Linux for zSeries

Device Drivers and Installation Commands April 30, 2002 (Draft)

Linux Kernel 2.4

LNUX-1103-08

Linux for zSeries

Device Drivers and Installation Commands April 30, 2002 (Draft)

Linux Kernel 2.4

LNUX-1103-08

Note

Before using this document, be sure to read the information in “Notices” on page 231.

Ninth Edition – (May 2002)

This edition applies to the Linux for zSeries kernel 2.4 patch (made in September 2001) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures

Tables

Summary of changes

This revision focuses on hardware and software enhancements for the zSeries environment as announced in April 2002.

Edition 9 changes

New Information

- Networking
  - Source VIPA
  - Virtual LAN (VLAN) support[1]
  - OSA-Express SNMP subagent[1]
  - IP Version 6 (IPv6)[1]
  - Query/Purge ARP[1]
  - Proxy ARP
  - Broadcast[1]
- DASD
  - DASD reserve and release
  - DASD device access by VOLSER
- Tape
  - 3590 tape drive support
  - Tape disciplines as modules
  - Tape display unit support
- Other
  - SCSI-over-Fibre-Channel driver (zfcp)[1]
  - Response to VARY ON/OFF events
  - Toggling CHPIDs logically on- and offline
  - Guide to interpreting syntax diagrams

Changed Information

- The kernel level is now 2.4.17.
- Clarified that the command specified by the ″vmhalt″ parameter must be in uppercase.
- Deleted sentence in DASD driver kernel parameter syntax description stating that ′autodetect,probeonly′ is the default if there is no ′dasd′ parameter. By default, autodetect is disabled.
- The QDIO module is now supplied in source-code form.

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

Deleted Information

- Information on ″make config″ and ″menuconfig″, and related examples

1. These functions are new as of the April 2002 announcement and require an appropriate processor and microcode level.

Edition 8 changes

New Information

- snIPL tool
- Setting up XPRAM dynamically
- Reference to *Dump Tools* manual

Changed Information

- Open-source LCS driver
- Corrected the separator (colon) between multiple guest-machine IDs in the `iucv` kernel parameter and `insmod` command line.
- Added references to VIPA requirement for kernel built with CONFIG_DUMMY kernel option.

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

Deleted Information

- References to CONFIG_IP_ALIAS kernel option

Edition 7 changes

New Information

- Option **u** added to fdasd. The option re-creates VTOC labels and keeps the partitions.
- VIPA (virtual IP addressing) chapter
- Installation instructions for qdio and qeth modules
- z90crypt cryptographic device driver

Changed Information

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

Deleted Information

- Parameter *buffer_count* (replaced by *mem_in_k*) in QETH

Edition 6 changes

New Information

- DASDview
- HiperSockets

Changed Information

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

- OSA-Express – Clarifying editorial changes
- fdasd – **r** and **s** options added

- zIPL – Use of quotes

Edition 5 changes

New Information

- VIPA
- IP address takeover
- HiperSockets

Changed Information

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

- OSA-Express – Updates for new cards
- IUCV – New API
- Kernel building restrictions.

Edition 4 changes

New Information

- Channel device layer
- Device file system
- Tape driver
- zIPL utility
- modprobe, lsmod, depmod summarized

Changed Information

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

- DASD – Add commands for creating device nodes and more details of naming scheme and channel device layer description added
- XPRAM – note on reusing partitions
- CTC channel device layer description added
- Gigabit Ethernet section expanded for all OSA-Express devices and channel device layer description added
- Console section expanded

Edition 3 changes

New Information

- Gigabit Ethernet driver restriction

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

Edition 2 changes

New Information

- CTC/ESCON VM channel subset selection for TCP/IP

Changed Information

- CTC/ESCON module parameter syntax
- VM Minidisk driver revisions
- ’mem’ parameter additional option
- Console parameter change for P/390

About this document

This document describes the drivers available to Linux for the control of zSeries devices and attachments. It also provides information on commands and parameters relevant to the installation process.

The drivers described herein have been developed with version 2.4.17 of the Linux kernel. If you are using a later version of the kernel, the kernel parameters may be different from those described in this document.

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

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

Note: For tools related to taking and analyzing system dumps, see the manual *Linux for zSeries Using the Dump Tools*, LNUX-1108.

How this document is organized

The first part of this document contains general information relevant to all Linux for zSeries device drivers.

Parts two and three consist of chapters specific to individual device drivers. (Part two describes the drivers for zSeries hardware; part three describes the network device drivers.)

Part four contains information on the Linux and zSeries commands and parameters used in installing.

These chapters are followed by a reference section containing summaries of the command syntax of the drivers, a glossary and an index.

Who should read this document

This document is intended for :

- System administrators who wish to configure a Linux for zSeries system

Assumptions

The following general assumptions are made about your background knowledge:

- You have an understanding of Linux and zSeries terminology.
- You are familiar with Linux device driver software.
- You have an understanding of basic computer architecture, operating systems, and programs.
- You are familiar with the zSeries devices attached to your system. (zSeries knowledge should not be required, as the code specific to the zSeries hardware is provided by IBM.)

Understanding syntax diagrams

This section describes how to read the syntax diagrams in this manual.

To read a syntax diagram follow the path of the line. Read from left to right and top to bottom.

- The ►►─── symbol indicates the beginning of a syntax diagram.
- The ───► symbol, at the end of a line, indicates that the syntax diagram continues on the next line.
- The ►─── symbol, at the beginning of a line, indicates that a syntax diagram continues from the previous line.
- The ───►◄ symbol indicates the end of a syntax diagram.

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

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

Case sensitivity
Unless otherwise noted, entries are case sensitive.

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

*	Asterisk
:	Colon
,	Comma
=	Equals sign
-	Hyphen
//	Double slash
()	Parentheses
.	Period
+	Add
$	Dollar sign

For example:

```
iucv LINUX2:VMTCPID
```

Variables
An *italicized* lowercase word indicates a variable that you must substitute with specific information. For example:

```
►►── -p interface─────────────────────────────────────────►◄
```

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

```
►►──iucv=USERID───────────────────────────────────────────►◄
```

Repetition

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

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

Defaults

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

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

Required Choices

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

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

Optional Choice

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

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

Understanding syntax diagrams

Part 1. Linux for zSeries Device drivers overview

This section describes principles common to different device drivers.

Chapter 1. Common device support

Before Linux for zSeries can use a device the associated device driver must be available to the Linux kernel. This can be achieved either by configuring the Linux kernel to use the channel device layer, compiling the device driver into the kernel or by invoking the driver as a module. The options for each driver are shown in the following table:

Device driver	**Able to use Channel device layer**	**Kernel**	**Module**
DASD	no	yes	yes
XPRAM	no	yes	yes
Hardware console	no	yes	no
3215 console	no	yes	no
Tape	no	yes	yes
CTC/ESCON	yes	yes	yes
IUCV	no	yes	no
LCS	yes	yes	yes
QETH	yes	no	yes
Crypto	no	no	yes
SCSI-over-Fibre Channel (zfcp)	no	yes	yes

A description of how to build the kernel including device drivers is given in Appendix B, "Kernel building" on page 213.

The parameters for the kernel resident device drivers are held in the parameter line file which is created during the installation of Linux .

- If you are using an LPAR or native installation this is parameter `-p` in the `zipl` parameter file.
- For a VM installation, include the parameter in the `PARM LINE A` file.

For the format of this file see Chapter 19, "Overview of the parameter line file" on page 207.

Drivers which are not kernel resident are loaded into Linux with their parameters by means of the `insmod` or `modprobe` command. See "insmod - Load a module into the Linux kernel" on page 171 or "modprobe - Load a module with dependencies into the Linux kernel" on page 173 for the syntax.

Because the zSeries architecture differs from that used by the Intel PC and other machines the I/O concepts used by zSeries device drivers are also different.

Linux was originally designed for the Intel PC architecture which uses two cascaded 8259 programmable interrupt controllers (PIC) that allow a maximum of 15 different interrupt lines. All devices attached to that type of system share those 15 interrupt levels (or IRQs). In addition, the bus systems (ISA, MCA, EISA, PCI, etc.) might allow shared interrupts, different polling methods or DMA processing.

Common device support

Unlike other hardware architectures, zSeries implements a channel subsystem that provides a unified view of the devices attached to the system. Although a large variety of peripheral attachments are defined for the zSeries architecture, they are all accessed in the same manner using I/O interrupts. Each device attached to the system is uniquely identified by a subchannel, and the zSeries architecture allows up to 64,000 devices to be attached.

To avoid the introduction of a new I/O concept to the common Linux code, Linux for zSeries preserves the IRQ concept and systematically maps the zSeries subchannels to Linux as IRQs. This allows Linux for zSeries to support up to 64,000 different IRQs, each representing a unique device.

The unified I/O access method incorporated in Linux for zSeries allows the operating system to implement all of the hardware I/O attachment functionality that each device driver would otherwise have to provide itself. A common I/O device driver is provided which uses a functional layer to provide a generic access method to the hardware. The driver comprises a set of I/O support routines, some of which are common Linux interfaces, while others are Linux for zSeries specific:

`get_dev_info()`
: Allows a device driver to find out what devices are attached (visible) to the system, and to determine their current status.

`request_irq()`
: Assigns the ownership of a specific device to a device driver.

`free_irq()`
: Releases the ownership of a specific device.

`disable_irq()`
: Prevents a specific device from presenting interrupts to the device driver.

`enable_irq()`
: Allows a device to present I/O interrupts to the device driver.

`do_IO()`
: Initiates an I/O request.

`halt_IO()`
: Terminates the I/O request that is currently being processed by the device.

`do_IRQ()`
: This is an interrupt pre-processing routine that is called by the interrupt entry routine whenever an I/O interrupt is presented to the system. The `do_IRQ()` routine determines the interrupt status and calls the device specific interrupt handler according to the rules (flags) defined by `do_IO()`.

More information on these routines can be found in the Linux source directory, `/usr/share/doc/kernel-doc-2.4.17/s390/cds.txt` or the `Documentation/s390/cds.txt` file in the Linux kernel source tree.

Chapter 2. Partitioning DASD

Partitioning is a means of dividing a single DASD into several logical disks. A partition is a contiguous set of blocks on a DASD which is treated by Linux as an independent disk. The partition table defines the extents of partitions on a DASD.

To partition SCSI disks, use **fdisk** rather than **fdasd**.

Prerequisites

To set up and use DASD partitions you must take these steps:

1. Use `dasdfmt` tool (see “dasdfmt - Format a DASD” on page 132) with the (default) `'-d cdl'` option to format the DASD with the IBM compatible disk layout.
2. Use the `fdasd` tool (see “fdasd – DASD partitioning tool” on page 144) to create or add partitions. After this your partitions should appear in the device file system in the `/dev/dasd/...` directory.
3. Use the Linux `mke2fs` tool (see “mke2fs - Create a file system on DASD” on page 179) to create a file system on the partition or the `mkswap` tool to use the partition as swap space. If you use `mke2fs` you must ensure that the blocksize specified matches that which was defined with `dasdfmt`.
4. If you have created a file system you may mount the partition to the mount point of your choice in Linux .

If a DASD is formatted in the normal Linux disk layout (`dasdfmt` option `-d ldl`) it is not possible to create partitions on it and the whole DASD must be accessed as a single partition.

Why use partitions?

There are several reasons why you may want to partition your data. The most common of these are:

- **Encapsulate your data.** As corruption of the file system is likely to be local to a single partition, data in other partitions should survive.
- **Increase disk space efficiency.** You can have different partitions with different block sizes to optimize your usage. To improve performance a large blocksize is better, but this can be wasteful of space. In general wastage amounts to half a block for each file, which becomes significant for small files. For these reasons it is usually better to store small files in a partition with a small blocksize and large files in one with a larger blocksize.
- **Limit data growth.** Runaway processes or undisciplined users can consume so much disk space that the operating system no longer has room on the hard drive for its bookkeeping operations. This will lead to disaster. By segregating space you ensure that processes other than the operating system die when the disk space allocated to them is exhausted.

Restrictions

There are some limitations to the current implementation and some precautions you should take in using it. These are:

Partitioning DASD

- You can only partition ECKD disks formatted with the new disk layout (`dasdfmt` option `-d cdl`).
- No more than three partitions can be created on any one physical volume. This restriction is a result of the scheme of allocating Linux major and minor numbers to the partitions. (Increasing the number of partitions per DASD would drastically reduce the number of DASD that could be mounted in a system).
- You are advised to use `fdasd` to create or alter partitions as it checks for errors. If you use another partition editor it is your responsibility to ensure that partitions do not overlap. If they do, data corruption will occur.
- To avoid wasting disk space you should leave no gaps between adjacent partitions. Gaps are not reported as errors, but a gap can only be reclaimed by deleting and recreating one or other of the surrounding partitions and rebuilding the file system on it.
- A disk need not be partitioned completely. You may begin by creating only one or two partitions at the start of your disk and convert the remaining space to a partition later (perhaps when performance measurements have given you a better value for the blocksize).
- There is no facility for moving, enlarging or reducing partitions as `fdasd` has no control over the file system on the partition. You only can delete and recreate them. If you change your partition table you will lose the data in all altered partitions. It is up to you to preserve the data by copying it to another medium.

The partition table

The partition table is an index of all the partitions on a DASD. In Linux for zSeries we do not use the normal Linux partition table, but as in other zSeries operating systems we use a VTOC (Volume Table Of Contents) to store this index information.

In the zSeries a VTOC is used to access data on any DASD. It is an index in a special format which contains pointers to the location of every data set on the volume. In Linux for zSeries these data sets form our Linux partitions.

zSeries disk layout (VTOC)

Operating systems on a mainframe (z/OS, OS/390, VM/ESA and VSE/ESA) expect a standard DASD format. In particular the format of the first two tracks of a DASD is defined by this standard. In order to share data with other operating systems Linux for zSeries can use DASD in the common format. The first two tracks are then unavailable to Linux (they have non-Linux -standard variable blocksizes for example) but this is transparent to the user (apart from a slight loss in disk capacity).

Volume label

The third block of the first track of the DASD (cylinder 0, track 0, block 2) contains the volume label. This block has a four byte key and an eighty byte data area. The contents are:

1. **Key**

   This identifies the block as a volume label. It must contain the four EBCDIC[2] characters 'VOL1'.
2. **Label identifier**

2. The conversion to EBCDIC will be carried out by the `fdasd` tool.

This 4-byte field is identical to the Key field.

3. **VOLSER (volume serial number)**

   This identifies by serial number the volume on which the partition resides or will reside. A volume serial number is one to six alphanumeric, national ($, #, @) or special characters in the EBCDIC[2] code. If it contains special characters other than hyphens it must be enclosed in apostrophes. If the `VOLSER` is shorter than six characters it is padded with trailing blanks (converted to EBCDIC code 0x40). Do not code a volume serial number as `SCRTCH`, `PRIVAT`, `MIGRAT` or `L`*nnnnn* (L with five digits) as these are reserved labels in other zSeries operating systems.

4. **VTOC address**

   This is a five byte field containing 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 will contain EBCDIC space characters (code 0x40).

VTOC

The VTOC is located in the second track (cylinder 0, track 1). It contains a number of 144 byte labels which consist of a 44 byte key and a 96 byte data area.

The first label is a format 4 DSCB describing the VTOC itself. The second label is a format 5 DSCB containing free space information. (If the volume has more than 65536 tracks the format 5 DSCB will contain binary zeroes and will be followed by a format 7 DSCB containing the free space information.) After these follow format 1 DSCBs for each of the partitions. Each label is written in a separate block.

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

The VTOC can be displayed with standard zSeries tools such as VM/DITTO. A Linux DASD with physical device number 0x'0193', volume label 'LNX001', and three partitions might be displayed like this:

```
                            VM/DITTO DISPLAY VTOC                  LINE 1 OF 5
===>                                                          SCROLL ===> PAGE

CUU,193 ,VOLSER,LNX001   3390, WITH   100 CYLS, 15 TRKS/CYL, 58786 BYTES/TRK

--- FILE NAME --- (SORTED BY =,NAME  ,) ---- EXT    BEGIN-END     RELTRK,
1...5...10...15...20...25...30...35...40.... SQ   CYL-HD   CYL-HD      NUMTRKS
 *** VTOC EXTENT ***                         0     0  1     0  1      1,1
LINUX.VLNX001.PART0001.NATIVE                0     0  2    46 11      2,700
LINUX.VLNX001.PART0002.NATIVE                0    46 12    66 11    702,300
LINUX.VLNX001.PART0003.NATIVE                0    66 12    99 14   1002,498
 *** THIS VOLUME IS CURRENTLY 100 PER CENT FULL WITH      0 TRACKS AVAILABLE

PF  1=HELP      2=TOP       3=END       4=BROWSE    5=BOTTOM    6=LOCATE
PF  7=UP        8=DOWN      9=PRINT    10=RGT/LEFT 11=UPDATE   12=RETRIEVE
```

In Linux (using the device file system) this DASD might appear so:

```
[root@host /root]# ls -l /dev/dasd/0193/
total 0
brw-------   1 root     root      94,  12 Jun  1  2001 device
brw-------   1 root     root      94,  12 Jun  1  2001 disc
brw-------   1 root     root      94,  13 Jun  1  2001 part1
brw-------   1 root     root      94,  14 Jun  1  2001 part2
brw-------   1 root     root      94,  15 Jun  1  2001 part3
```

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

Part 2. Linux for zSeries — zSeries device drivers

The zSeries device drivers are:

Chapter 3. Linux for zSeries DASD device driver

DASD overview

The DASD device driver in Linux for zSeries takes care of all real or emulated DASD (Direct Access Storage Device) that can be attached to a zSeries via the channel subsystem.[3] The class of devices named DASD includes a variety of physical media, on which data is organized in blocks and/or records which can be accessed (read or written) in random order.

Traditionally these devices are attached to a control unit connected to a zSeries I/O channel. In modern systems these have been largely replaced by emulated DASD, such as the internal disks of the Multiprise family, the volumes of the RAMAC virtual array, or the volumes of the Enterprise Storage Server. These are completely virtual representations of DASD in which the identity of the physical device is hidden.

Each device protocol supported by the DASD device driver is supplied as a separate module, which can be added and removed at run-time. The DASD core module is named `dasd_mod` and the device format modules are named `dasd_eckd_mod`, `dasd_fba_mod` and `dasd_diag_mod`. All modules are loaded by issuing the command '`modprobe` *`module_name`*'. As well as the parameter '`dasd=`' which specifies the volumes to be operated by the DASD device driver, the core module has an additional parameter '`dasd_disciplines=`*`mod_names`*' which enables a selection of device protocols to be auto-loaded during initialization of the core module.

The DASD device driver is capable of accessing an arbitrary number of devices. The default major number for DASD (94) can only address 64 DASD (see below for details), so additional major numbers (typically descending from 254) are allocated dynamically at initialization or run time. The only practical limit to the number of DASD accessible is the range of major numbers available in the dynamic allocation pool.

Each DASD configured to the system uses 4 minor numbers.

- The first minor number always represents the entire device, including IPL, VTOC and label records.
- The remaining three minor numbers represent partitions of the device as defined in the partition table.

DASD naming scheme using devfs

We recommend that you use the device filesystem (`devfs`) to have a comfortable and easy-to-use naming scheme for DASD. (The older naming scheme is described in "DASD naming scheme without devfs" on page 12.) DASD nodes generated by `devfs` have the general format '`/dev/dasd/`<*devno*>/<*type*>', where '*devno*' is the unit address of the device and '*type*' is a name denoting either a partition on that device or the entire device. *devno* must be four hexadecimal digits, padded with leading zeroes if necessary.

3. SCSI disks attached via an OpenFCP adapter are not classified as DASD. They are handled by the zfcp driver (see Chapter 8, "SCSI-over-Fibre Channel driver" on page 55).

For example /dev/dasd/01a1/disc refers to the whole of the disk with device address 0x01a1 and /dev/dasd/01a1/part1 refers to the first partition on that disk.

The entire physical device can also be referred to as node '/dev/dasd/<*devno*>/device' which is equivalent to '.../disc'; the difference being that the /device node is always available whereas the /disc node is only available after the device has been formatted. (The /part<*x*> nodes are only available after the device has been formatted and partitioned.)

For example a device with address 0x0150 and 2 partitions will have these device node entries:

```
94   0   /dev/dasd/0150/device   - for the entire device (before formatting)
94   0   /dev/dasd/0150/disc     - for the entire device (after formatting)
94   1   /dev/dasd/0150/part1    - for the first partition
94   2   /dev/dasd/0150/part2    - for the second partition
```

DASD naming scheme without devfs

A Linux 2.2 or 2.4 system is restricted to 256 major device numbers, each holding 64 blocks of 4 minor numbers, giving a maximum of 16,384 DASD even if no numbers are used for other types of device. Every major number used for other devices reduces the maximum number of DASD by 64. If the device file system (devfs) is not activated the DASD device driver has a built in naming scheme for DASD according to Table 1. (You can override the built in scheme by creating customized nodes in the Linux /dev/ subdirectory.) These names are sufficient to access the maximum number of DASD accessible.

Table 1. DASD naming convention

Names	Number	Major/minor numbers (assuming dynamic allocation from 254)
dasda – dasdz	26	94:0 — 94:100
dasdaa – dasdbl	38	94:104 — 94:252
dasdbm – dasdzz	638	254:0 — 245:244
dasdaaa – dasdzzz	17576	245:248 — 131:148
Sum:	18278	

General DASD nodes have the format dasd<*x*>, or dasd<*x*><*p*>, where <*x*> is a letter identifying the device and <*p*> is a number denoting the partition on that device. The first form, dasd<*x*>, is used to address the entire disk. The second, dasd<*x*><*p*>, is used to address the partitions on this device.

For example /dev/dasda refers to the whole of the first disk in the system and /dev/dasda1 to the first partition on that disk.

They are typically created by:

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

If you have a large number of DASD you may wish to use a script to create them. An example of this for bash is:

```
`cat /proc/dasd/devices |
 sed 's/^.*[(]\([ 0-9]*\)[:]\([ 0-9]*\)[)].*\(dasd[a-z]*\)[:].*$/\1 \2 \3/g' |
 awk ' $1 {
printf "mknod /dev/%s b %d %d; mknod /dev/%s1 b %d %d;",$3,$1,$2,$3,$1,$2+1;
 }'`
```

A similar script may be written for csh or ksh.

Partitioned DASD

The DASD device driver is embedded into the Linux generic support for partitioned disks. This implies that you can have any kind of partition table known to Linux on your DASD, such as the MSDOS or Amiga partition scheme. However none of the partition schemes built in to Linux to support platforms other than zSeries will preserve zSeries IPL, VTOC and label records.

'IBM label' partition scheme:

To ensure compatibility with other zSeries operating systems the IBM-label partition scheme has been added to Linux . This scheme currently supports VOL1 (zSeries), LNX (Linux) and CMS (VM/ESA) labeled disks, as well as unlabeled disks which are treated equivalently to LNX-labeled disks. The disk layout of the different types is shown in Figure 1.

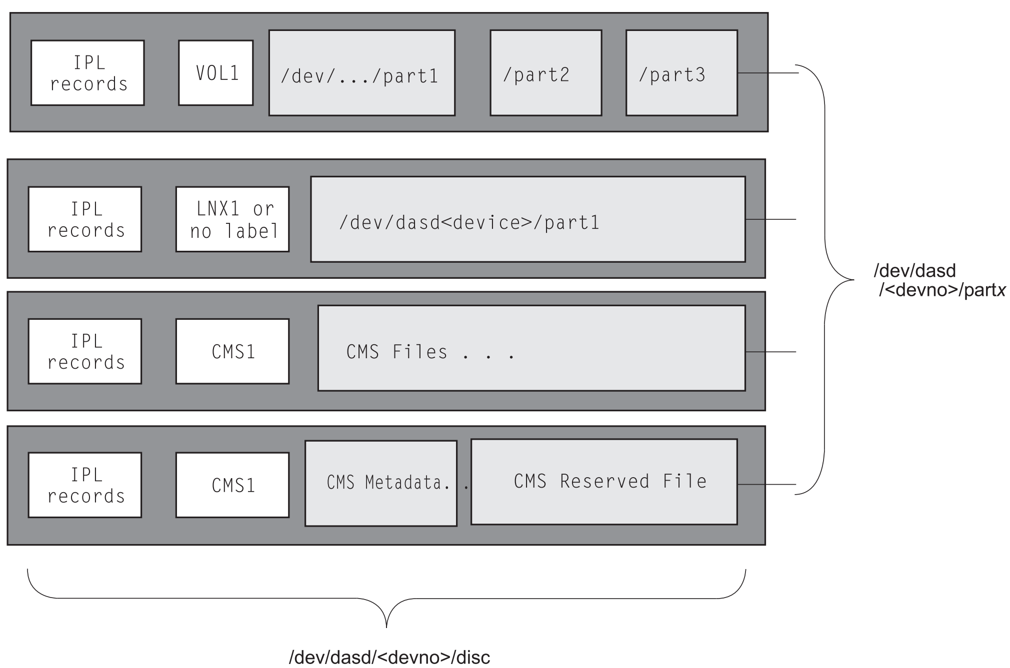


Figure 1. Partition scheme for VOL1, LNX and CMS labeled disks

The first of these examples shows a DASD with compatible disk layout. The second shows a disk in an LPAR or native mode, or a full pack minidisk (dedicated DASD) in VM. The third and fourth examples are VM specific.

VOL1 labeled disk:

DASD device driver

This disk layout (also known as the *compatible disk layout*, or CDL) is compliant with IBM guidelines for volume labeling of ECKD volumes. This enables non-Linux operating systems to access Linux volumes online, for example for backup and restore.

Partitioning support for such disks means that the VTOC contains data in the IBM standard, namely one 'format 4' label describing the VTOC, one 'format 5' label[4] and one to three 'format 1' labels[5] describing the extents of the volume (partitions to Linux). The partitions are created and modified by the `fdasd` tool (see "fdasd – DASD partitioning tool" on page 144).

LNX1 labeled disk or non-labeled volume:

These disks are implicitly reserved for use by Linux . The disk layout reserves the IPL and label records for access through the 'entire disk' device. All remaining records are grouped into the first (and only) partition.

CMS1 labeled disk:

Handling of these disks depends on the content of the CMS filesystem. If the volume contains a CMS filesystem it will be treated equivalently to a LNX labeled volume. If the volume is a CMS reserved volume [6] the CMS reserved file is represented by the first and only partition. IPL and label records as well as the metadata of the CMS filesystem are reserved for access through the 'entire disk' device.

See Chapter 2, "Partitioning DASD" on page 5 for more information.

DASD features

The DASD device driver can access devices according to Table 2 by its built in CCW interface.

Table 2. Supported DASD devices

Device format	Control unit type/model	Device type/model
ECKD (Extended Count Key Data)	3990(2105)/**	3380/**
	3990(2105)/**	3390/**
	9343/**	9345/**
FBA (Fixed Block Access)	6310/??	9336/??
	3880/**	3370/**

'*' signifies any digit.

The DASD device driver is also known to work with these devices:
- Multiprise internal disks
- RAMAC
- RAMAC RVA
- Enterprise Storage Server (Seascape) virtual ECKD-type disks

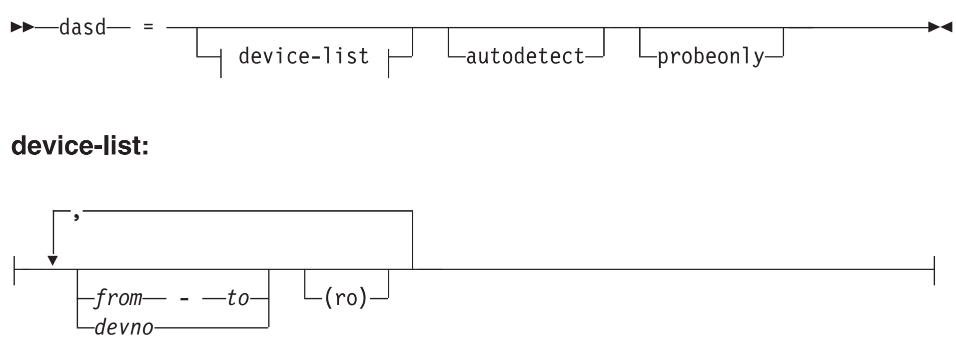
4. The 'format 5' label is required by other operating systems but is unused by Linux and set to zeroes by `fdasd`.
5. Other operating systems may create up to seven 'format 1' labels.
6. CMS reserved volume means a volume that has been reserved by a 'CMS RESERVE fn ft fm' command.

Linux for zSeries implements a maximum of three partitions per volume. The available disk space for partitions is the whole volume, skipping the first blocks according to the scheme outlined in Figure 1 on page 13.

DASD kernel parameter syntax

The DASD driver is configured by a kernel parameter added to the parameter line:

DASD kernel parameter syntax



where:

autodetect
: causes the driver to consider any device operational at the time of IPL as a potential DASD and allocate a device number for it. Nevertheless the devices which are not DASD, or do not respond to the access methods known to the kernel, will not be accessible as DASD. Any ’open’ request on such a device will return `ENODEV`. In `/proc/dasd/devices` these devices will be flagged ’unknown’.

probeonly
: causes the DASD device driver to reject any ’`open`’ syscall with `EPERM`.

autodetect,probeonly
: behaves in the same way as above, but additionally all devices which are accessible as DASD will refuse to be opened, returning `EPERM`.

from-to defines the first and last DASD in a range. All DASD devices with addresses in the range are selected. It is not necessary for the *from* and *to* addresses to correspond to actual DASD.

devno defines a single DASD address.

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

The DASD addresses must be given in hexadecimal notation with or without a leading `0x`, for example 0191 or 5a10.

DASD device driver

If you supply one or more kernel parameters `dasd=`*`device-list1`* `dasd=`*`device-list2`* `...` the devices are processed in order of appearance in the parameter line. Devices are ignored if they are unknown to the machine, non-operational, or set off-line.[7]

If autodetection is turned on a DASD device is allocated in Linux for every device operational at the time of initialization of the driver, in order of ascending subchannel numbers.

Note that the autodetection option may cause confusing results if you change your I/O configuration between two IPLs, or if you are running as a guest operating system in VM/ESA, because the devices might appear with different major/minor combinations in the new IPL .

DASD kernel example (using devfs)

```
dasd=192-194,5a10(ro)
```

This reserves major/minor numbers and nodes as follows:

```
94  0  /dev/dasd/0192/device - for the entire device  192
94  0  /dev/dasd/0192/disc   - for the entire device  192 (formatted)
94  1  /dev/dasd/0192/part1  - first  partition on    192
94  2  /dev/dasd/0192/part2  - second partition on    192 (if used)
94  3  /dev/dasd/0192/part3  - third  partition on    192 (if used)

94  4  /dev/dasd/0193/device - for the entire device  193
94  4  /dev/dasd/0193/disc   - for the entire device  193 (formatted)
94  5  /dev/dasd/0193/part1  - first partition on     193
94  6  /dev/dasd/0193/part2  - second partition on    193 (if used)
94  7  /dev/dasd/0193/part3  - third  partition on    193 (if used)

94  8  /dev/dasd/0194/device - for the entire device  194
94  8  /dev/dasd/0194/disc   - for the entire device  194 (formatted)
94  9  /dev/dasd/0194/part1  - first partition on     194
94 10  /dev/dasd/0194/part2  - second partition on    194 (if used)
94 11  /dev/dasd/0194/part3  - third  partition on    194 (if used)

94 12  /dev/dasd/5a10/device - for the entire device 5a10 (read only)
94 12  /dev/dasd/5a10/disc   - for the entire device 5a10 (formatted,read only)
94 13  /dev/dasd/5a10/part1  - first  partition on   5a10 (read only)
94 14  /dev/dasd/5a10/part2  - second partition on   5a10 (if used,read only)
94 15  /dev/dasd/5a10/part3  - third  partition on   5a10 (if used,read only)
```

The '`/device`' node is registered by the DASD device driver during initialization and is always available.

All other nodes are generated by the device filesystem support and are only available after formatting (`/disc`) and partitioning (`/part1` to `/part3`).

7. Currently there is no check for duplicate occurrences of the same device number.

DASD module parameter syntax

The following are the DASD driver module parameters:

DASD module parameter syntax

```
>>--insmod--dasd_mod--dasd-- = --+-------------------+--+-----------+-->
                                 '-| device-list |-'    '-probeonly-'

>--+------------+-----------------------------------------------------><
   '-autodetect-'
```

device-list:

```
   .-,------------------------------.
   V                                |
|----+-from-- - --to-+--+------+----+--------------------------------|
     '-devno---------'  '-(ro)-'
```

where:

dasd_mod
: is the name of the device driver module

dasd
: is the start of the parameters

and all other parameters are the same as the DASD kernel parameters described in “DASD kernel parameter syntax” on page 15.

DASD module example

```
insmod dasd_mod dasd=192-194,5a10(ro)
```

The details are the same as “DASD kernel example (using devfs)” on page 16.

DASD dynamic attach and enable/disable

Dynamic device attach enables Linux for zSeries to deal with DASD devices which are dynamically attached to a running system. In addition it allows the operator to dynamically enable or disable devices.

The system will take appropriate action automatically when a dynamic attach occurs. When a device is attached, the system will try to initialize it according to the configuration of the DASD device driver.

Note

Detachment in VM of a device still open or mounted in Linux may trigger a limitation in the Linux kernel 2.4 common code and cause the system to hang or crash.

DASD device driver

The /proc filesystem node /proc/dasd/devices provides an interface which can be used to dynamically configure the DASD device driver's settings. This interface provides some elementary support and does not provide a base for full DASD management. For example, there is the capability to add a device range by using the add directive, but there is no corresponding remove directive. Therefore, the following commands should be used with care, as some configuration errors can not be corrected without a reboot of the system.

- To add a range to the list of known devices:

```
echo "add device range=devno-range" >> /proc/dasd/devices
```

  This updates the currently running system. It does not update any persistent information on the volumes.

- To disable devices manually:

```
echo "set device range=devno-range off" >> /proc/dasd/devices
```

  This resets the state of the devices as if they had never been defined as DASD. All buffers referring to the devices are flushed unconditionally or terminated with an error.

- To enable devices manually. :

```
echo "set device range=devno-range on" >> /proc/dasd/devices
```

  This tries to initialize devices as if they had just been added to the system.

DASD reserve and release

Especially for cluster solutions, the access to a volume must be coordinated and sometimes is necessary to get exclusive access to such a volume. To enable this, the DASD device driver supports the 'Device Reserve', 'Device Release' and 'Unconditional Reserve' functions for all ECKD devices.

The functions are implemented according to the ECKD architecture. This means that the function waits for the interrupt (synchronous I/O) and returns as soon as the related interrupt occurs. If the interrupt is missing for some reason, e.g., a reserve command was given and the device is locked by someone else, the I/O returns with an I/O error after a certain time.

DASD 'reserve', 'release' and 'unconditional reserve' functionality is implemented using the DASD IOCTL interface:

- BIODASDRSRV - to reserve a device
- BIODASDRLSE - to release a device
- BIODASDSLCK - to reserve a device even if it was already reserved by someone else (steal lock)

Notes:

1. Use these functions with care because you might block others. You can block entire users or even worse the entire OS if you reserve a shared device.

   At least the 'unconditional release' ('steal lock') function should be used very carefully to prevent data corruption if a lock is stolen and the OS does not realize it.

2. A reserved device cannot be analyzed during IPL and therefore cannot be accessed.

DASD device access by VOLSER

This section describes how to access DASD devices by VOLSER.

Description

Devices (like DASD) are accessed by user space programs by a special file (the device node). These files are mapped to the kernel space representation of that device (a major and a minor number). The DASD driver assigns these numbers according to the sequence the devices are attached or found, which can change between kernel boots. The same filename, therefore, can lead to a completely different device if someone changes the number of devices that are visible to Linux. With devfs, this is prevented by the dynamic creation of device nodes. These include the (host) device number as a part of the file (path) name. For example, the device with the device number 0192 is given the filename /dev/dasd/0192/device. As long as the device number does not change, the same filename will always specify the same device.

If for some reason the device number is changed (with the attach command in VM or by changing the IOCP definition), the old filename either does not exist or it can also specify another device if the new device is given the old device number. This problem is addressed by device access by VOLSER.

If devfs is enabled, this will create a link in /dev/labels with a name equal to the volume serial number (which is written to the VTOC area) and pointing to the correct device directory. For example, if there is a device with device number 0192 and the VOLSER is DSK001, the link /dev/labels/DSK001 will point to /dev/dasd/0192.

Usage

The VOLSER can be set by the tool dasdfmt when formatting (the default is to use ″0X<device number>″) or later by fdasd when creating the partitions. Changing the VOLSER will automatically remove and create the correct link in /dev/labels. There is one exception, however. It is possible to assign two devices the same VOLSER. In this case, only one link for the first device (in the order the DASDs are initialized) is created. This will make the link ambiguous again and should be avoided. Furthermore, if the VOLSER on the device that created the link is changed, the previously hidden device will not automatically create the link. This will happen on the next reboot or can be forced by entering

```
blockdev --rereadpt /dev/dasd/<devno>/device
```

Notes:

1. When creating VOLSERs, all lowercase characters are automatically converted to uppercase. Special characters (neither alphabetic nor numeric) permitted by **dasdfmt** and **fdasd** should be valid in the /dev/labels node, but the behavior could be unexpected. The node might not even be created if unsuitable characters are encountered.
2. The specification of VOLSER is case-sensitive and must always match the entry in /dev/labels. VOLSERs are normally uppercase.

Example

To mount DSK001, the entry in /etc/fstab could be changed from

```
/dev/dasd/0192/part1 /usr ext2 defaults 0 0
```

to

```
/dev/labels/DSK001/part1 /usr ext2 defaults 0 0
```

DASD – Preparing for use

1) Low level format

Before using an ECKD type DASD as a Linux for zSeries disk the device must be formatted. This should be done from Linux for zSeries by issuing an `ioctl` called `BIODASDFMT` on the file descriptor of the opened volume `/dev/dasd/<devno>/<device>`. The utility `dasdfmt` is provided as an interface to this `ioctl` with additional checking.

Caution: Using `dasdfmt` or the raw `ioctl` can potentially destroy your running Linux for zSeries system, forcing you to reinstall from scratch.

See the help given by `dasdfmt -help` and “dasdfmt - Format a DASD” on page 132 for further information. The `dasdfmt` utility calls several processes sequentially. Take care to allow sufficient time for each process to end before attempting to enter an additional command.

We recommend you set `blksize` to 1024 or higher (ideally 4096) because the `ext2fs` file system uses 1KB blocks and 50% of capacity will be unusable if the DASD blocksize is 512 bytes.

The formatting process can take a long time (hours) for large DASD.

2) Create partitions

After formatting the device with the common disk layout (CDL), (this is the default option for `dasdfmt`), the partitions have to be created. This is done by the `fdasd` tool which writes some labels to the device (see “Partitioned DASD” on page 13) and calls the device driver to re-read the partition table. `fdasd` is a user-space program with a command-line interface. See “fdasd – DASD partitioning tool” on page 144 for more information. The restriction of four minor numbers per DASD in the current implementation means that no more than three partitions can be created on a single DASD.

Note: If you do not use the common disk layout you are not able to partition the device. In this case only one partition per DASD is supported.

3) Make a file system

Before using a DASD as a Linux for zSeries data disk, you must create a file system on it. (A DASD for use as a swap device or paging space only needs to be defined as such.) Using `mkxxfs` (replacing *xx* with the appropriate identifier for the file system – for example use `mke2fs` for an ext2 file system) you can create the file system of your choice on that volume or partition .

It is recommended that you build your file system on the partitions of the DASD (`/dev/dasd/<devno>/<part>`, and so on), rather than the whole volume.

Note that the blocksize of the file system must be larger than or equal to the blocksize given to the `dasdfmt` command. It is recommended that the two blocksize values be equal.

You must enable `CONFIG_DASD`, `CONFIG_DASD_ECKD` and `CONFIG_DASD_FBA` in the configuration of your current kernel to access IBM DASD.

DASD API (ioctl interface)

The `ioctl` interface of the DASD device driver follows the common format:

```
int ioctl (int fd, int command, xxx)
```

The argument '`fd`' is a descriptor of an open file. '`command`' is the action requested and the third argument '*`xxx`*' is a pointer to a data structure specific to the request.

The `ioctl` commands specific to the DASD device driver are:

DASDAPIVER
returns the version of the DASD device driver API.

BIODASDDISABLE
disables the device.

BIODASDENABLE
enables the device

BIODASDFMT
formats the device with a specified blocksize.

BIODASDINFO
returns status information for the device.

BIODASDINFO2
returns status information for the device (including format and features).

BIODASDPRRD
reads profiling information.

BIODASDPRRST
resets profiling information.

BIODASDPSRD
returns performance statistics for the device.

BIODASDRLSE
requests release of a reserved device.

BIODASDRSRV
requests reserve of a device.

BIODASDSATTR
sets attributes (cache operations) of the device.

BIODASDSLCK
requests unconditional reserve ('steal lock') of a device.

In addition, the DASD device driver shares a number of common ioctl commands with most other block device drivers:

HDIO_GETGEO
get the device geometry.

BLKGETSIZE
get the device size in blocks.

BLKGETSIZE64
get the device size in blocks (64-bit return value).

BLKRRPART
re-read the partition table.

BLKSSZGET
get the sector size of the device.

BLKROSET
set or change the read-only flag of the device.

BLKROGET
get the current setting of the read-only flag of the device.

BLKRASET
set or change the number of read-ahead buffers of the device.

BLKRAGET
get the current number of read-ahead buffers of the device

BLKFLSBUF
flush the buffers.

BLKPG
handle the partition table and disk geometry.

BLKELVGET
get elevator.

BLKELVSET
set elevator.

If you need more `ioctl` functionality for your applications you may register your own `ioctl` commands to the DASD device driver. This is done using the function:

```
dasd_ioctl_no_register   (struct module     *owner,
                          int               no,
                          dasd_ioctl_fn_t   handler)
```

A previously added `ioctl` command can be deleted using:

```
dasd_ioctl_no_unregister (struct module     *owner,
                          int               no,
                          dasd_ioctl_fn_t   handler)
```

These dynamically added `ioctls` are scanned if none of the statically defined commands fulfils the requested command. If no related command is found in the static or in the dynamic list the driver returns `'ENOTTY'`.

For more information about `ioctl` see the `ioctl` man page or the public Linux documentation.

Examples of the implementation of the DASD `ioctl` interface can be found in the sections about DASD tools, in particular `dasdfmt` (“dasdfmt - Format a DASD” on page 132) and `fdasd` (“fdasd – DASD partitioning tool” on page 144).

DASD restrictions

- Note that the `dasdfmt` utility can only format volumes containing a standard record zero on all tracks. If your disk does not fulfill this requirement (for example if you re-use an old volume, or access a brand new disk or one having an unknown history), you should additionally use a device support facility such as ICKDSF (in z/OS, OS/390, VM/ESA, VSE/ESA or stand-alone) before doing the `dasdfmt` for the low-level format.
- The DASD device driver does not support Parallel Access Volumes (PAV), either static or dynamic. Even if this capability might work under VM using LVM, this is not supported and has not been tested.
- The size of any swap device or file may not exceed 2 GB. Similarly, the limit for the main memory that can be defined is slightly less than 2 GB.

Chapter 4. Linux for zSeries XPRAM device driver

The zSeries architecture in 31 bit mode supports the access of only 2 GB (gigabytes) of main memory. To overcome this limitation additional memory can be declared and accessed as expanded memory. For compatibility reasons this expanded memory can also be declared in the 64-bit mode of zSeries. The zSeries architecture allows applications to access up to 18 EB (exabytes) of expanded storage (although the current hardware can be equipped with at most 64 GB of real memory). The memory in the expanded storage range can be swapped in or out of the main memory in 4 KB blocks.

An IPL (boot) of Linux for zSeries does not reset expanded storage, so it is persistent through IPLs and could be used, for example, to store diagnostic information. The expanded storage is reset by an IML (power off/on).

The XPRAM device driver is a block device driver that enables Linux for zSeries to access the expanded storage. Thus XPRAM can be used as a basis for fast swap devices and/or fast file systems.

XPRAM features

- Automatic detection of expanded storage.

  (If expanded storage is not available, XPRAM fails with a log message reporting the lack of expanded storage.)
- Storage can be subdivided into up to 32 partitions.
- Device driver major number: 35.
- Partition minor numbers: 0 through 31.
- Hard sector size: 4096 bytes.
- The device file system (`devfs`) is supported. If `devfs` is switched on during kernel build `XPRAM` automatically generates the device nodes `/dev/slram/0` through `/dev/slram/31`.

Note on reusing XPRAM partitions

It is possible to reuse the filesystem or swap device on an XPRAM device or partition if the XPRAM kernel or module parameters for the new device or partition match the parameters of the previous use of XPRAM. If you change the XPRAM parameters for a new use of XPRAM you must make a new filesystem (for example with `mke2fs`) or a new swap device for all partitions that have changed. A device or partition has changed if its size has changed. All partitions following one which has changed are treated as changed as well (even if their sizes have not been changed).

XPRAM kernel parameter syntax

The kernel parameter is optional. If omitted the default is to define the whole expanded storage as one partition. The syntax is:

XPRAM kernel parameter

```
                                          ┌─,──────────────┐
                                          ▼                │
►►──xpram_parts── = ──number_of_partitions───partition_size─┴───────────────►◄
```

where *number_of_partitions* defines how many partitions the expanded storage is split into. The i-th *partition_size* defines the size of the i-th partition. Blank entries are inserted if necessary to fill *number_of_partitions* values. Each size may be blank, specified as a decimal value, or a hexadecimal value preceded by `0x`, and may be qualified by a magnitude:

- `k` or `K` for kilo (1024) is the default
- `m` or `M` for Mega (1024*1024)
- `g` or `G` for Giga (1024*1024*1024)

The size value multiplied by the magnitude defines the partition size in bytes. The default size is `0`.

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

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

XPRAM kernel example

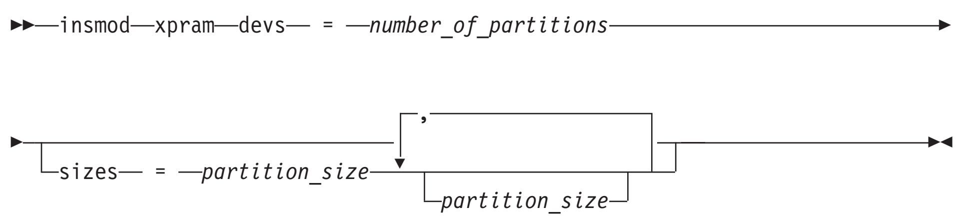
```
xpram_parts=4,0x800M,0,0,0x1000M
```

This allocates the extended storage into four partitions. Partition 1 has 2 GB (hex 800M), partition 4 has 4 GB, and partitions 2 and 3 use equal parts of the remaining storage. If the total amount of extended storage was 16 GB, then partitions 3 and 4 would each have approximately 5 GB.

XPRAM module parameter syntax

If it is not included in the kernel XPRAM may be loaded as a module. The syntax of the module parameters passed to `insmod` or `modprobe` differs from the kernel parameter syntax:

XPRAM module call



where:

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

XPRAM module example

```
insmod xpram devs=4 sizes=2097152,8388608,4194304,2097152
```

This allocates a total of 16 GB of extended storage into four partitions, of (respectively) size 2 GB, 8 GB, 4 GB, and 2 GB.

Support for loading XPRAM dynamically

In order to load the xpram module dynamically either using the `modprobe` command or by mounting an xpram partition for the first time, an entry in `/etc/modules.conf` (formerly also `/etc/conf.modules`) is required.

The following is an example of an xpram entry in `/etc/modules.conf`:

```
alias block-major-35 xpram
options xpram devs=4 sizes=4096,0,2048
```

With the above entry, four xpram partitions will be created. The first partition (minor 0) will have a size of 4096 KB, the third partition (minor 2) will have a size of 2048 KB, and partitions 2 and 4 (minors 1 and 3) will each use half the size of the remaining expanded storage.

Chapter 5. Linux for zSeries Console device drivers

The zSeries hardware requires a line-mode terminal (the hardware console) for overall system control. The Linux for zSeries console device drivers enable Linux to use this console for basic Linux control as well.

You can use a 3215 or a 3270 console instead of the hardware console if Linux is running under VM/ESA. You can use a 3215 console if Linux is running on a P/390.

The zSeries **system console** is the device which gives the zSeries operator access to the SE (Service Element) which is in overall control of the zSeries system. This can be a real device physically attached to the zSeries, or it can be emulated in software, for example by running an HMC (Hardware Management Console) in a Web browser window.

A zSeries **terminal** is any device which gives a zSeries user access to applications running on the zSeries system. This could be a real device such as a 3270 linked to the zSeries through a controller, or again it can be a terminal emulator on a networked device.

Note that both ’terminal’ and ’console’ have special meanings in Linux which should not be confused with the zSeries usage. The Linux console and the Linux terminals are different applications which both run on zSeries **terminal**s.

The drivers for the 3215, 3270 and hardware consoles can be compiled into the Linux kernel. If more than one console is present the default console driver will be chosen at run time according to the environment:

- In an LPAR or native environment, the hardware console will be made the default.
- In VM/ESA either the 3215 or the 3270 console driver will be made the default, depending on the guest’s console settings (the ″`CONMODE`″ field in the output of ″`#CP QUERY TERMINAL`″).
- On a P/390 the 3215 console will be made the default.

The default driver can be overridden with the ″`conmode=`″ kernel parameter (see “Console kernel parameter syntax” on page 30).

The intended use of the console device drivers is solely to launch Linux . When Linux is running, the user should access Linux for zSeries via a terminal emulation such as Telnet or `ssh`, because the console is a line-mode terminal and is unable to support applications such as `vi`. Therefore it is strongly recommended that you assign *`dumb`* to the *`TERM`* environment variable so that at least applications like `less` give appropriate output.

Note that there are different options that must be selected during kernel configuration to enable the Linux terminal on the hardware console or to enable the Linux console on the 3215 or 3270 console.

Console features

- Provides a line mode typewriter terminal.
- Console output on the first terminal.

Console kernel parameter syntax

The hardware console device driver does not require any parameters.

The 3215 console device driver does not require any parameters if it is used under VM/ESA. If it is used with a P/390 system, you have to specify the `condev` kernel parameter. This supplies the device driver with the subchannel number of the 3215 device. The reason that this parameter is needed is that there is no guaranteed method of recognizing a 3215 device attached to a P/390.

The kernel parameter syntax is:

3125 device driver syntax

```
►►──condev── = ──cuu──────────────────────────────────────────────────►◄
```

where *cuu* is the device 'Control Unit and Unit' address, and may be expressed in hexadecimal form (preceded by `0x`) or in decimal form.

Note: Early releases of the driver will not accept this parameter in hexadecimal form.

Console kernel examples

```
condev=0x001f
```

or

```
condev=31
```

Both of these formats tell the device driver to use device number hex 1F for the 3215 terminal.

Using the console

The console is a line mode terminal. The user enters a complete line and presses enter to let the system know that a line has been completed. The device driver then issues a read to get the completed line, adds a new line and hands over the input to the generic TTY routines.

Console special characters

The console does not have a control key. That makes it impossible to enter control characters directly. To be able to enter at least some of the more important control characters, the character `'^'` has a special meaning in the following cases:

- The two character input line `^c` is interpreted as a `Ctrl+C`. This is used to cancel the process that is currently running in the foreground of the terminal.
- The two character input line `^d` is interpreted as a `Ctrl+D`. This is used to generate an end of file (EOF) indication.

- The two character input line `^z` is interpreted as a `Ctrl+Z`. This is used to stop a process.
- The two characters `^n` at the end of an input line suppresses the automatic generation of a new line. This makes it possible to enter single characters, for example those characters that are needed for yes/no answers in the ext2 file system utilities.
- The two characters '^-' followed by a third character invoke the so called "magic sysrequest" function. Various debugging and emergency functions are performed specified by the third character. This feature can be switched on or off during runtime by echoing '1' or '0' to `/proc/sys/kernel/sysrq`. The third character can be:
  - 'b' – re-IPL immediately,
  - 's' – emergency sync all filesystems,
  - 'u' – emergency remount all mounted filesystems readonly,
  - 't' – show task info,
  - 'm' – show memory,
  - '0' to '9' – set console log level,
  - 'e' – terminate all tasks,
  - 'i' – kill all tasks except `init`,
  - 'l' – kill all tasks including `init`.

If you are running under VM without a 3215 console you will have to use the `CP VINPUT` command to simulate the **ENTER** and **SPACE** keys.

The **ENTER** key is simulated by entering:

```
#CP VInput VMSG \n
```

The **SPACE** key is simulated by entering:

```
#CP VInput VMSG  \n    (two blanks followed by \n).
```

If the special characters do not appear to work, make sure you have the correct codepage in your terminal emulator. One known to work is codepage 037 (United States).

VM console line edit characters

When running under VM, the control program (CP) defines five characters as line editing symbols. Use the `CP QUERY TERMINAL` command to see the current settings. The defaults for these depend on the terminal emulator you are using, and can be reassigned by the CP system operator or by yourself using the `CP TERMINAL` command to change the setting of `LINEND`, `TABCHAR`, `CHARDEL`, `LINEDEL` or `ESCAPE`. The most common values are:

LINEND #
: The end of line character (this allows you to enter several logical lines at once).

TABCHAR |
: The logical tab character.

CHARDEL @
: The character delete symbol (this deletes the preceding character).

LINEDEL [(ASCII terminals) or ¢ (EBCDIC terminals)
The line delete symbol (this deletes everything back to and including the previous LINEND symbol or the start of the input).

ESCAPE ″
The escape character (this allows you to enter a line edit symbol as a normal character).

To enter the line edit symbols `# | @ [ "` (or `# | @ ¢ "`) you need to type the character pairs `"# "| "@ "[ ""` (or `"# "| "@ "¢ ""`). Note in particular that to enter the quote character (″) you must type it twice (″″).

Example:

If you should type in the character string:

```
#CP HALT#CP ZIPL 190[#CP IPL 1@290 PARM VMHALT=""MSG OP REBOOT"#IPL 290""
```

the actual commands received by CP will be:

```
CP HALT
CP IPL 290 PARM VMHALT="MSG OP REBOOT#IPL 290"
```

Console 3270 emulation

If you are accessing the VM console using the `x3270` emulator, then you should add the following settings to the `.XDefaults` file in order to get the correct code translation:

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

Console – Use of VInput

Note: ’VInput’ is a VM CP command. It may be abbreviated to ’VI’ but should not be confused with the Linux command ’vi’.

If you use the hardware console driver running under VM it is important to consider how the input is handled. Instead of writing into the suitable field within the graphical user interface at the service element or HMC you have to use the `VInput` command provided by VM. The following examples are written at the input line of a 3270 terminal or terminal emulator (for example, x3270).

Note that, in the examples, capitals within `VInput` and its parameters processed by VM/CP indicate the characters you have to type. The lower case letters are optional and are shown for readability. These examples assume that you enter the `CP READ` mode first. If you are not in this mode you must prefix all of the examples with the command `#CP`.

```
VInput VMSG LS -L
```

(the bash will call `ls -l` after this command was sent via `VInput` to the hardware console as a non-priority command - `VMSG`).

```
VInput PVMSG ECHO *
```

(the bash will execute `echo *` after this command was sent via `VInput` to the hardware console as a priority command - `PVMSG`).

The hardware console driver is capable to accept both if supported by the hardware console within the specific machine or virtual machine. Please inspect your boot messages to check the hardware console's capability of coping with non-priority or priority commands respectively on your specific machine. Remember that the hardware console is unable to make its own messages available via `dmesg`.

Features of `VInput`.

1. Use ′""′ to output a single ′"′.

   `VInput` example: `VInput PVMSG echo ""Hello world, here is ""$0`

   (on other systems: `echo "Hello world, here is "$0`)

2. Do not use `#` within `VInput` commands.

   This character is interpreted as an end of line character by `VM CP`, and terminates the `VInput` command. If you need the **#** character it must be preceded by the escape character (″**#**).

3. All characters in lower case are converted by VM to upper case.

   If you type `VInput VMSG echo $PATH`, the driver will get `ECHO $PATH` and will convert it into `echo $path`. Linux and the bash are case sensitive and cannot execute such a command. To resolve this problem, the hardware console uses an escape character (%) under VM to distinguish between upper and lower case characters. This behavior and the escape character (%) are adjustable at build-time by editing the driver sources, or at run time by use of the `ioctl` interface. Some examples:

   - input: `VInput VMSG ECHO $PATH`

     output: `echo $path`

   - input: `VInput vmsg echo $%PATH%`

     output: `echo $PATH`

   - input: `VInput pvmsg echo ""1/11/02ello, here is ""$0 #name of this process`

     output: `VINPUT PVMSG ECHO "1/11/02ELLO, HERE IS "$0`

     `NAME OF THIS PROCESS`

     `HCPCMD001E Unknown CP command: NAME`

     `echo "Hello, here is "$0`

     `Hello, here is -bash`

Console limitations

- The 3215 driver only works in combination with VM/ESA. In a single image or in LPAR mode the 3215 terminal device driver initialization function just exits without registering the driver.
- Due to a problem with the translation of code pages (500, 037) on the host, the pipe command character (**|**) cannot be intercepted by the console. If you need to use this command execute it from a Telnet session.
- Displaying large files might cause some missing sections within the output because of the latency of the hardware interface employed by the device.
- In native or LPAR environments, you occasionally have to use the **Delete** button of the GUI on the Service Element or Hardware Management Console to enable further output. This is relevant to:

- SE version 1.6.1 or older on G5, G6, and Multiprise 3000.
- SE version 1.5.2 or older on G3, G4, and Multiprise 2000.

- Messages concerning the hardware console operation generated by the hardware console driver cannot be provided to the `syslog` and are therefore unavailable with `dmesg`.
- Output from the head/top is deleted if the amount exceeds approximately 30 kilobytes per LPAR (or image) on SE or HMC.
- Applications such as `vi` are not supported because of the console's line-mode nature.

Chapter 6. Channel-attached tape device driver

The 3590 tape discipline module ('tape_3590_mod.o') is subject to license conditions as reflected in: "International License Agreement for Non-Warranted Programs" on page 233.

The Linux for zSeries tape device driver manages channel-attached tape drives which are compatible with IBM 3480, 3490, and 3590 magnetic tape subsystems. Various models of these devices are handled (for example the 3490E).

Tape driver features

The device driver supports a maximum of 128 tape devices.

No official Linux device major number is assigned to the zSeries tape device. The driver allocates major numbers dynamically and lists them on initialization. Currently, without devfs, major number 254 is allocated for both the character and block device front-ends. With devfs, the major numbers are 4 for block and 152 for character. Minor numbers will be allocated in pairs from zero.

The driver may search for all tape devices attached to the Linux machine, or it may be given a list of device addresses to use.

If it is not given a list the numbers allocated are volatile – the number allocated to any particular physical device may change if the system is rebooted or the device driver is reloaded. In particular a device brought online during a Linux session will be allocated the next available number at the time it comes online, but at the next reboot it will be given a number according to the sequence of device addresses.

If a `"tape="` parameter is present at system startup or module load, all tape devices in the ranges of the specified parameter list will be used. The devices are then numbered (sequentially from zero) according to the order in which their subchannel numbers appear in the list.

If the Device File System (`devfs`) is used the user need not be concerned about the major and minor numbers used since each device will be assigned a node name based on the channel address of the device automatically. When the driver is initialized in autodetect mode without parameters (at system startup or module load) all supported tape devices attached will be detected.

Tape disciplines as modules

The tape driver separates the general tape infrastructure from the device-specific part, which is encapsulated in tape disciplines.

The names of the discipline kernel modules are as follows:
- tape_3480_mod.o
- tape_3490_mod.o
- tape_3590_mod.o

Tape driver as a kernel module
When the tape driver (tape390) is built as a kernel module, the disciplines

(except tape_3590_mod.o) are also built as kernel modules. When the tape390 kernel module initializes, it tries to load all known disciplines (currently 3480/3490/3590). Therefore, the user need not load the modules for these disciplines separately.

```
> modprobe tape390
> lsmod
Module              Size  Used by
tape_3590_mod       12384   0  (autoclean)
tape_3490_mod        1472   0  (autoclean) (unused)
tape_3480_mod        1472   0  (autoclean) (unused)
tape390             46384   0  [tape_3590_mod tape_3490_mod tape_3480_mod]
```

Tape driver built into the kernel

The tape driver (tape390) and discipline modules (except tape_3590_mod) can also be built directly into the kernel. Any disciplines built or supplied as modules must then be loaded separately:

```
> modprobe tape_3590_mod
> modprobe tape_3480_mod
> modprobe tape_3490_mod
```

Tape device front-ends

Two front-ends are provided for each physical device:

- character device front-end
- block device front-end

Character device front-end

The character device front-end is used in multi-step procedures to leave the tape in position for the subsequent step at the close of each step. You will usually read or write to the tape device using this front-end.

If `devfs` is not used, the node names for these devices should be constructed from the standard label `tibm`, with a prefix indicating the close function `r` (rewind) or `n` (non-rewind), and a suffix from the device number (starting at zero). Thus the names given to the first two devices are `/dev/rtibm0`,

```
rtibm0 -> r     rewind
          tibm  label
          0     device number
```

`/dev/ntibm0`, `/dev/rtibm1` and `/dev/ntibm1`.

If `devfs` is used, the node names for these devices are constructed from the standard label `tape`, the device number, and `rewinding` or `nonrewinding`. Thus, if the I/O device addresses are 0181 and 0182, the names assigned are

```
/dev/tape/0181/char/rewinding
/dev/tape/0181/char/nonrewinding
/dev/tape/0182/char/rewinding
/dev/tape/0182/char/nonrewinding
```

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

Most Linux tape software should work with both the rewinding and non-rewinding devices.

Block device front-end

You can also use the tape driver as a block device, but this is restricted to read-only mode.

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

One block device node is allocated to each physical device. They follow a similar naming convention to the character devices. Without `devfs` the prefix `b` is used – `/dev/btibm0` for the first device, `/dev/btibm1` for the second and so on. With `devfs` a device type of `/block/disc` is used, such as:

```
/dev/tape/0181/block/disc
/dev/tape/0182/block/disc
```

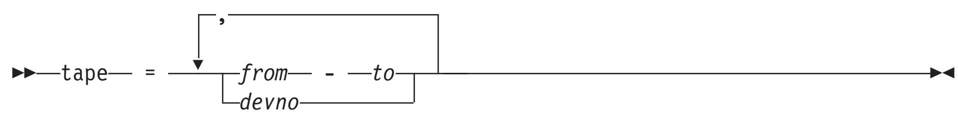
for devices 0181 and 0182.

It is advisable to use only the ISO9660 filesystem on Linux for zSeries tapes, since this filesystem is optimized for CD-ROM devices, which – just like 3480, 3490, or 3590 tape devices – cannot perform fast seeks.

Tape driver kernel parameter syntax

You do not need to give the tape device driver any kernel parameters if you want to use tape auto-detection. You should not give it parameters if you are using `devfs`. If you want to specify the physical tape devices to be used you must configure the tape driver by adding a parameter to the kernel parameter line:

Tape driver kernel parameter syntax



where:

from-to defines the first and last tape device in a range. All valid tape devices with addresses in this range are selected. It is not necessary for the *from* and *to* addresses to correspond to actual devices.

devno defines a single tape device.

The tape addresses must be given in hexadecimal notation (without a leading `0x`), for example 0181 or 5a01.

If you supply one or more kernel parameters, for example `tape=`*`from-to`*`,tape=`*`devno`*`,...`, the devices are processed in the order in which they appear in the parameter line. Devices are ignored if they are unknown to the device driver, non-operational, or set offline. You should specify no more than 128 devices in the parameter line as this is the maximum number of devices manageable by the driver.[8]

Note that the auto-detection option may cause confusing results if you change your I/O configuration between two IPLs, or if you are running as a guest operating system in VM/ESA, because the devices might appear with different names (major/minor combinations) in the new IPL if you are not using `devfs`.

Tape driver kernel example

If `devfs` is not used the kernel parameter could be:

```
tape=181-184,19f
```

This reserves devices as follows:

```
0181     will be    /dev/ntibm0   /dev/rtibm0   /dev/btibm0
0182     will be    /dev/ntibm1   /dev/rtibm1   /dev/btibm1
0183     will be    /dev/ntibm2   /dev/rtibm2   /dev/btibm2
0184     will be    /dev/ntibm3   /dev/rtibm3   /dev/btibm3
019f     will be    /dev/ntibm4   /dev/rtibm4   /dev/btibm4
```

If `devfs` is used the device names will be:

8. Currently there is no check for duplicate occurrences of the same device number.

```
0181     will be    /dev/tape/0181/char/rewinding
                    /dev/tape/0181/char/nonrewinding
                    /dev/tape/0181/block/disc
0182     will be          .../0182/...
0183     will be          .../0183/...
0184     will be          .../0184/...
019f     will be          .../019f/...
```

Tape driver module parameter syntax

The syntax of the module call to load the tape device driver is:

Tape module parameter syntax

```
>>──┬─modprobe─┬──tape390──┬──────────────────────────────────┬──────────><
    └─insmod───┘           │              ┌─,──────────────┐  │
                           │              ▼                │  │
                           └─tape── = ────┬─from── - ──to─┬┴──┘
                                          └─devno─────────┘
```

where:

tape390
: is the name of the device driver module

and the rest of the parameters are the same as those of the tape driver kernel syntax.

Tape driver module example

```
modprobe tape390 tape=181-184,19f
```

The details are the same as “Tape driver kernel example” on page 38.

Tape device driver API

The tape device driver uses the POSIX-compliant tape interface similar to the Linux SCSI tape device driver.

Some differences in the MTIO interface do exist due to the different hardware:

- `MTSETDENSITY` has no effect as the recording density is automatically detected.
- `MTSETDRVBUFFER` has no effect as the drive automatically switches to unbuffered mode if buffering is unavailable.
- `MTLOCK` and `MTUNLOCK` have no effect as the tape device hardware does not support media locking.
- `MTLOAD` waits until a tape is inserted rather than loading a tape automatically.
- `MTUNLOAD`. The drives do not support an unload command, but if `MTUNLOAD` is used, the next tape in the stacker will be inserted automatically if the mode of operation is automatic and another tape is present. Otherwise, the tape is unloaded.
- `MTCOMPRESSION` controls the IDRC (Improved Data Recording Capability). Compression is off after system start.

  To switch compression on, use

  ```
  mt -f tape compression          or
  mt -f tape compression 1
  ```

  To switch compression off, use

  ```
  mt -f tape compression 0
  ```

  Any other numeric value has no effect, and any other argument switches compression off.
- `MTSETPART` and `MTMKPART` have no effect as the devices do not support partitioning.
- `MTIOCGET` returns a structure that, aside from the block number, contains mostly SCSI-related data not applicable to this driver.
- `TAPE390_DISPLAY` enables programs to write to the display on zSeries and S/390 tape drives (see "Tape display support" on page 41).

/proc/tapedevices Interface

The tape driver creates the entry "/proc/tapedevices" in the proc file system. This entry provides information about all tape devices in the Linux system. The information can be displayed by issuing, for example:

```
cat /proc/tapedevices

TapeNo  DevNo   CuType  CuModel DevType DevMod  BlkSize State   Op      MedState
0       1A10    3490    10      3490    40      auto    UNUSED  ---     UNLOADED
1       1A28    3480    01      3480    04      auto    UNUSED  ---     UNKNOWN
2       1A32    3590    50      3590    11      auto    IN_USE  ---     LOADED
3       1A33    3590    50      3590    11      1024    UNUSED  ---     LOADED
```

where:

TapeNo
: Driver internal number for the tape device

DevNo
: Device number of the tape device in hex

CuType
Control unit type number

CuModel
Control unit model number

DevType
Device type number

DevMod
Device model number

BlkSize
Currently used block size (auto or number of bytes)

State State of the tape device:

UNUSED
Device is currently not being used by any process

IN_USE
Device is being used by a process

NOT_OP
Device is not operational

Op Tape operation which is currently active, for example:

--- No operation

WRI Write operation

RFO Read operation

There are several other operation codes, such as for rewind and seek.

MedState
Current state of the cartridge

LOADED
Cartridge is loaded into the tape device

UNLOADED
No cartridge is loaded

UNKNOWN
The tape driver does not have information about the current cartridge state

Tape display support

Description

Tape display support enables user programs to specify messages (up to two 8-byte messages) to be displayed on the display unit of an addressed tape drive.

To enable this functionality, the `TAPE390_DISPLAY` IOCTL function has been added, and a new tool exploiting the new function, `tape390_display`, has been added to the s390-tools package. See the common Linux documentation on how to use the IOCTL interface.

Note: `TAPE390_DISPLAY` is supported for IBM 3480, 3490, and 3590 tape devices.

User interfaces

tape390_display tool

tape390_display

```
►►──tape390_display──┬──┬─────┬──message1──┬──device─┬──────────────────►◄
                     │  │     └ -b ┘              │         │
                     │  └─message1──message2─┘         │
                     └─ -h──────────────────────────────┘
```

where:

-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

message2
: is a second message to be displayed alternately with the first, at 2 second intervals

device
: is the target tape device

-h or --help
: prints help text

Notes:

1. Symbols that can be displayed include:

   Alphabetic characters:
   A through Z (uppercase only) and spaces. Lowercase letters will be converted to uppercase.

   Numeric characters:
   0 1 2 3 4 5 6 7 8 9

   Special characters:
   @ $ # , . / ' ( ) * & + - = % : ; < > ?
2. If only one message is defined, it remains displayed until the tape 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.

Example (with devfs):

```
tape390_display BACKUP COMPLETE /dev/tape/181/char/nonrewinding
```

Tape driver examples

The following examples illustrate the operation of the tape driver on the basis of the `mt` utility.

Example 1 – Creating a single-volume tape (without devfs)

In this example a tape with an ISO9660 filesystem is created using the first tape device. For this the ISO9660 filesystem support must be built into your system kernel.

Use the `mt` command to issue tape commands, and the `mkisofs` command to create an ISO9660 filesystem:

- Create a Linux directory (`somedir`) and fill it with the contents of the filesystem

  ```
  mkdir somedir
  cp contents somedir
  ```

- Insert a tape
- Ensure the tape is positioned at the beginning

  ```
  mt -f /dev/ntibm0 rewind
  ```

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

  ```
  mt -f /dev/ntibm0 setblk 2048
  ```

- Write the filesystem to the character device driver

  ```
  mkisofs -l -f -o file.iso somedir
  dd if=file.iso of=/dev/ntibm0 bs=2048
  ```

- Rewind the tape again

  ```
  mt -f /dev/ntibm0 rewind
  ```

- Now you can mount your new filesystem as a block device:

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

Example 2 – Creating a multivolume tape (with devfs)

In this example files are backed up onto a multivolume tape using the Linux facility `tar`.

- Insert a tape medium in the device (here: `/dev/tape/019f/char/nonrewinding`).
- If necessary, rewind and erase the tape:

```
mt -f /dev/tape/019f/char/nonrewinding rewind
mt -f /dev/tape/019f/char/nonrewinding erase
mt -f /dev/tape/019f/char/nonrewinding rewind
```

- Open a new Telnet session to trace the content of `/var/log/messages`

```
tail -f /var/log/messages &
```

- In the first Telnet session backup the files to tape using the tar command with the option `-M` (multi-volume), for example:

```
tar -cvMf /dev/tape/019f/char/nonrewinding /file_specs
```

- If more tape volumes are required you will be prompted to prepare the next medium. Go to a third Telnet session and enter the command:

```
mt -f /dev/tape/019f/char/nonrewinding offl
```

- Insert a new tape manually (if not using a tape unit magazine with autoload).
- In the second session, wait for a message of the form:

```
Mar  1 14:34:13 gfree06 kernel: T390:(1a32): Tape has been mounted
```

  in `/var/log/messages` (see the 'tail' command above). When you see this message, press the return key in the `tar` session.
- Repeat this process with any remaining tapes until the backup is complete.

Tape driver restrictions

The driver is unable to detect manual operations on the tape device, in particular manual tape unloads, and these operations will lead to errors in reading and writing. The driver provides `ioctl` functions to control the device and these must be used, either through the API or by using the Linux `mt` utility.

Tape driver further information

Basic Linux tape control is handled by the `mt` utility, which is described in the `mt` man page. Note that the sections on SCSI tape devices are not applicable to channel-attached zSeries devices. They do apply to tape devices accessed via the zfcp driver, however.

Chapter 7. Generic cryptographic device driver

Note on license

This driver is subject to license conditions as reflected in "International License Agreement for Non-Warranted Programs" on page 233.

This Linux driver (z90crypt) is a generic character device driver for a cryptographic device. This virtual device will route work to any physical devices (for example, PCI Cryptographic Coprocessor, PCICC, or PCI Cryptographic Accelerator, PCICA) installed on the system.

Overview

This driver controls the PCICC or PCICA in a Linux environment. Its current function is RSA-PKCS 1.2 decryption using a private key. The owner of the key can decrypt messages that were encrypted using the corresponding public key. This function is however confined to ″insecure cryptography″ – the private key is not itself encrypted.

For RSA–PKCS 1.2 see http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1/. The context consists of four steps:

- Encoding – also termed ″padding″ – which adds material to a plain-text message
- Encryption, with a public or private key, by arithmetic operations involving very large numbers
- Decryption, closely related to encryption, with the counterpart of the key used for encryption
- Decoding, which for PKCS 1.2 means stripping the padding from the message.

z90crypt performs the last two operations using the private member of a key-pair.

When using a PCICC and invoking z90crypt directly (not via libica), the generation of public/private key pairs, encryption, signing and signature verification, and even decryption using a public key are not supported by Linux for zSeries at present. When using libica, this library supplies these functions via software, with a speed tradeoff. The PCICA, on the other hand, performs these functions in hardware.

The following figure illustrates the software relationships:

Cryptographic device driver

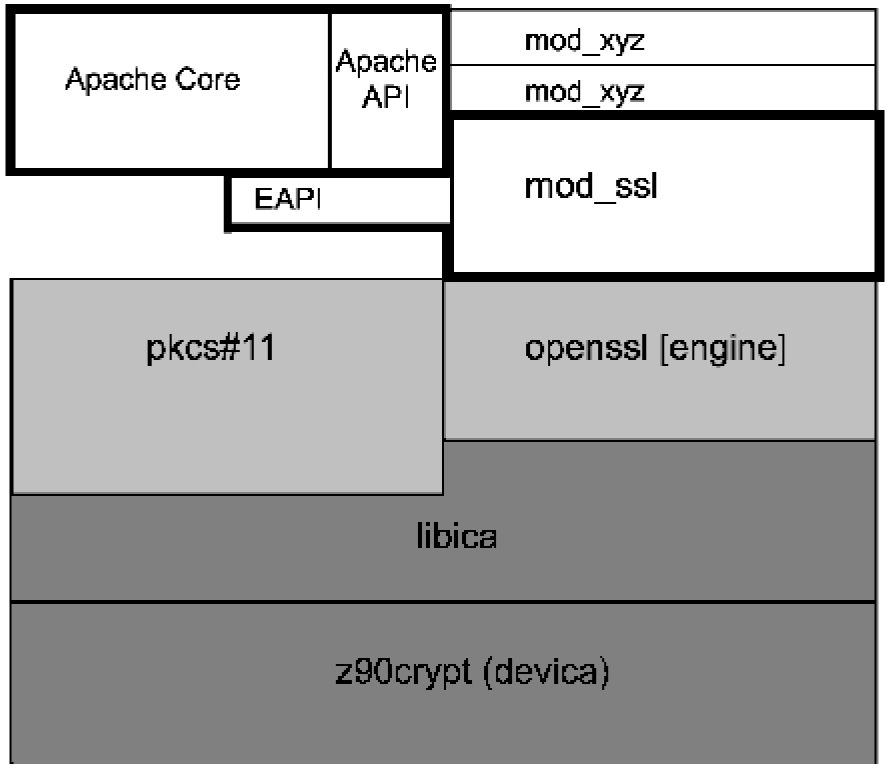


Figure 2. z90crypt device driver interfaces

HMC settings

On the HMC you can determine what your current settings for cryptographic cards are, and specify settings for a new card.

Checking your current crypto hardware settings on the HMC

To check that you have the correct settings:

1. Go into single object operation mode with your machine.
2. Right click on the **LPAR** icon. It must show three parts (CHPIDs, CPs, and PCI crypto).
3. Click on crypto. This shows the crypto processors. They must be ’online’.

Alternatively:

1. Go into single object operation mode with your machine.
2. Select the **LPAR** icon.
3. Click **customize/delete activation profiles** icon on the right.
4. Double click on active profile.
5. Click on the **crypto** tab (at the bottom). If you do not see this use the small right arrow in the upper right corner.

   You can see the currently set ’control domain index’ and ’usage domain index’ (highlighted number). If the **enable modify authority** check box is marked then you have the rights to change settings.
6. Click the **pci crypto** tab.

You see the current coprocessor number in the 'pci cryptographic coprocessor candidate list' and 'pci cryptographic coprocessor online list'.

Defining a crypto device to an LPAR

The cryptographic device must be defined to the LPAR. To define the device to the LPAR, on the HMC, do the following:

1. With the PCICC card installed, bring up the PR/SM™ Panel for your Linux LPAR. On this panel there is an initial Processor tab. This offers a choice of two built-in S/390 Cryptographic Co-processor Features (CCF's). One or both must be chosen for the PCICC card to operate, but the built-in feature does not operate under Linux. You must click on one or both, but it makes no difference which you choose.
2. Select a unique "cryptographic domain":

   When the CCF(s) are selected, two further tabs appear to the right. They are:
   - A "crypto" tab. On this tab, select a "cryptographic domain" for the LPAR. This is a number between 0 and 15. The cryptographic domain must be unique to the LPAR — different from the cryptographic domain for any other LPAR. The choice, however, has no visible effect.
   - The PCI card tab. This offers a choice of PCICCs and/or PCICAs. Be aware, however, that z90crypt uses only PCICAs when both types of adapters are installed.
3. If you are using a zSeries z900 (Freeway) GA2 machine: On the PR/SM panel where you choose which architecture your LPAR will support, choose **ESA390**. Do *not* choose 'Linux only'. If you choose Linux only, no crypto devices will be available to your LPAR.

Installing z90crypt

Prerequisites

To use the crypto device driver, you will need the driver module and the libica library. The library is the interface between the application and the z90crypt driver module. The module and the library can be obtained from the DeveloperWorks Web site:

- The crypto module:
  http://www.ibm.com/developerworks/oss/linux390/download_obj.html
- The libica library:
  http://www-124.ibm.com/developerworks/projects/libica
  Obtain the latest version of the 390 (31-bit) binary (.rpm). Source code is also available here in tar form.
- pkcs#11:
  http://www-124.ibm.com/developerworks/projects/openCryptoki
  Obtain the latest version of the 390 (31-bit) binary (.rpm). This file is designated as "openCryptoki". Source code is also available here in tar form.

z90crypt is distributed as a package with a name of the form "z90crypt ... .tar.gz".

Contents of the tar file

The tar file contains:

- The object module z90crypt.o, which can be loaded into the Linux kernel.
- The script z90crypt_install, which copies the object module into a suitable directory.

- The script z90crypt_load, which loads it.
- The script z90crypt_unload, which unloads it.
- A text README file (if there is any new information augmenting the description in this chapter).
- The header file z90crypt.h. Structures in this header are essential to interface your program with z90crypt.

Installing and loading

To install and load the z90crypt driver, follow these steps.

VM Requirements

APAR VM62905 for z/VM 4.2 is required in order to install and use the crypto function in Linux guests. The Linux guest must also have the following entry defined in the User Directory Table:

```
CRYPTO APVIRT
```

Note

The following procedure may need to be adapted to the distribution you are using. For certain actions, you will need root authority, which is indicated by the shell prompt character '#'.

1. You will need to gunzip the driver file (name in the format "z90crypt ... .tar.gz") and then perform:

   ```
   $ tar -xvf z90crypt ... .tar
   ```

   to obtain its constituents.
2. Make the shell scripts executable:

   ```
   $ chmod 744 z90crypt_install
   $ chmod 744 z90crypt_load
   $ chmod 744 z90crypt_unload
   ```
3. Run the install script for z90crypt:

   ```
   # ./z90crypt_install
   ```
4. Run the script to load z90crypt:

   ```
   # ./z90crypt_load
   ```
5. (Optional) Check whether z90crypt is in the kernel list of modules:

   ```
   $ /sbin/lsmod
   ```

   This should result in output like the following:

   ```
   Module             Size   Used by
   z90crypt          34160    0 (unused)
   ```

   If the correct choices have not been made in the PR/SM panel, the module will not load.
6. Test the hardware response of the driver module.

   ```
   $ cat /proc/driver/z90crypt
   ```

   You should have an output similar to the following:

   ```
   bash-2.04$ cat /proc/driver/z90crypt

   Cryptographic domain: 14
   ```

```
Total device count: 2
PCICA count: 0
PCICC count: 2
requestq count: 0
pendingq count: 0
total open handles: 0


Mask of online devices: 01 means PCICA, 02 means PCICC
    0202000000000000 0000000000000000 0000000000000000 0000000000000000
    0000000000000000 0000000000000000 0000000000000000 0000000000000000


Mask of waiting work element counts
    0000000000000000 0000000000000000 0000000000000000 0000000000000000
    0000000000000000 0000000000000000 0000000000000000 0000000000000000
```

7. Unload the driver.

   Note: Always load the driver before using it and always unload it after use.

   ```
   # ./z90crypt_unload
   ```
8. Unpack the libica library
9. Compile and install the libica library

   ```
   make -f Makefile.linux
   ```

You may choose to program, as outlined below, between the 2nd and 3rd steps.

Decryption using z90crypt

How to perform decryption using z90crypt:

Outline of decryption program

The following steps are required:

1. Get a device handle, say "dh" for z90crypt:

   ```
   dh= open("/dev/z90crypt", O_RDWR)
   ```
2. Create and load a structure of the type ica_rsa_modexpo_t or ica_rsa_modexpo_crt_t as described below. You will define the private key to be used and set the input buffer pointer to the message you want decrypted.
3. Invoke ioctl to activate z90crypt:

   ```
   rc= ioctl(dh, <function code>, <structure name>)
   ```

   where:
   - <**function code**> is ICARSAMODEXPO or ICARSACRT
   - <**structure name**> is the name of the structure you created, of the type ica_rsa_modexpo_t or ica_rsa_modexpo_crt_t
   - **rc** is your name for the return code

   For example:

   ```
   rc= ioctl(dh, ICARSAMODEXPO, mycryptmex)
   ```

   where:
   - **rc** is your name for the ioctl return code
   - **dh** is your name for the z90crypt device handle
4. Obtain the decrypted and decoded message from the output buffer in the structure. The original message must have been PKCS 1.2 encoded – that is, the

decrypted message must have correct padding. If so, z90crypt strips the padding, but if not it gives an error message.

.

The ica_rsa_modexpo_t structure

The ica_rsa_modexpo_t structure is shown in “z90crypt header file” on page 51, along with the other structures for use with z90crypt. The (private) key consists of the exponent in *b_key and the modulus in *n_modulus. Both of these are hexadecimal representations of large numbers. The length L of *n_modulus must be in the range 64 – 256.

The input data and the exponent b_key must both be of length L. The output data must be of length L or greater. In all these cases, padding on the left with zeroes is allowed.

There are mathematical rules for the construction of public/private key-pairs, constraining the modulus and exponent in each member of a pair, but we omit these, as z90crypt will not generate key-pairs anyway. See http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1/ for a summary of the mathematics.

The ica_rsa_modexpo_crt_t structure

The only formal difference between this structure and the previous one is in the definition of the key. This is defined so that the Chinese Remainder Theorem can be used in decryption/encryption. z90crypt decrypts about 4 times faster if the CRT definition is used. The key-definition fields are all in hexadecimal representation. They have these meanings and limitations:

- *bp_key, *bq_key: Prime factors of the modulus. In z90crypt the modulus is (*bp_key) * (*bq_key). The resulting length L of the modulus, in hexadecimal representation, must be found before these fields are defined.
- *np_prime, *nq_prime: Exponents used for *bp_key and *bq_key respectively.
- *u_mult_inv: A coefficient used in the z90crypt implementation of decryption by CRT.
- *bp_key, *np_prime, *u_mult_inv must all be of length 8 + L/2
- *bq_key, *nq_prime must both be of length L/2

The input data length must be L, and the output data length must be at least L, as in ica_rsa_modexpo_t.

Other functions of z90crypt

Checking hardware status

There is an ioctl interface for checking on underlying hardware in z90crypt. The function code is ICAZ90STATUS, and you supply a struct ica_z90_status_t (see “z90crypt header file” on page 51 for the definition) as the argument. When control returns you will have the number of cards installed and a mask showing which cards are installed.

Example:

```
rc= ioctl(dh, ICAZ90STATUS, my_z90crypt_status);
```

where:

rc is your name for the ioctl return code

dh is your name for the z90crypt device handle

my_z90crypt_status
is your name for your structure of type ica_z90_status, in which the status data is to be written.

Quiescing

You need root authority to do this. You can 'quiesce' z90crypt by executing an ioctl with function code ICAZ90QUIESCE. This lets the driver finish any outstanding work, but prevents any new work from being submitted. After 30 seconds, either all outstanding requests will be completed or else the requesting process will be notified that its work will not be completed. The only way to 'un-quiesce' z90crypt is to unload it and then re-load it.

Example:

```
rc = ioctl(dh, ICAZ90QUIESCE);
```

Random number generation

If you do a read from z90crypt, you will get back a string of more-or-less random bytes. For read, you cannot specify a buffer length of more than 256 bytes.

Example:

```
 rc = read (dh, my_buffer, number_of_bytes);
```

where:

rc is your name for the ioctl return code

dh is your name for the z90crypt device handle

my_buffer is your name for the byte-array where the random bytes will be loaded in response to the read

Returns from read and ioctl

0 means everything went well and the result is in your output buffer. Return codes greater than 0 have the following meanings:

- 8 -- invalid operand
- 16 -- device not available.
- 17 -- error in pkcs 1.2 padding (no room for required padding)
- 24 -- error copying to or from user space

A return code of '-1' means that errno is one of the following:

- 13 -- permission denied (attempted quiesce but not root)
- 22 -- invalid operand
- 132 -- device is quiescing; no new work being accepted.
- 134 -- unknown error (this usually means a transient error in a crypto device; a retry may succeed).

For any other error code, look in /usr/include/asm/errno.h

z90crypt header file

Refer to file 'z90crypt.h' for other important information.

Crypto support for 31-bit emulation

31-bit applications can access the 64-bit z90crypt driver by using the 31-bit emulation feature.

The best way to build a 31-bit emulation support for hardware cryptography is to install a 31-bit Linux system and a second with a 64-bit Linux system and the necessary emulation layer.

1. Compile all necessary programs (like libica, pkcs11, openssl and apache) under the 31-bit system.
2. Copy the 31-bit versions of the following files to the directory for 31-bit emulation in your distribution. You can find both files on your 31-bit system in the /usr/lib directory. The files are:
```
libica.so
libcrypto.so
```
3. Copy the rest of your applications (like openssl, apache, ...) to your target 64-bit system (including the 31-bit emulation layer).
4. Now you should be able to run the test programs from OpenSSL, such as:
```
openssl speed rsa -engine ibmca
```

Example of use: Apache

This section illustrates the installation of the Apache Web server with SSL and crypto-card support.

Note

The information in this section applied to IBM's internal testing. You may need to adapt the information depending on the distribution you are using.

Specific software version numbers listed are those that applied at that time and may no longer be current.

Prerequisites

Table 3. Requirements for z90crypt usage example

Software	
Item	**Source**
Operating system: Linux for S/390, version: kernel 2.4.7	The DeveloperWorks Web site: http://www.ibm.com/developerworks/oss/linux390/alpha_src.html
Patches: • Configure-390.patch • ibmca.patch Install the ibmca patch with patch -p5.	The OpenSSL Web site: http://www-124.ibm.com/developerworks/projects/libica Select the **Files** tab. There is a link to download the OpenSSL patches. Download Patchset1. All files are source.
Apache Web server version 1.3.20	Download the Apache Web server. The Web site is: http://www.apache.com/
Modssl version 2.8.4	Download from the ModSSL Web site: http://www.modssl.org/

Table 3. Requirements for z90crypt usage example (continued)

Software	
Item	**Source**
OpenSSL (engine) version 0.9.6b	Download from the OpenSSL Web site: http://www.openssl.org/
pkcs#11	Download from: http://www-124.ibm.com/developerworks/projects/openCryptoki The 390 (31-bit) binary is in openCryptoki-1.3-2B.s390.rpm. Source code is also available here in tar form.
Hardware	
Customized crypto card, for example, PCICC or PCICA	

Installing the Apache Web server

1. Create the directory `my_web_server`
2. Change into the directory `my_web_server`
3. Download the Apache Web server and copy it into the directory my_web_server:
   ```
   wget http://httpd.apache.org/dist/httpd/apache_1.3.20.tar.gz
   ```
4. Download the Openssl [engine] library and copy it into the directory my_web_server
   ```
   wget http://www.openssl.org/source/openssl-0.9.6b-engine.tar.gz
   ```
5. Download the Modssl module and copy it into the directory my_web_server
   ```
   wget http://www.modssl.org/source/mod_ssl-2.8.4-1.3.20.tar.gz
   ```
6. Unpack the Apache Web server.
7. Unpack the Openssl [engine] library.
8. Unpack the Modssl module.
9. Now you should have the following directory tree:
   ```
   my_web_server
       /apache-<VERSION>
       /mod_ssl-<VERSION>
       /openssl-engine-<VERSION>
   ```
10. Change to the Openssl [engine] directory
11. Configure, compile, test and install Openssl.

    Prerequisites: Two patches are needed:
    - Configure-390.patch
    - ibmca.patch Install the ibmca patch with patch -p5.

    Makefiles: You may need to edit some makefiles and include some extra libs with -ldl.
    ```
    $ ./config
    $ make
    $ make test
    # make install
    ```
12. Change to the Modssl module directory
13. Configure the Modssl module:

```
$ ./configure  --with-apache=../apache-<VERSION> \
                 --with-ssl=../openssl-engine-<VERSION> \
                 --enable-module=ssl \
                 --enable-rule=SSL_EXPERIMENTAL \
```

14. Change to the Apache directory
15. Configure, compile, create self-sign certificate and install Apache:

```
$ ./configure  --prefix=/usr/local/apache \
                 --enable-module=ssl
$ make
$ make certificate TYPE=custom # Here you can type in
                                  # the self-sign certificate data
$ make install
```

16. Define the crypto device in the httpd.conf file:

```
<IfModule mod_ssl.c>
SSLCryptoDevice ibmca
...
something else
...
</IfModule>
```

This will activate the hardware.

To check your set-up:
1. Start Apache
2. With your favorite Web browser, try to establish a secure connection (an https connection)

If Apache comes up on your browser, the connection and encryption are OK.

Chapter 8. SCSI-over-Fibre Channel driver

The SCSI-over-Fibre Channel driver (zfcp) provides support for Fibre Channel attached SCSI devices on Linux for zSeries. This chapter details the usage of the SCSI driver (zfcp) for the QDIO-based zSeries FCP (zSeries SCSI over Fibre Channel) adapter.

The zfcp driver is a back-end (i.e., low-level or host-bus adapter driver) supplementing the Linux SCSI I/O subsystem (SCSI stack). Thus, Linux for zSeries can make use of all SCSI device types currently supported by Linux on other platforms. These include SCSI disks, tapes, CD-ROMs, and DVDs, for example. SCSI-over-Fibre Channel support via zfcp requires a zSeries z900, z800, or later mainframe. zSeries FCP adapters do not work with older S/390 models. Both ESA (31 bit Linux) and ESAME (64 bit Linux) are supported.

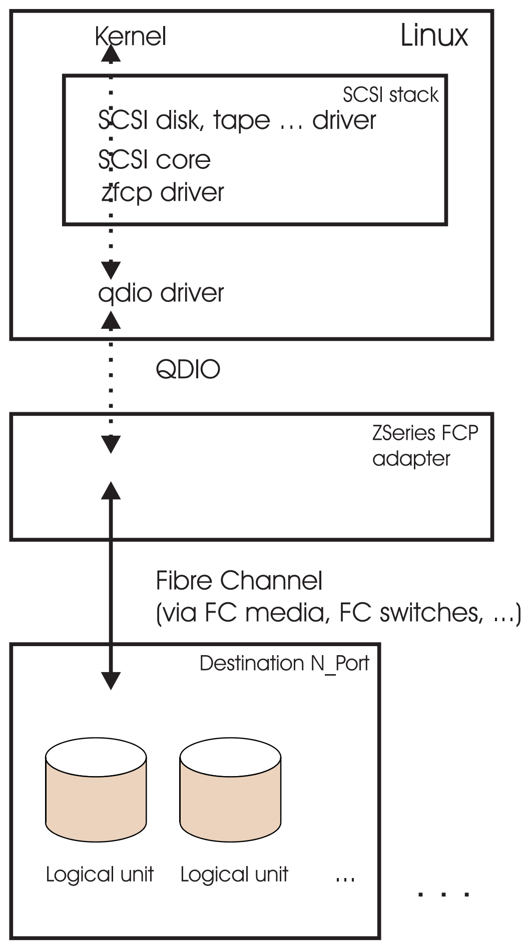


Figure 3. General layout of zSeries FCP environment

Configuring zfcp

There are different parameters which can or must be supplied by the user to allow for proper zfcp operation:

- address mappings between Linux SCSI and FCP schemes (optional for each SCSI target)
- logging level to determine the verbosity of the zfcp device driver (optional, default value is used if not supplied)

For these purposes, the zfcp driver provides different means of configuration:
- kernel parameters
- module parameters (such as for use in modules.conf)
- various proc-fs entries in /proc/scsi/zfcp

The following table shows corresponding module parameters and proc-fs entries. These allow configuring parameters or displaying configured parameters.

Table 4. zfcp functions, parameters, and proc_fs entries

Function	Module parameter	Kernel parameter	proc-fs entry
Set logging level	loglevel	zfcp_loglevel	/proc/scsi/zfcp/mod_parm
Get logging level (and other global module information)	(not applicable)	(not applicable)	/proc/scsi/zfcp/mod_parm
Add address mapping(s)	map	zfcp_map	/proc/scsi/zfcp/add_map
Get all existing address mappings	(not applicable)	(not applicable)	/proc/scsi/zfcp/map

In general, the syntax/format of module/kernel parameters and the input to and output from corresponding proc-fs entries are very similar. This is explained by the following examples.

Usage of the ″map″ module parameter and ″zfcp_map″ kernel parameter:

```
Module:
map="<devno> <port scsi-id>:<wwpn> <unit scsi-lun>:<fcp-lun>; ..."

Kernel:
zfcp_map="<devno> <port scsi-id>:<wwpn> <unit scsi-lun>:<fcp-lun>; ..."
```

where:

devno S/390 device number of the zSeries FCP adapter (16 bit, see /proc/subchannels or IOCDS)

port scsi-id
Linux internal SCSI ID assigned to the Fibre Channel port of the SCSI target device (32 bits, must not be 0, must be unique one-to-one mapping for each WWPN)

Port SCSI IDs must be unique for a particular adapter. It is valid to assign identical SCSI IDs to ports attached via different adapters (devnos). For example:

```
0x1200 1:0x3628745827682600 0:0x1234000000000000
0x1300 1:0x9862876876000000 0:0x1235000000000000
```

Similarly, SCSI LUNs are assigned in the scope of a particular port.

wwpn World Wide Port Number identifying the Fibre Channel port of the SCSI target device (64 bits, according to the FCP/SCSI-3 specifications (see “References” on page 65)

unit scsi-lun
: Linux internal SCSI logical unit number (32 bit, must be a unique one-to-one mapping for each FCP LUN, must start from 0 and increment by 1 for each new LUN on the same port and adapter)l.

  Unit SCSI LUNs must be unique for a particular part. It is valid to assign identical SCSI LUNs to units attached via different ports (WWPN).

fcp-lun
: Number of the logical unit as defined by FCP (64 bits, according to the FCP/SCSI-3 specifications (see “References” on page 65)

The following requirements apply to address mappings:

- **Syntax:** Allowed separators between device number and port mapping are spaces and/or horizontal tabs. This is also true for the separation of port mapping and unit mapping. Spaces and/or horizontal tabs are allowed at the beginning and end of each record. The elements comprising the two mapping groups (port and unit) must be separated by colons; here, no spaces are allowed. Several address mapping records (consisting of device number, port mapping, and unit mapping) may be specified by separating them with a semicolon as indicated. Comments may be used at the end of any address mapping line or on separate lines. They start with a ’#’ character and continue up to the end of the current line. Comments are discarded by zfcp on address mapping processing. That is why user comments are not available via /proc/scsi/zfcp/map.
- **Number formats:** Numbers can be in the decimal, octal, or hexadecimal systems following the C programming language conventions regarding prefixes.
- **Allowed values:** The zfcp driver assigns SCSI ID 0 to each zSeries FCP adapter by default. *For this reason, the ID 0 is not available for port mappings and must not be assigned by the user.*
- **Duplicate or conflicting device declarations:** The user must not configure more than one address mapping record for a single SCSI device. Here, a SCSI device is uniquely identified by the combination of adapter device number, port WWPN, and FCP LUN.

If an invalid address mapping is specified, the module load or kernel-based FCP functionality will fail. It is then the user’s responsibility to resolve the problem by reloading the module, or rebooting the kernel, using the corrected address mappings.

Examples:

```
Module:

insmod zfcp.o map="\
0x4200 0x1:0x4268426842684268 0x0:0x420a000000000000;\
0x4200 0x1:0x4268426842684268 0x1:0x420b000000000000"
```

Two devices would be configured on the adapter of device number 0x4200, on the port with SCSI ID 0x1, WWPN 0x4268426842684268. More specifically, the two devices have SCSI LUNs 0x0 and 0x1, which correspond to FCP LUNs 0x420a000000000000 and 0x420b000000000000, respectively.

```
Kernel:

zfcp_map="0x4200 0x1:0x4268426842684268 0x0:0x4209000000000000"
root=/dev/scsi/host0/bus0/target1/lun0/part1
```

This would configure one device on the adapter of device number 0x4200, on the port with SCSI ID 0x1, WWPN 0x4268426842684268. The device will have SCSI LUN 0x0, corresponding to FCP LUN 0x4209000000000000, and be addressable by the indicated devfs node.

Usage of the ″/proc/scsi/zfcp/add_map″ proc-fs entry:

This interface accepts address mapping records with a syntax similar to that of the corresponding module parameter. See above for the syntax of a record describing a particular address mapping. In contrast to the corresponding module parameter, new-line characters may be used instead of or in addition to semicolons to separate multiple address mapping records. This renders address mapping records maintained in a user defined configuration file more readable.

A second difference is that any invalid input will result in the complete or partial rejection of the erroneous record (e.g., a new adapter may be configured before an erroneous port-mapping in the same record is detected) and the complete rejection of all subsequent records. Records prior to an erroneous record are processed normally.

Examples for specifying address mappings using the ″/proc/scsi/zfcp/add_map″ proc-fs entry:

```
echo "\
0x4200 0x1:0x4268426842684268 0x2:0x420c000000000000;\
0x4200 0x1:0x4268426842684268 0x3:0x420d000000000000"\
> /proc/scsi/zfcp/add_map

or

cat my_mapping.txt > /proc/scsi/zfcp/add_map
```

where the content of my_mapping.txt could be:

```
# my 2 Shark disks
0x4200 0x1:0x4268426842684268 0x2:0x420c000000000000
0x4200 0x1:0x4268426842684268 0x3:0x420d000000000000   # mounted on /home
```

Usage of the ″/proc/scsi/zfcp/map″ proc-fs entry:

This file provides all configured address mappings. Its record format is the same as that used for specifying new mappings with ″/proc/scsi/zfcp/add_map″ (note: only line-breaks are used as separators for this proc-fs entry).

Example for the output of ″/proc/scsi/zfcp/map″:

```
cat /proc/scsi/zfcp/map

0x4200 0x1:0x4268426842684268 0x0:0x420a000000000000
0x4200 0x1:0x4268426842684268 0x1:0x420b000000000000
0x4200 0x1:0x4268426842684268 0x2:0x420c000000000000
0x4200 0x1:0x4268426842684268 0x3:0x420d000000000000
```

How to trigger a reopen error recovery procedure:

To force a reopen procedure of an adapter, port, or unit by the error recovery subsystem, the string ″reset erp″ can be echo'ed into the respective ″status″ file in the proc-filesystem. For example, the command

```
echo "reset erp" >/proc/scsi/zfcp/devno0x4200/id0x1/lun0x0/status
```

triggers a reopen procedure on LUN 0 of port 1 of adapter 0x4200. The error recovery subsystem will consequently try to close the LUN (if it is still open) and reopen it afterwards.

Note: Error recovery might impact the ongoing I/O of
- the current Linux system, regardless of how error recovery is triggered (manually by the user or automatically by zfcp)
- other Linux systems (sharing environment)

Usage of the "loglevel" module parameter and "zfcp_loglevel" kernel parameter:
These parameters allow specifying the level of verbosity of the zfcp driver. The logging level consists of several sub-levels for the various functional parts of the module. Each higher level implicitly also includes all lower levels, e.g., specifying a 1 in a particular category will supply the user with the output for problems reported with either levels 0 and 1.

Currently, the following levels are specified:

0 **normal** (default: maximum performance, only critical problems are reported)
- zfcp version
- syntax/semantic errors in address mappings
- detected external conditions which restrict or prevent zfcp operation (e.g., another OS instance owns exclusive access to a configured logical unit via same Fibre Channel link)
- unexpected hardware behavior
- unexpected zfcp behavior

1 **info** (supplies extra info about the origin of problems)
- various memory constraints

2 **debug** (not intended for user interpretation: information relevant to program flow and debugging)

3 **trace** (not intended for user interpretation: all output is present)

The functional categories these numbers pertain to are defined by their position within the hexadecimal string. Each category is represented by one nibble of the 4-byte value:

```
Module:
loglevel=0xueqdcfso

Kernel:
zfcp_loglevel=0xueqdcfso
```

where the letters describing the output categories are defined as:

u Currently unassigned

e Error handler (functions concerned with FCP error handling)

q QDIO (functions dealing with the QDIO interface/module)

d Dynamic I/O (functions dealing with dynamic attachment and detachment of adapters)

c Configuration (functions dealing with configuration changes caused by the utilization of user interfaces)

f zSeries FCP (protocol related functions)

s SCSI (interface functionality to the Linux SCSI stack)

o Other (miscellaneous functionality)

It is recommended to always use a loglevel of 0x00000000 for normal use. All serious problems will be caught in this way. If a problem occurs repetitively and the cause cannot be identified, the level may be increased to 1 in the corresponding area.

All output is written to the VM terminal or service element console. It should also be present in /var/log/messages (assuming a correct setup in /etc/syslog.conf).

Examples:

```
insmod zfcp.o loglevel=0x00123001
```

This example would load the module with the log level set to ″normal″ for the categories ″error handler″, ″zSeries FCP″, and ″SCSI″, ″info″ for ″QDIO″ and ″other″, ″debug″ for ″dynamic I/O″ and ″trace″ for ″configuration″.

```
zfcp_loglevel=0x01111221 root=/dev/dasd/4abc/part1
```

This example would activate logging within the kernel with the log level set to ″info″ for the categories ″error handler″, ″QDIO″, ″dynamic I/O″, ″configuration″, and ″other″, and ″debug″ for the categories ″OpenFCP″ and ″SCSI″.

Usage of the ″/proc/scsi/zfcp/mod_parm″ proc-fs entry to display/modify the current logging level:

This interface can be used to set or display the current logging level. The syntax is similar to that of the corresponding module/kernel parameter. See above for details on the logging level specifications.

At the time of writing, the ″loglevel″ parameter is the only global parameter of the zfcp driver configurable via ″/proc/scsi/zfcp/mod_parm″.

Example for specifying a logging level using the ″/proc/scsi/zfcp/mod_parm″ proc-fs entry:

```
echo "loglevel=0x00123001" > /proc/scsi/zfcp/mod_parm
```

This will adjust the level of output in all sections to the same values used in the above module-load example.

Example for displaying the current logging level and other global module information using the ″/proc/scsi/zfcp/mod_parm″ proc-fs entry:

```
cat /proc/scsi/zfcp/mod_parm

Module Information:

Module Version $Revision: 1.11 $ running in mode: FULL FEATURED
Debug proc output enabled:   YES

Full log-level is:    0x00222222 which means:
QDIO log-level:              2     Dynamic IO log-level:             2
```

```
Configuration log-level:       2     zSeries FCP log-level:            2
SCSI log-level:                2     Other log-level:                  2

Registered Adapters:           1
```

Device-specific information is present in the status file for each of the three different device categories: adapter, port, and unit. Any one of them can be found within a dynamic device directory and device file tree under the

```
/proc/scsi/zfcp/
```

directory.

Within this directory, new entries are created whenever a new device is added to the existing configuration and removed whenever the device is removed from the module, according to the following scheme:

```
devno0x<no>/status
devno0x<no>/id0x<no>/status
devno0x<no>/id0x<no>/lun0x<no>/status
```

where <no> is a placeholder for an actual number and

devno0x<no>
: specifies the device number of the adapter

id0x<no>
: specifies the port SCSI ID

lun0x<no>
: specifies the unit SCSI LUN

Thus, the hexadecimal numbers following devno, id, or lun are simply those with which the device has been specified previously, via module parameter 'map', kernel parameter 'zfcp_map', or the proc file system (add_map).

If more than one adapter is configured, several devno0x<no> directories will be present, each containing a status file, which upon read will reveal details about the underlying Fibre Channel topology, version information, Fibre Channel protocol parameters, and SCSI-stack setup.

Similarly, various id0x<no> directories can exist under each devno0x<no> directory in case of multiple ports per adapter. From the file within these, SCSI, Fibre Channel and general information of relevance to the port may be determined.

Finally, different lun0x<no> directories can be present in each id0x<no> directory, one for each unit connected to the above port. Not surprisingly, the status file found at this location reveals Fibre Channel, SCSI, and general information pertaining to the unit under consideration.

Consider two sample uses of this scheme:

Example 1: Finding all SCSI units (identified by SCSI LUNs) connected to the adapter with devno0x6842 and port of SCSI ID 0x12345678:

```
ls /proc/scsi/zfcp/devno0x6842/id0x12345678
```

This will display one status file (for the port), and various directory entries for all the SCSI units.

Example 2: Assuming that example 1 revealed a SCSI unit of lun0xd0d0d0d0, among other things, its status information is displayed with

```
cat /proc/scsi/zfcp/devno0x6842/id0x12345678/lun0xd0d0d0d0/status
```

devfs and zfcp mapping

On a system without devfs, there is no obvious correlation between the SCSI mapping in zfcp (/proc/scsi/zfcp/map) and the device nodes in the /dev directory.

However, if devfs is enabled, such a relation exists. In this case, each /dev entry for a SCSI device looks like

```
/dev/scsi/host<no>/bus0/target<no>/lun<no>/<device-type or partition>
```

Here, the host number is an integer incrementing from 0, identifying each SCSI adapter as it appears. For example, the first adapter would be 0, the second 1, and so forth.

The bus number is not used for devices accessed via zfcp and will always be 0.

The target number is the SCSI port ID as specified in the corresponding map entry.

The LUN number is the SCSI LUN as specified in the corresponding map entry.

Example: The /dev entry:

```
/dev/scsi/host0/bus0/target1/lun0/disc
```

should be present when the zfcp module has been loaded, for example:

```
insmod zfcp.o \
map="0x1234 0x1:0x345adf3322443525 0x0:0x01f000000000000"
```

and sd_mod is also loaded (for the /disc identification) and the device specified is actually a SCSI disk. The 0x1/target1 and 0x0/lun0 correspondence can be seen. The bus number is 0 as usual. Finally, only one adapter (0x1234) is specified and hence is the first to appear; its host-number is 0.

Sample walkthrough

Pre-requisites: A zSeries z900 (Freeway) or z800 (Raptor) with a zSeries FCP adapter, a correctly configured Fibre Channel SAN.

1. Install Linux on an LPAR (native or VM). Details about the installation procedure can be found at the DeveloperWorks site.
2. Obtain the zfcp.o module and copy it to your system.
3. Load the modules required by zfcp, e.g.:
   ```
   modprobe qdio
   modprobe scsi_mod
   ```
4. Load zfcp, specifying the initial disk and/or loglevel, if desired (both optional), e.g.:
   ```
   insmod zfcp.o  \
   loglevel=0x00000000 map="0x4202 0x1:0x2422067800001000 \
   0x0:0x510f000000000000"
   ```

The devno (0x4202), WWPN (0x2422067800001000) and FCP LUN (0x510f000000000000) should be taken from your current SAN setup and the zSeries IOCDS.

5. Specify additional devices via add_map (optional), e.g.:
```
echo "0x4202 0x1:0x2422067800001000 0x1:0x5112000000000000 \
0x4202 0x1:0x2422067800001000 0x2:0x5103000000000000" \
> /proc/scsi/zfcp/add_map
```
6. Make the devices known to the SCSI stack:
```
echo "scsi add-single-device 0 0 1 1" > /proc/scsi/scsi
echo "scsi add-single-device 0 0 1 2" > /proc/scsi/scsi
```
   Note that the numbers represent host (0), bus (0), target (1), lun(1/2) in the same manner and order as explained for devfs above.

   `add-single-device` is required only for devices that have been added manually (via `/proc/scsi/zfcp/add_map`). For more information on the use of the Linux SCSI stack, see the SCSI-stack HOWTO.
7. Load the module(s) for the media you want to access:
   - SCSI disk:
     ```
     modprobe sd_mod
     ```
   - SCSI tape:
     ```
     modprobe st
     ```
   - SCSI CD/DVD (read-only):
     ```
     modprobe sr_mod
     ```
   - SCSI generic (special interface to all device types, e.g. for CD burning):
     ```
     modprobe sg
     ```
8. Create and mount a filesystem on a SCSI disk:
   - Partition a disk (see ″man fdisk″):
     ```
     fdisk /dev/scsi/host0/bus0/target1/lun0/disc
     ```
   - Create a filesystem on the first partition:
     ```
     mke2fs /dev/scsi/host0/bus0/target1/lun0/part1
     ```
   - Mount it:
     ```
     mount /dev/scsi/host0/bus0/target1/lun0/part1 /mnt
     ```

Installation considerations

Notes:

1. Currently, SCSI/FCP is not enabled in the microcode by default. If you are getting ″definition errors″ for your FCP CHPIDs in the CHPIDs Work Area, please contact your hardware support for a recent microcode fix that enables SCSI/FCP.
2. The zfcp driver requires version 2 of QDIO for Linux and is not compatible with any older version.
3. Relevant higher-level, common-code SCSI drivers must be present either as modules or compiled into the kernel:

Table 5. SCSI module dependencies

Functionality	Kernel config option	Module name
zfcp	CONFIG_ZFCP (mandatory)	zfcp.o

Table 5. SCSI module dependencies (continued)

qdio	CONFIG_QDIO (mandatory)	qdio.o
SCSI core	CONFIG_SCSI (mandatory) CONFIG_SCSI_MULTI_LUN (strongly recommended) CONFIG_SCSI_LOGGING (recommended)	scsi_mod.o
SCSI disks	CONFIG_BLK_DEV_SD	sd_mod.o
SCSI tapes	CONFIG_CHR_DEV_ST	st.o
SCSI CD-ROM	CONFIG_BLK_DEV_SR	sr_mod.o
SCSI generic	CONFIG_BLK_DEV_SG	sg.o

Furthermore, the kernel configuration options for the required filesystems (e.g. ISO 9660 for SCSI CD-ROM support) and partition types (PC-BIOS disk layout) should be selected if desired. For partitioning SCSI disks, the `fdisk` application should be used and not `fdasd`.

4. Partitioning of SCSI disks will only work if the PC-BIOS disk layout is compiled into the kernel.

Restrictions

- The zfcp module only supports FC switched fabric. FC point-to-point attachments are not supported by zfcp even if the zSeries FCP adapter could do so. Direct FC-arbitrated loop attachment is not supported by zfcp or the zSeries FCP adapter.
- Address mappings issued to zfcp currently cannot be removed ’on the fly’. The recommended way to change existing address mappings is to reload the zfcp module. This implies disruption and restart of SCSI operation of a particular Linux instance.
- The data transfer mechanism provided by the zSeries FCP adapter restricts the maximum data transfer size of single SCSI commands. This maximum amount of data is 58 page-aligned pages with 4096 bytes each. This amount decreases accordingly for non-aligned data buffers. This is not an issue for SCSI block devices (SCSI disks driven by `sd` or SCSI CD-ROM/DVD devices driven by `sr`) in the current Linux implementation. It might be an issue for SCSI character devices (SCSI tape devices driven by `st` or other SCSI devices, like CD writers or scanners, driven by sg, depending on the particular SCSI stack utilization of these device drivers. The SCSI tape device driver (`st`) allows users to configure the actual data transfer size per SCSI command. The default value (30 KB) is sufficient and recommended from the zfcp perspective. The actual data transfer size of SCSI commands routed via the generic SCSI device driver (`sg`) may differ for several SCSI applications.
- Vary on/off support is not yet implemented.
- VM support by zfcp has not yet been defined.

Considerations for future distributions

The following are suggestions for inclusion in Linux distributions:

Device filesystem

The legacy naming scheme used for SCSI devices (e.g. /dev/hda, ...) is not sufficient for SAN environments. For example, a single device which is temporarily unavailable and which could not be scanned might cause large inconsistencies regarding device identification. The device filesystem would resolve such SCSI device identification issues, except for HBA identification (/dev/scsi/host0/bus0/target1/lun0). The issue of HBA identification is resolved in the scope of zfcp.

'sd_many' patch
The SCSI disk driver of current Linux 2.4 kernels only supports a maximum of 128 SCSI disks. The so-called 'sd_many' patch (Richard Gooch) raises this limit to approximately 2000 SCSI disks. This external patch is not supported by IBM, but it might be used at one's own risk.

Persistent configuration
A configuration file should be used by the distribution to store zfcp address mappings and thus allow for persistent configurations (across reboot).

References

The following are sources of additional information for zSeries FCP:

FCP/SCSI-3 specifications
Describes SCSI-3, the Fibre Channel Protocol, and related information (www.t10.org and www.t11.org).

SCSI-stack HOWTO
Information about SCSI and devfs, parameters and proc-fs (The Linux 2.4 SCSI subsystem HOWTO, by Douglas Gilbert, http://www.linuxdoc.org/HOWTO/SCSI-2.4-HOWTO/index.html)

Part 3. Linux for zSeries Network device drivers

These chapters describe the channel device layer and the device drivers available to connect zSeries systems to your network.

The drivers described are:

- Chapter 10, “Linux for zSeries CTC/ESCON device driver” on page 79
- Chapter 11, “Linux for zSeries IUCV device driver” on page 87
- Chapter 12, “Linux for zSeries LCS device driver” on page 109
- Chapter 13, “QETH device driver for OSA-Express (QDIO) and HiperSockets” on page 113

License conditions

Some of these drivers are subject to license conditions as reflected in: “International License Agreement for Non-Warranted Programs” on page 233.

Chapter 9. Linux for zSeries Channel device layer

The channel device layer provides a common interface to Linux for zSeries channel devices. You can use this interface to configure the devices and to handle machine checks (devices appearing and disappearing).

The drivers using the channel device layer at the time of writing are:

1. **LCS** – supports OSA-2 Ethernet Token Ring, and OSA-Express Fast Ethernet in non-QDIO mode.
2. **CTC/ESCON** – high speed serial link
3. **QETH** – supports OSA-Express features in QDIO mode and HiperSockets.

The channel device layer draws together the configuration of the drivers and resolves conflicts. These could, for example, result in the `LCS` and `CTC` drivers in contention for 3088/08 and 3088/1F devices (which could be either 2216/3172 `LCS` compatible devices or `ESCON/CTC`). To resolve the clashing without the channel device layer, each of these device drivers had to be configured separately, with a check for conflicts performed visually.

The channel device layer is used on a per-driver basis, not on a system basis. For example a CTC driver which is not configured to use the channel device layer can be used in conjunction with an LCS driver which is configured to use it.

Description

The current configuration of the channel device layer is held (in human readable form) in the file `/proc/chandev`.

You can pass arguments to the channel device layer in three ways:

1. Piping them to `/proc/chandev`, for example:
   ```
   echo reprobe >/proc/chandev
   ```
   will cause un-initialized channel devices to be probed.
2. Editing them into `/etc/chandev.conf` – this will only take effect after a reboot of after executing the sequence of commands mentioned in “Read configuration” on page 77. You can also add comments to the configuration file. Comment lines must be prefixed with a ’#’ character.
3. Using the `'chandev='` keyword on the Linux boot command line, for example:
   ```
   chandev=noauto,0x0,0x480d;noauto,0x4810,0xffff
   ```
   will exclude all devices from auto-detection except for subchannels 0x480e and 0x480f.

Multiple options can be passed, separated by commas, but no spaces are allowed between parameters.

To be consistent with other hot-pluggable architectures, the script pointed to by `/proc/sys/kernel/hotplug` (this will normally be `/sbin/hotplug`) will be called automatically on startup or on a device machine check as follows:.

Channel device layer

```
/sbin/hotplug chandev <start starting_devnames>
                      <machine_check (devname last/pre_recovery_status)
                                        (current/post_recovery_status)>.
```

The channel device layer does not open `stdin`, `stdout`, or `stderr` so it is advisable that you open them at the start of your script, as in this sample which starts devices as they become available:

```
#!/bin/bash
exec >/dev/console 2>&1 0>&1
# Remove the comment symbol from the line below for debugging purposes.
# echo $*
if [ "$1" = "chandev" ] && [ "$2" = "start" ]
then
    shift 2
    while [ "$1" != "" ]  && [ "$1" != "machine_check" ]
    do
        isup='ifconfig $1 2>/dev/null | grep UP'
        if [ "$isup" = "" ]
        then
            ifup $1
        fi
        shift
    done
fi
```

For example if devices `tr0` and `ctc0` become active at a time when `eth0` and `eth1` are subject to a `device_gone` machine check and `eth2` is subject to a `revalidate` machine check (which is normally fully recoverable), the parameters passed to `hotplug` would be:

```
/sbin/hotplug chandev start tr0 ctc0
                  machine_check eth0 gone gone eth1 gone gone
                                eth2 revalidate good
```

This script can be used, for example, to call `/etc/rc.d/init.d/network start` when a device appears. (This makes the `ipldelay` kernel boot parameter obsolete when Linux is running native.) It may also be used to recover from bad machine checks if the default machine check handling is inadequate. The machine checks that can be presented as parameters to the channel device layer are `good`, `not_operational`, `no_path`, `revalidate` and `device_gone`.

The channel device layer will wait a few seconds after machine checks before running `/sbin/hotplug` because a machine check on one device is often followed by checks on others. It is better to handle multiple devices with a single script, rather than with individual scripts for each device, which could compete for resources.

Channel device layer options

Terminology

devno a 16 bit unsigned number (usually expressed as hexadecimal) which uniquely identifies a subchannel connected to a device.

force list
a term (specific to channel device layer) describing a range of *devno* which are to be configured specifically (as opposed to configuration by auto-detection).

auto machine check recovery bitfield

The bits in this field signify:

not_operational
0x1

no_path
0x2

revalidate
0x4

device_gone
0x8

chan_type bitfield

The bits in this field signify:

ctc 0x01

escon 0x02

lcs 0x04

osad 0x08 – reserved, not used in this release

qeth 0x10

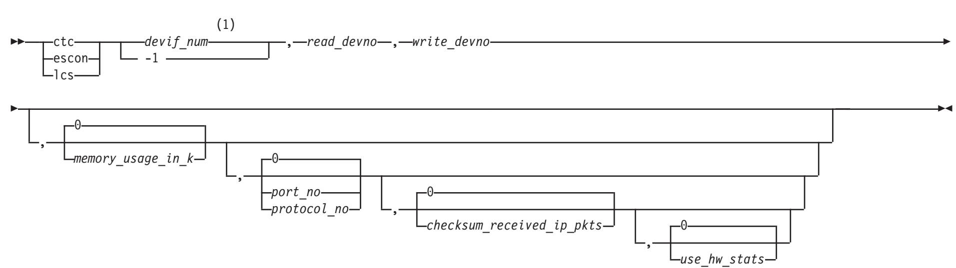
A single device driver may handle more than one type of device. In this case the values corresponding to each device handled are summed to create the parameter

Device identification (CTC/ESCON and LCS)

This section describes how to use the channel device layer to control `CTC`, `ESCON` and `LCS` devices.

The CTC/ESCON and LCS drivers are configured (for a single device) with the command:

CTC, ESCON, LCS device identification



Notes:

1 *devif_num* is concatenated to the device type, for example `ctc1`, `escon7` or `lcs2`.

Where:

ctc | escon | lcs
specifies the channel device type

devif_num
is the device interface number.

This can be 0 to 255 for a specific number, or '-1' to indicate you do not care which device interface number is chosen.

read_devno
is the read device address.

write_devno
is the write device address.

memory_usage_in_k
is the memory to be allocated for buffers. The default (zero) means 'let the driver decide'.

port_no
is the relative adapter number for LCS.

protocol_no
is the protocol number for CTC or ESCON. This can take the values:
- 0 for compatibility mode (the default; used with non-Linux peers other than OS/390 and z/OS)
- 1 for extended mode,
- 2 meaning "CTC-based tty" (this is only supported on Linux -Linux connections),
- 3 for compatibility mode with OS/390 and z/OS.

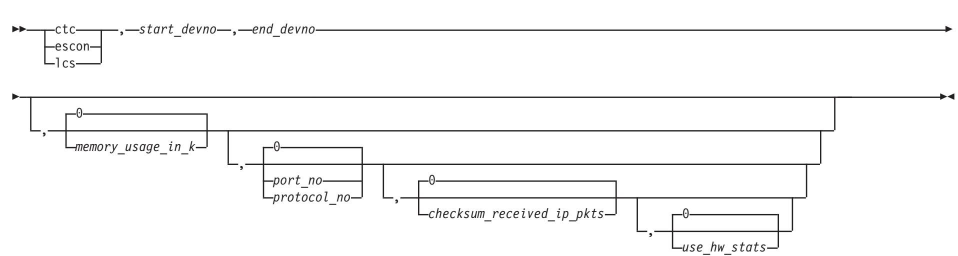
checksum_received_ip_pkts
is a flag: '1' = true; '0' (default) = false.

use_hw_stats
is a flag: '1' = true; '0' (default) = false.

For examples of device identification see:
- CTC, ESCON: "Configuration examples" on page 79
- LCS: "LCS channel device layer configuration example" on page 109

The CTC/ESCON and LCS drivers are configured for a range of devices with the command:

CTC, ESCON, LCS device range



Where:

start_devno
is the start address of a range.

end_devno
is the end address of a range.

The rest of the parameters are as above. All addresses within the specified range will be scanned and any devices found which match the device type specified will be assigned.

For examples of device range identification see:
- CTC, ESCON: “Configuration examples” on page 79
- LCS: “LCS channel device layer configuration example” on page 109

Device identification (QDIO)

For the syntax of the QETH device driver for the OSA-Express feature or HiperSockets with the channel device layer see “Configuring QETH for OSA-Express and HiperSockets using the channel device layer” on page 115.

Commonly used options

These options are used to set up the system.

add parameters

```
                               ┌─all devnos────────────────┐
►►──add_parms──,──chan_type────┼───────────────────────────┼──,──string──────►◄
                               └─,──lo_devno──,──high_devno─┘
```

`chan_type` is defined in ***chan_type bitfield*** in the terminology on page 71.

This is for device driver specific options which are passed as a string to the driver and are not dealt with by the channel device layer. This string cannot contain spaces. ***lo_devno*** and ***high_devno*** are optional parameters to specify a range.

The *string* is interpreted by the driver (see the particular driver chapter for details).

delete parameters

```
              ┌─all────────────────────────────────────────────┐
►►──del_parms─┼────────────────────────────────────────────────┼──────►◄
              └─,──chan_type─┬───────────────────────────────┬─┘
                             └─,──exact_match─┬────────────┬─┘
                                              └─,──lo_devno─┘
```

chan_type is defined in the terminology on page 71.

This deletes some or all device driver specific options. If `chan_type` is not specified all the strings will be deleted. If ***exact_match*** is set to ’1’ the driver parameters will only be removed where ***chan_type*** is exactly equal. If ***exact_match*** is set to ’0’ the parameters are to be removed where any bit matches ***chan_type***. ***lo_devno*** is an optional parameter to specify that the delete is only to happen if this parameter matches a ***lo_devno*** in a defined range.

no auto-detection

```
         ┌─all──────────────────────────┐
►►─noauto─┴─,─lo_devno─,─high_devno─┴──────────────────────►◄
```

This stops auto-detection of channel devices in the given range of device numbers. `noauto` without a device range will stop auto-detection of all channel devices.

use device names

```
►►─use_devno_names──────────────────────────────────────────►◄
```

This instructs the channel device layer to assign device names based on the `cuu` number of the read channel. For example a Token Ring read channel with `cuu` number 0x7c00 would be assigned an interface name of `tr0x7c00`. This may be used to avoid device name conflicts. The default is to generate device names in sequence, so the default name for the channel above might be `tr2`.

Options for power users

These options are used for maintenance or fine-tuning.

delete no-auto ranges

```
             ┌─all─────┐
►►─del_noauto─┴─,─devno─┴───────────────────────────────────►◄
```

Delete the range containing `devno`, or all `noauto` ranges if `devno` is not given.

delete forced device

```
►►─del_force─,─read_devno───────────────────────────────────►◄
```

Remove a forced channel device from the force list.

do not use device names

```
►►─dont_use_devno_names─────────────────────────────────────►◄
```

Cancel a **`use_devno_names`** command.

add a device

```
►►—add_model—,—chan_type—,—cu_type—,—cu_model—,—dev_type—,—dev_model—,——————►
►—max_port_no—,—automatic_machine_check_handling——————————————————————————►◄
```

Probe for the device specified. '-1' may be used as a wildcard for any of the parameters except ***chan_type*** or ***automatic_machine_check_handling***. Set ***max_port_no*** to zero ('0') for non LCS devices.

chan_type and ***automatic_machine_check_handling*** are defined in the terminology on page 70.

delete a device

```
►►—del_model—,—cu_type—,—cu_model—,—dev_type—,—dev_model——————————►◄
```

Remove the device specified. '-1' may be used as a wildcard for any of the parameters.

delete all devices

```
►►—del_all_models——————————————————————————————————————————►◄
```

Remove all devices.

auto-detect any devices

```
►►—non_cautious_auto_detect————————————————————————————————►◄
```

Attempt to auto-detect devices even if their type/model pairs do not unambiguously identify the device. For example 3088/1F's can either be CTC/ESCON or 3172 LCS compatible devices. If the wrong device driver attempts to probe these channels there may be long delays on startup or even a kernel lockup, so use this option with caution.

auto-detect known devices

```
►►—cautious_auto_detect————————————————————————————————————►◄
```

Do not attempt to auto-detect devices unless their type/model pairs unambiguously identify the device. (This is the default behavior.)

machine check recovery

```
            ┌─all devnos──────────────────┐
►►──auto_msck──┼─────────────────────────────┼──,──auto_msck_recovery──────►◄
            └─,──lo_devno──,──high_devno──┘
```

Specify the kind of machine check recovery to be performed over a range of devices. ***auto_msck_recovery*** is defined in the terminology on page 70.

delete machine check recovery

```
                ┌─all──────┐
►►──del_auto_msck──┼──────────┼──────────────────────────────►◄
                └─,──devno─┘
```

Delete machine check recovery for the range of devices including ***devno***, or all machine check recovery if ***devno*** is not specified.

null model information

```
►►──reset_clean──────────────────────────────────────────────►◄
```

Reset all model information, forced devices and `noauto` lists to null.

default model information

```
►►──reset_conf───────────────────────────────────────────────►◄
```

Reset all model information, forced devices and `noauto` lists to default settings.

empty model information

```
►►──reset_conf_clean─────────────────────────────────────────►◄
```

Reset all model information, forced devices and `noauto` lists to empty.

shutdown device

```
>>--shutdown--+-all-----------+-------------------------------><
              +-device_name-+-+
              '-read_devno--'
```

Shut down the particular device identified by ***device_name*** or ***read_devno***, de-register it and release its interrupts. If no parameter is given all devices are shut down.

reprobe

```
>>--reprobe---------------------------------------------------><
```

Call probe method for channels whose interrupts are not owned.

unregister general probe

```
>>--unregister_probe--+-all------------+----------------------><
                      '-probefunc_addr-'
```

Unregister a probe function, or unregister all probe functions if no address given.

unregister specific probe

```
>>--unregister_probe_by_chan_type--chan_type------------------><
```

Unregister all probe functions which match the `chan_type` bitfield exactly. This is useful if you want a configuration to survive a kernel upgrade.

read configuration

```
>>--read_conf-------------------------------------------------><
```

Read instructions from `/etc/chandev.conf`.

This is used to make the channel device layer read from `/etc/chandev.conf` on boot, or to cause the channel device layer to re-read its configuration during operation.

do not read configuration

```
►►──dont_read_conf──────────────────────────────────────────────►◄
```

Do not read instructions from `/etc/chandev.conf` on boot.

For example the following sequence of commands piped to `/proc/chandev` should have the same effect as rebooting for channel devices:
- `shutdown`
- `reset_conf`
- `read_conf`
- `reprobe`

See also

If you wish to write a driver which is compatible with the channel device layer see:
- `/linux/include/asm-s390/chandev.h` – for the API (which is commented), and
- `/linux/drivers/s390/misc/chandev.c` – for the code.

Files

`/proc/chandev`
This holds the current configuration. Use

```
cat /proc/chandev
```

to see the configuration, and

```
echo command >/proc/chandev
```

to enter a new command.

`/etc/chandev.conf`
This file can be used to configure the channel device layer kernel parameters.

`/sbin/hotplug`
This is a user script or executable which is run whenever devices come online or go offline ('appear' or 'disappear').

Chapter 10. Linux for zSeries CTC/ESCON device driver

A CTC connection or an ESCON connection is the typical high speed connection between mainframes. The data packages and the protocol of both connections are the same. The difference between them is the physical channel used to transfer the data.

Both types of connection may be used to connect a mainframe, an LPAR, or a VM guest to another mainframe, LPAR or VM guest, where the peer LPAR or VM guest may reside on the same or on a different system.

A third type of connection is virtual CTC which is a software connection between two VM guests on the same VM system and which is faster then a physical connection.

The Linux for zSeries CTC device driver supports all three types of connection and can be used to establish a point-to-point TCP/IP connection between two Linux for zSeries systems or between a Linux for zSeries system and another operating system such as VM/ESA, VSE/ESA, Linux for S/390, OS/390 or z/OS.

CTC/ESCON features

- Any number of CTC and/or ESCON connections available.
- Autosense mode available (the driver will pick all available channels starting with the lowest device numbers).
- If built monolithically (not as a module) the parameters can be used to describe a maximum of 16 devices. If more channels are available and 'noauto' is not specified the additional channels are auto-sensed and used in ascending order.

CTC/ESCON with the channel device layer

Channel device layer configuration

The default for this driver is to select channels in order (automatic channel selection). If you need to use the channels in a different order, or do not want to use automatic channel selection, you can specify alternatives using the `ctc=` kernel parameter.

For the syntax of the `CTC/ESCON` channel device layer configuration see "Device identification (CTC/ESCON and LCS)" on page 71.

Configuration examples

For one network device (CTC):

```
ctc0,0x7c00,0x7c01,200,0,0,0
```

This tells the channel device layer to force `ctc0` (if detected) to use device addresses 7c00 and 7c01. 200 kilobytes are to be allocated for buffers. The usual protocol id (0) will be used, checksumming is not to be done on received `ip` packets and hardware statistics are not to be used. (For devices such as `ctc` which do not have hardware statistics this parameter is ignored.)

Or for two network devices (CTC + ESCON):

CTC/ESCON device driver

```
ctc0,0x601,0x600
escon3,0x605,0x608
```

This forces `ctc0` to use device addresses 601 and 600 and `escon3` to use 605 and 608. All other parameters are defaulted.

To scan a range of devices:

```
ctc,0x700,0x7ff,100
```

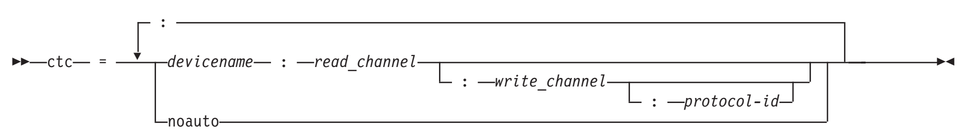
will scan the range 0x700 to 0x7ff for all `CTC` devices and allocate a buffer of 100 kilobytes for every device found. A device name will be generated for each device.

CTC/ESCON without the channel device layer

Kernel parameter syntax

The default for this driver is to select channels in order (automatic channel selection). If you need to use the channels in a different order, or do not want to use automatic channel selection, you can specify alternatives using the `ctc=` kernel parameter.

CTC kernel parameter



Note: The entire parameter is repeated (separated by spaces) for each `CTC/ESCON` device.

Where:

devicename
: is `ctc` or `escon` concatenated with the channel number, for example `ctc1` or `escon99`.

read_channel
: is the read channel address (in hexadecimal preceded by 0x).

write_channel
: is the write channel address (in hexadecimal preceded by 0x). If omitted the default is the read channel address plus 1.

protocol-id
: is the protocol number for CTC or ESCON. This can take the values:

- 0 for compatibility mode (the default; used with non-Linux peers other than OS/390 and z/OS)
- 1 for extended mode,
- 2 meaning ″CTC-based tty″ (this is only supported on Linux -Linux connections),
- 3 for compatibility mode with OS/390 and z/OS.

Using `noauto` as the device name disables automatic channel selection. If the only parameter given is `noauto` the CTC driver is disabled. This might be necessary, for example, if your installation uses 3271 devices or other such devices that use the CTC device type and model, but operate with a different protocol.

Kernel example

For one network device (CTC):

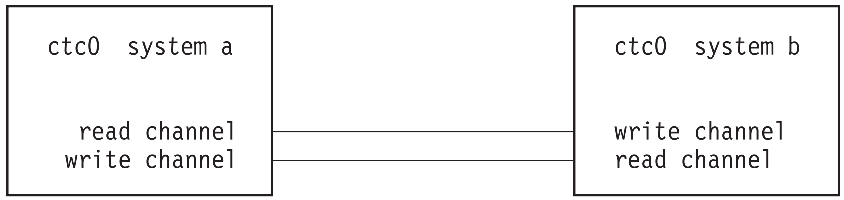


Figure 4. Connection of two systems via CTC (kernel)

```
ctc=ctc0:0x600
```

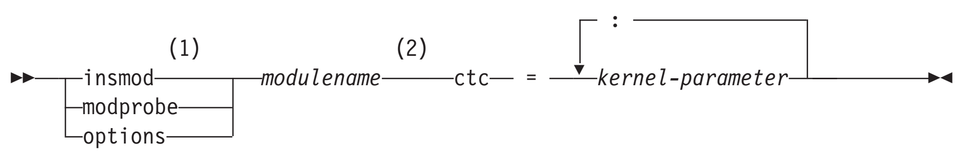
Or for two network devices (CTC + ESCON):

```
ctc=ctc0:0x601:0x600:escon3:0x605:0x608,
```

Module parameter syntax

These parameters can be passed to the CTC/ESCON driver module by `insmod`, or can be specified in the parameter file `"/etc/modules.conf"` or `"/etc/conf.modules"` (the file name depends on the Linux distribution).

CTC module options



Notes:

1 **insmod** or **modprobe** on the command line or **options** in the parameter file.

2 When using **insmod**, *modulename* includes the path to the module's object file (for example `/lib/modules/.../ctc.o`). When using **modprobe**, *modulename* is the module name without the path (`ctc`).

Where:

kernel-parameter
: is as defined above in "Kernel parameter syntax" on page 80

Note: If the parameter line file is used, the CTC driver may be loaded by typing `modprobe ctc` on the command line.

Module example

For one network device (CTC):

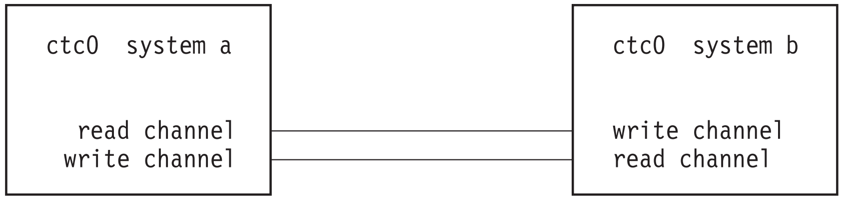


Figure 5. Connection of two systems via CTC (module)

Command line example:

```
insmod ctc ctc=ctc0:0x0600:0x0601
```

or

```
insmod /lib/modules/ctc.o ctc=ctc0:0x0600
```

Parameter file example:

```
options ctc ctc=ctc0:0x0600
```

Or for two network devices (CTC + ESCON):

Command line example:

```
insmod /lib/modules/ctc.o ctc=ctc0:0x0601:0x0600:escon3:0x0605:0x0608
```

or

Parameter file example:

```
options ctc ctc=ctc0:0x0601:0x0600:escon3:0x0605:0x0608
```

CTC/ESCON – Preparing the connection

1. Connection

   Prior to activation a channel connection is required. This can be a real or virtual connection :

   - Real Channels

     Connect the systems with a pair of channels to the remote system. Verify that the read channel of one is connected to the write channel of the other.
   - LPAR to LPAR Channels

     Select a pair of channels on each system. Verify that the read channel of one is connected to the write channel of the other and vice-versa.
   - VM Channels

     a. Obtain a subnet from your TCP/IP communications staff. It is important that the subnet used by your Linux guests is not the same as that used by VM/ESA on the LAN. The Linux system is a separate network and should be treated as such.

     b. Take one address from that subnet and assign it to VM.

     c. Define two virtual channels to your user ID. The channels may be defined in the VM User Directory using directory control `SPECIAL` statements, for example:

     ```
     special 0c04 ctca
     special 0c05 ctca
     ```

     or by using the CP commands:

```
define ctca as 0c04
define ctca as 0c05
```

from the console of the running CMS machine (preceded by #CP if necessary), or from an EXEC file (such as PROFILE EXEC A).

d. Add the necessary 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 VM ETC GATEWAYS file to include "permanent" host entries for each Linux guest.
e. Bring these updates online by using OBEYFILE or by recycling TCPIP and/or ROUTED as needed.

Connect the virtual channels to the channels of the VM TCP/IP target user ID. You must couple the Linux read channel to the VM TCP/IP write channel and vice versa. The coupling can be done with the following CP commands (following the previous example)

```
couple 0c04 to tcpip 0c05
couple 0c05 to tcpip 0c04
```

The VM TCP/IP channel numbers depend on the customization on the remote side. In this example, the CTC read channel 0c04 is connected to the VM TCP/IP write channel 0c05. Similarly, CTC write (0c05) is connected to VM TCP/IP read (0c04).

You can write the define and couple commands into the CMS PROFILE EXEC A script. The Linux for zSeries virtual machine must always be IPLed as CMS before IPLing as Linux in order for these commands to take effect.

Instead of connecting to the VM TCP/IP user ID, you can connect to any other virtual machine in which a Linux for zSeries, z/OS, OS/390, or VSE system is running.

2. Definitions on the remote side

   Set up the TCP/IP on the remote side, as described in the reference manuals. This will vary depending on which operating system is used on the remote side.

   Note: It is important that you have IOBUFFERSIZE 32678 defined because the Linux for zSeries CTC driver works with 32k internally. This is configurable for each device by writing the value to the buffersize file for that device (/proc/net/ctc/<devicename>/buffersize), for example

   ```
   echo 32768 > /proc/net/ctc/ctc0/buffersize
   ```

3. Activation on the remote side

   Activate the channels on the remote side. This again will vary depending on the operating system used on the remote side.

4. Activation on the Linux for zSeries side

   The network devices are activated with the ifconfig command. It is necessary to define the right MTU size for the channel device, otherwise it will not work properly. Please use the same MTU size (default 1500) that is defined on the remote side:

   The syntax of this command is:

CTC ifconfig command

```
►►──ifconfig──device_id──ip_address──pointopoint──to_address────────────►

   ┌─mtu 32760──────────────────┐
►──┴─mtu──max_transfer_unit─────┴──┬──────┬──────────────────────────►◄
                                   ├─up───┤
                                   └─down─┘
```

Where:

device_id
: identifies the device. (`ctc0` to `ctcn` or `escon0` to `esconn`)[9]

ip_address
: is the IP address of the local side.

to_address
: is the IP address of the remote side.

max_transfer_unit
: is the size of the largest IP packet which may be transmitted

up activates the interface

down deactivates the interface

An example of the use of `ifconfig` is:

```
ifconfig ctc0 10.0.51.3 pointopoint 10.0.50.1 mtu 32760
```

If you are using a CTC-based tty connection you must create a device node with major number 43 in the Linux `/dev` directory:

```
mknod /dev/ttyZ0 c 43 0
mknod /dev/ttyZ1 c 43 1
```

and so on

No network device setup is needed in this case. The CTC-based tty emulates a standard serial port including the usual handshake lines (RTS/CTS/DTR/DSR/CD). To establish a connection, simply open the previously created device (`/dev/ttyZx`) on both peers using a standard terminal emulator or activate a standard `getty` on it.

Notes

Device major number 43 is reserved on PC architecture for `/dev/isdn`. This number has been allocated to CTC/ESCON on Linux for zSeries because on zSeries there is no ISDN support. The connection is established when the tty device is opened. Following closure of the tty device, shutdown of the connection is delayed for about ten seconds. This delay has been implemented to avoid unnecessary initialization sequences if programs quickly open and

9. When using the channel device layer only, all `CTC` network devices are named `ctcn`, regardless of whether they are virtually defined or a real `ESCON`.

close the device . For this reason, if the driver is loaded as a module, it can only be unloaded after first closing all CTC-based ttys and then waiting for this delay to expire.

CTC/ESCON – Recovery procedure after a crash

In a native Linux for zSeries system if one side of a CTC connection crashes it is not possible to simply reconnect to the other side after a reboot. The correct procedure is:

1. Stop the CTC connection on the Linux for zSeries side using (for instance):
```
ifconfig escon0 down
```
2. Activate the channels on the remote side.
3. Activate the channels on the Linux for zSeries side, for example:
```
ifconfig escon0 10.0.0.1 pointopoint 10.0.50.1 mtu 32760
```

Chapter 11. Linux for zSeries IUCV device driver

The Inter-User Communication Vehicle (IUCV) is a VM/ESA communication facility that enables a program running in one virtual machine to communicate with another virtual machine, or with a control program, or even with itself. The communication takes place over a predefined linkage called a path.

The Linux for zSeries IUCV device driver is a network device, which uses IUCV to connect Linux kernels running on different VM user IDs, or to connect a Linux kernel to another VM guest such as a TCP/IP service machine.

IUCV features

The following features are supported:

- Multiple output paths from a Linux guest
- Multiple input paths to a Linux guest
- Simultaneous transmission and reception of multiple messages on the same or different paths
- Network connections via a TCP/IP service machine gateway

IUCV kernel parameter syntax

The driver must be loaded with the IDs of the guest machines you want to connect to:

IUCV kernel parameter

```
                  ┌─:──────┐
                  ▼        │
►►──iucv── = ───USERID─┴──────────────────────────────►◄
```

Parameter:

USERID
: Name of the target VM guest machine (in capital letters)

IUCV kernel parameter example

The following diagram shows the possible connection of two Linux for zSeries machines:

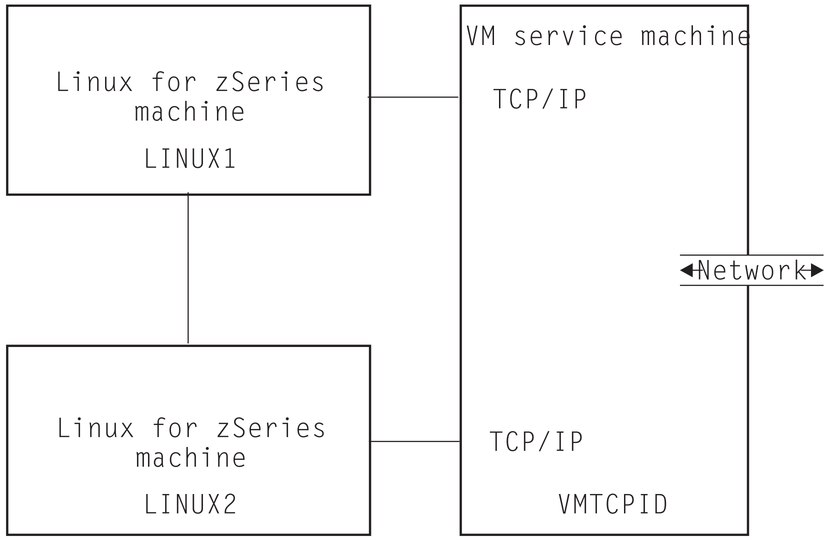


Figure 6. Connection of two systems using IUCV

The command

```
iucv=VMTCPID:LINUX2
```

connects the LINUX1 system to the TCP service machine and the other Linux system.

IUCV module parameter syntax

The driver must be loaded with the IDs of the guest machines you want to connect to:

IUCV module parameter

Parameter:

USERID
: Name of the target VM guest machine (in capital letters)

IUCV module parameter example

The example of "IUCV kernel parameter example" could be set up by starting the IUCV module with:

```
modprobe netiucv iucv=VMTCPID:LINUX2
```

IUCV – Preparing the connection

This is an additional task that you must perform before you can use the IUCV network link. If Linux is being used as a network hub instead of VM TCP/IP, the concepts discussed remain the same, though the syntax will be different.

The following steps must be undertaken in VM:

1. Obtain a subnet from your TCP/IP communications staff. It is important that the subnet used by your Linux guests not be the same as that used by VM on the LAN. It is a separate network and should be treated as such.
2. Take one address from that subnet and assign it to VM. Update your `PROFILE TCPIP` file with a home entry, device, link, and start statements for each guest, for example:

```
Home
 vm_ip_address link_name1
 vm_ip_address link_name2

Device device_name1 IUCV 0 0 linux_virtual_machine1 A
Link link_name1 IUCV 0  device_name1

Device device_name2 IUCV 0 0 linux_virtual_machine2 A
Link link_name2 IUCV 0  device_name2

Start device_name1
Start device_name2
```

3. Add the necessary 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 VM `ETC GATEWAYS` file to include ″permanent″ host entries for each Linux guest.
4. Bring these updates online by using `OBEYFILE` or by recycling `TCPIP` and/or `ROUTED` as needed.
5. Add the statement

```
IUCV ALLOW
IUCV ANY
```

to your VM user directory entry.

The Linux commands needed to start communications through a TCP/IP service machine are:

TCP/IP ifconfig command

```
                                  (1)
>>──ifconfig──iucv──iucv_number──────────────────────────────────────────>

>──your_address──pointopoint──service_address──┬─────────────┬──────────><
                                               └─┤ options ├─┘
```

options:

```
   ┌─mtu 9216─┐
├──┴─mtu──n───┴──netmask──mask_value─────────────────────────────────────┤
```

Notes:

1 *iucv_number* is concatenated to the interfacce name, for example `iucv0`.

Parameters:

iucv_number
: Path number (for example 0)

your_address
: TCP/IP address of your machine

pointopoint
: required to establish a point-to-point connection to a service machine

service_address
: Address of the TCP/IP service machine to connect to

n
: maximum transfer unit size. The default is 9216, which is suitable for use with the zSeries Virtual Image Facility for Linux (VIF). The maximum value is 32764.

mask_value
: Mask to identify addresses served by this connection

route command

```
>>──route──add── -net──default──iucv──iucv_number────────────────────────><
```

Parameters:

iucv_number
: Path number defined above

inetd command

```
         (1)
►►──inetd──────────────────────────────────────────────────────►◄
```

Notes:

1 Not required if the IUCV driver is started during boot.

The commands needed to start direct communications to another guest are:

user-to-user ifconfig command

```
►►──ifconfig──iucv──iucv_number────────────────────────────────►

►──guest_0_address──pointopoint──guest_1_address───────────────►◄
```

Parameters:

iucv_number
: Path number (for example 0)

guest_0_address
: TCP/IP address of your machine

guest_1_address
: TCP/IP address of target machine

IUCV – Further information

The standard definitions in the VM TCP/IP configuration files apply.

For more information of the VM TCP/IP configuration see: *VM/ESA TCP/IP Planning and Customization* , SC24-5847-01.

IUCV restrictions

- This device driver is only available to Linux for zSeries systems running as guests under VM/ESA or z/VM.

IUCV Application Programming Interface (API)

Linux IUCV is a full duplex, event driven facility which transfers whole records at a time. To exploit any of the IUCV functions one must first register with IUCV using the function `iucv_register_program()`. For more information on all IUCV functionality refer to the CP Programming Services book, available on the Web as manual number SC24-5760 at `http://www.ibm.com/s390/vm/pubs` .

IUCV API

In this description of the API parameters which are pointers do not necessarily require a value. A 'NULL' pointer will be ignored by the functions. If a parameter is not a pointer a value must be provided. All addresses passed to IUCV must be real addresses in the VM guest machine.

iucv_register_program

Purpose: To register an application with IUCV.

Note: pgmmask

- When pgmname, userid and pgmmask are provided the mask is used as is.
- When pgmname and userid are provided and pgmmask is not provided the default mask is all 0xff
- When pgmname and pgmmask are provided and userid is not provided the first 8 bytes of the mask are 0x00 and the last 16 bytes are copied from the last 16 bytes of pgmmask.
- When pgmname is provided and userid and pgmmask are not provided the first 8 bytes of the mask are 0x00 and the last 16 bytes are 0xff.

API Descriptor:

Name	Type	Input/Output	Description
pgmname	uchar [16]	input	User identification
userid	uchar[8]	input	Machine Identification
pgmmask	uchar[24]	input	Indicates which bits in the userid and pgmname combined will be used to determine who is given control
ops	iucv_interrupt_ops_t	input	Address of vector of interrupt handlers
pgm_data	* void	input	Application data passed to interrupt handlers. (token)

Return value: type iucv_handle_t

This is a token used as input value for iucv_connect, iucv_accept and iucv_unregister

A value of zero (0) indicates that an error occurred in registration. Check syslog for details. The reasons for the error could be:

- Machine size is greater than 2 GB
- new_handler kmalloc failed
- pgmname was not provided
- ops is not defined
- pathid_table kmalloc failed
- An application with this pgmname, userid and pgmmask is already registered
- iucv_declare_buffer failed

iucv_unregister_program

Purpose: Unregister application with IUCV

API Descriptor:

Name	Type	Input/Output	Description
handle	iucv_handle_t	input	Token which was returned during registration to identify application to be unregistered

Return value: type int

This should be zero (0) to indicate a normal return.

iucv_accept

Purpose: After the user has received a Connection Pending external interrupt this function is issued to complete the IUCV communication path

API Descriptor:

Name	Type	Input/Output	Description
pathid	u16	input	path identification number
msglim_reqstd	u16	input	the number of outstanding messages requested
user_data	uchar[16]	input	16-bytes of user data
flags1	int	input	Contains options for the path:
IPPRTY	0X20	input	specifies that you want to send a priority message
IPRMDATA	0X80	input	specifies that your program can handle a message in the parameter list
IPQUSCE	0X40	input	specifies that you want to quiesce the path being established
handle	iucv_handle_t	input	address of token
pgm_data	* void	input	application data passed to interrupt handlers
flags1_out	* int	output	0x20 byte ON, indicates you may send a priority message
msglim	* u16	out○put	number of outstanding messages

Return value: type int

A zero (0) or positive value is the return code from CP IUCV.

A return code of -EINVAL means the handle given was NULL.

iucv_connect

Purpose: This function establishes an IUCV path. Although the connect may have completed successfully you are not able to use the path until you receive an IUCV ″Connection Complete″ external interrupt.

API Descriptor:

Name	Type	Input/Output	Description
pathid	u16	output	path identification number
msglim_reqstd	u16	input	the number of outstanding messages requested
user_data	uchar[16]	input	16-bytes of user data
userid	uchar[8]	input	User identification
system_name	uchar[8]	input	8-bytes identifying the system
flags1	int	input	Contains options for the path:
IPPRTY	0X20	input	specifies that you want to send a priority message
IPRMDATA	0X80	input	specifies that your program can handle a message in the parameter list
IPQUSCE	0X40	input	specifies that you want to quiesce the path being established
IPLOCAL	0X01	input	allows an application to force the partner to be on the local system. If IPLOCAL is specified then target class cannot be specified.
flags1_out	* int	output	0x20 byte ON indicates you may send a priority message
msglim	* u16	out○put	number of outstanding messages
handle	iucv_handle_t	input	address of handler

pgm_data	void *	input	application data passed to interrupt handlers

Return value: type int

A zero (0) or positive value is the return code from CP IUCV or the internal function add_pathid.

A return code of -EINVAL means an invalid handle passed by application or the pathid address is NULL.

iucv_purge

Purpose: This function cancels a message that you have sent

API Descriptor:

Name	Type	Input/Output	Description
pathid	u16	input	path identification number
msgid	u32	input	specifies the ID of the message to be purged. If msgid is specified then pathid and srccls must also be specified.
srccls	u32	input	specifies the source message class
audit	uchar[3]	output	contains information about any asynchronous error that may have affected the normal completion of this message.

Return value: type int

This is the return code from CP IUCV.

iucv_query_bufsize

Purpose: This function determines how large an external interrupt buffer IUCV will require to store information.

API Descriptor: Void

Return value: type ulong

This is the required size of the external interrupt buffer.

iucv_query_maxconn

Purpose: This function determines the maximum number of connections that may be established by the virtual machine.

API Descriptor: Void

Return value: type ulong

Maximum number of connections.

iucv_quiesce

Purpose: This function temporarily suspends incoming messages on an IUCV path. You can later reactivate the path by invoking the iucv_resume function.

API Descriptor:

Name	Type	Input/Output	Description
pathid	u16	input	path identification number
user_data	uchar[16]	input	16-bytes of user data

Return value: type int

This is the return code from CP IUCV.

iucv_receive

Purpose: This function receives messages that are being sent to you over established paths. To receive data buflen must be 8-bytes or greater.

API Descriptor:

Name	Type	Input/Output	Description
pathid	u16	input	path identification number
msgid	u32	input	specifies the message ID. If msgid is specified then pathid and srccls must also be specified
trgcls	u32	input	specifies target class
buffer	* void	input	address of buffer to receive
buflen	ulong	input	length of buffer to receive
flags1_out	* int	output	Contains information about the path:
IPNORPY	0x10		specifies that a reply is required
IPPRTY	0x20		specifies that you want to send a priority message

IPRMDATA	0x80		specifies the data is contained in the parameter list
residual_buffer	* void	output	address of buffer updated by the number of bytes you have received
residual_length	* ulong	*output	Contains one of the following values depending on whether the receive buffer is: • The same length as the message – this field is zero. • Longer than the message – this field contains the number of bytes remaining in the buffer. • Shorter than the message, this field contains the residual count (that is, the number of bytes remaining in the message that does not fit into the buffer. In this case return code is 5.

Return value: type int

A zero (0) or positive value is the return code from CP IUCV.

A return code of -EINVAL means the buffer address is pointing to NULL.

iucv_receive_array

Purpose: This function receives messages that are being sent to you over established paths. To receive data the first entry in the array must be 8-bytes or greater.

API Descriptor:

Name	Type	Input/Output	Description
pathid	u16	input	path identification number
msgid	u32	input	specifies the message ID. If msgid is specified then pathid and srccls must also be specified
trgcls	u32	input	specifies target class

buffer	* iucv_array_t	input	address of array of buffers
buflen	ulong	input	total length of buffers
flags1_out	* int	output	Contains information about the path:
IPNORPY	0x10		specifies that a reply is required
IPPRTY	0x20		specifies that you want to send a priority message
IPRMDATA	0x80		specifies the data is contained in the parameter list
residual_buffer	* void	output	address points to the current list entry IUCV is working on
residual_length	* ulong	*output	Contains one of the following values depending on whether the receive buffer is: • The same length as the message – this field is zero. • Longer than the message – this field contains the number of bytes remaining in the buffer. • Shorter than the message, this field contains the residual count (that is, the number of bytes remaining in the message that does not fit into the buffer. In this case return code is 5.

Return value: type int

A zero (0) or positive value is the return code from CP IUCV.

A return code of -EINVAL means the buffer address is pointing to NULL.

iucv_reply

Purpose: This function is used to respond to a two-way message that you have received. You must specify completely the message to which you wish to reply (pathid, msgid and trgcls). The msgid and trgcls are the values returned by the previous IUCV receive.

API Descriptor:

Name	Type	Input/Output	Description
pathid	u16	input	path identification number
msgid	u32	input	specifies the message ID.
trgcls	u32	input	specifies the target class
flags1	int	input	Contains options for the path:
IPPRTY	0x20		specifies that you want to send a priority message
buffer	* void	input	address of reply buffer
buflen	* ulong	input	length of reply buffer
residual_buffer	* ulong	output	Address of buffer updated by the number of bytes you have moved
residual_length	* ulong	output	Contains one of the following values depending on whether the receive buffer is: • The same length as the message – this field is zero. • Longer than the reply – this field contains the number of bytes remaining in the buffer. • Shorter than the message, this field contains the residual count (that is, the number of bytes remaining in the reply which do not fit into the buffer. In this case b2f0_result = 5.

Return value: type `int`

A zero (0) or positive value is the return code from `CP IUCV`.

A return code of `-EINVAL` means the buffer address is pointing to `NULL`.

iucv_reply_array

Purpose: This function is used to respond to a two-way message array that you have received. You must specify completely the array to which you wish to reply (`pathid`, `msgid` and `trgcls`). The `msgid` and `trgcls` are the values returned by the previous `iucv_receive_array`.

The array contains a list of non-contiguous buffer addresses and their lengths. These buffers contain the reply data.

API Descriptor:

Name	Type	Input/Output	Description
pathid	u16	input	path identification number
msgid	u32	input	specifies the message ID.
trgcls	u32	input	specifies the target class
flags	int	input	Contains options for the path:
IPPRTY	0x20		specifies that you want to send a priority message
buffer	* iucv_array_t	input	address of array of reply buffers
buflen	ulong	input	total length of reply buffers
residual_address	* ulong	output	Address of buffer which IUCV is currently working on

residual_length	* ulong	output	Contains one of the following values depending on whether the answer buffer is: • The same length as the reply – this field is zero. • Longer than the reply – this field contains the number of bytes remaining in the buffer. • Shorter than the message, this field contains the residual count (that is, the number of bytes remaining in the reply which do not fit into the buffer. In this case b2f0_result = 5.

Return value: type `int`

A zero (0) or positive value is the return code from `CP IUCV`.

A return code of `-EINVAL` means the buffer address is pointing to `NULL`.

iucv_reply_prmmsg

Purpose: This function is used to respond to a two-way message that you have received. You must specify completely the message to which you wish to reply (`pathid`, `msgid` and `trgcls`).`prmmsg` signifies the data has been moved into the parameter list.

API Descriptor:

Name	Type	Input/Output	Description
pathid	u16	input	path identification number
msgid	u32	input	specifies the message ID.
trgcls	u32	input	specifies the target class
flags1	int	input	Contains options for the path:
IPPRTY	0x20		specifies that you want to send a priority message

prmmsg	uchar[8]	input	8-bytes of data to be placed into the parameter list.

Return value: type int

This is the return code from CP IUCV.

iucv_resume

Purpose: This function restores communications over a quiesced path.

API Descriptor:

Name	Type	Input/Output	Description
pathid	u16	input	path identification number
user_data	uchar[16]	input	16-bytes of user data

Return value: type int

This is the return code from CP IUCV.

iucv_send

Purpose: This function transmits data from a buffer to another application. This is a one-way process – the receiver will not reply to the message.

API Descriptor:

Name	Type	Input/Output	Description
pathid	u16	input	path identification number
msgid	* u32	output	specifies the message ID.
trgcls	u32	input	specifies the target class
srccls	u32	input	specifies the source message class
msgtag	u32	input	specifies a tag to be associated with the message
flags1	int	input	Contains options for the path:
IPPRTY	0x20		specifies that you want to send a priority message
buffer	* void	input	address of send buffer
buflen	ulong	input	length of send buffer

Return value: type int

A zero (0) or positive value is the return code from `CP IUCV`.

A return code of `-EINVAL` means the buffer address is `NULL`.

iucv_send_array

Purpose: This function transmits data to another application. The array holds the addresses and lengths of discontiguous buffers which hold the message text. This is a one-way process – the receiver will not reply to the message.

API Descriptor:

Name	Type	Input/Output	Description
`pathid`	`u16`	input	path identification number
`msgid`	`* u32`	output	specifies the message ID.
`trgcls`	`u32`	input	specifies the target class
`srccls`	`u32`	input	specifies the source message class
`msgtag`	`u32`	input	specifies a tag to be associated with the message
`flags1`	`int`	input	Contains options for the path:
`IPPRTY`	`0x20`		specifies that you want to send a priority message
`buffer`	`* iucv_array_t`	input	address of array of send buffers
`buflen`	`ulong`	input	total length of send buffers

Return value: type `int`

A zero (0) or positive value is the return code from `CP IUCV`.

A return code of `-EINVAL` means the buffer address is `NULL`.

iucv_send_prmmsg

Purpose: This function transmits data to another application. `prmmsg` signifies the 8 bytes of data are to be moved into the parameter list. This is a one-way message – the receiver will not reply to this message.

API Descriptor:

Name	Type	Input/Output	Description
`pathid`	`u16`	input	path identification number
`msgid`	`* u32`	output	specifies the message ID.

trgcls	u32	input	specifies the target class
srccls	u32	input	specifies the source message class
msgtag	u32	input	specifies a tag to be associated with the message
flags1	int	input	Contains options for the path:
IPPRTY	0x20		specifies that you want to send a priority message
prmmsg	uchar[8]	input	8-bytes of data to be placed into the parameter list.

Return value: type int

This is the return code from CP IUCV.

iucv_send2way

Purpose: This function transmits data to another application. The data to be transmitted is in a buffer. The receiver of the message is expected to reply, and a buffer is provided into which IUCV will move the reply.

API Descriptor:

Name	Type	Input/Output	Description
pathid	u16	input	path identification number
msgid	* u32	output	specifies the message ID.
trgcls	u32	input	specifies the target class
srccls	u32	input	specifies the source message class
msgtag	u32	input	specifies a tag to be associated with the message
flags1	int	input	Contains options for the path:
IPPRTY	0x20		specifies that you want to send a priority message
buffer	* void	input	address of send buffer
buflen	ulong	input	length of send buffer
ansbuf	* void	input	address of reply buffer
anslen	ulong	input	length of reply buffer

Return value: type int

A zero (0) or positive value is the return code from CP IUCV.

A return code of -EINVAL means the buffer address is NULL.

iucv_send2way_array

Purpose: This function transmits data to another application. The array holds the addresses and lengths of discontiguous buffers which hold the message text. The receiver of the message is expected to reply, and a buffer is provided into which IUCV will move the reply.

API Descriptor:

Name	Type	Input/Output	Description
pathid	u16	input	path identification number
msgid	* u32	output	specifies the message ID.
trgcls	u32	input	specifies the target class
srccls	u32	input	specifies the source message class
msgtag	u32	input	specifies a tag to be associated with the message
flags1	int	input	Contains options for the path:
IPPRTY	0x20		specifies that you want to send a priority message
buffer	* iucv_array_t	input	address of array of send buffers
buflen	ulong	input	total length of send buffer
ansbuf	* iucv_array_t	input	address of array of reply buffers
anslen	ulong	input	total length of reply buffers

Return value: type int

A zero (0) or positive value is the return code from CP IUCV.

A return code of -EINVAL means the buffer address is NULL.

iucv_send2way_prmmsg

Purpose: This function transmits data to another application. prmmsg specifies that the 8-bytes of data are to be moved into the parameter list. The receiver of the message is expected to reply, and a buffer is provided into which IUCV will move the reply.

API Descriptor:

Name	Type	Input/Output	Description
pathid	u16	input	path identification number
msgid	* u32	output	specifies the message ID.
trgcls	u32	input	specifies the target class
srccls	u32	input	specifies the source message class
msgtag	u32	input	specifies a tag to be associated with the message
flags1	int	input	Contains options for the path:
IPPRTY	0x20		specifies that you want to send a priority message
prmmsg	uchar[8]	input	8-bytes of data to be placed into parameter list
ansbuf	* void	input	address of reply buffer
anslen	ulong	input	length of reply buffer

Return value: type int

A zero (0) or positive value is the return code from CP IUCV.

A return code of -EINVAL means the buffer address is NULL.

iucv_send2way_prmmsg_array

Purpose: This function transmits data to another application. prmmsg specifies that the 8-bytes of data are to be moved into the parameter list. The receiver of the message is expected to reply, and an array of addresses and lengths of discontiguous buffers is provided into which IUCV will move the reply.

API Descriptor:

Name	Type	Input/Output	Description
pathid	u16	input	path identification number
msgid	* u32	output	specifies the message ID.
trgcls	u32	input	specifies the target class
srccls	u32	input	specifies the source message class

msgtag	u32	input	specifies a tag to be associated with the message
flags1	int	input	Contains options for the path:
IPPRTY	0x20		specifies that you want to send a priority message
prmmsg	uchar[8]	input	8-bytes of data to be placed into parameter list
ansbuf	* iucv_array_t	input	address of array of reply buffers
anslen	ulong	input	total length of reply buffers

Return value: type int

A zero (0) or positive value is the return code from CP IUCV.

A return code of -EINVAL means the buffer address is NULL.

iucv_setmask

Purpose: This function enables or disables message interrupts and reply interrupts (priority or non-priority). A zero value disables the interrupts; any non-zero value enables them.

API Descriptor:

Name	Type	Input/Output	Description
SetMaskFlag	int	input	Flag for setting interrupt types.
Nonpriority_ MessagePending InterruptsFlag	0x40		
Priority_ MessagePending InterruptsFlag	0x80		
Nonpriority_ MessageCompletion InterruptsFlag	0x20		
Priority_ MessageCompletion InterruptsFlag	0x10		

Return value: type int

This is the return code from CP IUCV.

iucv_sever

Purpose: This function terminates an IUCV path.

API Descriptor:

Name	Type	Input/Output	Description
pathid	u16	input	path identification number
user_data	uchar[16]	input	16-bytes of user data

Return value: type int

This is the return code from CP IUCV.

IUCV Interrupt Handler API

The following are declarations of IUCV interrupts. To ignore an interrupt, declare it as NULL.

Input parameters:

eib pointer to an external interrupt buffer

pgm_data
pointer to token that was passed by the application.

Output and Return: void

```
typedef struct {
        void (*ConnectionPending) (iucv_ConnectionPending * eib,
                                   void* pgm_data);
        void (*ConnectionComplete) (iucv_ConnectionComplete * eib,
                                    void* pgm_data);
        void (*ConnectionSevered) (iucv_ConnectionSevered * eib,
                                   void* pgm_data);
        void (*ConnectionQuiesced) (iucv_ConnectionQuiesced * eib,
                                    void* pgm_data);
        void (*ConnectionResumed) (iucv_ConnectionResumed * eib,
                                   void* pgm_data);
        void (*MessagePending) (iucv_MessagePending * eib,
                                void* pgm_data);
        void (*MessageComplete) (iucv_MessageComplete * eib,
                                 void* pgm_data);
} iucv_interrupt_ops_t;
```

Chapter 12. Linux for zSeries LCS device driver

> **Note**
> The LCS driver is now provided in source-code form.

This Linux network driver supports OSA–2 Ethernet Token Ring, and OSA-Express Fast Ethernet in non-QDIO mode.

To configure this device driver, use the channel device layer. For details on how to use the channel device layer, see Chapter 9, "Linux for zSeries Channel device layer" on page 69.

The LAN Channel Station (LCS) network interface has two channels, one read channel and one write channel. This is very similar to the zSeries `CTC` interface (see Chapter 10, "Linux for zSeries CTC/ESCON device driver" on page 79). The read channel must have model type 0x3088 and an even `cua` number. The write channel also has a model type of 0x3088 and has a `cua` number one greater than the read `cua` number. Only certain `cua` types are supported so as not to clash with a `CTC` control unit type.

The driver always has a read outstanding on the read subchannel. This is used to receive command replies and network packets (these are differentiated by checking the type field in the `LCS` header structure). Any network packets that arrive during the startup and shutdown sequence have to be discarded. During normal network I/O, the driver will intermittently retry reads in order to permanently keep a read outstanding on the read channel. (This is in case an `-EBUSY` or the like occurs, in which case the driver would stop receiving network packets.)

The default configuration is to use software statistics, with IP checksumming off (this improves performance) and to have network hardware checking using a CRC32 check which should guarantee integrity for normal use. However, financial institutions or the like might want the additional security of IP checksumming.

Additional `cua` model types can be added later so that new `LCS` compatible cards will be supported even if not available when the driver was developed.

LCS supported functions

- Supports Ethernet and Token Ring
- Auto detects whether card is connected to Token Ring or Ethernet

LCS channel device layer configuration

For the syntax of `LCS` configuration with the channel device layer see "Device identification (CTC/ESCON and LCS)" on page 71.

LCS channel device layer configuration example

```
chandev=noauto,lcs0,0x7c00,0x7c01,1,1,1
```

This will make LCS device 0 use addresses 7c00 and 7c01 for input and output, use relative adapter 1, set IP checksumming on and collect network statistics from the hardware.

LCS kernel parameter syntax

If the channel device layer is not used and the LCS driver is compiled directly into the kernel, the LCS boot parameters are as follows:

lcs kernel parameter

```
>>--lcs-- = --hw_stats--,--ip_checksumming------------------------->

>--+-------------------------------+------------------------------><
   |   .-,---------------------.   |
   |   V                       |   |
   '---model_no,max_adapter_no-+---'
```

lcs devno kernel parameter

```
                  .-,---------------------.
                  V                       |
>>--lcs_devno-- = ---model_no,rel_adapter_no-+--------------------><
```

lcs noauto kernel parameter

```
>>--lcs_noauto----------------------------------------------------><
```

The meanings of these parameters are:

hw_stats
: If set to 1 it is equivalent to the **use_hw_stats** keyword for the module call. If set to 0 it is ignored.

ip_checksumming
: If set to 1 it is equivalent to the **do_sw_ip_checksumming** keyword for the module call. If set to 0 it is ignored.

model_no,max_adapter_no
: is identical to the **additional_model_info** keyword described for modules.

model_no,rel_adapter_no
: is identical to the **devno_portno_pairs** keyword described for modules. The keyword **lcs_devno** is short because of limitations in the allowable length of kernel parameter names.

lcs_noauto
: Put this parameter in the kernel parameter line if you want to set auto-detection off.

It is important that the parameters are entered in pairs (2, 4, 6 or 8 parameters) as the `cu model` and `max rel adapter no` must go together.

LCS warning

Under some circumstances, LCS initialization can generate a message such as:

```
"failed to add multicast address"
```

This message is for information purposes only and can be ignored. The driver and card continue to operate normally.

LCS restrictions

- To use OSA devices when running Linux for zSeries on a basic mode machine (no LPARs) you may need to specify an `ipldelay=`*`xyz`* boot parameter. We recommend a value between `2m` and `5m` for *`xyz`* for the OSA card to initialize fully after IPL.

Chapter 13. QETH device driver for OSA-Express (QDIO) and HiperSockets

The QETH driver is subject to license conditions as reflected in: "International License Agreement for Non-Warranted Programs" on page 233. The QDIO driver is now supplied in source-code form and is not subject to these license provisions.

The QETH Linux network driver supports HiperSockets virtual devices as well as the OSA-Express Fast Ethernet, Gigabit Ethernet, High Speed Token Ring, and ATM (running Ethernet LAN emulation) features in QDIO mode. HiperSockets enable the zSeries to connect to virtual networks on a shared zSeries.

A HiperSockets device is controlled by the same device driver as the OSA-Express feature in QDIO mode. Most of the device driver parameters are common to the two devices.

The OSA-Express feature in QDIO mode is described in detail in *OSA-Express Customer's Guide and Reference*, SA22-7476.

This driver has been developed for the zSeries 64-bit architecture and S/390 31-bit architecture with version 2.4 of the Linux kernel.

Naming conventions

Different features used will generate different interface base names:
- Ethernet features will generate an interface name starting with "eth"
- HiperSockets devices will generate an interface name starting with "hsi"
- Token Ring features will generate an interface name starting with "tr"

The 'eth' interface name is used for the OSA-Express ATM feature when emulating Ethernet in QDIO mode.

Numbers will be appended to the base names according to the following rules:
- If a device interface number, `devif_num`, is specified during device configuration, that number will be used. For example, a device configuration like:

  ```
  qeth7, <read_devno>, <write_devno>, <data_devno>
  ```

  will cause the interface name to be `eth7` for an Ethernet feature, `hsi7` for HiperSockets, and `tr7` for a Token Ring feature.

  If the `devif` number is `-1`, the next available number will be used. See "Configuring QETH for OSA-Express and HiperSockets using the channel device layer" on page 115 for the description of `devif_num`.
- If chandev is instructed to use device number names (`use_devno_names`), the interface number will be the cuu number of the read channel. For example, if the read channel has the cuu `0xfd0c`, the interface name would be `eth0xfd0c` for an Ethernet feature, `hsi0xfd0c` for a HiperSockets device, and `tr0xfd0c` for a Token Ring feature. See "Commonly used options" on page 73 for the description of `use_devno_names`.

Introduction

You need two modules to configure HiperSockets as well as the OSA-Express feature in QDIO mode:

- The QDIO protocol governs the interface between the zSeries and the OSA-Express feature. You need only load the QDIO module, or build it into the kernel; no configuration is necessary.
- The QETH module controls the feature itself. Configuring the QETH module is described in this chapter.

For HiperSockets and the OSA-Express feature in QDIO mode three I/O subchannels must be available to the driver. One subchannel is for control reads, one for control writes, and the third is for data.

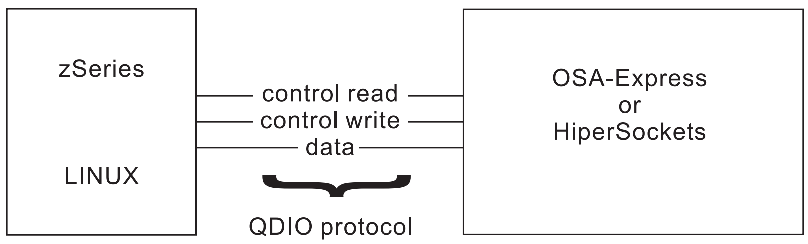


Figure 7. I/O subchannel interface

Installing QETH

To install the QETH (and possibly QDIO) module, follow these steps:

1. Check if the QDIO OSA or HiperSockets devices are online in Linux (check if they appear in /proc/subchannels). If not, attach them to the Linux guest or bring them online to the Linux LPAR.
2. Make sure the devices are correctly defined in the chandev.conf file. If they are not, define them and then reset chandev by issuing:
   ```
   echo reset_conf;read_conf > /proc/chandev
   ```
3. If the QDIO driver is to be loaded as a module, issue the **insmod** command:
   ```
   insmod qdio
   ```
4. Issue the **insmod** command for the qeth.o module:
   ```
   insmod qeth
   ```
5. Issue the **ifup** command or the **ifconfig** command with the desired parameters for any QDIO OSA or HiperSockets interfaces.

Now the interfaces to the devices are set up.

Note: See “QETH restrictions” on page 126 for information on possible failures when setting an IP address on an interface handled by QETH.

QETH supported functions

The following functions are supported:

- Auto-detection of devices
- Primary and secondary routers
- Priority queueing (does not apply to HiperSockets)

- Individual device configuration. It is possible to configure different triples of channels on the same CHPID differently. For example, if you have CHPID fc, then you can configure 0xfc00,0xfc01,0xfc02 differently from 0xfc04,0xfc05,0xfc06, for example, with different `mem_usage_in_k` values.
- IP address takeover. See "IP address takeover" on page 123.
- Virtual IP addresses (VIPA) , including Source VIPA (does not apply to HiperSockets). See Chapter 15, "VIPA – minimize outage due to adapter failure" on page 183.
- HiperSockets. See "Examples: HiperSockets" on page 121 and "Example: HiperSockets" on page 124.
- Broadcast. See "Broadcast support" on page 128.
- VLAN. See Chapter 17, "Virtual LAN (VLAN) support" on page 191.
- Query and purge of ARP data. See "qetharp - Query and purge OSA and HiperSockets ARP data" on page 156.
- Internet Protocol Version 6 (IPv6) (Gigabit Ethernet and Fast Ethernet only). See "Support for IP Version 6 (IPv6)" on page 127.
- SNMP (via the OSA-Express feature). See "osasnmpd – Start OSA-Express SNMP subagent" on page 151.

Configuring QETH for OSA-Express and HiperSockets using the channel device layer

This section describes how to configure QETH for the OSA-Express feature in QDIO mode and HiperSockets with the channel device layer. Only the most common options are given here to illustrate the syntax; see Chapter 9, "Linux for zSeries Channel device layer" on page 69 for full details of all channel device options.

The driver will normally use auto-detection to find all QDIO OSA-Express features and HiperSockets devices in the system. (The `noauto` option can be used to exclude address ranges from auto-detection.) In some circumstances it may be necessary to configure the driver explicitly for a device. This is done with the **qeth** command.

OSA-Express and HiperSockets device identification

```
                     (1)
>>--qeth----devif_num------,--read_devno--,--write_devno--,--data_devno------------->
          '- -1 ------'

>-+----------------------------------------------------------------------------+-><
  '-,--+-(device dependent)-+--+-------------------------------------------+---'
       '-memory_usage_in_k--'  '-,--+-0-------+--+------------------------+-'
                                    '-port_no-'  '-,--+-0---------------------+-'
                                                      '-checksum_received_ip_pkts-'
```

Notes:

1 *devif_num* is concatenated to the driver name, for example `qeth0` or `qeth1`.

Note: All characters must be entered in lower case as shown, except for hexadecimal numbers, where either case may be used.

Where:

devif_num
: is the device interface number. This is concatenated with `qeth`, for example `qeth1`.

  A value of -1 indicates that the next available number is to be allocated automatically.

read_devno
: is the read channel address (in hexadecimal preceded by 0x)

  This address must be an even number.

write_devno
: is the write channel address (in hexadecimal preceded by 0x)

  This address must be one greater than the read channel address.

data_devno
: is the data channel address (in hexadecimal preceded by 0x)

memory_usage_in_k
: is the number of kilobytes to be allocated for read and write buffers. (The allocation between read and write is determined by the driver.)

  Note: The default value is device dependent. For an OSA device (in QDIO mode), it is 8192 KB. For HiperSockets, it varies between 2048 KB and 8192 KB, depending on the OS value specified in the IOCDS. The value of this parameter (default or explicit) limits the maximum MTU size (“ifconfig - Configure a network interface” on page 167). For more information, see “QETH restrictions” on page 126.

port_no
: is the relative port number of the device. The OSA-Express feature in QDIO mode and HiperSockets use only port 0, the default port. The OSA-Express ATM feature in QDIO mode set up for Ethernet LAN emulation allows configuration of two emulated ports through OSA/SF.

checksum_received_ip_pkts
: is 1 to perform software checksumming or 0 to suppress it.

Port identification

Each OSA-Express feature in QDIO mode must be associated with a port name. To do this, use the channel device layer **add_parms** command as shown below. (HiperSockets devices do not require a port name.)

OSA-Express feature in QDIO mode port identification

```
                       ┌─all──────────────────────┐
►►─add_parms─,─0x10─┤                          ├─,─portname─:─PORT_NAME─┬──────────────┬──────►◄
                       └─,─lo_devno─,─hi_devno─┘                           └─,─qeth_parms─┘
```

Where:

add_parms
: Used to pass additional parameters to the driver.

0x10
: Identifies the device as an OSA-Express feature in QDIO mode.

lo_devno,hi_devno
: Specifies the address range containing the device.

port_name
: Required. Identifies the port for sharing by other OS images. *port_name* can be 1 to 8 characters long and must be uppercase. All operating systems sharing the port must use the same *port_name*. Example: NETWORK1.

qeth_parms
: are additional QETH parameters that can be added by using the chandev **add_parms** command as described in "QETH parameter syntax".

QETH parameter syntax

QETH parameters

```
        (1)   ┌─no_router─────────┐  ┌─,no_checksumming──────┐
►►────────────┼───────────────────┼──┼───────────────────────┼──────────────►
              ├─primary_router────┤  └─,──sw_checksumming────┘
              ├─secondary_router──┤
              └─multicast_router──┘

   ┌─,no_prio_queueing───────────────────┐
►──┼─────────────────────────────────────┼──────────────────────────────────►
   └─┬─,──prio_queueing_tos─────────────┬┘
     ├─,──prio_queueing_prec────────────┤
     └─,──no_prio_queueing── : ──number─┘

                                              ┌─,dont_fake_broadcast──┐
►──┬───────────────────────────────────────┬──┼───────────────────────┼─────►
   └─,──polltime── : ──┬─500───────┬───────┘  └─,──fake_broadcast─────┘
                       └─poll_time─┘

   ┌─,broadcast_allrings──┐  ┌─,macaddr_noncanon──┐
►──┼──────────────────────┼──┼────────────────────┼─────────────────────────►
   └─,──broadcast_local───┘  └─,──macaddr_canon───┘

   ┌─,enable_takeover───┐
►──┼────────────────────┼───────────────────────────────────────────────────►
   └─,──disable_takeover┘

►──┬────────────────────────────────────────────────────┬─────────────────►◄
   └─,──add_hhlen── : ──hardware_header_length──────────┘
```

Notes:

1 All options except the first used must be preceded by a comma. All characters must be entered in lower case as shown, except for hexadecimal numbers where either case may be used.

The meanings of the parameters of this command are as follows:

primary_router | secondary_router | multicast_router | no_router
: For HiperSockets: Only the multicast_router value is honored, if the

microcode provides this feature. It causes the driver to receive all multicast packets of the HiperSockets CHPID. This is to be used for multicast routers.

For OSA-Express feature in QDIO mode: Specifies whether the device is used to interconnect networks. A ″primary router″ is the principal connection between two networks; a ″secondary router″ is used as backup in case of problems with the primary. Both of these options require the Linux system to be configured as a router. The ″multicast_router″ parameter is ignored. The default for this parameter is ″No router″ – the OSA-Express feature will only be used to connect the Linux for zSeries system to a single network.

It is possible to add routing status dynamically. This is done with the command:

```
echo primary_router ifname > /proc/qeth
```

or

```
echo secondary_router ifname > /proc/qeth
```

ifname is the name of the interface in Linux , for example `eth0`.

It is not possible to reset routing status with the current hardware.

Note: To configure Linux running as a VM guest or in an LPAR as a router, not only must the ’primary_router’ or ’secondary_router’ option be set, but IP forwarding must also be enabled. This can be done by:

```
sysctl -w net.ipv4.conf.all.forwarding=1
sysctl -w net.ipv6.conf.all.forwarding=1
```

or in distribution-dependent configuration files. Refer to your distribution manual for the procedure.

sw_checksumming | no_checksumming
Specifies whether error detection is to be performed by the driver, or is not required.

prio_queueing_tos | prio_queueing_prec | no_prio_queueing | no_prio_queueing: *number*
For HiperSockets: This parameter is ignored.

Specifies the type of priority queuing to be used. See “Priority queuing” on page 127 for details. **no_prio_queueing** is equivalent to **no_prio_queueing:** *2* (the default queue).

polltime
Specifies the maximum duration of background polling (in microseconds) used by `QDIO`. The default is 500. This parameter is effective only for OSA under LPAR and V=R guests under VM in native mode. When using V=V guests, HiperSockets, or VM Guest LAN, the poll time will be adjusted by the qdio.o module.

dont_fake_broadcast | fake_broadcast
: `fake_broadcast` sets the 'broadcast capable' device flag. This is necessary for the `gated` routing daemon. If the feature supports broadcast, this parameter is meaningless.

broadcast_allrings | broadcast_local
: `broadcast_local` restricts Token Ring broadcasts to the local LAN segment. `broadcast_allrings` propagates Token Ring broadcasts to all rings connected via bridges.

macaddr_noncanon | macaddr_canon
: Interpret MAC addresses in canonical or non-canonical form in a Token Ring.

enable_takeover | disable_takeover
: allow/do not allow IP address takeover.

enable_takeover | disable_takeover
: allow/do not allow IP address takeover.

all_hhlen
: Reserves additional hardware-header space in front of every packet in socket buffers ("sk_buff"). This can be beneficial for Linux virtual servers (LVS) and IPIP tunnels, in general for software that makes use of free space in front of packets.

OSA-Express feature in QDIO mode channel device layer configuration example

```
add_parms,0x10,0x7c00,0x7c02,portname:NETWORK1
qeth1,0x7c00,0x7c01,0x7c02,4096,-1
```

This tells the channel device layer to force `qeth1` (if detected) to use device addresses 7c00, 7c01 and 7c02, allocate four megabytes of buffer space, name the connection 'NETWORK1', and use the default port.

HiperSockets channel device layer configuration example

```
qeth1,0x7c00,0x7c01,0x7c02,4096,-1
```

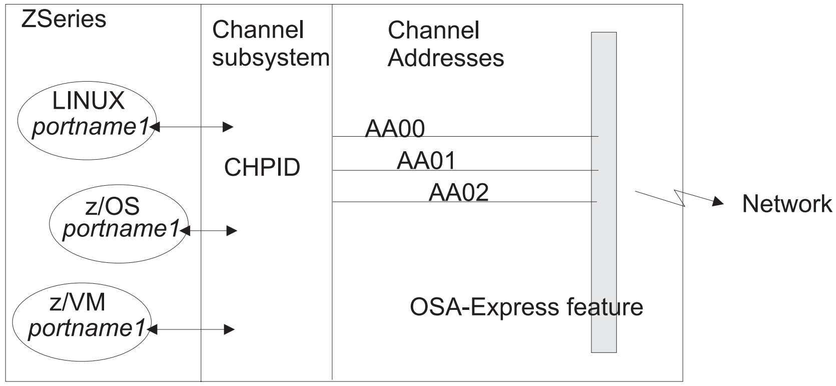
This tells the channel device layer to force `qeth1` (if detected) to use device addresses 7c00, 7c01 and 7c02, allocate four megabytes of buffer space, and use the default port.

Examples: OSA-Express feature in QDIO mode

1: Basic configuration

In this example a single OSA-Express CHPID is being used to connect a Linux for zSeries system to a network.

Hardware configuration – OSA-Express connecting Linux for zSeries to a network



Software configuration – OSA-Express connecting Linux for zSeries to a network

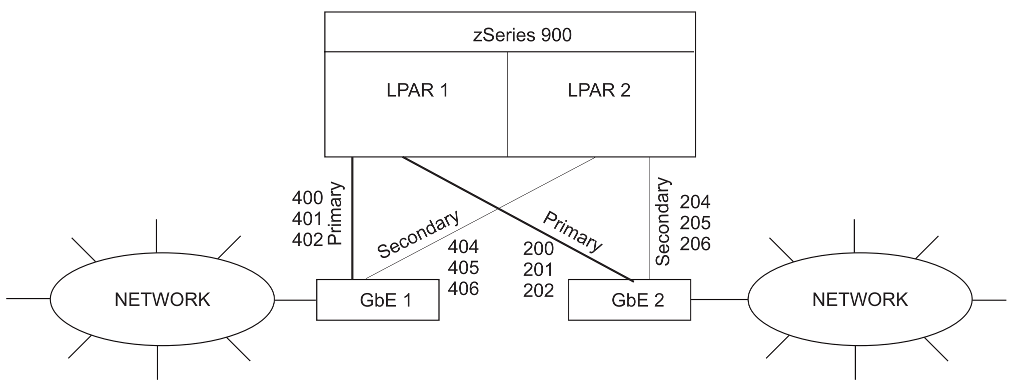
With the channel device layer the load commands for this configuration are:

```
add_parms,0x10,0xAA00,0xAA02,portname:NETWORK
qeth-1,0xAA00,0xAA01,0xAA02
```

2: Router configuration

This example shows how Linux systems running on different LPARs in a zSeries may use OSA-Express to communicate with a network or to act as a router between networks. Note that the LPAR needs to have IP forwarding switched on in order to act as a router. This is usually done in the proc filesystem or a configuration file; refer to your distribution manual for details.

Hardware configuration – OSA-Express and Linux for zSeries as a router



In this example it is assumed that Linux is configured as a router in both LPARs.

Software configuration – OSA-Express and Linux for zSeries as a router

LPAR 1 – This LPAR is configured to be the primary routing LPAR:

```
add_parms,0x10,0x400,0x402,primary_router,portname:OSACARD1
qeth-1,0x400,0x401,0x402
add_parms,0x10,0x200,0x202,primary_router,portname:OSACARD2
qeth-1,0x200,0x201,0x202
```

LPAR 2 – This LPAR is configured to be the secondary routing LPAR:

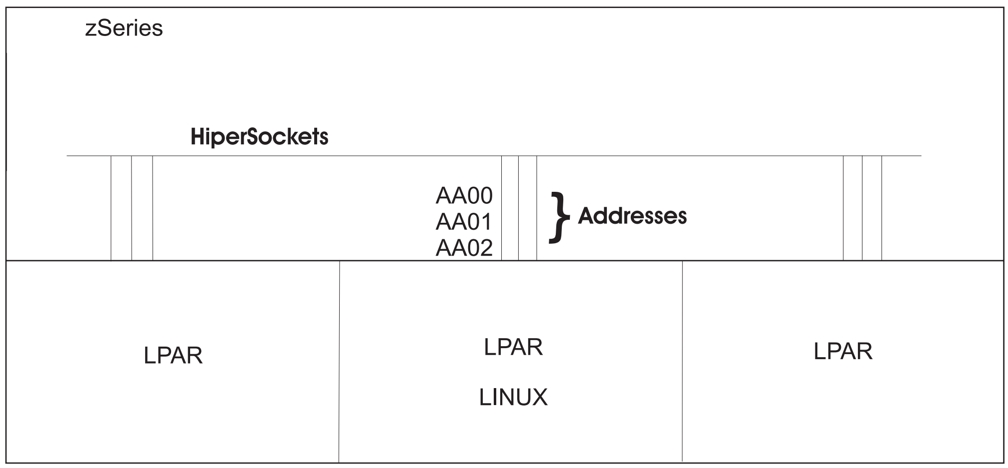
```
add_parms,0x10,0x404,0x406,secondary_router,portname:OSACARD1
qeth-1,0x404,0x405,0x406
add_parms,0x10,0x204,0x206,secondary_router,portname:OSACARD2
qeth-1,0x204,0x205,0x206
```

Examples: HiperSockets

1: Basic configuration

In this example a single HiperSockets is being used to connect a Linux for zSeries system to a network.

Hardware configuration – HiperSockets connecting Linux for zSeries to a network



Software configuration – HiperSockets connecting Linux for zSeries to a network

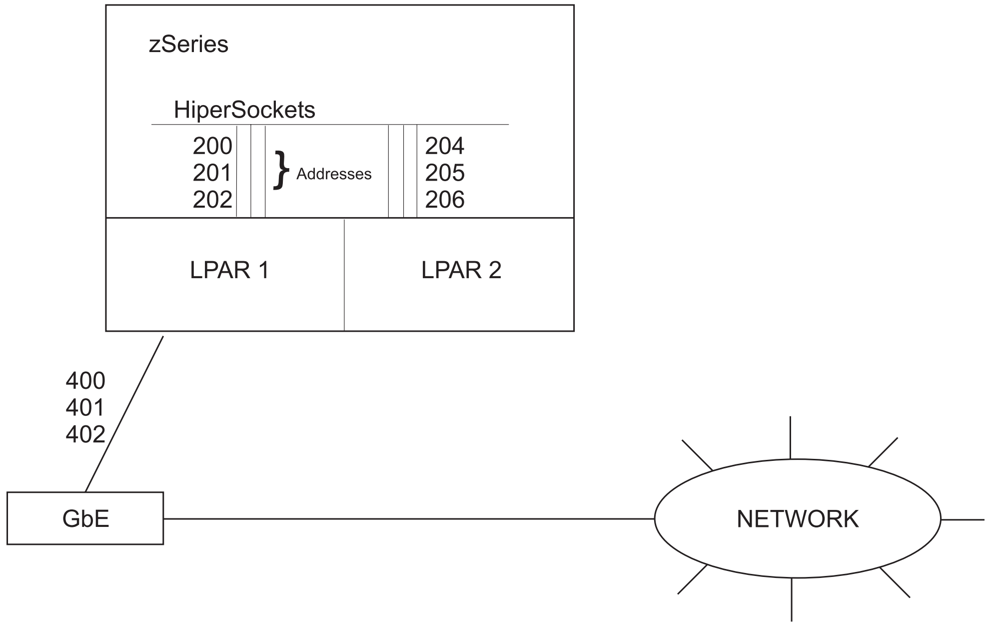
With the channel device layer the load commands for this configuration are:

```
qeth-1,0xAA00,0xAA01,0xAA02
```

2: Router configuration

This example shows how Linux systems running on different LPARs in a zSeries may use HiperSockets to communicate with a network or to act as a router between networks.

Hardware configuration – HiperSockets and Linux for zSeries as a router



In this example it is assumed that Linux is configured as a router in both LPARs.

Software configuration – HiperSockets, an OSA-Express feature, and Linux for zSeries as a router

LPAR 1 – uses subchannels 200 - 202 for connecting to HiperSockets and 400 - 402 to connect to the OSA-Express feature. There it is able to route packets in and out.

```
add_parms,0x10,0x400,0x402,primary_router
qeth-1,0x400,0x401,0x402
qeth-1,0x200,0x201,0x202
```

LPAR 2 – uses subchannels 204 - 206 as a network client:

```
qeth-1,0x204,0x205,0x206
```

OSA-Express feature in QDIO mode and HiperSockets – Preparing the connection

Activating the OSA-Express feature in QDIO mode and HiperSockets connection:

The network devices can be activated with the `ifconfig` command. It is necessary to define the right MTU size for the channel device, otherwise it will not work properly. You must use the same MTU size as that which is defined on the remote side.

For HiperSockets, the right MTU size is selected automatically based on the HiperSockets maximum frame size. The HiperSockets maximum frame size is a hardware definition parameter for HiperSockets CHPID definition.

For details of the ifconfig command, see the ifconfig man page.

An example of the use of `ifconfig` for HiperSockets is:

```
ifconfig hsi0 192.168.100.11 netmask 255.255.255.0
```

An example of the use of `ifconfig` for an OSA-Express feature in QDIO mode is:

```
ifconfig eth0 192.168.100.11 netmask 255.255.255.0
          broadcast 192.168.100.255 mtu 1492 up
```

or, more simply:

```
ifconfig eth0 192.168.100.11
```

IP address takeover

It is possible to add and remove ranges of IP addresses for the OSA-Express feature in QDIO mode or HiperSockets by writing to the `/proc/qeth_ipa_takeover` file. When a command is written to this file the driver calls on the OSA ″Address Takeover″ mechanism. This overrides any previous allocation of the specified address to another LPAR or feature. If another LPAR on the same feature has already registered for that IP address this association will be removed.

Note: Using HiperSockets, only IP addresses of another Linux operating system in the same CEC (Central Electronics Complex) can be taken over, not IP addresses of other zSeries operating systems. IP takeover must be enabled both on the image taking over the address, and on the image that gives up its address.

The registered addresses are held in this file in plain text and can be read to see the current associations.

Only one command at a time can be written to the file. Subsequent commands in the same write action are ignored.

The following commands are available:
- `add4` <*addr*>/<*mask_bits*>[:<*interface*>]
- `add6` <*addr*>/<*mask_bits*>[:<*interface*>]
- `inv4`
- `inv6`
- `del4` <*addr*>/<*mask_bits*>[:<*interface*>]
- `del6` <*addr*>/<*mask_bits*>[:<*interface*>]

`add4` or `add6` adds an address range. `del4` or `del6` deletes an address range. <*addr*> is an 8- or 32-character hexadecimal IP address. <*mask_bits*> specifies the number of bits which are set in the network mask. <*interface*> is optional and specifies the interface name to which the address range is bound.

For example

```
echo add4 c0a80a00/24 > /proc/qeth_ipa_takeover
```

activates all addresses in the 192.168.10 subnet for address takeover.

`inv4` or `inv6` inverts the selection of address ranges done with add4 or `add6`. Issue `inv4` or `inv6` once to set all addresses which have been specified with `add4` or `add6` not to use the takeover mechanism; all other addresses will be set to use it.

Notes:

1. The address is not actually taken over until a corresponding `ifconfig` command is executed.
2. See also the note concerning setting IP addresses under "QETH restrictions" on page 126.

Example: OSA-Express feature in QDIO mode

Assuming all addresses in subnet 192.168.10 are activated for address takeover, issue an `ifconfig` command to take over a specific address, for example:

```
ifconfig eth0 192.168.10.5
```

This sets the IP address 192.168.10.5 on the feature `eth0`, and removes it from other LPARs if necessary.

The IP address must be different to that previously set on the device or no action will be taken. To take over a device at the same address `ifconfig` must be called twice; the first time with a dummy address (for example 0.0.0.0) to notify the device of the takeover and the second time with the original address to reset it. To re-capture the `eth0` device with the IP address in the previous example you could use:

```
ifconfig eth0 0.0.0.0
ifconfig eth0 192.168.10.5
```

The second line alone will not take over the device from another LPAR if the IP address is the same as that set by the other LPAR.

Example: HiperSockets

Assuming all addresses in subnet 192.168.10 are activated for address takeover, issue an `ifconfig` command to take over a specific address, for example:

```
ifconfig hsi0 192.168.10.5
```

The IP address must be different to that previously set on the device or no action will be taken. To take over a device at the same address `ifconfig` must be called twice; the first time with a dummy address (for example 0.0.0.0) to notify the device of the takeover and the second time with the original address to reset it. To re-capture the `hsi0` device with the IP address in the previous example you could use:

```
ifconfig hsi0 0.0.0.0
ifconfig hsi0 192.168.10.5
```

The second line alone will not take over the device from another LPAR if the IP address is the same as that set by the other LPAR.

Proxy ARP

The following commands implement a proxy ARP capability:

- add_rxip4 <*addr*>:<*interface*>
- add_rxip6 <*addr*>:<*interface*>
- del_rxip4 <*addr*>:<*interface*>
- del_rxip6 <*addr*>:<*interface*>

The specified address is set/unset as "do not fail a gratuitous ARP" on the specified interface.

Example:

```
[Gateway to Internet=GW]
   |
[Linux/zSeries router=R]
   |    |    |
 [G1]  [G2]  [G3]
```

G1, G2, G3 are Linux guests (connected, for example, via IUCV to R), reached from GW (or the outside world) via R. R is the ARP proxy for G1, G2, G3. That is, it agrees to take care of packets destined for G1, G2 and G3. The advantage is that GW does not need to know that G1, G2, G3 are behind a router. Assuming that G1 has address 1.2.3.4, R would issue

```
echo add_rxip4 01020304:eth0 > /proc/qeth_ipa_takeover
```

to receive packets for 1.2.3.4, so that it can forward them to G1.

Note: See the note concerning setting IP addresses under "QETH restrictions" on page 126.

More information on Proxy ARP is available from

```
http://www.sjdjweis.com/linux/proxyarp/
http://ibiblio.org/pub/Linux/docs/HOWTO/unmaintained/mini/Proxy-ARP
http://www.tldp.org/HOWTO/mini/Proxy-ARP-Subnet/
```

.

OSA-Express feature in QDIO mode – Virtual IP address (VIPA)

Note: This does not apply to HiperSockets.

zSeries and S/390 use virtual IP addresses to protect against certain types of hardware connection failure. These virtual IP addresses (VIPAs) can be built under Linux using "dummy" devices (for example `dummy0` or `dummy1`) or loopback aliases (`lo:0` or `lo:1`). (To enable these virtual devices the kernel must be compiled with the special option CONFIG_DUMMY.)

In order to make the OSA-Express feature accept packets for the VIPAs on the system the `qeth` driver must be informed about the VIPAs using the `/proc` interface.

The virtual devices are configured with the following commands:

- add_vipa4 <*addr*>:<*interface*>
- add_vipa6 <*addr*>:<*interface*>

- `del_vipa4 <addr>:<interface>`
- `del_vipa6 <addr>:<interface>`

These commands must be piped to `/proc/qeth_ipa_takeover`, for example:

```
echo add_vipa4 c0a80a00 > /proc/qeth_ipa_takeover
```

`add_vipa4` or `add_vipa6` add an address range. `del_vipa4` or `del_vipa6` delete an address range. *<addr>* is an 8 character or 32 character hexadecimal IP address. *<interface>* is optional and specifies the interface name to which the address range is bound.

For an example of how to use VIPA, see Chapter 15, “VIPA – minimize outage due to adapter failure” on page 183.

Note: See also the note concerning setting IP addresses under “QETH restrictions”.

QETH restrictions

- The MTU range is 576 – 61440. However, depending on the medium and networking hardware settings, it may be restricted to 1492, 1500, 8992 or 9000. For HiperSockets the MTU range extends to 57344. This may be restricted by the framesize announced by the microcode.
- The maximum MTU size is limited by the value of the *memory_usage_in_k* parameter, which, together with the maximum frame (buffer) size, determines the number of buffers. The frame size for OSA-Express is fixed at 64 KB. For HiperSockets, the maximum frame size is defined during HiperSockets CHPID definition in the IOCDS. If the hardware configuration specifies the maximum frame size as 40 KB, the MTU can be configured up to 32 KB (frame size minus 8 KB) using ifconfig. Possible frame sizes are 16, 24, 40, and 64 KB.

  The total memory usage is

  ```
  (number of buffers) X (Linux memory usage per buffer)
  ```

  The Linux memory usage per buffer is 16 KB for frame size 16 KB, 32 KB for frame size 24 KB, and 64 KB for frame sizes 40 and 64 KB. Linux will calculate the number of buffers from the total memory usage given in the chandev statement (where the number of buffers is <=128 and >=16. If a parameter is too high, Linux will allocate 128 buffers of 64 KB each.
- There is a restriction in Linux that the packet size of a multicast packet cannot be greater than the MTU size of the interface used.
- There may be circumstances that prevent **ifconfig** (or other commands) from setting an IP address on an OSA-Express network feature. The most common one is that another system in the network has set that IP address already. As a result, the IP address will be indicated by **ifconfig** as being set on the interface, but the address is not actually set on the feature. Since the design of the network stack in Linux does not allow feedback about IP address changes, there is no means of notifying the user of the problem other than to log a message. This message (usually containing text such as ″could not set IP address″ or ″duplicate IP address″) will appear in the kernel messages (displayable using ″dmesg″). For most distribution settings, this will also trigger a message in /var/log/messages. If you are not sure whether the IP address was set properly or experience a networking problem, you should always check these logs to see if an error was encountered when setting the address.

This requirement applies to both IPv4 and IPv6 addresses, and to IP takeover, VIPA, HiperSockets, and Proxy ARP.

Priority queuing

Note: This does not apply to HiperSockets.

The OSA-Express feature in QDIO mode has four output queues (queues 0 to 3) in central storage. The feature gives these queues different priorities (queue 0 having the highest priority) which is relevant mainly to high traffic situations. When there is little traffic queuing has no impact on processing.

The device driver can put data on one or more of the queues. By default it uses queue 2 for all data. However, the driver can look at outgoing IP packets and select a queue for the data according to the IP type of service (if `prio_queueing_tos` is specified in the options) or IP precedence (if `prio_queueing_prec` is specified in the options) fields. These fields are part of the IP datagram header and can be set with a `setsockopt` call.

Some applications use these fields to tag data. The mapping the driver performs between IP type of service is as follows:

IP type of service	Queue used when IP TOS queueing is switched on
not important	3
low latency	0
high throughput	1
high reliability	2
default	2 (The default queue may be changed by a kernel option.)

When IP precedence is selected as the queueing type, the two most significant bits of each IP header precedence field are used to determine the queue for this packet.

Support for IP Version 6 (IPv6)

Note: Support for IPv6 applies to Gigabit Ethernet (GbE) and Fast Ethernet (FENET) only.

There are noticeable differences between the IP stacks for versions 4 and 6. Some concepts in IPv6 are different from IPv4, such as neighbor discovery, broadcast, and IPSec. IPv6 uses a 16-byte address field, while the addresses under IPv4 are 4 bytes in length.

From a user point of view, however, the impact of IPv6 is largely limited to the specification of IP addresses and the use of commands that are specific to a particular version (see “IP address takeover” on page 123, “OSA-Express feature in QDIO mode – Virtual IP address (VIPA)” on page 125, and “Proxy ARP” on page 125, for example).

Note: See the note concerning setting IP addresses under “QETH restrictions” on page 126.

Stateless autoconfiguration in IPv6

Stateless autoconfiguration uses a feature-generated unique identification value to generate different addresses on each Linux instance. This requires a kernel patch affecting `struct net.device` and thus all networking device drivers. Consult your distributor if you do not build a kernel on your own.

Querying and purging OSAand HiperSockets ARP data

This function is implemented by the `qetharp` utility. See "qetharp - Query and purge OSA and HiperSockets ARP data" on page 156 for details.

Broadcast support

If a feature supports broadcast traffic, this will be enabled automatically and reflected in the interface flags (seen in `ifconfig`). Broadcast traffic can then flow in and out of the device.

Broadcasts are useful when a host needs to find information without knowing exactly what other host can supply it, or when a host wants to provide information to a large set of hosts in a timely manner.

When a host needs information that one or more of its neighbors might have, it could have a list of neighbors to ask, or it could poll all of its possible neighbors until one responds. Use of a wired-in list creates obvious network management problems (early binding is inflexible). On the other hand, asking all of one's neighbors is slow if one must generate plausible host addresses, and try them until one works. On the ARPANET, for example, there are roughly 65,000 plausible host numbers. Most IP implementations have used wired-in lists (for example, addresses of "prime" gateways.)

Fortunately, broadcasting provides a fast and simple way for a host to reach all of its neighbors.

A host might also use a broadcast to provide all of its neighbors with some information; for example, a gateway might announce its presence to other gateways.

One way to view broadcasting is as an imperfect substitute for multicasting, the sending of messages to a subset of the hosts on a network. In practice, broadcasts are usually used where multicasts are what is wanted; datagrams are broadcast at the hardware level, but filtering software in the receiving hosts gives the effect of multicasting.

Some Samba services are using broadcasts. Samba is an open-source Linux for zSeries application that lets you run your entire file and print system on a single machine--while still delivering zSeries' outstanding performance. This means you can reduce costs and simplify administration while getting zSeries scalability, reliability and availability.

Part 4. Installation/configuration commands and parameters

This section describes configuration parameters for Linux for zSeries and the tools available for configuration.

These are described in the chapters:
- Useful Linux commands:
  - dasdfmt - Format a DASD
  - dasdview - Display DASD characteristics
  - fdasd - Partition a DASD
  - osasnmpd - Start OSA-Express SNMP subagent
  - qetharp - Query and purge OSA and HiperSockets ARP data
  - snIPL - Remote control of Support Element functions
  - zipl - Make DASD bootable
  - ifconfig - Configure a network interface
  - insmod - Load a module into the Linux kernel
  - modprobe - Load a module with dependencies into the Linux kernel
  - lsmod - List loaded modules
  - depmod - Create dependency descriptions for loadable kernel modules
  - mke2fs - Create a file system
  - vconfig - Configure VLANs
- VIPA (Virtual IP Address) implementation, to minimize outage due to adapter failure.
- Kernel parameters:
  - cio_ignore
  - cio_msg
  - cio_notoper_msg
  - ipldelay
  - maxcpus
  - mem
  - noinitrd
  - ramdisk_size
  - ro
  - root
  - vmhalt
- Overview of the parameter line file.

Note: For tools related to taking and analyzing system dumps, see the manual *Linux for zSeries Using the Dump Tools*, LNUX-1108.

Chapter 14. Useful Linux commands

This chapter describes commands which have been created to install and configure Linux for zSeries:

- dasdfmt
- dasdview
- fdasd
- osasnmpd
- qetharp
- snIPL
- zipl

It also summarizes standard Linux commands used during the installation, configuration and initial startup of Linux for zSeries. These are:

- ifconfig
- insmod
- modprobe
- lsmod
- depmod
- mke2fs
- vconfig

Note: For tools related to taking and analyzing system dumps, see the manual *Linux for zSeries Using the Dump Tools*, LNUX-1108.

dasdfmt - Format a DASD

Purpose

This tool is used to give a low-level format to direct access storage devices (DASD). Note that this is a software format. To give a hardware format to raw DASD you must use another zSeries device support facility such as ICKDSF, either in stand-alone mode or through another operating system.

`dasdfmt` uses an `ioctl` call to the DASD driver to format tracks. A blocksize (hard sector size) can be specified. Remember that the formatting process can take quite a long time. Use the -p option to check the progress.

See Chapter 2, “Partitioning DASD” on page 5 for further information about partitioning DASD.

Usage

Prerequisites:

(see Chapter 2, “Partitioning DASD” on page 5 for terminology and further information)

- You must have installed the library `libvtoc.so` in the Linux `/lib` directory, and
- You must have root permissions.

Format

dasdfmt syntax

```
                                                      ┌─-d cdl─┐
►►──dasdfmt──┬──────┬──┬─-f ──diskspec─┬──┼────────┼──┬──────────────┬──────►
             ├─ -t ─┤  └─-n ──devno────┘  └─-d ldl─┘  ├─ -L ─────────┤
             ├─ -v ─┤                                  └─ -l ──volid─┘
             ├─ -y ─┤
             └─ -F ─┘

   ┌─-b 4096──────────┐
►──┼──────────────────┼──┬──────────────────┬───────────────────────────────►◄
   └─-b ──blocksize───┘  ├─ -p ─────────────┤
                         └─ -m ──hashstep───┘
```

or

```
►►──dasdfmt── -h ──────────────────────────────────────────────────────────►◄
```

or

```
►►──dasdfmt── -V ──────────────────────────────────────────────────────────►◄
```

The parameters are:

-f *diskspec* or **`--device=`***diskspec*
: Specifies the device to be formatted. *diskspec* takes the form: `/dev/dasd/`*`xxxx`*`/device` where *xxxx* is the four-character hexadecimal device address of the DASD.

-n *devno* **or --devno** *devno*
: Specifies the four-character hexadecimal device address of the disk to format, for example `-n 01a3`.

The following parameters are necessary. If you do not specify values for them, you will be prompted. You can use the default values by pressing the <enter> key:

-b *block_size* or **`--blocksize=`***block_size*
: Specifies the blocksize. The minimum blocksize is 512 bytes and increases in powers of 2 (512, 1024, 2048, and 4096). The default blocksize is 4096.

The following parameters are optional:

-d *disklayout* or **`--disk_layout=`***disklayout*
: Formats the device with compatible disk layout (`cdl`) or Linux disk layout (`ldl`).

-F or --force
: Formats the device without checking if it is mounted.

-h or --help
: Prints out an overview of the syntax. Any other parameters will be ignored.

-l *volid* or **`--label=`***volid*
: Specifies the volume identifier (see 3 on page 7) to be written to the disk.

-L or **`--no_label`**
: Valid for `-d ldl` only, where it suppresses the default `LNX1` label.

-m *hashstep* or **`--hashmarks=`***hashstep*
: prints a hash mark (#) after every *hashstep* cylinders are formatted. *hashstep* must be in the range 1 to 1000. The default is 10.

  This may be used if the progress bar cannot be displayed, for example on a 3270 console

-p or **`--progressbar`**
: prints a progress bar. Do not use this option if you are using a 3270 console.

-t or **`--test`**
: (test mode) Analyzes parameters and prints out what would happen, but does not modify the disk.

-v
: Prints out extra information messages.

-V or --version
: Prints the version number of `dasdfmt` and exits.

-y
: Starts formatting immediately without prompting for confirmation.

Restrictions

Note: The following is technically not a restriction of `dasdfmt`, since formatting interruptions are detected and the device is reset to the "active, not formatted" state. Nevertheless, it could happen that a disk goes into the "accepted" state, and the following procedure can be used to remedy this situation.

During the formatting process the DASD is placed in a temporary 'accepted' state. If it is left in this state (for example if DASDFMT is interrupted) the DASD cannot be accessed. You can check the state of a DASD by entering:

```
# cat /proc/dasd/devices
```

If you see "accepted" in the output the corresponding DASD is disabled and not available for use. You can re-enable the DASD with the command:

```
# echo 'set <devno> on' > /proc/dasd/devices
```

where <*devno*> is the device number of the DASD you want to enable.

Example:

```
# cat /proc/dasd/devices
0700(ECKD) at ( 94:  0) is   dasda:active at blocksize: 4096, 601020 blocks, 2347 MB
0800(ECKD) at ( 94:  4) is   dasdb:active at blocksize: 4096, 601020 blocks, 2347 MB
0900(ECKD) at ( 94:  8) is   dasdc:accepted
```

Disk 900 is in the accepted state and cannot be formatted with the dasdfmt command.

```
# echo 'set 900 on' > /proc/dasd/devices
```

enables the disk and changes its state to "active, not formatted".

```
# cat /proc/dasd/devices
0700(ECKD) at ( 94:  0) is   dasda:active at blocksize: 4096, 601020 blocks, 2347 MB
0800(ECKD) at ( 94:  4) is   dasdb:active at blocksize: 4096, 601020 blocks, 2347 MB
0900(ECKD) at ( 94:  8) is   dasdc:active n/f
#
```

You now will be able to format the DASD.

Examples

Example 1

To format a 100 cylinder VM minidisk with the standard Linux disk layout and a 4 KB blocksize at address 0193:

```
[root@host /root]# dasdfmt -b 4096 -d ldl -n 193
Drive Geometry: 100 Cylinders * 15 Heads = 1500 Tracks
I am going to format the device 193 in the following way:
   Device number of device : 0x193
   Major number of device  : 94
   Minor number of device  : 12
   Labeling device         : yes
   Disk label              : LNX1
   Disk identifier         : 0X0193
   Extent start (trk no)   : 0
   Extent end (trk no)     : 1499
   Compatible Disk Layout  : no
   Blocksize               : 4096

--->> ATTENTION! <<---
All data in the specified range of that device will be lost.
Type "yes" to continue, no will leave the disk untouched: yes
Formatting the device. This may take a while (get yourself a coffee).

cyl   100 of   100 |##################################################| 100%

Finished formatting the device.
Rereading the partition table... ok
[root@host /root]#
```

Example 2

To format the same disk with the compatible disk layout (using the default value of the -d option). The device is specified by device node.

```
[root@host /root]# dasdfmt -b 4096 -f /dev/dasd/0193/device
Drive Geometry: 100 Cylinders * 15 Heads = 1500 Tracks
I am going to format the device /dev/dasd/0193/device in the following way:
Device number of device    : 0x193
Major number of device     : 94
Minor number of device     : 12
labeling device            : yes
Disk label                 : VOL1
Disk identifier            : 0X0193
Extent start (trk no)      : 0
Extent end (trk no)        : 1499
Compatible Disk Layout     : yes
Blocksize                  : 4096

--->> ATTENTION! <<---
All data in the specified range of that device will be lost.
Type "yes" to continue, no will leave the disk untouched: yes
Formatting the device. This may take a while (get yourself a coffee).

cyl   100 of   100 |##################################################| 100%

Finished formatting the device.
Rereading the partition table... ok
[root@host /root]#
```

dasdview - Display DASD Structure

Purpose

This tool is used to send DASD and VTOC information to the system console. Such information might be the volume label, VTOC entries, or disk geometry details.

If you specify a start point and size, you can also display the contents of a disk dump.

Usage

Prerequisites:

(See Chapter 2, “Partitioning DASD” on page 5 for terminology and further information.)

You must have:

- Installed the library `libvtoc.so` in the Linux `/lib` directory.
- Root permissions.

dasdview displays this DASD information:

- The volume label.
- VTOC details (general information, and FMT1, FMT4, FMT5 and FMT7 labels).
- The content of the DASD, by specifying:
  - Starting point
  - Size

  You can display these values in hexadecimal, EBCDIC, and ASCII format.

Format

dasdview syntax (1)

```
►►──dasdview──┬──────┬──────────────────────────────────────────►◄
              ├─ -h ─┤
              ├─ -? ─┤
              └─ -v ─┘
```

dasdview syntax (2)

```
►►──dasdview─────────────────────────────────────────────────────►

►─┬──────────────────────────────────────────────┬──┬─ -n ── devno ─┬──►◄
  │  ┌─0───────────┐ ┌─128────────┐ ┌─ -1 ─┐     │  └─ -f ── node ──┘
  └─┬┴─ -b ── begin ─┴─┴─ -s ── size ─┴─┴─ -2 ─┴─────┤
    ├─ -i ───────────────────────────────────────┤
    ├─ -x ───────────────────────────────────────┤
    ├─ -l ───────────────────────────────────────┤
    └─ -t ── spec ───────────────────────────────┘
```

The parameters are:

-h or --help
Display short usage text on console, to see manpage enter **man dasdview**.

-? or --help
Display short usage text on console, to see manpage enter **man dasdview**.

-v or --version
Display version number on console, and exit.

-b *begin* or **--begin=***begin*
Display disk content on the console, starting from *begin*. The content of the disk are displayed as hexadecimal numbers, ASCII text and EBCDIC text. If *size* is not specified (see below), **dasdview** will take the default size (128 bytes). You can specify the variable *begin* as:

```
begin[k|m|b|t|c]
```

The default for *begin* is 0.

dasdview displays a disk dump on the console using the DASD driver. The DASD driver might suppress parts of the disk, or add information that is not relevant. This might occur, for example, when displaying the first two tracks of a disk that has been formatted as **cdl**. In this situation, the DASD driver will pad shorter blocks with zeros, in order to maintain a constant blocksize. All Linux applications (including **dasdview**) will process according to this rule.

Here are some examples of how this option can be used:

```
-b 32   (start printing at Byte 32)
-b 32k  (start printing at kByte 32)
-b 32m  (start printing at MByte 32)
-b 32b  (start printing at block 32)
-b 32t  (start printing at track 32)
-b 32c  (start printing at cylinder 32)
```

-s *size* or **--size=***size*
Display a disk dump on the console, starting at *begin*, and continuing for **size** = *size*). The content of the dump are displayed as hexadecimal numbers, ASCII text, and EBCDIC text. If a start value (*begin*) is not specified, **dasdview** will take the default. You can specify the variable *size* as:

```
size[k|m|b|t|c]
```

The default for *size* is 128 bytes.

Here are some examples of how this option can be used:

```
-s 16   (use a 16 Byte size)
-s 16k  (use a 16 kByte size)
-s 16m  (use a 16 MByte size)
-s 16b  (use a 16 block size)
-s 16t  (use a 16 track size)
-s 16c  (use a 16 cylinder size)
```

-1 Display the disk dump using format 1 (as 16 Bytes per line in hexadecimal, ASCII and EBCDIC). A line number is not displayed. You can only use option **-1** together with **-b** or **-s**.

Option **-1** is the default.

-2 Display the disk dump using format 2 (as 8 Bytes per line in hexadecimal, ASCII and EBCDIC). A decimal and hexadecimal byte count are also displayed. You can only use option **-2** together with **-b** or **-s**.

-i or --info
Display basic information such as device node, device number, device type, or geometry data.

-x or --extended
Display the information obtained by using **-i** option, but also open count, subchannel identifier, and so on.

-l or --label
Display the volume label.

-t *spec* or **--vtoc=***spec*
Display the VTOC's table-of-contents, or a single VTOC entry, on the console. The variable *spec* can take these values:

info Display overview information about the VTOC, such as a list of the dataset names and their sizes.
f1 Display the contents of all *format 1* data set control blocks (DSCBs).
f4 Display the contents of all *format 4* DSCBs.
f5 Display the contents of all *format 5* DSCBs.
f7 Display the contents of all *format 7* DSCBs.
all Display the contents of *all* DSCBs.

-n *devno* or **--devno=***devno*
Specify the device using the device number *devno*. You can only use this option if your system is using the device file system. Here is an example of the use of this option:

```
dasdview -i -n 193
```

-f *node* or **--devnode=***node*
Specify the device using the device node *devnode*. If your system is not using the device file system you must use this option to specify a device..

An example of the use of this option might be as follows. Assume you wish to display the information about a DASD that has a device node `/dev/dasda` or `/dev/dasd/0193/device`. You could then issue these commands:

```
dasdview -i -f /dev/dasda
```

or

```
dasdview -i -f /dev/dasd/0193/device
```

Examples

To display basic information about a DASD:

```
bash-2.04# dasdview -i -n 193
```

This displays:

```
--- general DASD information --------------------------------------------------
device node            : /dev/dasd/0193/device
device number          : hex 193       dec 403
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
bash-2.04#
```

To include extended information:

```
bash-2.04# dasdview -x -n 193
```

This displays:

```
--- general DASD information -----------------------------------------------------
device node            : /dev/dasd/0193/device
device number          : hex 193        dec 403
type                   : ECKD
device type            : hex 3390       dec 13200

--- DASD geometry ----------------------------------------------------------------
number of cylinders    : hex 64         dec 100
tracks per cylinder    : hex f          dec 15
blocks per track       : hex c          dec 12
blocksize              : hex 1000       dec 4096

--- extended DASD information ----------------------------------------------------
real device number     : hex 452bc08    dec 72530952
subchannel identifier  : hex e          dec 14
CU type  (SenseID)     : hex 3990       dec 14736
CU model (SenseID)     : hex e9         dec 233
device type  (SenseID) : hex 3390       dec 13200
device model (SenseID) : hex a          dec 10
open count             : hex 1          dec 1
req_queue_len          : hex 0          dec 0
chanq_len              : hex 0          dec 0
status                 : hex 5          dec 5
label_block            : hex 2          dec 2
FBA_layout             : hex 0          dec 0
characteristics_size   : hex 40         dec 64
confdata_size          : hex 100        dec 256

characteristics        : 3990e933 900a5f80  dff72024 0064000f
                         e000e5a2 05940222  13090674 00000000
                         00000000 00000000  24241502 dfee0001
                         0677080f 007f4a00  1b350000 00000000

configuration_data     : dc010100 4040f2f1  f0f54040 40c9c2d4
                         f1f3f0f0 f0f0f0f0  f0c6c3f1 f1f30509
                         dc000000 4040f2f1  f0f54040 40c9c2d4
                         f1f3f0f0 f0f0f0f0  f0c6c3f1 f1f30500
                         d4020000 4040f2f1  f0f5c5f2 f0c9c2d4
                         f1f3f0f0 f0f0f0f0  f0c6c3f1 f1f3050a
                         f0000001 4040f2f1  f0f54040 40c9c2d4
                         f1f3f0f0 f0f0f0f0  f0c6c3f1 f1f30500
                         00000000 00000000  00000000 00000000
                         00000000 00000000  00000000 00000000
                         00000000 00000000  00000000 00000000
                         00000000 00000000  00000000 00000000
                         00000000 00000000  00000000 00000000
                         00000000 00000000  00000000 00000000
                         800000a1 00001e00  51400009 0909a188
                         0140c009 7cb7efb7  00000000 00000800
bash-2.04#
```

To display volume label information:

```
bash-2.04# dasdview -l -n 193
```

This will display:

```
--- volume label -----------------------------------------------------------------
volume label key        : ascii  'åÖÖñ'
                        : ebcdic 'VOL1'
                        : hex    e5d6d3f1

volume label identifier : ascii  'åÖÖñ'
                        : ebcdic 'VOL1'
```

```
                        : hex    e5d6d3f1

volume identifier       : ascii  'ðçðñùó'
                        : ebcdic '0X0193'
                        : hex    f0e7f0f1f9f3

security byte           : hex    40


VTOC pointer            : hex    0000000101
                                 (cyl 0, trk 1, blk 1)

reserved                : ascii  '@@@@@'
                        : ebcdic '     '
                        : hex    4040404040

CI size for FBA         : ascii  '@@@@'
                        : ebcdic '    '
                        : hex    40404040

blocks per CI (FBA)     : ascii  '@@@@'
                        : ebcdic '    '
                        : hex    40404040

labels per CI (FBA)     : ascii  '@@@@'
                        : ebcdic '    '
                        : hex    40404040

reserved                : ascii  '@@@@'
                        : ebcdic '    '
                        : hex    40404040

owner code for VTOC     : ascii  '@@@@@@@@@@@@@@'
                          ebcdic '              '
                          hex    40404040 40404040  40404040 4040

reserved                : ascii  '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@'
                          ebcdic '                                '
                          hex    40404040 40404040  40404040 40404040
                                 40404040 40404040  40404040 40
bash-2.04#
```

To display partition information:

```
bash-2.04# dasdview -t info -n 193
```

This will display:

```
--- VTOC info ---------------------------------------------------------------
The VTOC contains:
  3 format 1 label(s)
  1 format 4 label(s)
  1 format 5 label(s)
  0 format 7 label(s)
Other S/390 and zSeries operating systems would see the following data sets:
 +----------------------------------------------+--------------+--------------+
 | data set                                     | start        | end          |
 +----------------------------------------------+--------------+--------------+
 | LINUX.V0X0193.PART0001.NATIVE                |          trk |          trk |
 | data set serial number : '0X0193'            |            2 |          500 |
 | system code            : 'IBM LINUX    '     |      cyl/trk |      cyl/trk |
 | creation date          :  year 2001, day 317 |        0/  2 |       33/  5 |
 +----------------------------------------------+--------------+--------------+
 | LINUX.V0X0193.PART0002.NATIVE                |          trk |          trk |
 | data set serial number : '0X0193'            |          501 |          900 |
 | system code            : 'IBM LINUX    '     |      cyl/trk |      cyl/trk |
 | creation date          :  year 2001, day 317 |       33/  6 |       60/  0 |
 +----------------------------------------------+--------------+--------------+
```

```
 | LINUX.V0X0193.PART0003.NATIVE                   |          trk |          trk |
 | data set serial number : '0X0193'               |          901 |         1499 |
 | system code            : 'IBM LINUX    '        |      cyl/trk |      cyl/trk |
 | creation date          :  year 2001, day 317    |       60/  1 |       99/ 14 |
 +-------------------------------------------------+--------------+--------------+
bash-2.04#
```

To display VTOC information:

```
bash-2.04# dasdview -t f4 -n 193
```

This will display:

```
--- VTOC format 4 label -------------------------------------------------------
DS4KEYCD    : 0404040404040404040404040404040404040404040404040404040404040404...
DS4IDFMT    : dec 244, hex f4
DS4HPCHR    : 0000000105 (cyl 0, trk 1, blk 5)
DS4DSREC    : dec 7, hex 0007
DS4HCCHH    : 00000000 (cyl 0, trk 0)
DS4NOATK    : dec 0, hex 0000
DS4VTOCI    : dec 0, hex 00
DS4NOEXT    : dec 1, hex 01
DS4SMSFG    : dec 0, hex 00
DS4DEVAC    : dec 0, hex 00
DS4DSCYL    : dec 100, hex 0064
DS4DSTRK    : dec 15, hex 000f
DS4DEVTK    : dec 58786, hex e5a2
DS4DEVI     : dec 0, hex 00
DS4DEVL     : dec 0, hex 00
DS4DEVK     : dec 0, hex 00
DS4DEVFG    : dec 48, hex 30
DS4DEVTL    : dec 0, hex 0000
DS4DEVDT    : dec 12, hex 0c
DS4DEVDB    : dec 0, hex 00
DS4AMTIM    : hex 0000000000000000
DS4AMCAT    : hex 000000
DS4R2TIM    : hex 0000000000000000
res1        : hex 0000000000
DS4F6PTR    : hex 0000000000
DS4VTOCE    : hex 01000000000100000001
              typeind    : dec 1, hex 01
              seqno      : dec 0, hex 00
              llimit     : hex 00000001 (cyl 0, trk 1)
              ulimit     : hex 00000001 (cyl 0, trk 1)
res2        : hex 000000000000000000000
DS4EFLVL    : dec 0, hex 00
DS4EFPTR    : hex 0000000000 (cyl 0, trk 0, blk 0)
res3        : hex 000000000000000000
bash-2.04#
```

To print the content of a disk to the console starting at block 2 (volume label):

```
bash-2.04# dasdview -b 2b -s 128 -n 193
```

This will display:

```
+---------------------------------------+-----------------+-----------------+
| HEXADECIMAL                           | EBCDIC          | ASCII           |
|  01....04 05....08  09....12 13....16 | 1.............16| 1.............16|
+---------------------------------------+-----------------+-----------------+
|  E5D6D3F1 E5D6D3F1  F0E7F0F1 F9F34000 | VOL1VOL10X0193?. | ??????????????@. |
|  00000101 40404040  40404040 40404040 | ................ | ................ |
|  40404040 40404040  40404040 40404040 | ???????????????? | @@@@@@@@@@@@@@@@ |
|  40404040 40404040  40404040 40404040 | ???????????????? | @@@@@@@@@@@@@@@@ |
|  40404040 40404040  40404040 40404040 | ???????????????? | @@@@@@@@@@@@@@@@ |
|  40404040 88001000  10000000 00808000 | ????h........... | @@@@?........... |
|  00000000 00000000  00010000 00000200 | ................ | ................ |
|  21000500 00000000  00000000 00000000 | ?............... | !............... |
+---------------------------------------+-----------------+-----------------+
bash-2.04#
```

To display the content of a disk on the console starting at block 14 (first FMT1 DSCB) using format 2:

```
bash-2.04# dasdview -b 14b -s 128 -2 -n 193
```

This will display:

```
+--------------+--------------+---------------------+----------+----------+
|     BYTE     |     BYTE     |     HEXADECIMAL     |  EBCDIC  |  ASCII   |
|   DECIMAL    | HEXADECIMAL  |  1 2 3 4   5 6 7 8  | 12345678 | 12345678 |
+--------------+--------------+---------------------+----------+----------+
|        57344 |         E000 | D3C9D5E4  E74BE5F0  | LINUX.V0 | ?????K?? |
|        57352 |         E008 | E7F0F1F9  F34BD7C1  | X0193.PA | ?????K?? |
|        57360 |         E010 | D9E3F0F0  F0F14BD5  | RT0001.N | ??????K? |
|        57368 |         E018 | C1E3C9E5  C5404040  | ATIVE??? | ?????@@@ |
|        57376 |         E020 | 40404040  40404040  | ???????? | @@@@@@@@ |
|        57384 |         E028 | 40404040  F1F0E7F0  | ????10X0 | @@@@???? |
|        57392 |         E030 | F1F9F300  0165013D  | 193.???? | ???.?e?= |
|        57400 |         E038 | 63016D01  0000C9C2  | ??_?..IB | c?m?..?? |
|        57408 |         E040 | D440D3C9  D5E4E740  | M?LINUX? | ?@?????@ |
|        57416 |         E048 | 40404065  013D0000  | ??????.. | @@@e?=.. |
|        57424 |         E050 | 00000000  88001000  | ....h.?. | ....?.?. |
|        57432 |         E058 | 10000000  00808000  | ?....??. | ?....??. |
|        57440 |         E060 | 00000000  00000000  | ........ | ........ |
|        57448 |         E068 | 00010000  00000200  | .?....?. | .?....?. |
|        57456 |         E070 | 21000500  00000000  | ?.?..... | !.?..... |
|        57464 |         E078 | 00000000  00000000  | ........ | ........ |
+--------------+--------------+---------------------+----------+----------+
bash-2.04#
```

To see what is at block 1234 (in this example there is nothing there):

```
bash-2.04# dasdview -b 1234b -s 128 -n 193
```

This will display:

```
+---------------------------------------+-----------------+-----------------+
| 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  | ................ | ................ |
+----------------------------------------+------------------+------------------+
bash-2.04#
```

To try byte 0 instead:

```
bash-2.04# dasdview -b 0 -s 64 -n 193
```

This will display:

```
+----------------------------------------+------------------+------------------+
| HEXADECIMAL                            | EBCDIC           | ASCII            |
|  01....04 05....08  09....12 13....16  | 1............16  | 1............16  |
+----------------------------------------+------------------+------------------+
|  C9D7D3F1 000A0000  0000000F 03000000  | IPL1............ | ????............ |
|  00000001 00000000  00000000 40404040  | ................ | ................ |
|  40404040 40404040  40404040 40404040  | ???????????????? | @@@@@@@@@@@@@@@@ |
|  40404040 40404040  40404040 40404040  | ???????????????? | @@@@@@@@@@@@@@@@ |
+----------------------------------------+------------------+------------------+
bash-2.04#
```

fdasd – DASD partitioning tool

Purpose

The new disk layout (z/OS compatible disk layout, or ’cdl’) now allows you to split DASD into several partitions. Use `fdasd` to manage partitions on a DASD. You can use this to create, change and delete partitions, and also to change the volume identifier label.

Note: To partition a SCSI disk, use **fdisk** rather than **fdasd**.

Usage

Prerequisites:

(see Chapter 2, “Partitioning DASD” on page 5 for terminology and further information)

- You must have installed the library `libvtoc.so` in the Linux `/lib` directory,
- You must have root permissions, and
- The disk must be formatted with `dasdfmt` with the (default) `-d cdl` option.

Attention: Incautious use of `fdasd` can result in loss of data.

`fdasd` is a menu-driven tool that you call with the command `fdasd` followed by the device node of the DASD you want to partition.

Overview of functionality:

1. `fdasd` checks that the volume has a valid volume label and VTOC. If either is missing or incorrect, `fdasd` will recreate it.
2. You are given a menu through which you can display DASD information, add or remove partitions, or change the volume identifier.
3. Your changes will not be written to disk until you type the ’write’ option on the menu. You may quit without altering the disk at any time prior to this. The items written to the disk will be the volume label, the ’format 4’ DSCB , a ’format 5’ DSCB and one to three ’format 1’ DSCBs.

Format

Command line syntax

fdasd command

```
>>--fdasd--+--+------------------------+--+-------------+--dasd--+-------><
           |  +-- -l --volid-----------+  +-- --verbose-+        |
           |  +-- -a ---------+--+----+                          |
           |     +-- -c --conf_file-+  +- -s -+                  |
           +-- -h ---------------------------------------------+
           +-- -? ---------------------------------------------+
           +-- -v ---------------------------------------------+
```

Where:

-l *volid*
: Is the volume label (see 3 on page 7).

-a or --auto
Auto-create one partition using the whole disk in non-interactive mode.

-c *conf_file* **or --config** *conf_file*
This option enables you to create several partitions, controlled by the plain text configuration file *conf_file*. Using this option, **fdasd** automatically switches to non-interactive mode and creates all the partitions specified in *conf_file*. The configuration file *conf_file* contains the following line for each partition you want to create:

```
[x,y]
```

where x is the keyword **first**, for the first possible track on disk, or a track number. y is either the keyword **last**, for the last possible track on disk, or a track number.

The example configuration file below would allow you to create three partitions:

```
[first,1000]
[1001,2000]
[2001,last]
```

-s or --silent
Suppresses messages. This is appropriate only in non-interactive mode (options -a or -c).

--verbose
Prints additional messages that are normally suppressed.

dasd Is the device node of the DASD you want to partition. If the device file system is used this has the form `/dev/dasd/`*`xxxx`*`/device`, where *xxxx* is the device number (devno) of the DASD. If the device file system is not used it has the form `/dev/dasd`*`xxx`* where *xxx* is one to three letters. See "DASD overview" on page 11 for more information.

-h or -? or --help
Displays help on command line arguments.

-v or --version
Displays the version of `fdasd`.

Processing

fdasd menu

When you have called `fdasd` using the syntax described in "Command line syntax" on page 144, the following menu will appear:

```
Command action
   m print this menu
   p print the partition table
   n add a new partition
   d delete a partition
   v change volume serial
   t change partition type
   r re-create VTOC and delete all partitions
   u re-create VTOC re-using existing partition sizes
   s show mapping (partition number - data set name)
   q quit without saving changes
   w write table to disk and exit

Command (m for help):
```

Menu commands:

m Re-displays the `fdasd` command menu.

p Displays the following information about the DASD:
- Number of cylinders
- Number of tracks per cylinder
- Number of blocks per track
- Blocksize
- Volume label
- Volume identifier
- Number of partitions defined

and the following information about each partition (including the free space area):
- Linux node
- Start track
- End track
- Number of tracks
- Partition id
- Partition type (1 = filesystem, 2 = swap)

n Adds a new partition to the DASD. You will be asked to give the start track and the length or end track of the new partition.

d Deletes a partition from the DASD. You will be asked which partition to delete.

v Changes the volume identifier. You will be asked to enter a new volume identifier. See page 144 for the format.

t Changes the partition type. You will be asked to identify the partition to be changed. You will then be asked for the new partition type (Linux native or swap). Note that this type is a guideline; the actual use Linux makes of the partition depends on how it is defined with the `mkswap` or `mk`*xx*`fs` tools. The main function of the partition type is to describe the partition to other operating systems so that, for example, swap partitions can be skipped by backup programs.

r Re-creates the VTOC and thereby deletes all partitions.

u Re-creates all VTOC labels without removing all partitions. Existing partition sizes will be re-used. This is useful to repair damaged labels or migrate partitions created with older versions of `fdasd`.

s Displays the mapping of partition numbers to data set names. For example:

```
Command (m for help): s

disk             : /dev/dasd/0193/device
volume label     : VOL1
volume identifier: 0X0193

WARNING: This mapping may be NOT up-to-date,
         if you have NOT saved your last changes!

/dev/dasd/0193/part1  -  LINUX.V0X0193.PART0001.NATIVE
/dev/dasd/0193/part2  -  LINUX.V0X0193.PART0002.NATIVE
/dev/dasd/0193/part3  -  LINUX.V0X0193.PART0003.NATIVE
```

q Quits `fdasd` without updating the disk. Any changes you have made (in this session) will be discarded.

w Writes your changes to disk and exits. After the data is written Linux will re-read the partition table.

Examples

Example using the menu

This section gives an example of how to use `fdasd` to create two partitions on a VM minidisk, change the type of one of the partitions, save the changes and check the results.

In this example we will format a VM minidisk with the compatible disk layout (cdl) using the device file system. The minidisk has device number 193.

1. Call `fdasd`, specifying the minidisk:

```
[root@host /root]# fdasd /dev/dasd/0193/device
```

`fdasd` reads the existing data and displays the menu:

```
reading volume label: VOL1
reading vtoc : ok

Command action
   m print this menu
   p print the partition table
   n add a new partition
   d delete a partition
   v change volume serial
   t change partition type
   r re-create VTOC and delete all partitions
   u re-create VTOC re-using existing partition sizes
   s show mapping (partition number - data set name)
   q quit without saving changes
   w write table to disk and exit
Command (m for help):
```

2. Use the `p` option to verify that no partitions have yet been created on this DASD:

```
Command (m for help): p

Disk /dev/dasd/0193/device:
     100 cylinders,
     15 tracks per cylinder,
     12 blocks per track
   4096 bytes per block
volume label: VOL1, volume identifier: 0X0193
maximum partition number: 3

             -----------tracks----------
             Device start    end   length   Id   System
                        2   1499     1498        unused
```

3. Define two partitions, one by specifying an end track and the other by specifying a length. (In both cases the default start tracks are used):

```
Command (m for help): n
First track (1 track = 48 KByte) ([2]-1499):
Using default value 2
Last track or +size[c|k|M] (2-[1499]): 700
You have selected track 700
```

```
Command (m for help): n
First track (1 track = 48 KByte) ([701]-1499):
Using default value 701
Last track or +size[c|k|M] (701-[1499]): +400
You have selected track 1100
```

4. Check the results using the p option:

```
Command (m for help): p

Disk /dev/dasd/0193/device:
     100 cylinders,
     15 tracks per cylinder,
     12 blocks per track
   4096 bytes per block
volume label: VOL1, volume identifier: 0X0193
maximum partition number: 3

              -----------tracks----------
              Device    start    end    length   Id   System
/dev/dasd/0193/part1        2    700       699    1   Linux native
/dev/dasd/0193/part2      701   1100       400    2   Linux native
                         1101   1499       399        unused
```

5. Change the type of a partition:

```
Command (m for help): t

Disk /dev/dasd/0193/device:
     100 cylinders,
      15 tracks per cylinder,
      12 blocks per track
    4096 bytes per block
volume label: VOL1, volume identifier: 0X0193
maximum partition number: 3

              -----------tracks----------
              Device   start   end   length   Id   System
/dev/dasd/0193/part1       2   700      699    1   Linux native
/dev/dasd/0193/part2     701  1100      400    2   Linux native
                        1101  1499      399        unused

change partition type
partition id (use 0 to exit):
```

Enter the ID of the partition you want to change; in this example partition 2:

```
partition id (use 0 to exit): 2
```

6. Enter the new partition type; in this example type 2 for swap:

```
current partition type is: Linux native

    1 Linux native
    2 Linux swap

new partition type: 2
```

7. Check the result:

```
Command (m for help): p


Disk /dev/dasd/0193/device:
     100 cylinders,
      15 tracks per cylinder,
      12 blocks per track
    4096 bytes per block
volume label: VOL1, volume identifier: 0X0193
maximum partition number: 3

              -----------tracks----------
              Device   start   end   length   Id   System
/dev/dasd/0193/part1       2   700      699    1   Linux native
/dev/dasd/0193/part2     701  1100      400    2   Linux swap
                        1101  1499      399        unused
```

8. Write the results to disk using the `w` option:

```
Command (m for help): w
writing VTOC...
rereading partition table...
[root@host /root]#
```

Results:

You can check this in Linux by listing the directory `/dev/dasd/0193/` (in this case). After all changes have been written to disk the new device nodes should appear in the device file system. The first entry represents the whole disk, and the following entries represent one partition each.

```
[root@host /root]# ls -l /dev/dasd/0193/
total 0
brw-------   1    root    root    94,   12 May 2 2001 device
brw-------   1    root    root    94,   12 May 2 2001 disc
brw-------   1    root    root    94,   13 May 2 2001 part1
brw-------   1    root    root    94,   14 May 2 2001 part2
[root@host /root]#
```

Example using options

You can partition using the **-a** or **-c** option without entering the menu mode. This is useful for partitioning using scripts, if you need to partition several hundred DASDs, for example.

With the **-a** parameter you can create one large partition on a DASD:

```
bash-2.04# fdasd -a /dev/dasd/0193/device
auto-creating one partition for the whole disk...
writing volume label...
writing VTOC...
rereading partition table...
bash-2.04#
```

This will create a partition as follows:

```
              Device      start      end   length   Id  System
  /dev/dasd/0193/part1        2     1499     1498    1  Linux native
```

Using a configuration file you can create several partitions. For example, the configuration file `config` below will create three partitions:

```
[first,500]
[501,1100]
[1101,last]
```

Submitting the command with the -c option creates the partitions:

```
bash-2.04# fdasd -c config /dev/dasd/0193/device
parsing config file 'config'...
writing volume label...
writing VTOC...
rereading partition table...
bash-2.04#
```

This will create partitions as follows:

```
              Device      start      end   length   Id  System
  /dev/dasd/0193/part1        2      500      499    1  Linux native
  /dev/dasd/0193/part2      501     1100      600    2  Linux native
  /dev/dasd/0193/part3     1101     1499      399    3  Linux native
```

osasnmpd – Start OSA-Express SNMP subagent

Purpose

The OSA-Express SNMP[10] subagent `osasnmpd` supports management information bases (MIBs) provided by OSA-Express Fast Ethernet, Gigabit Ethernet, High Speed Token Ring, and ATM Ethernet LAN Emulation features in QDIO mode only. The subagent extends the capabilities of the ucd-snmp master agent (snmpd) and therefore cannot run without having this package installed on a Linux system. The subagent communicates with the ucd-snmp master agent via the AgentX protocol.

The structure of the MIBs may change when updating the OSA-Express microcode to a newer level. New MIBs may even be introduced by the microcode itself. Whenever MIBs change in the future, however, an update of the subagent is not needed, because it receives the MIB objects supported by an OSA-Express feature from the feature itself.

The subagent is compatible with the ucd-snmp 4.2.x package. More information about the ucd-snmp project can be found at

```
http://net-snmp.sourceforge.net
```

Notes:

1. ucd-snmp/net-snmp is an Open Source project hosted under SourceForge.net, which is owned by the Open Source Development Network, Inc. (OSDN).
2. This subagent capability via the OSA-Express feature is sometimes referred to in non-Linux zSeries and S/390 environments as ″Direct SNMP″, which distinguishes it from another method of accessing OSA SNMP data via OSA/SF, a package for monitoring and managing OSA features that does not run on Linux. This is the first Linux implementation of the subagent function.

Usage

Prerequisites

- ucd-snmp package 4.2.x (recommended 4.2.3 or higher)
- Linux OSA-Express (QDIO) device driver (qeth)
- OSA-Express feature running in QDIO mode
- Latest OSA-Express microcode level

Once the ucd-snmp master agent (snmpd) is started on a Linux system, it binds to a port (default 161) and awaits requests from SNMP management software. Upon receiving a request, it processes the request(s), collects the requested information and/or performs the requested operation(s), and returns the information to the sender.

When the `osasnmpd` subagent is started, it retrieves the MIB objects of the OSA-Express features currently present on the Linux system and registers them with the master agent. If an OID is requested that belongs to a subagent, the master agent passes the SNMP request to that subagent, which is then responsible for handling this OID and returning the response data to the master agent. In case of OSA-Express MIB features, the request is routed to the OSA-Express SNMP subagent.

10. Simple Network Management Protocol

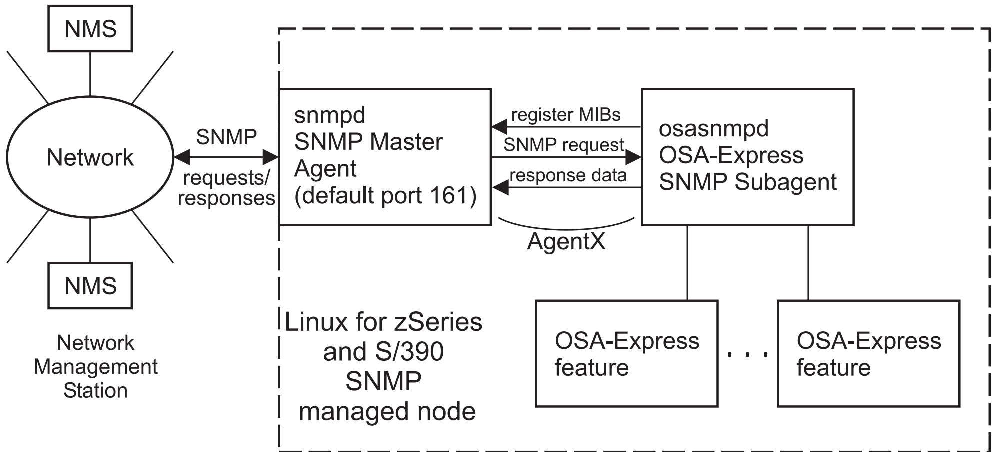


Figure 8. OSA-Express SNMP agent flow

Processes running after the ucd-snmp master agent and the OSA-Express subagent are started:

```
ps -ef | grep snmp

USER       PID
root       687     1  0 11:57 pts/1    00:00:00 snmpd
root       729   659  0 13:22 pts/1    00:00:00 osasnmpd
root       730   729  0 13:22 pts/1    00:00:00 osasnmpd
root       731   730  0 13:22 pts/1    00:00:00 osasnmpd
```

where:

[PID 687]
is the SNMP master agent

[PID 729]
is the OSA-Express SNMP subagent main thread. Awaits incoming requests from the SNMP master agent.

[PID 730]
is the OSA-Express SNMP subagent manager thread.

[PID 731]
is the OSA-Express SNMP subagent update MIB thread. It detects any network interface changes on a Linux machine and refreshes the supported MIB objects.

How to stop the OSA-Express subagent

The subagent can be stopped by sending either a SIGINT or SIGTERM signal to the main thread. For example:

```
killall osasnmpd
kill    [PID of subagent main thread]
```

Note: Do not use kill -9 or kill -SIGKILL to stop the subagent under normal circumstances. This will prevent the OSA-Express MIB objects from being unregistered by the SNMP master agent and may cause problems when restarting the subagent.

If the subagent is killed, the master agent must be restarted as well. On the other hand, if the master agent is restarted and any subagents are still running, the newly started master agent will not look for existing subagents. They must also be restarted in this case.

Communication between the master agent and subagent

The communication between the master agent and subagent is done via the AgentX (Agent eXtensibility) protocol. AgentX support is compiled into the ucd-snmp master agent by default as of version 4.2.2, but it must be explicitly enabled. Therefore, add or uncomment the directive

```
master agentx
```

in the subagent control section of the snmpd.conf configuration file.

Note: Due to restrictions imposed by ucd-snmp, there is no recovery of communication between snmpd and osasnmpd if snmpd is killed and restarted while the osasnmpd daemon is still running. In this case, kill and restart osasnmpd as well.

Configuring access control

During subagent startup or when network interfaces are added or removed, the subagent has to query MIB objects from the interfaces group of the standard MIB-II via SNMP. Therefore the access control must be set up properly to guarantee read access for the subagent on the localhost to the standard MIB-II. Otherwise, the subagent will not start. Access control directives go into the file snmpd.conf as well. Normally this file contains an access control section for that purpose. For example, when using the simpler wrapper directives instead of the view-based access control model (vacm), there should be a line like

```
rocommunity public 127.0.0.1 or rocommunity public localhost
```

or, when using the vacm model:

```
...
com2sec mylocal 127.0.0.1 public
group   mygroup v2c mylocal
...
```

in the access control section.

Note: A basic setup for the ucd-snmp configuration files snmpd.conf and snmp.conf can be created using the Perl script utility

```
snmpconf -g basic_setup
```

Downloading the IBM OSA-Express MIB

This section explains how to add the IBM OSA-Express MIB to the powerful ucd-snmp command line tools (snmpget, snmpset, snmptranslate, etc.). You might want to do this if you would like to deal with textual OIDs instead of numerical OIDs, for example. First, you must download the appropriate OSA-Express MIB from the Internet (when available, the URL will be provided via the DeveloperWorks site).

Rename the MIB file to 'IBM-OSA-MIB.txt', for example. Then place the MIB into an appropriate ucd-snmp MIB directory and tell the tools to load the MIB. For example:

```
cp IBM-OSA-MIB.txt /usr/share/snmp/mibs (default MIB path)
```

and then

```
export MIBS="$MIBS:{path}/mibs/IBM-OSA-MIB"
```

or add a line to the {path}/snmp.conf file:

```
mibs +IBM-OSA-MIB
```

where {path} could be

```
/usr/local/share/snmp          [if compiled from ucd-snmp source package]
/etc/snmp/ or /usr/share/snmp   [if vendor-supplied ucd-snmp rpm package]
```

The MIB file can have any name, but if you add it in this way, the name must match that in the definition line in the MIB file, for example:

```
==>IBM-OSA-MIB DEFINITIONS ::= BEGIN
```

See also the FAQ (How do I add a MIB to the tools?) for the ucd-snmp package at `http://net-snmp.sourceforge.net/FAQ.html`

Note: This step is not essential to correctly set up the OSA-Express subagent on your system, and most of the ucd-snmp tools will work without having a reference to the textual MIB (apart from snmptable).

Format

Start the OSA-Express SNMP subagent

```
>>--osasnmpd--+-------------+-----------------------------------------><
              '-| OPTIONS |-'
```

OPTIONS:

```
|--+-------------------------------------------------------------+--|
   +--+----------------+--+----+--+----+--+--------------+--------+
   |  +- -l --logfile-+  '-A -'  '-f -'  '-P --pidfile--'         |
   |  '- -L ----------'                                           |
   +- -h ---------------------------------------------------------+
   '- -v ---------------------------------------------------------'
```

-l ***logfile***
: Specifies a file to log all messages/warnings from the subagent (including stdout/stderr) to *logfile*. If a path is not entered, the file is created in the current directory.

  If this option is not entered, logging is nevertheless done to file `/var/log/osasnmpd.log`.

-L
: Print messages and warnings to stdout/stderr.

-A
: Append to logfile rather than truncating it.

-f
: Do not fork() from the calling shell.

-P ***pidfile***
: Save the process ID of the subagent in *pidfile*. If a path is not specified, the current directory is used.

-h
: Print usage message and exit.

-v
: Print version information and exit.

Examples

```
# cat /var/log/osasnmpd.log
IBM OSA-E ucd-snmp 4.2.x series subagent version 1.0.1
Mar 21 16:04:20 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.1.
```

```
Mar 21 16:04:20 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.2.
Mar 21 16:04:20 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.3.
Mar 21 16:04:20 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.7.
Mar 21 16:04:20 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.5.
Mar 21 16:04:20 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.4.
OSA-E microcode level is 309
Initialization of OSA-E subagent successful...
Mar 22 09:38:55 *** OSA-E interface change indicated ***
Mar 22 09:38:55 *** Reinitialized MIB information ***
#
```

From time to time, it is advisable to look at the logfile of either the master agent or the OSA-Express subagent. Warnings and messages go into these log files. The above logfile excerpt shows the messages after a successful OSA-Express subagent initialization. In the following example, several MIB tables were registered by the subagent:

```
.1.3.6.1.4.1.2.6.188.1.1 = ibmOSAExpChannelTable
.1.3.6.1.4.1.2.6.188.1.4 = ibmOSAExpEthPortTable
```

The following is an SNMP GET request for the ifDescr (interface description) MIB object from the MIB-II interfaces group using ifIndex 5. eth1 is the result string for that query. The GET request was handled by the master agent, which supports the SNMP standard MIBs.

```
# snmpget -v 2c 10.0.0.2 public interfaces.ifTable.ifEntry.ifDescr.5
interfaces.ifTable.ifEntry.ifDescr.5 = eth1
#
```

The following SNMP GET requests query ibmOsaExpEthPortName from ibmOSAExpEthPortTable for interface eth1 (ifIndex 5). This is an OID that is not supported by the master agent itself. The master agent now checks whether one of its subagents, if present, has registered for this OID. In this case, the OSA-Express subagent is the owner of that MIB object and provides data for the OID instance.

```
# snmpget -v 2c -OS 10.0.0.2 public ibmOsaExpEthPortName.5
IBM-OSA-MIB::ibmOsaExpEthPortName.5 = MIKE
#

# snmpget -v 2c -OS 10.0.0.2 public .1.3.6.1.4.1.2.6.188.1.4.1.16.5
IBM-OSA-MIB::ibmOsaExpEthPortName.5 = MIKE
#
```

qetharp - Query and purge OSA and HiperSockets ARP data

Purpose

The `qetharp` utility is used to query and purge address data such as MAC and IP addresses from the ARP cache of the OSA and HiperSockets hardware.

Format

qetharp parameters

```
>>--qetharp--+------------------------+--------------------------><
             +-+------+-- -q --interface-+
             | '- -n -'                  |
             +- -p --interface-----------+
             +- -v ----------------------+
             '- -h ----------------------'
```

The meanings of the parameters of this command are as follows:

-q or --query
: Shows the address resolution protocol (ARP) information found in the ARP table of the OSA or HiperSockets, which depends on *interface*. If it is an OSA device, it shows the ARP entries stored in the OSA feature's ARP table, otherwise, the ones from the HiperSockets ARP table. If the IP address is an IPv4 address, qetharp tries to determine the symbolic host name. If it fails, the IP address will be shown. In case of IPv6, there is currently no attempt to determine host names, so that the IP address will be shown directly.

-n or --numeric
: Shows numeric addresses instead of trying to determine symbolic host names. This option can only be used in conjunction with the -q option.

-p or --purge
: Flushes the ARP table of the OSA, causing the hardware to regenerate the addresses. This option works only with OSA devices. qetharp returns immediately.

-v or --verbose
: Shows version information and exits

-h or --help
: Shows usage information and exits

Examples

- Show all ARP entries of the OSA defined as eth0:

  ```
  qetharp -q eth0
  ```

- Show all ARP entries of the OSA defined as eth0, without resolving host names:

  ```
  qetharp -nq eth0
  ```

- Flush the OSA's ARP table for eth0:

  ```
  qetharp -p eth0
  ```

snIPL – Remote control of Support Element (SE) functions

Purpose

`snIPL` (**s**imple **n**etwork **IPL**) is an interactive tool used for remotely controlling Support Element (SE) functions. It allows you to:
- Boot Linux for zSeries in LPAR mode.
- Send and retrieve operating system messages.
- Deactivate an LPAR for I/O fencing purposes.

Using `snIPL`, you overcome the limitations of the SE graphical interface when `snIPL` is used for I/O fencing from within a clustered environment of Linux systems that run in LPAR mode.

`snIPL` uses the network management application programming interface (API) which:
1. establishes an SNMP network connection.
2. uses the SNMP protocol to send and retrieve data.

For details, refer to *zSeries Application Programming Interfaces*, SB10-7030, which you can obtain from the following Web site:

```
http://www.ibm.com/server/resourcelink/
```

> **Note!**
> `snIPL` currently supports a direct connection to the SE, and does not yet support direct communication with the Hardware Management Console (HMC).

Usage

Prerequisites:

You must configure the SE to allow the initiating host system to access the SE API. For details of how to do so, refer to the *Application Programming Interfaces* book.

Attention: Careless use of `snIPL` can result in loss of data.

Overview of functionality:
To communicate with the SE, `snIPL` uses the application programming interface provided by the HMC to:
1. establish an SNMP network connection.
2. use the SNMP protocol to send and retrieve data.

Connection Errors and Exit Codes:
If a connection error occurs (such as a timeout or communication failure):
1. `snIPL` sends to `stderr`:
   - the error code of the management API
   - a message
2. The shell exit code is set to 99.

If no error occurs, a shell exit code of 0 is returned after `snIPL` has completed its processing.

Format

Command line syntax

snIPL command

```
>>--snipl--+----------------------------------------+--ipaddr--------><
           +- -u -----------------------------------+
           |  .- -c --public-----.                  |
           +--+------------------+------------------+
           |  '- -c --community--'                  |
           +- -n ----LPARname--+--------------------+
           |                   +- -i --|OS Msgs dialog|-+
           |                   '- -d ---------------'   |
           |  .- -t --10000-----.                   |
           +--+-----------------+-------------------+
           |  '- -t --timeout---'                   |
           |  .- -p --5000-------.                  |
           '--+------------------+------------------'
              '- -p --interval---'
```

Parameters:

-u
Display the usage and exit.

-c *community*
Specify the *community* (which is the HMC term for password) of the initiating host system. The default for *community* is `public`. The value entered here must match the entry contained in the SNMP configuration settings on the SE.

-n *LPARname*
Specify the name of the target LPAR (also referred to as a CPC image). If you do not specify the **–n** option, `snIPL` displays the *LPAR dialog* in which the LPARs it can find are listed. If you again do not enter a value for *LPARname* but instead press **Ctrl-D**, the *Command dialog* is displayed.

-i
Start an *Operating System Messages dialog*, which you can then use to enter your own commands (this option is used together with **-n**). These commands are read from `stdin` and then sent to the target LPAR by the management API. `snIPL` also initiates a background thread (using pthreads) which continuously retrieves operating system messages as they are generated. The output of the polling thread is sent to `stdout`. To terminate the *Operating System Messages dialog*, press **Ctrl-D**. `snIPL` will then exit.

Note: The *Operating System Messages dialog* is *line-oriented* and therefore use of a screen-based editor is not recommended.

-d
Instruct `snIPL` to issue a deactivate command for the target LPAR, and then to exit (this option is used together with **-n**).

-t *timeout*
Specify the *timeout* in milliseconds, for management API calls. The default is 10000.

-p *interval*
Specify the *interval* in milliseconds, for management API calls that retrieve operating system messages. The default is 5000.

ipaddr Specify the IP address of the target Support Element (SE). In order to successfully establish a connection (using a valid *community*), you must configure in the SE's *SNMP configuration* task, the:
- IP-address of the initiating system
- *community*.

In addition, you must activate SNMP support in the SE's *Settings* task.

If `snIPL` repeatedly returns the error code 19 (timeout), the target SE is probably
- not reachable, or
- incorrectly configured.

Processing

snIPL menu

You call `snIPL` using the syntax described in "Command line syntax" on page 158.

1. If you specify **–n** together with a *valid* value for *LPARname*, `snIPL` displays the *Command dialog* (shown below).
2. If you specify **–n** together with an *invalid* value for *LPARname*, `snIPL` re-displays the *LPAR dialog*.
3. If you do *not* specify the **–n** option and a value for *LPARname*, `snIPL` displays the *LPAR dialog*.
   a. If you then press **Ctrl-D**, `snIPL` displays the *Command dialog*.
   b. Otherwise, you can select an LPAR, and press Enter. `snIPL` then proceeds to the *Command dialog*.

```
n select LPAR image
i operating system messages interaction
l perform a load
d perform a deactivate
m print this menu
x exit
```

4. Now you can select an option from the *Command dialog*.

Option	Result
n	The *LPAR dialog* is re-displayed, and you can select another LPAR.
i	The *Operating System Messages dialog* is displayed. To abort the *Operating System Messages dialog*, you again press **Ctrl-D**. The polling thread then terminates after the last management API call that was issued has returned. This means that, in the worst case, the polling thread quits after waiting for the number of milliseconds contained in *interval*.
l	A Linux for zSeries system is started in the target LPAR. You are then prompted to enter: • load address • load parameters • clear indicator • timeout value • store status indicator These arguments are similar to those which you enter directly at the SE. If you press **Ctrl-D**, a previously entered default value will be re-used.

You are prompted to confirm that the parameters you specified are correct. Then the respective `load` command is issued.

d Causes a deactivate command to be issued on the target LPAR.

m The *Command dialog* is printed to the screen.

x `snIPL` exits.

Restrictions when using snIPL:

- `snIPL` does *not* recover from:
  - connection failures
  - errors in API call execution.

  In both of these situations, you should simply restart `snIPL`.

  However, if the problem persists, a networking failure has probably occurred. In this situation you should try to increase the `timeout` values.
- `snIPL` does acknowledge that a load command has been accepted by the management API on the SE. However you must also check whether or not the load command has completed successfully. For example, `snIPL` cannot determine if an incorrect load address has been used as input.
- Currently, `snIPL` only supports direct communication with the SE.

Examples

This section provides a typical sequence of commands for using `snIPL`, where entering **Ctrl-D**, a previously-entered default is used.

1. The user selects option **l** to perform a load.
2. These parameters are specified:
```
Load address (as XXXX in HEX): 5119
Load parameter: CTRL-D
Clear indicator (0/1): 1
Timeout: 60
Store status indicator (0/1): CTRL-D
```
3. The parameters that were selected are now confirmed:
```
Load address: 5119
Load parameter: <default>
Clear indicator: 1
Timeout: 60s
Store status indicator: <default>
```
4. The user is then prompted to confirm that a LOAD command should be performed:
```
Perform a LOAD command on partition MYLPAR with these
parameters? (y/n) y
```
5. The LOAD command is processed, and an acknowledgement sent when completed:
```
processing.... acknowledged.
Command (m for help):
```

zIPL – zSeries initial program loader

Purpose

Because of changes in the DASD disk layout for kernel 2.4, the boot utility `SILO` is replaced by the zSeries initial program loader (`zIPL`) for Linux for zSeries.

`zIPL` can be used to prepare a device for two purposes:
- For booting Linux (as a Linux program loader)
- For dumping.

Usage

Prerequisites:
- The Linux kernel version must be equal to or later than 2.4.3.
- You should have the `parsecfg` package from your Linux distributor.

  For more details see the `Readme.zipl` document in the kernel source tree at `...linux/Documentation/s390`.

zIPL supports devices as shown in Table 6.

Table 6. Supported devices

As a boot loader	As a dump tool
ECKD[1] DASDs with fixed block AIX-compliant layout	ECKD DASDs with fixed block AIX-compliant layout
ECKD DASDs with z/OS-compliant layout	ECKD DASDs with z/OS-compliant layout
Fixed Block Access (FBA) DASDs	Magnetic tape subsystems compatible with IBM3480, IBM3490, or IBM3590.
VM minidisks using DIAG250 on ECKD hardware	
VM minidisks using DIAG250 on FBA hardware	
[1]Enhanced Count Key Data	

`zIPL` uses configuration data that it retrieves from:
1. The command line. Settings at the command line override settings in the configuration file.

   Exception: Parameter settings in the configuration file override parmfile settings on the command line.
2. The configuration file if it is accessible. The default configuration file is `/etc/zipl.conf`. You can also use an environment variable, `ZIPLCONF`, to specify the configuration file.

For every operation a target directory must be specified, either on the command line or in the configuration file. This directory must contain all `zIPL` boot-loader files and must be accessible as read-write. `zIPL` reads the boot loaders from the target directory, and might create the boot-map file and temporary device nodes there.

You use the `image` parameter to tell `zIPL` to install a boot loader. You use the `-d` (or `--dumpto`) parameter to tell `zIPL` to create a dump device. If none or both of these options are set (on either the command line or in the configuration file), `zIPL` will issue an error message and exit.

Creating a boot loader

Prerequisite: If the boot loader is to be installed on an AIX-compliant ECKD, FBA, or VM minidisk, the corresponding stage 2 boot loaders `eckd2`, `fba2`, or `diag2` must reside on the same device as the target directory. This is normally the case if the target directory does not contain symbolic links to other devices.

To use `zIPL` as a boot loader, you must specify the following either on the command line or in the configuration file:

1. The `image` parameter.

   If you specify an optional address on the `image` parameter, the kernel image will be loaded to that address. The default address 0x10000 is used otherwise.

   The image, and if specified, the ramdisk and the parmfile, must reside on the same device, otherwise an error is reported.
2. The `target` parameter.

   This specifies the target directory in which the boot loader will be installed. This means that you do not need to specify the device on which the boot loader should be installed. The device on which the target directory resides will be automatically detected and used.
3. The location of the boot parameters or the parmfile. You can do this in several ways:
   - You can set parameters under the ipl section in the configuration file by using `parameters=`. A parmfile will be created in the target directory, and will be loaded at IPL to the default address 0x1000.

     Note: This overrides any parmfile setting both on the command line and in the configuration file.
   - You can specify the name of a parmfile by using `parmfile=` in the configuration file, or by using `-p` or `--parmfile=` on the command line. If a parmfile is specified and no parameter setting is present, the parmfile will be loaded to either the default address 0x1000 or the address specified.

Creating a dump device

Prerequisite: A partition must be available to `zIPL`. Any existing data on the partition will be lost.

To create a dump device with `zIPL`, you must specify the following parameters either on the command line or in the configuration file:

1. The partition on which the dump device should be created with the `dumpto` parameter. The partition is used to automatically define the device. `zIPL` will delete all data on this partition and install the `zIPL` boot loader here.

   Notes:

   a. If the dump device is a ECKD disk with fixed-block layout (AIX-compliant), the corresponding stage 2 dump utility must be located on the same device as the one selected using the `dumpto` parameter. The dump utility will be overwritten by the dump, and you must re-install it in order to take another dump.

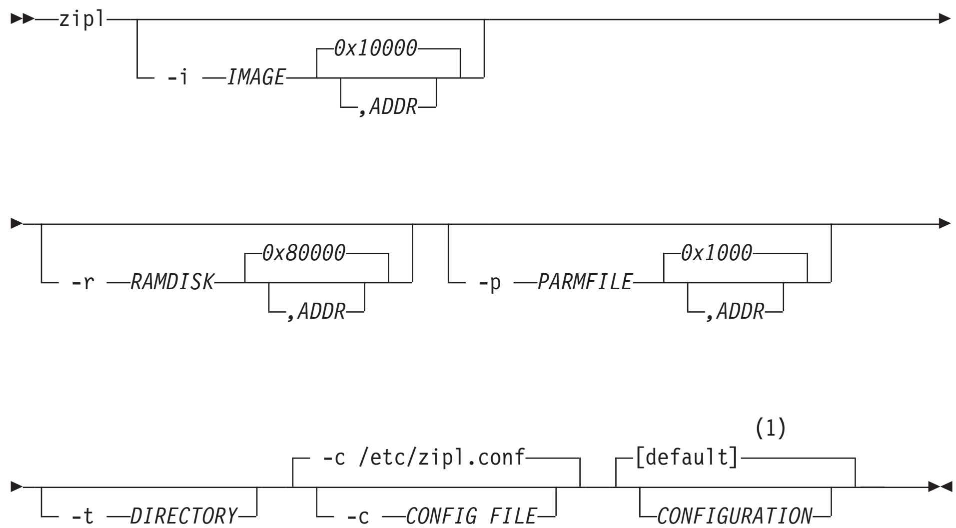
   b. If the dump device is a disk with z/OS-compliant layout, the corresponding stage 2 dump utility can reside in a (possibly very small) partition, and the dump can go into another, larger partition. Here you do not need to re-install the dump utility after every dump.

2. The target directory that contains the boot loader, for example `eckd2dump`, with the `target` parameter.

Format

Command line syntax

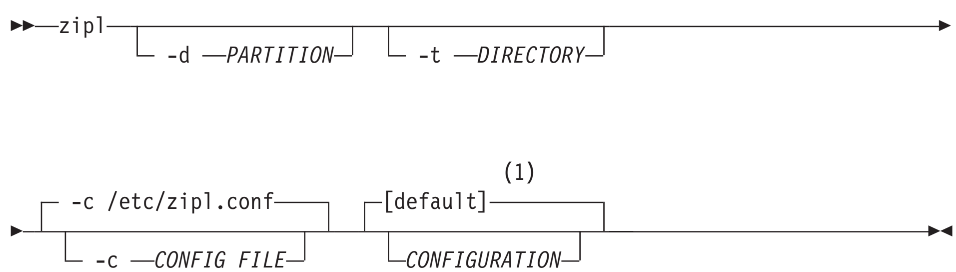
zIPL boot command



Notes:

1 If no configuration name is given, zIPL will use the configuration specified in the `defaultboot` section of the configuration file.

zIPL dump command



Notes:

1 If no configuration name is given `zIPL` will use the configuration specified in the `defaultboot` section of the configuration file.

Where:

-c *CONFIG FILE* **or --config=***CONFIG FILE*
: Specifies the name of the configuration file. This overrides the default `/etc/zipl.conf` as well as the environment variable `ZIPLCONF`.

CONFIGURATION
: Selects a section from the configuration file to be read.

-i *IMAGE[,ADDRESS]* **or --image=***IMAGE[,ADDRESS]*
: Specifies the location of the Linux system kernel on the file system as well as in memory after IPL. The default address is 0x10000.

-r *RAMDISK[,ADDRESS]* **or --ramdisk=***RAMDISK[,ADDRESS]*
: Specifies the location of the initial ramdisk image (initrd) on the file system as well as in memory after IPL. The default address is 0x80000.

-p *PARMFILE[,ADDRESS]* **or --parmfile=***PARMFILE[,ADDRESS]*
: Specifies the location of the parameter file on the file system as well as in memory after IPL. The default address is 0x1000.

-d *PARTITION* **or --dumpto=***PARTITION*
: Specifies the partition on which the dump will be located after IPL.

-t *DIRECTORY* **or --target=***DIRECTORY*
: Specifies the target directory where zIPL finds boot loaders, for example, eckd2dump, as well as various other temporary and permanent files. For example the parmfile will be created in this directory.

-h or --help
: Displays help on command line arguments.

Configuration file syntax

Location: By default `zIPL` retrieves the configuration file from `/etc/zipl.conf`. You can set an environment variable, `ZIPLCONF`, which overrides the default setting. If the name of the configuration file is given on the command line, this setting overrides both the default setting and the environment variable.

Configuration file sections:

defaultboot
: The configuration file can contain a default boot section. If no configuration is specified on the command line, the `defaultboot` section is accessed and the default configuration section is activated.

Example:

```
[defaultboot]
default=myconfig
```

configuration
: A section describing a configuration always has the same name as the configuration name which is specified in the `defaultboot` section or on the command line.

Example:

```
[myconfig]
```

A section describing a configuration can contain one or more of the following options:

target=*DIRECTORY*
: Specifies the directory where `zIPL` finds boot loaders, for example `eckd2dump`, as well as various other temporary and permanent files.

image=*IMAGE[,ADDRESS]*
: Specifies the location of the Linux system kernel on the file system as well as in memory after IPL. The default address is 0x10000

ramdisk=*RAMDISK[,ADDRESS]*
: Specifies the location of the initial ramdisk image (`initrd`) on file system as well as in memory after IPL. The default address is 0x80000.

parmfile=*PARMFILE[,ADDRESS]*
: Specifies the location of the parameter file on the file system as well as in memory after IPL. The default address is 0x1000.

parameters='*PARAMETERS***'**
: Specifies the parameters for the kernel. Surround the parameter list with either quotation marks or apostrophes. For example, if you need to use quotation marks within the parameter list, you can surround the parameter list with apostrophes. Setting this parameter causes a file called ″parmfile″ containing the given parameters to be created in the target directory. At IPL that file will be loaded to the default address 0x1000.

Note: This setting overrides parmfile settings both on the command line and in the configuration section. Any existing parmfile will be overwritten if zIPL creates a new parmfile from the parameter settings.

dumpto=*PARTITION*
: Specifies the partition on which the dump will be located after IPL.

See "Examples" for an example configuration file.

Examples

Example `zIPL` command: This section shows how to use `zIPL` from the command line, equivalently to `SILO`. The following example shows how to install a boot loader with an image and a parmfile.

```
zipl --target=/mnt/boot --image=/mnt/boot/image --parmfile=/mnt/boot/parmfile
```

The parmfile could look like this:

```
root=/dasda1 ro dasd=fd00
```

Results:

1. `zIPL` will issue a warning ″No configuration file found″. This can be ignored.
2. `zIPL` will install the boot loader on the device of the target directory.

Example configuration file: This example configuration file has two configurations. The first one (the default) is called `ipl` and installs a boot loader. The second is called `dumptape` and prepares a tape device, /dev/rtibm0, for dumping.

```
[defaultboot]
default=ipl
[ipl]
target=/boot
image=/boot/image
ramdisk=/boot/initrd
```

```
parameters='root=/dev/ram0 ro'
[dumptape]
target=/boot
dumpto=/dev/rtibm0
```

Example zIPL commands with configuration file: This section shows the effects some commands would have assuming that the configuration file above is used.

- Calling zIPL to use the default configuration file settings:

  ```
  zipl
  ```

  Result: zIPL reads the default option from the defaultboot section and selects section ipl. Then it installs a boot loader that will load /boot/image, /boot/initrd and /boot/parmfile. A parmfile will be created with the following contents:

  'root=/dev/ram0 ro'
- Calling zIPL to create a dump tape:

  ```
  zipl dumptape
  ```

  Result: zIPL selects section dumptape and prepares a dump tape on /dev/rtibm0.
- Calling zIPL to create a boot loader with another image rather than the one in the configuration file:

  ```
  zipl --image=/boot/otherimage
  ```

  Result: zIPL reads the default option from defaultboot and selects section ipl. Then it installs a boot loader that will load /boot/otherimage, /boot/initrd and /boot/parmfile. A parmfile will be created with the following contents:

  'root=/dev/ram0 ro'

ifconfig - Configure a network interface

Usage

`ifconfig` is used to configure the kernel-resident network interfaces. It is used at startup time to set up interfaces as necessary. After that, it is usually only needed when debugging or when system tuning is needed.

- If no arguments are given, `ifconfig` displays the status of the currently active interfaces.
- If a single interface argument is given, it displays the status of the given interface only
- Otherwise, it configures an interface.

Notes:

1. Since kernel release 2.2, there are no longer explicit interface statistics for alias interfaces. The statistics printed for the original address are shared with all alias addresses on the same device. If you want per-address statistics you should add explicit accounting rules for the address using the `ipchains` command.
2. See also "QETH restrictions" on page 126 for information on potential failures when setting IP addresses on interfaces associated with OSA-Express in QDIO mode (QETH driver).

Format

Display active interface status

```
►►──ifconfig──────────────────────────────────────────────────►◄
```

Display status of given interface

```
►►──ifconfig──interface───────────────────────────────────────►◄
```

Activate or shut down an interface

```
►►──ifconfig──┬────────────────┬──┤ OPTIONS ├──┬─────────┬──────────────►◄
              └─address_family─┘               └─address─┘
```

OPTIONS:

```
                             (1)            (1)
├──interface──┬─up───┬──┬───────────┬──┬─────────┬──┬─────────────────┬──►
              └─down─┘  └─metric──N─┘  └─mtu──N──┘  └─netmask─address─┘

►──┬──────────────────────┬──┬──────────────────────┬──┬──────────────┬──►
   └─add─addr/prefixlen───┘  └─del─addr/prefixlen───┘  │           (2)│
                                                       └─multicast────┘

     (1)
►──┬─────────────────────┬───────────────────────────────────────────────┤
   └─txqueuelen──length──┘
```

Notes:

1. Distribution dependent
2. This should not normally be needed as the drivers set the flag correctly themselves.

address_family
: If the first argument after the interface name is recognized as the name of a supported address family, that address family is used for decoding and displaying all protocol addresses.

  On zSeries, supported address families include:
  - inet

interface
: The name of the interface. This is usually a driver name followed by a unit number, for example eth0 for the first Ethernet interface.

  The configurable interfaces are:
  - `ctc`*i* (*i* = 0 to 255)
  - `escon`*i* (*i* = 0 to 255)
  - `eth`*i* (*i* = 0 to 255)
  - `hsi`*i* (*i* = 0 to 255)
  - `iucv0`
  - `lo`
  - `tr`*i* (*i* = 0 to 255)

up This flag causes the interface to be activated. It is implicitly specified if an address is assigned to the interface.

down This flag causes the driver for this interface to be shut down.

metric N
This parameter sets the interface metric.[11]

mtu N This parameter sets the maximum transfer unit (MTU) of an interface. See "QETH restrictions" on page 126 for information on the maximum MTU size for OSA-Express interfaces via QDIO (QETH driver).

netmask addr
Set the IP network mask for this interface. This value defaults to the usual class A, B or C network mask (as derived from the interface IP address), but it can be set to any value.

add addr/prefixlen
Add an IPv6 address to an interface. *prefixlen* is the number of leading contiguous bits set in the network mask.

This option can be given only once per invocation.

del addr/prefixlen
Remove an IPv6 address from an interface. *prefixlen* is the number of leading contiguous bits set in the network mask.

This option can be given only once per invocation.

multicast
Set the multicast flag on the interface. This should not normally be needed as the drivers set the flag correctly themselves.

address
The IP address to be assigned to this interface.

txqueuelen length
Set the length of the transmit queue of the device. It is useful to set this to small values for slower devices with a high latency (modem links, ISDN) to prevent fast bulk transfers from disturbing interactive traffic like Telnet too much.

Files:

```
/proc/net/socket
/proc/net/dev
```

11. Metric: Cost of an OSPF interface. The cost is a routing metric that is used in the OSPF link-state calculation. To set the cost of routes exported into OSPF, configure the appropriate routing policy.
    - Range: 1 through 65,535
    - Default: 1

    All OSPF interfaces have a cost, which is a routing metric that is used in the OSPF link-state calculation. Routes with lower total path metrics are preferred over those with higher path metrics. When several equal-cost routes to a destination exist, traffic is distributed equally among them. The cost of a route is described by a single dimensionless metric that is determined using the following formula:

    ```
    cost = ref-bandwidth / bandwidth
    ```

    Where `ref-bandwidth` is the reference bandwidth. Its default value is 100 Mbps (which you specify as 100000000), which gives a metric of 1 for any bandwidth that is 100 Mbps or greater.

Examples

To start or modify an ESCON interface in Linux:

```
ifconfig escon0 x.x.x.x pointopoint y.y.y.y netmask 255.255.255.255 mtu mmmmm
```

where:

x.x.x.x is the IP address of the Linux side

y.y.y.y is the IP address of the remote partner z/OS

mmmmm
is the MTU size which could be up to 32760 - make sure the other side of the channel uses the same MTU size

The ESCON CTC device addresses are defined in the kernel parameter file, or when loading the module:

```
...... ctc=0,0xddd,0xyyy,escon0
```

Another example, showing how to set up an Ethernet connection:

```
ifconfig eth0 192.168.100.11 netmask 255.255.255.0 broadcast 192.168.100.255 mtu 1492 up
```

or, simply:

```
ifconfig eth0 192.168.100.11
```

insmod - Load a module into the Linux kernel

Usage

`insmod` installs a loadable module in the running kernel. It tries to link a module into the kernel by resolving global symbols in the module with values from the kernel's symbol table. If the object file name is given without extension, `insmod` will search for the module in common default directories. The environment variable MODPATH can be used to override this default.

Format

Load a module into the kernel

```
>>--insmod--+-------------+--+-------------+--object_file------------>
            '-| OPTIONS |-'  '-module_name-'

   .-----------------------------------------.
   V                  (1)              (1)   |
>----symbol-- = --'--"--------value--"--'----+------------------------><
```

OPTIONS:

```
|--+----+--+----+--+----+--+----+--+----+--+----+--+----+--+----+------|
   '-f -'  '-k -'  '-m -'  '-p -'  '-s -'  '-x -'  '-X -'  '-v -'
```

Notes:

1 Both quotes are needed to prevent the shell from stripping the quote characters from the parameter. This can also be accomplished using the escape character '\'.

object_file
: Name of module source file. (Name by which module is invoked.)

module_name
: Explicit name of module if not derived from the name of the source file.

symbol Name of parameter specific to module.

value Value of parameter to be passed to module.

-f Attempt to load the module even if the version of the running kernel and the version of the kernel for which the module was compiled do not match.

-k Set the auto-clean flag on the module. This flag will be used to remove modules that have not been used in some period of time (usually one minute).

-m Output a load map, making it easier to debug the module in the event of a kernel panic.

-p Probe the module to see if it could be successfully loaded. This includes locating the object file in the module path, checking version numbers, and resolving symbols.

-s Output everything to `syslog` instead of the terminal.

-X Export the external symbols of the module.

-x Do not export the external symbols of the module.

-v Verbose mode.

Examples

DASD module

```
insmod dasd_mod dasd=192-194,5a10
```

XPRAM module

```
insmod xpram devs=4 sizes=2097152,8388608,4194304,2097152
```

CTC module

```
insmod ctc setup=\"ctc=0,0x0600,0x0601,ctc0\"
```

LCS module

```
insmod lcs additional_model_info=0x70,3,0x71,5
           devno_portno_pairs=0x1c00,0,0x1c02,1,0x1d00,-1
```

QDIO module

```
insmod qdio
```

OSA Express module

```
insmod qeth qeth_options=noauto,0x400,0x401,0x402,0x200,0x201,0x202,secondary_router
```

modprobe - Load a module with dependencies into the Linux kernel

Usage

`modprobe` installs a loadable module and all its dependencies in the running kernel. The dependency information is created by `depmod` (see “depmod - Create dependency descriptions for loadable kernel modules” on page 177). It tries to link a module into the kernel by resolving global symbols in the module with values from the kernel’s symbol table. If there are still unresolved symbols it will try to satisfy these by loading further modules. If the object file name is given without extension, `modprobe` will search for the module in common default directories. The environment variable `MODPATH` can be used to override this default.

Format

Load a module with dependencies into the kernel

```
                            ┌─-C ──/etc/modules.conf─┐
►►──modprobe──┬─────────────┬┼────────────────────────┼──module_name──────►
              └─┤ OPTIONS ├─┘└─-C ──configfile────────┘

   ┌──────────────────────────────────────┐
   ▼                    (1)           (1) │
►──symbol── = ──’──"──────value──"──’─────┴──────────────────────────────►◄
```

OPTIONS:

```
├──┬────┬──┬────┬──┬────┬──┬────┬──┬────┬──┬────┬──┬────┬──────────┤
   └─-a─┘  └─-d─┘  └─-k─┘  └─-n─┘  └─-q─┘  └─-s─┘  └─-v─┘
```

Notes:

1 Both quotes are needed to prevent the shell from stripping the quote characters from the parameter. This can also be accomplished using the escape character ’\’.

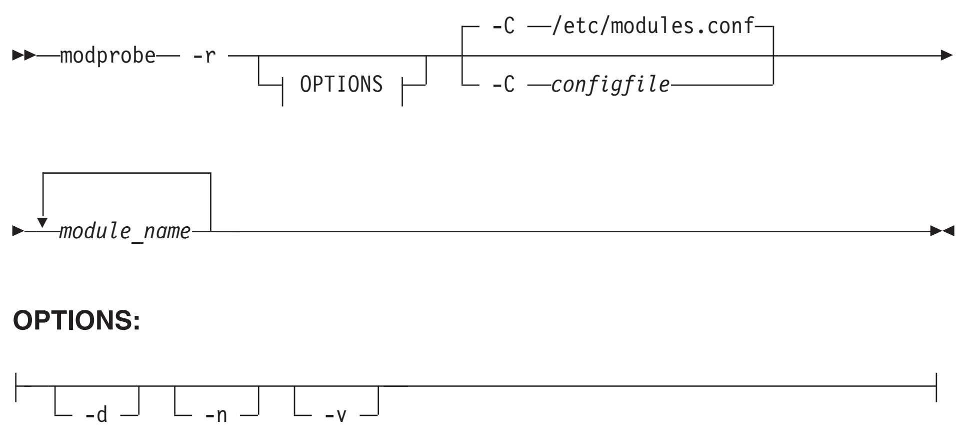
List matching modules

```
                   ┌─-C ──/etc/modules.conf─┐
►►──modprobe── -l ─┼────────────────────────┼──┬───────────────┬─pattern────►◄
                   └─-C ──configfile────────┘  └─-t ──type─────┘
```

Show configuration

```
                   ┌─-C ──/etc/modules.conf─┐
►►──modprobe── -c ─┼────────────────────────┼──────────────────────────────►◄
                   └─-C ──configfile────────┘
```

Remove module (stacks) or do autoclean



The parameters for the `modprobe` command are:

module_name
: Explicit name of module. (Name by which module is invoked.)

symbol Name of parameter specific to module.

value Value of parameter to be passed to module.

pattern Module name pattern, which may include wildcard characters.

-a Load all matching modules (default is to stop after first success).

-d Show debugging information.

-k Set the auto-clean flag on the module. This flag will be used to remove modules that have not been used in some period of time (usually one minute).

-n Probe the module to see if it (and its dependencies) could be successfully loaded, but do not load the module.

-q Suppress error messages.

-s Send the report to `syslog` instead of `stderr`.

-t Only consider modules of type <*type*>.

-v Verbose mode.

Comments

Note that this list is not comprehensive. See `man modprobe` for information on the full set of parameters.

Examples

OSA-Express module

```
modprobe qeth
```

This will attempt to load the `qeth` network driver. It will find a dependency on `qdio`, load the `qdio` base support automatically, and then load `qeth`.

lsmod - List loaded modules

Usage

lsmod lists all loaded modules in the running kernel.

Format

list modules

```
►►──lsmod──────────────────────────────────────────────────────────►◄
```

Examples

```
# lsmod
Module                  Size  Used by
qeth                  135680   1  (autoclean)
qdio                   22992   1  (autoclean) [qeth]
#
```

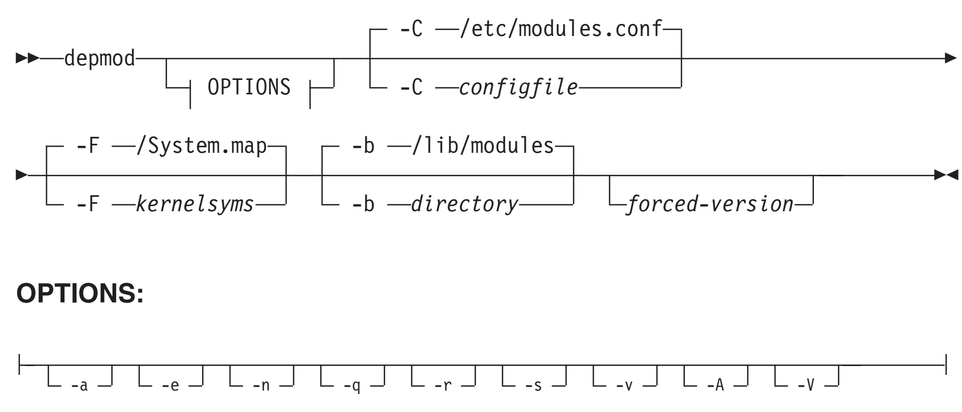
depmod - Create dependency descriptions for loadable kernel modules

Usage

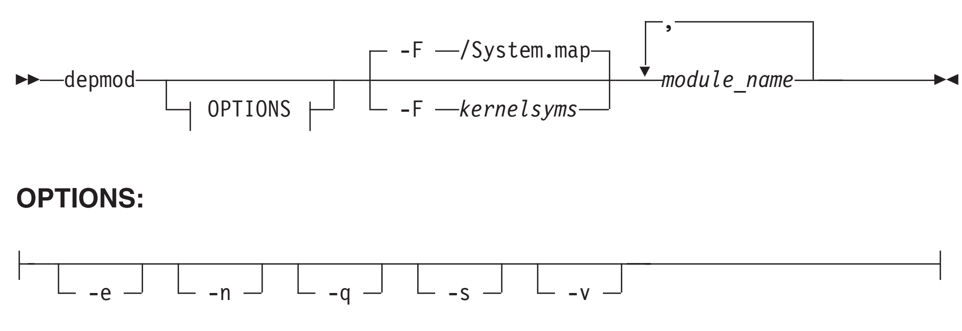
`depmod` creates a dependency file based on the symbols found in a set of modules. This information is used by `modprobe` (qv).

Format

Create dependencies for all files in a directory



Create dependencies for files



module_name
: Explicit name of module

-a Search for modules in all directories specified in `/etc/modules.conf` or <*`configfile`*>.

-b Use `/lib/modules` or <*`directory`*> as the starting point for the subtree containing the modules.

-e Show all the unresolved symbols for each module.

-n Write the dependency file on `stdout` instead of in the `/lib/modules` tree.

-q (quiet) Suppress error messages about missing symbols.

-s Write all error messages via the `syslog` daemon instead of `stderr`.

-v Show the name of each module as it is analyzed.

-A Like `depmod -a`, but compare file timestamps and only update the dependency file if anything has changed.

-C Use the file <*configfile*> instead of `/etc/modules.conf`.

-F Use the symbol information found in <*kernelsyms*>.

-V Show the release version of `depmod`.

Examples

```
depmod -e -n mymodule
```

displays the unresolved references in *mymodule* on `stdout`.

mke2fs - Create a file system on DASD

Usage

This utility creates an ext2 file system on a DASD hard disk. The device must already have a low-level format.

Format

mke2fs syntax

```
            ┌─-b 4096──────────┐
►►──mke2fs──┼──────────────────┼──device──────────────────────────────►◄
            └─-b ──blocksize───┘
```

-b *blocksize*
: Specifies the blocksize. Default is 4096. The blocksize needs to be a multiple of the blocksize specified on the **dasdfmt** command. This allows you to use block sizes up to the hardware maximum of 4096.

device
: Specifies the device node.

Examples

```
mke2fs -b 4096 /dev/dasd/<devno>/part<x>
```

vconfig - VLAN (802.1Q) configuration program

Usage

The `vconfig` program allows you to create and remove VLAN devices on a VLAN-enabled kernel. VLAN devices are virtual Ethernet devices that represent the virtual LANs on the physical LAN.

Format

Configure a VLAN

```
►►──vconfig──┬─add──interface──vlan_id────────────────────────────┬────────►◄
             ├─rem──vlan_device───────────────────────────────────┤
             ├─set_flag──vlan_device──┬─0─┬───────────────────────┤
             │                        └─1─┘                       │
             ├─set_egress_map──vlan_device──skb_priority──vlan_qos─┤
             ├─set_ingress_map──vlan_device──skb_priority──vlan_qos┤
             │                ┌─DEV_PLUS_VID_NO_PAD──┐             │
             └─set_name_type──┼─VLAN_PLUS_VID────────┼─────────────┘
                              ├─VLAN_PLUS_VID_NO_PAD─┤
                              └─DEV_PLUS_VID─────────┘
```

interface
: interface name on which the VLAN is to be created

vlan_id
: Number (1-4095) of the resulting VLAN. Also referred to as VID. Depending on the setting of name type, this results in creation of a new device with a name bearing that number (*vlan_device*).

vlan_device
: Device name assigned to the VLAN with the 'add' option.

set_flag
: The value 1 enables Ethernet header reordering. When dumping the device, it will appear as a common Ethernet device without VLANs. The value 0 disables Ethernet header reordering, resulting in VLAN-tagged packets when dumping the device.

set_egress_map
: Specifies that outbound packets with a particular socket buffer priority (*skb_priority*) should be tagged with the particular VLAN priority *vlan_qos*. The default VLAN priority is 0.

set_ingress_map
: Specifies that inbound packets with a particular VLAN priority (*vlan_qos* should be tagged with the particular socket buffer priority (*skb_priority*). The default socket buffer priority is 0.

set_name_type
: Defines the way VLAN ID names are created.

VLAN_PLUS-VID
: vlan*vlan_id* with leading zeroes (e.g., 'vlan0005')

VLAN_PLUS_VID_NO_PAD
: vlan*vlan_id* without leading zeroes (e.g., 'vlan5')

DEV_PLUS_VID
interface.vlan_id with leading zeroes (e.g., ’eth0.0005’)

DEV_PLUS_VID_NO_PAD
interface.vlan_id without leading zeroes (e.g., ’eth0.5’)

Note: VLAN uses Broadcom’s NICE interface when the network device supports it. This is necessary because the hardware of these devices removes the VLAN tag from the Ethernet packet. The set_flag option on VLAN devices created on such a physical network device will be ignored. Dumping the network device will show only untagged (non-VLAN) traffic, and dumping the VLAN devices will show only traffic intended for that VLAN, without the tags.

Examples

Refer to Chapter 17, “Virtual LAN (VLAN) support” on page 191.

Chapter 15. VIPA – minimize outage due to adapter failure

This chapter describes how you use VIPA (Virtual IP Address) to assign IP addresses to a *system*, instead of to *individual adapters*. Using VIPA, you can minimize outage caused by adapter failure.

Notes:

1. The VIPA functionality requires a kernel built with the CONFIG_DUMMY option.
2. See the information in "QETH restrictions" on page 126 concerning possible failure when setting IP addresses for OSA-Express features in QDIO mode (QETH driver).

There are two types of VIPA described here:

- Standard VIPA is usually sufficient for applications, such as Web Server, that do *not* open connections to other nodes.
- Source VIPA can be used for applications that open connections to other nodes.

Standard VIPA

Purpose

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

Usage

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

1. Create a dummy device using the Linux `insmod` command.
2. Assign a *virtual IP address* to the dummy device.
3. Ensure that your service (for example, the Apache Web server) listens to the virtual IP address assigned above.
4. 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 9 on page 184).
   - Dynamic routing. For details of how to configure routes, you must refer to the documentation delivered with your *routing daemon* (for example, `zebra` or `gated`).

If outage of an adapter occurs, you must *switch adapters*.

- To do so under static routing, you should:
  1. Delete the route that was set previously.
  2. Create an alternative route to the virtual IP address.
- To do so under dynamic routing, you should refer to the documentation delivered with your *routing daemon* for details.

Example

This example assumes static routing is being used, and shows you how to:

1. Configure VIPA under static routing.

2. Switch adapters when an adapter outage occurs.

Figure 9 shows the network adapter configuration used in the example.

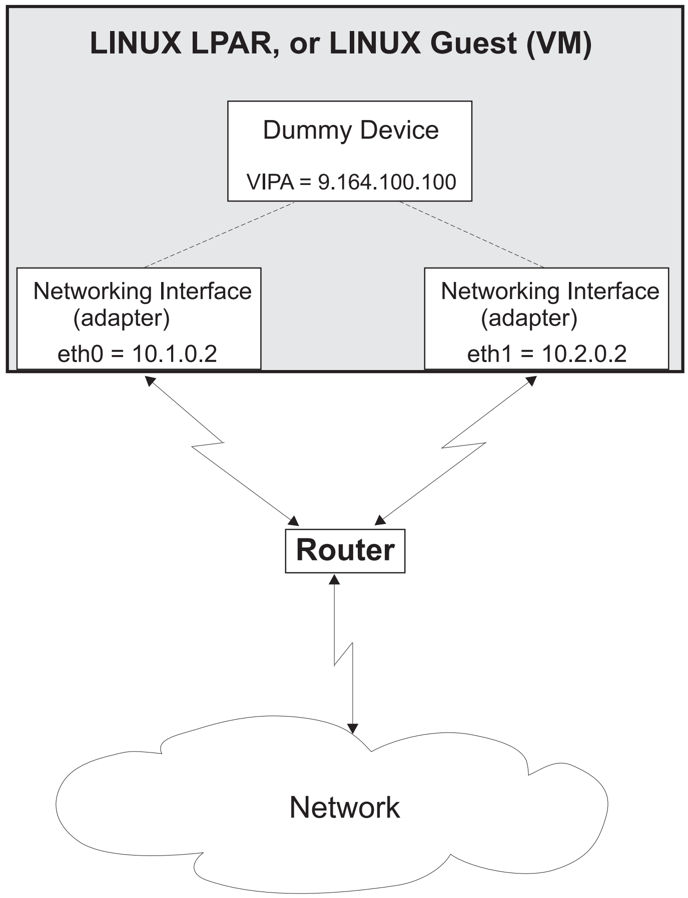


Figure 9. Example of using Virtual IP Address (VIPA)

1. Create a dummy device.
   ```
   insmod dummy
   ```
2. Assign a virtual IP address (9.164.100.100) to the dummy device.
   ```
   ifconfig dummy0 9.164.100.100
   ```
3. Enable the network devices for this VIPA so that it accepts packets for this IP address. This is necessary only on OSA-Express devices in QDIO mode.
4. Ensure that your service (such as the Apache Web server) listens to the virtual IP address.
5. Set up a route to the virtual IP address (static routing), so that VIPA can be reached via the gateway with address 10.1.0.2.
   ```
   route add -host 9.164.100.100 gw 10.1.0.2
   ```

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

1. Delete the previously-created route.
   ```
   route delete -host 9.164.100.100
   ```
2. Create the alternative route to the virtual IP address.
   ```
   route add -host 9.164.100.100 gw 10.2.0.2
   ```

3. Enable the VIPA on the network devices:

```
echo add_vipa4 09a46464:eth0 >/proc/qeth_ipa_takeover
echo add_vipa4 09a46464:eth1 >/proc/qeth_ipa_takeover
```

where 09a46464 is the IP address in hexadecimal format. For IPv6, the address is specified in IPv6 format:

```
add_vipa6 2002::1234:5678:eth0 >/proc/qeth_ipa_takeover
add_vipa6 2002::1235:5678:eth1 >/proc/qeth_ipa_takeover
```

where `2002::1234:5678` is equivalent to entering `20020000000000000000000012345678`.

Source VIPA

Purpose

Source VIPA provides a functionality specially required in high-performance environments. Normally, IP packets are tagged with the IP address of the adapter through which they leave the system. In case of an adapter failure, IP packets cannot be traced back and thus become "lost" in the network. With Source VIPA, they are tagged with the VIPA instead. Usually, applications have had to provide source VIPA support on their own, or the kernel had to provide Source VIPA functionality like the AIX or MVS kernel does. With the Source VIPA package, Source VIPA can be used selectively on a per-application basis. This is done by dynamically linking the Source VIPA shared object file to the application to be Source-VIPA-enabled before linking the runtime library to it. This will catch some socket calls and tweak them to yield a Source VIPA effect. The result is a per-application configuration of Source VIPA.

The Source VIPA solution does not affect kernel stability. Source VIPA is controlled by a configuration file containing flexible rules for when to use Source VIPA, based on destination IP address ranges.

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

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

```
ip route x src y
```

Usage

Installation

```
make
make starter
make install
```

Paths can be changed in the Makefile. Defaults are:

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

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

Configuration

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

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

`D1.D2.D3.D4/MASK` specifies a range of destination addresses and the number of bits set in the subnet mask (MASK). As soon as a socket is opened and connected to these destination addresses and the application does not do an explicit bind to a source address, src_vipa does a bind to S1.S2.S3.S4. Instead of IP addresses in dotted notation, hostnames can be used and will be resolved using DNS.

.INADDR_ANY P1-P2 S1.S2.S3.S4, or the same command with only one port P, will cause bind calls with INADDR_ANY as a local address to be intercepted if the port the socket is bound to is between P1 and P2 (inclusive). In this case, INADDR_ANY will be replaced by S1.S2.S3.S4 (which can be 0.0.0.0). All .INADDR_ANY statements will be read and evaluated in order of appearance. This means that two .INADDR_ANY statements can be used to have bind be intercepted for every port outside a certain range. This is useful, for example, for `rlogin`, which uses `bind` to bind to a local port but INADDR_ANY as a source address to use automatic source address selection.

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

Enabling an application

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

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

Example

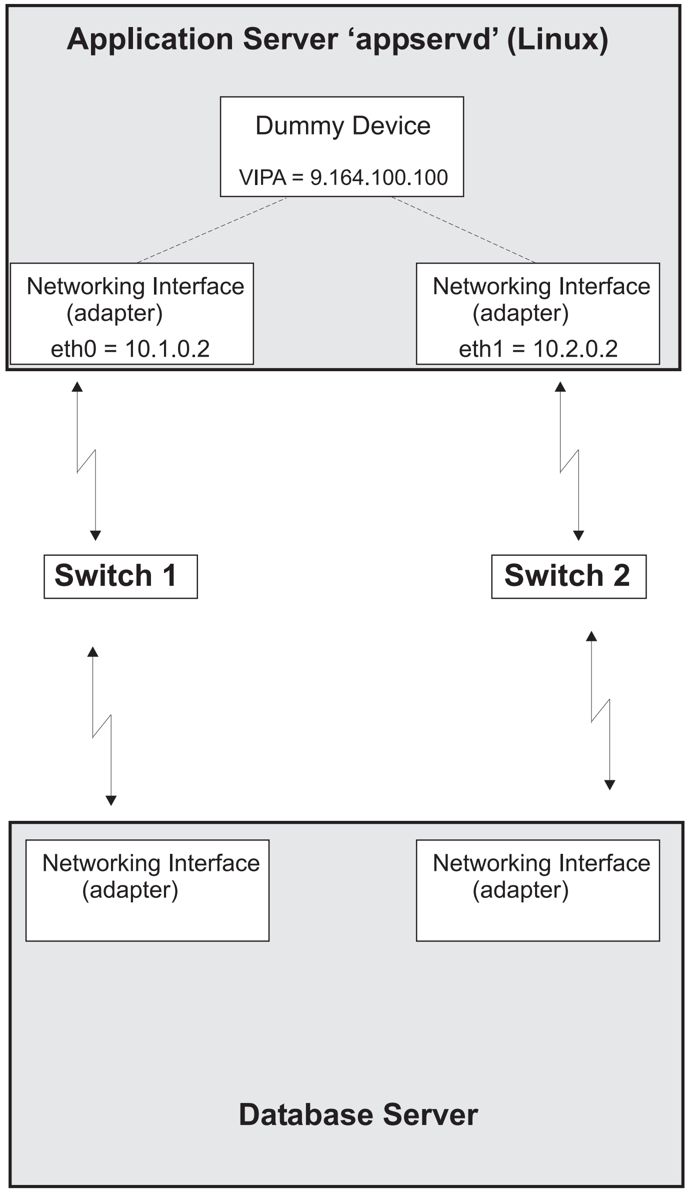


Figure 10. Example of using source VIPA

The command:

```
src_vipa.sh appservd start
```

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

Chapter 16. VARY ON/OFF events and toggling CHPIDs online/offline

This chapter describes the notification provided to drivers after VARY ON/OFF events and the toggling of CHPIDs logically on- or offline under Linux.

Response to VARY ON/OFF events

Certain VARY ON/OFF events on which information can be obtained via a channel subsystem call are made known to device drivers. This includes link incidents and resource accessibility events. These events are made known to Linux via a machine check, and thus the machine check handler is enabled to process CRWs from the channel subsystem (CSS).

Note: Processing of CSS machine checks is done for native and LPAR modes only. The code will not be triggered under VM, since VM does not pass CSS machine checks through.

Device drivers are called in a manner similar to that when devices on a subchannel become attached/detached, but only when the last path to a device is gone (physically or logically) or a formerly inaccessible device has become accessible. This is achieved by calls to

- nopfunc() for devices becoming unavailable
- oper_func() for devices becoming available

Device driver writers should be prepared to handle link incidents and resource accessibility events.

Toggling CHPIDs logically on and offline

If the kernel option CONFIG_CHSC has been set to 'Y', an interface in the /proc file system is exported that allows the user to

- see the current status of CHPIDs
- control the logical status of CHPIDs from a Linux perspective (i.e., set them online/offline) via piping commands to the /proc interface. This function requires root authorization.

Usage

/proc/chpids shows the current status of all known CHPIDs as online or logically offline. The following is sample output immediately after IPL:

```
bash-2.04# cat /proc/chpids
80 online
8C online
9C online
FD online
```

To set a known CHPID logically online/offline, issue the command

```
echo {on|off} <chpid> > /proc/chpids
```

An error message is generated if

- the CHPID is unknown to Linux, or

- the CHPID is already in the desired state.

Note: Only CHPIDs associated with a subchannel and physically online (not VARY'd offline) will appear in /proc/chpids.

Example

In the following example, CHPID '9C' is set logically offline:

```
bash-2.04# echo off 9c > /proc/chpids

bash-2.04# cat /proc/chpids
80 online
8C online
9C logically offline
FD online
```

Chapter 17. Virtual LAN (VLAN) support

VLAN technology works according to IEEE Standard 802.1Q by logically segmenting the network into different broadcast domains so that packets are switched only between ports designated for the same VLAN. By containing traffic originating on a particular LAN to other LANs within the same VLAN, switched virtual networks avoid wasting bandwidth, a drawback inherent in traditional bridged/switched networks where packets are often forwarded to LANs that do not require them.

Building a Linux kernel with VLAN and OSA-Express support is a prerequisite for using VLAN under Linux. VLAN support is integrated into the Linux kernel as of kernel version 2.4.14.

Introduction to VLANs

VLANs increase traffic flow and reduce overhead by allowing you to organize your network by traffic patterns rather than by physical location. In a conventional network topology, such as that shown in the following figure, devices communicate across LAN segments in different broadcast domains using routers. Although routers add latency by delaying transmission of data while using more of the data packet to determine destinations, they are preferable to building a single broadcast domain, which could easily be flooded with traffic.

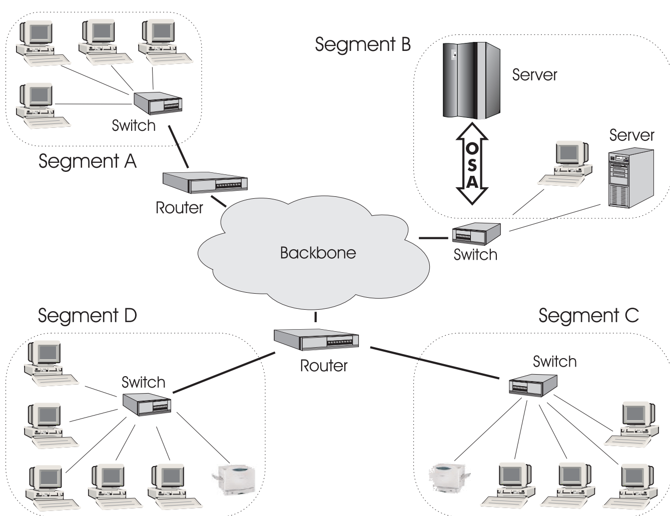


Figure 11. Conventional routed network

By organizing the network into VLANs through the use of Ethernet switches, distinct broadcast domains can be maintained without the latency introduced by multiple routers. As the following figure shows, a single router can provide the interfaces for all VLANs that appeared as separate LAN segments in the previous figure.

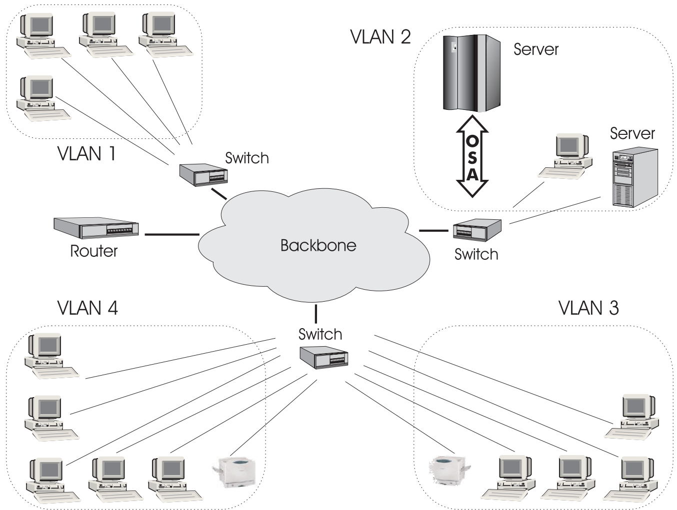


Figure 12. Switched VLAN network

The following figure shows how VLANs can be organized logically, according to traffic flow, rather than being restricted by physical location. If workstations 1-3 communicate mainly with the small server, VLANs can be used to organize only these devices in a single broadcast domain that keeps broadcast traffic within the group. This reduces traffic both inside the domain and outside, on the rest of the network.

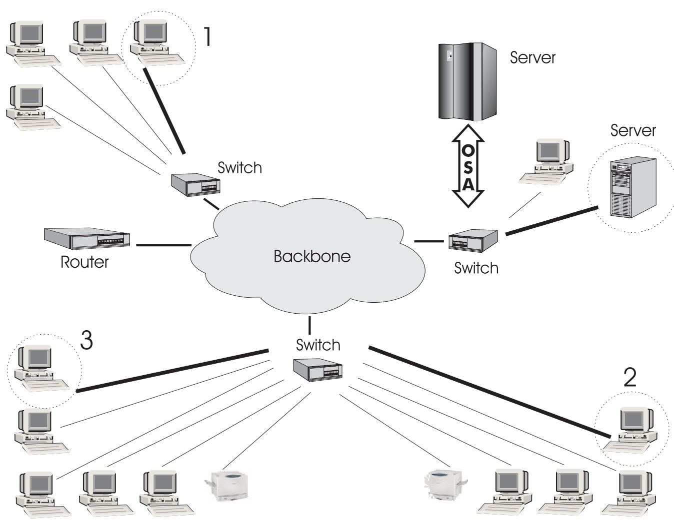


Figure 13. VLAN network organized for traffic flow

Configuring VLAN devices

VLANs are configured using the 'vconfig' command (see "vconfig - VLAN (802.1Q) configuration program" on page 180).

Information on the current VLAN configuration is available by listing the files in
`/proc/net/vlan/*`

with `cat` or `more`. For example:

```
bash-2.04# cat /proc/net/vlan/config
VLAN Dev name    | VLAN ID
Name-Type: VLAN_NAME_TYPE_RAW_PLUS_VID_NO_PAD  bad_proto_recvd: 0
eth2.100       | 100  | eth2
eth2.200       | 200  | eth2
eth2.300       | 300  | eth2
bash-2.04# cat /proc/net/vlan/eth2.300
eth2.300  VID: 300       REORDER_HDR: 1  dev->priv_flags: 1
         total frames received:     10914061
          total bytes received:   1291041929
      Broadcast/Multicast Rcvd:            6

      total frames transmitted:     10471684
       total bytes transmitted:   4170258240
            total headroom inc:            0
           total encap on xmit:     10471684
Device: eth2
INGRESS priority mappings: 0:0  1:0  2:0  3:0  4:0  5:0  6:0 7:0
EGRESS  priority Mappings:
bash-2.04#
```

Examples

VLANs are allocated in an existing interface representing a physical Ethernet LAN. The following example creates two VLANs, one with ID 3 and one with ID 5.

```
ifconfig eth1 9.164.160.23 netmask 255.255.224.0 up
vconfig add eth1 3
vconfig add eth1 5
```

The vconfig commands have added interfaces ″eth1.3″ and ″eth1.5″, which you can then configure:

```
ifconfig eth1.3 1.2.3.4 netmask 255.255.255.0 up
ifconfig eth1.5 10.100.2.3 netmask 255.255.0.0 up
```

The traffic that flows out of eth1.3 will be in the VLAN with ID=3 (and will not be received by other stacks that listen to VLANs with ID=4).

The internal routing table will ensure that every packet to 1.2.3.x goes out via eth1.3 and everything to 10.100.x.x via eth1.5. Traffic to 9.164.1xx.x will flow through eth1 (without a VLAN tag).

To remove one of the VLAN interfaces:

```
ifconfig eth1.3 down
vconfig rem eth1.3
```

The following example illustrates the definition and connectivity test for a VLAN comprising five different Linux systems, each connected to a physical Gigabit Ethernet LAN via eth1:

```
(LINUX1: LPAR 64bit)
     vconfig add eth1 5
     ifconfig eth1.5 10.100.100.1 broadcast 10.100.100.255 netmask 255.255.255.0 up


(LINUX2: LPAR 31bit)
     vconfig add eth1 5
     ifconfig eth1.5 10.100.100.2 broadcast 10.100.100.255 netmask 255.255.255.0 up


(LINUX3: VM Guest 64bit)
     vconfig add eth1 5
     ifconfig eth1.5 10.100.100.3 broadcast 10.100.100.255 netmask 255.255.255.0 up
```

```
(LINUX4: VM Guest 31bit)
     vconfig add eth1 5
     ifconfig eth1.5 10.100.100.4 broadcast 10.100.100.255 netmask 255.255.255.0 up


(LINUX5: Intel)
     vconfig add eth1 5
     ifconfig eth1.5 10.100.100.5 broadcast 10.100.100.255 netmask 255.255.255.0 up

Test the connections:

     ping 10.100.100.[1 - 5]          // Unicast-PING
     ping -I eth1.5 224.0.0.1         // Multicast-PING
     ping -b 10.100.100.255           // Broadcast-PING
```

Further information

More information on VLAN for Linux is available at

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

Chapter 18. Kernel parameters

There are two different ways of passing parameters to Linux :

- Passing parameters to your kernel at startup time. (The parameter line)
- Configuring your boot loader to always pass those parameters.

The kernel can only handle a parameter line file that is no larger than 896 bytes.

The parameters which affect Linux for zSeries in particular are:

- cio_ignore
- cio_msg
- cio_notoper_msg
- ipldelay
- maxcpus
- mem
- noinitrd
- ramdisk_size
- ro
- root
- vmhalt

cio_ignore

Usage

Specifies one or more device numbers or ranges of device numbers that will be ignored by the common I/O layer; no detection or device sensing will be done on any of these devices. The subchannel to which the device in question is attached will be treated as if no device were attached.

The device numbers must be specified as hexadecimal.

Devices that are not already known can later be ignored, or the "ignored" state for known devices reset, by piping commands to the `/proc/cio_ignore` interface. For more information, see the file `Documentation/s390/CommonIO` in the kernel source tree.

Format

cio_ignore syntax

```
                      ┌─,───────────────────────────────┐
                      ▼                                 │
►►──cio_ignore── = ────device or range of device numbers┴──────────────►◄
```

Examples

This example specifies that all devices in the range 0x23 to 0x42, and the device number 4711, if detected, are to be ignored:

```
cio_ignore=0x23-0x42,0x4711
```

cio_msg

Usage

Specifies whether I/O messages are to be sent to the console on boot-up.

These messages are usually suppressed (`cio_msg=no`) because on large machines with many attached devices the I/O layer generates a large number of these messages which can flood the console for a significant period of time. If you do need those messages (for example for debugging) you can switch them on manually using `cio_msg=yes`.

Format

cio_msg syntax

```
►►──cio_msg── = ──┬─ no ──┬──────────────────────────────────►◄
                  └─ yes ─┘
```

Examples

This example switches I/O messages to the console on boot:

```
cio_msg=yes
```

cio_notoper_msg

Usage

Specifies whether messages of the type "Device 4711 became 'not operational'" should be shown during startup. After startup, they are always shown.

These messages are usually displayed (`cio_notoper_msg=yes`).

Format

cio_notoper_msg syntax

```
>>──cio_notoper_msg── = ──┬─ no ──┬────────────────────────────────────><
                          └─ yes ─┘
```

Examples

This example suppresses 'not operational' messages on startup:

```
cio_notoper_msg=no
```

ipldelay

Usage

When you do a power on reset (POR), some activation and loading is done. This can cause Linux not to find the OSA-2 card. If you have problems with your OSA-2 card after booting, you might want to insert a delay to allow the POR, microcode load and initialization to take place in the OSA-2 card. The recommended delay time is two minutes. For example, `30s` means a delay of thirty seconds between the boot and the initialization of the OSA-2 card, `2m` means a delay of two minutes. The value `xy` must be a number followed by either `s` or `m`.

Format

ipldelay syntax

```
>>--ipldelay-- = --+- xy m -+------------------------------------------><
                   '- xy s -'
```

Examples

This example delays the initialization of the card by 2 minutes:

```
ipldelay=2m
```

This example delays the initialization of the card by 30 seconds:

```
ipldelay=30s
```

maxcpus

Usage

Specifies the maximum number of CPUs that Linux can use.

Format

maxcpus syntax

```
►►──maxcpus── = ──number──────────────────────────────────────────────►◄
```

Examples

```
maxcpus=2
```

mem

Usage

Restricts memory usage to the size specified. This must be used to overcome initialization problems on a P/390.

Format

mem syntax

```
►►──mem = ──┬─xx──M─────┬──────────────────────────────────────────►◄
            └─yyyyyy──K─┘
```

Examples

```
mem=64M
```

Restricts the memory Linux can use to 64MB.

```
mem=123456K
```

Restricts the memory Linux can use to 123456KB.

noinitrd

Usage

The `noinitrd` statement is required when the kernel was compiled with initial RAM disk support enabled. This command bypasses using the initial ramdisk.

This can be useful if the kernel was used with a RAM disk for the initial startup, but the RAM disk is not required when booted from a DASD

Format

noinitrd syntax

```
►►──noinitrd──────────────────────────────────────────────────────►◄
```

ramdisk_size

Usage

Specifies the size of the ramdisk in kilobytes.

Format

ramdisk_size syntax

```
►►──ramdisk_size── = ──size──────────────────────────────────────────►◄
```

Examples

```
ramdisk_size=32000
```

ro

Usage

Mounts the root file system read-only.

Format

ro syntax

```
▶▶──ro──────────────────────────────────────────────────────────▶◀
```

root

Usage

Tells Linux what to use as the root when mounting the root file system.

Format

root syntax

```
►►──root── = ──rootdevice──────────────────────────────────────────►◄
```

Examples

Without devfs this makes Linux use /dev/dasda1 when mounting the root file system:

```
root=/dev/dasda1
```

With devfs this could be:

```
root=/dev/dasd/0182/part1
```

vmhalt

Usage

Specifies a command to be issued after a shutdown on VM.

Format

vmhalt syntax

```
►►──vmhalt── = ──COMMAND──────────────────────────────────────────►◄
```

Examples

This example specifies that an initial program load of CMS should follow a shutdown on VM:

```
vmhalt="IPL CMS PARM AUTOCR"
```

AUTOCR avoids having to manually respond to each line.

Note: The command must be entered in uppercase.

Chapter 19. Overview of the parameter line file

The parameter line file contains kernel parameters which are read by Linux during the boot process. This chapter describes the format of the parameters in this file.

Parameters

Comments

The parameters allowed in the parameter line are described in Chapter 18, “Kernel parameters” on page 195 and/or together with the description of the device drivers.

Usage

The parameter line file contains data to be passed to the kernel for evaluation at startup time. The location from which the kernel reads this file varies with the IPL method as shown below:

IPL method	Location of parameter line file
DASD	Installed into the boot sector using `zipl` (option `–p`)
Tape	Second file on tape
VM reader	Second file in reader
CD-ROM	Third entry in the `.ins` file (with load address 0x00010480)

Format

The kernel parameter file consists of a single line containing at most 896 bytes. The line may be encoded in either ASCII or EBCDIC. It contains a list of kernel options (see kernel parameters, device driver parameters) separated by blanks.

For IPL from a VM reader the kernel parameter file must be broken into fixed length records of 80 bytes. Note that a record end does not separate two options. Therefore if an option ends at the end of a record the next record should begin with a blank character.

Examples

Here is an example of a parameter line file:

```
dasd=E0C0-E0C2 root=/dev/ram0 ro ipldelay=2m
```

This defines three DASD, a read-only root file system, and a two-minute delay to allow network connection.

Note that when loading from tape using an ASCII encoded parameter file (such as one generated on a UNIX or PC system) you must make sure that your parameter file does not span more than one line, is not larger than 896 bytes, and contains no special characters (for example tabs or new lines).

Appendix A. Reference information

LCS parameter syntax

The LCS channel device layer boot parameters are as follows:

chandev=
lcs

lcsnum	Device number.
lcs_read_devno	Read channel address.
lcs_write_devno	Write channel address.
memory_usage_in_k	Total buffer size to allocate.
relative_adapter_no	Relative adapter number.
checksum_received_ip_pkts	Perform checksum on inbound packets.
use_hw_stats	Get network statistics from the `LANSTAT LCS` primitive.

LCS module parameter syntax (without the channel device layer)

The following are the LCS device driver module parameters:

use_hw_stats	Get network statistics from the `LANSTAT LCS` primitive.
do_sw_ip_checksumming	Perform checksum on inbound packets.
ignore_sense	Boot devices which do not report a valid `sense_id`
additional_model_info	Model/maximum relative adapter number pairs.
devno_portno_pairs	Matching pairs of device numbers and port numbers.
noauto	`noauto=1` disables auto-detection.

For a description of the parameters see “Device identification (CTC/ESCON and LCS)” on page 71.

OSA-Express and HiperSockets parameter syntax

This driver is subject to license conditions as reflected in: “International License Agreement for Non-Warranted Programs” on page 233.

The OSA-Express and HiperSockets channel device layer boot parameters are as follows:

chandev=
qeth

osanum	Device number.
osa_read_devno	Read channel address.
osa_write_devno	Write channel address.
osa_data_devno	Data channel address.
memory_usage_in_k	Total buffer size to allocate.
port_no	Relative port number.
checksum_received_ip_pkts	Perform checksum on inbound packets.

OSA-Express and HiperSockets driver command syntax (without the channel device layer)

This driver is subject to license conditions as reflected in: “International License Agreement for Non-Warranted Programs” on page 233.

There is a single keyword parameter for the OSA-Express and HiperSockets driver:
`qeth_options`

This parameter is used as follows:

(Note that all characters must be in the case shown, except for hexadecimal numbers where case is irrelevant.)

```
qeth_options=[<driver options>,][<feature options>,[<feature options>,...]]
```

<driver options> is a comma separated list that sets the driver defaults. It can contain the following keywords:

auto	turns on auto-detection
noauto	turns off auto-detection
no_router	does not prepare the device as a router (default)
primary_router	makes the device a primary router
secondary_router	makes the device a secondary router
sw_checksumming	checksumming is to be performed by the software
hw_checksumming	checksumming is to be performed by the hardware
no_checksumming	checksumming is not to be used
prio_queueing_tos	priority queueing based on the IP type of service field
prio_queueing_prec	priority queueing based on the IP precedence field
no_prio_queueing	switch off priority queueing
no_prio_queueing: <*number*>	switch off priority queueing and set the default queue to <*number*>

<feature options> are used to override the global options for a particular device. These are also comma-separated lists and each list consists of three device numbers (decimal, or hex starting with 0x), an optional device name, and any of the driver options keywords except **auto** or **noauto**.

For a description of the parameters see "QETH parameter syntax" on page 117.

Linux for zSeries Major/Minor numbers

The major and minor numbers currently allocated to zSeries devices are:

Device	Major number	Minor numbers
DASD	94 + dynamic	0,4,8..252. – Volume, Other numbers (1...255) – Partitions
XPRAM	35	0–31

Appendix B. Kernel building

Building the kernel

Before deciding to change the details in the kernel source code, consider whether installing one of the Linux images provided in the Linux for zSeries kernel patches will be a more appropriate solution to your requirements.

Your build system must have the following software installed (as a minimum):

- kernel source 2.2.16 with the Linux for zSeries patch
- gcc version 2.95.2 or later supporting Linux for zSeries
- binutils version 2.9.1 or later supporting Linux for zSeries
- glibc 2.1.2 or later supporting Linux for zSeries
- sed
- bash
- make version 3.77 or later.

The following assumptions are made:

- You are confident in your ability to modify the kernel parameters without severely damaging the system
- You cannot locate a pre-compiled kernel image that meets your requirements (that is, a suitable kernel does not already exist)
- You are able to make an emergency IPL tape available (or preferably a complete backup on tape).

If you decide to modify your Linux for zSeries kernel, you should use the instructions outlined in the following sections. In this way you will be sure of completing all of the tasks necessary to ensure the system runs properly when you have finished. For example, you might have to install and link additional modules after you have compiled and installed the kernel.

Note: The OCO device drivers supplied by IBM have been compiled with ***chandev***, ***multicast***, ***token ring*** and ***Ethernet*** support. If you recompile the kernel without these options some or all of the IBM drivers will not work.

If you switch on *Loadable module support -> Set version information on all module symbols* you may also encounter problems, as these OCO modules are compiled without version information.

Preliminary steps

Before working with the kernel, there are a number of precautions that you should take:

- Make a backup copy of the current kernel and all modules corresponding to this kernel. It is important to make a backup even if you are replacing your kernel with a new version. This is because the new system might not run properly and you can use the backup to reload the old system
- Decide whether you want to modify the complete kernel, or only change some modules. If you only change some modules, you might not have to build the kernel.

If you are upgrading or replacing the kernel, obtain the new kernel or patch and load it into the directory `/usr/src`. This will probably create a new directory `usr/src/linux`, which will overwrite your last version of the kernel source. You will need to check the symbolic links to the `/usr/include` directory to ensure the following two links are valid:

- `ln -sf /usr/src/linux/include/linux /usr/include/linux`
- `ln -sf /usr/src/linux/include/asm /usr/include/asm`

Your first step in modifying the kernel, is to change to the `/usr/src/linux` directory and enter the command `make distclean`. This cleans up the Linux for zSeries distribution, resetting all options to their default values and removing all object files from the system. If you enter `make clean` instead of `make distclean`, you will only delete the object files.

Configuring the parameters

To configure the parameters, make sure you are in the `/usr/src/linux` directory and enter the command `make config`.

In `make config`, you select what you want to include in the resident kernel and what features you want to have available as dynamically loadable modules. You will generally select the minimal resident set that is needed to boot:

- The type of file system used for your root partition (for example, ext2)
- Normal hard disk drive support (for example, DASD)
- Network support

- TCP/IP support.

The set of modules is constantly increasing, and you will be able to select the option (M) when responding to the prompts shown in `make config` for those features that the current kernel can offer as loadable modules. You can completely enable or disable the use of your set of modules by using the `CONFIG_MODVERSIONS` option during `make config`.

`make config` requires `bash` to allow it work. Bash will be searched for in `$BASH`, `/bin/bash` and `/bin/sh` (in that order), so it must be located in one of these directories for it to work. `make config` must be performed even if you are only upgrading to the next patch. New configuration options are added in each release, and odd problems will turn up if the configuration files are not set up as expected. If you want to upgrade your existing configuration with minimal work, use `make oldconfig`, which will keep your old kernel and only ask you questions about new or modified options.

Alternative configuration commands are:
- `make menuconfig` — Text based menus, radiolists and windows
- `make oldconfig` — Same as `make config` except all questions based on the contents of your existing `./.config` file
- `make xconfig` — This X windows based configuration tool is currently not available with Linux for zSeries.

Notes on `make config`:
- Keeping unnecessary drivers in the Linux for zSeries kernel will make it bigger, and can cause problems: for example, unnecessary networking options might confuse some drivers.
- Selecting the kernel hacking option and changing the source code directly usually result in a bigger or slower Linux for zSeries kernel (or both). Thus you should probably answer (N) to the questions for development, experimental, or debugging features.

Checking the configuration

There are a pair of scripts that check the source tree for problems. These scripts do not have to be run each time you build the kernel, but it is a good idea to check for these types of errors and discrepancies at regular intervals.

`make checkconfig` checks the source tree for missing instances of `#include <linux>`. This needs to be done occasionally, because the C preprocessor will silently give bad results if these symbols haven't been included (it treats undefined symbols in preprocessor directives as defined to 0). Superfluous uses of `#include <linux>` are also reported, but you can ignore these, because smart `CONFIG_*` dependencies make them harmless. You can run `make checkconfig` without configuring the kernel. Also, `make checkconfig` does not modify any files.

`make checkhelp` checks the source tree for options that are in `Config.in` files but are not documented in `scripts/Configure.help`. Again, this needs to be done occasionally. If you have hacked the kernel and changed configuration options or are adding new ones, it is a good idea to `make checkhelp` (and add help as necessary) before you publish your patch. Also, `make checkhelp` does not modify any files.

Checking the dependencies

All of the source dependencies must be set each time you configure a new Linux for zSeries kernel.

Enter `make dep` to set up all the dependencies correctly. `make dep` is a synonym for the long form, `make depend`. This command performs two tasks:

- It computes dependency information about which `.o` files depend on which `.h` files. It records this information in a top-level file named `.hdepend` and in one file per source directory named `.depend`.
- If you have `CONFIG_MODVERSIONS` enabled, `make dep` computes symbol version information for all of the files that export symbols (note that both resident and modular files can export symbols). If you do not enable `CONFIG_MODVERSIONS`, you only have to run `make dep` once, right after the first time you configure the kernel. The `.hdepend` files and the `.depend` file are independent of your configuration. If you do enable `CONFIG_MODVERSIONS`, you must run `make dep` because the symbol version information depends on the configuration.

Compiling the kernel

Enter `make image` to create a Linux for zSeries kernel image. This compiles the source code and leaves the kernel image in the current directory (usually `/usr/src/linux/arch/s390/boot`).

Note that `make zImage` and `make bzImage` are not supported by the Linux for zSeries kernel.

Compiling for IPL from tape

If you want to make a boot tape, you must transfer a set of files to the IPL tape. The files, `image.tape.bin` (renamed as `image.txt`), `parm.line`, and `initrd.bin` (renamed as `initrd.txt`) are the ones used during the installation process.

If you want to IPL from tape, ensure that the following configuration settings are used:

- CONFIG_IPL is set
- CONFIG_IPL_TAPE is set
- CONFIG_BLK_DEV_RAM is set
- CONFIG_BLK_DEV_INITRD is set.

Additionally you should keep the default configuration settings to make sure that all requirements to get a running Linux for zSeries kernel are met.

Installing the modules

If you configured any of the parts of the Linux for zSeries kernel as modules by selecting (M) in the kernel parameter option, you will have to create the modules and then link them to the kernel.

You create the modules by entering the command `make modules`. This will compile all of the modules and update the `linux/modules` directory. This directory will now contain a set of symbolic links, pointing to the various object files in the kernel tree.

After you have created all your modules, you must enter `make modules_install`. This will copy all of the newly made modules into subdirectories under `/lib/modules/`*`kernel_release`*`/`, where *`kernel_release`* is 2.4 (the current kernel version).

As soon as you have rebooted the newly made kernel, you can use the utilities `insmod` and `rmmod` to install and remove modules without recompiling the kernel. Read the man-pages for `insmod` and `rmmod` to find out how to configure and remove a module.

Finishing up

You should always keep a backup Linux for zSeries kernel available in case something goes wrong. This backup must also include the modules corresponding to that kernel. If you are installing a new kernel with the same version number as your working kernel, make a backup of your modules' directory before you do a `make modules_install`.

In order to boot your new kernel, you'll need to copy the kernel image (found in `/usr/src/linux/arch/s390/boot/image` after compilation) to the place where your regular bootable kernel is located. This will be on your IPL tape.

To see how to create a tape from which you can perform an IPL, refer to the installation manual for your Linux for zSeries distribution.

Kernel parameter options

The Linux for zSeries specific kernel parameter options are described in the following sections:

IEEE FPU emulation

Configuration option
CONFIG_MATHEMU

Capable of being a module? -- (Module Name)
No

Value Required by Linux for zSeries
Dependent on zSeries version, see Description

Description
From S/390 versions G5 and G6 (or later), the zSeries systems are capable of calculating floating point numbers in IEEE-format.

Linux for zSeries provides an IEEE floating point emulation. This can be configured on (enter Y) or off (enter N) during kernel compilation time. If you have IEEE-emulation configured on, floating point arithmetic can be performed on any zSeries system, however it will be slow on those systems using the emulation.

On S/390 versions G3, G4 (or older ones) you must run with IEEE-emulation configured on (Y).

On S/390 versions G5, G6 (or later) you can make use of the hardware IEEE-arithmetic. VM/ESA has to be enabled to allow it to use and provide the hardware IEEE-arithmetic. This occurs automatically when you are running VM 2.4 or VM 2.3 with a PTF applied that enables the

IEEE-arithmetic. If you do not have VM 2.4 or did not apply the PTF to VM 2.3, then you still can (and have to) rely on the IEEE-emulation as on the older S/390 systems.

Built-in IPL record support

Configuration option
CONFIG_IPL

Capable of being a module? -- (Module Name)
No

Value Required by Linux for zSeries
Yes

Description
With this option turned on an IPL loader is generated at the start of the kernel image. That makes it possible to ’boot’ from the kernel image directly without the need of a separate loader. This makes sense for a medium that is sequentially read from the start at IPL time like a (VM) reader or a tape. The type of the loader generated to the head of the kernel image is chosen by the ’IPL method generated into head.S’ selection.

IPL method generated into head.S

Configuration option
CONFIG_IPL_VM

Capable of being a module? -- (Module Name)
No

Value Required by Linux for zSeries
See Description

Description
There are two loaders available for the generation into the kernel. ’tape’ selects the loader for an IPL from a tape device, ’vm_reader’ selects the loader for an IPL from a VM virtual reader.

Enable /proc/deviceinfo entries

Configuration option
CIO_PROC_DEVINFO

Capable of being a module? -- (Module Name)
No

Value Required by Linux for zSeries
No

Description
With many devices attached the `proc` filesystem runs out of inodes. Creating of the `/proc/deviceinfo/` entries is now disabled by default. If it is required it can be switched on again by setting this parameter to ’yes’.

Channel device layer support

Configuration option
CONFIG_CHANDEV

Capable of being a module? -- (Module Name)
No

Value Required by Linux for zSeries
No

Description
This option enables the new Channel Device Layer support. Currently the devices supported are:

1. **LCS**
2. **CTC/ESCON**
3. **QETH**
4. **OSAD**

See 'Channel Device Layer' on page 69 for more information.

Support for DASD devices

Configuration option
CONFIG_DASD

Capable of being a module? -- (Module Name)
`dasd_mod.o`

Value Required by Linux for zSeries
See Description

Description
This is used mainly in native installations.

Enable this option (Y) to support access to zSeries disks. These are known as DASD (Direct Access Storage Devices). You must enable this option to have disk access on a native or LPAR system.

When enabled (Y), this option lets you specify additional options for DASD access, see "Support for ECKD disks".

Support for ECKD disks

Configuration option
CONFIG_DASD_ECKD

Capable of being a module? -- (Module Name)
dasd_eckd_mod.o

Value Required by Linux for zSeries
See Description

Description
This is used mainly in native installations.

Enable this option (Y) if you have ECKD-type DASDs such as an IBM 3380 or 3390. ECKD devices are the most commonly used devices in zSeries, so you should enable this option unless you are very sure you do not have any ECKD devices.

This option is subordinate to `CONFIG_DASD`, see "Support for DASD devices".

Support for FBA disks

Configuration option
CONFIG_DASD_FBA

Capable of being a module? -- (Module Name)
dasd_fba_mod.o

Value Required by Linux for zSeries
See Description

Description
This is used mainly under VM/ESA for the virtual disk in storage VFB-512. Enable this option (Y) if you want to access your FBA devices.

This option is subordinate to `CONFIG_DASD`, see "Support for DASD devices" on page 220.

XPRAM device support

Configuration option
CONFIG_BLK_DEV_XPRAM

Capable of being a module? -- (Module Name)
`xpram.o`

Value Required by Linux for zSeries
See Description

Description
This is used to allow more than 2 GB of main storage to be accessed by Linux for zSeries. Enable this option (Y) to support access to expanded storage of up to 18 EB (although current hardware currently supports only 64 GB). The expanded storage can be subdivided into partitions.

See "XPRAM kernel parameter syntax" on page 26 for more information.

CTC/ESCON device support

Configuration option
CONFIG_CTC

Capable of being a module? -- (Module Name)
ctc.o

Value Required by Linux for zSeries
No

Description
If you want to use channel connections under Linux , enter (Y) here. This gives you the opportunity to make TCP/IP connections via virtual, parallel or ESCON channels between Linux for zSeries and other zSeries operating systems (Linux for zSeries, z/OS, OS/390, VM/ESA and VSE/ESA).

Read the CTC/ESCON device driver description on page79 for more information.

IUCV device support

Configuration option
CONFIG_IUCV

Capable of being a module? -- (Module Name)
netiucv.o

Value Required by Linux for zSeries
No

Description
This is a VM/ESA only device driver. Enter (Y) to enable a fast communication link between VM guests. At boot time the user ID of the guest needs to be passed to the kernel. Using `ifconfig` a point-to-point connection can be established to the Linux for zSeries system running on the other VM guest. Note that both kernels need to be compiled with this option and both need to be booted with the user ID of the other VM guest.

Basic QDIO support

Configuration option
CONFIG_QDIO

Capable of being a module? -- (Module Name)
qdio.o

Value Required by Linux for zSeries
Base support for any QDIO devices.

Description
If you want to use any QDIO based devices (OSA-Express, HiperSockets, or OpenFCP) under Linux, enter (Y) here.

Configuration option
CONFIG_QDIO_PERF_STATS

Capable of being a module? -- (Module Name)
qdio.o

Value Required by Linux for zSeries
No

Description
To get performance statistics in /proc/qdioi_perf, enter (Y) here.

Dummy device support

Configuration option
CONFIG_DUMMY

Value Required by Linux for zSeries
Required in order to use the VIPA functionality.

Description
To use OSA-Express or HiperSockets dummy connections under Linux , enter (Y) here.

Configuration option
CONFIG_SHARED_IPV6_CARDS

Value Required by Linux for zSeries
Required to facilitate IPv6 stateless address generation on OSA-Express feature using QDIO.

Description
To generate unique IPv6 stateless autoconfigured addresses on the same OSA-Express feature by different guest images, enter (Y) here.

Support for 3215 line mode terminal

Configuration option
CONFIG_TN3215

Capable of being a module? -- (Module Name)
No

Value Required by Linux for zSeries
No

Description
The 3215 console driver is used to read and write to a 3215 line mode console. Real 3215 devices are no longer available in a zSeries environment, so the 3215 driver can only be used under VM/ESA. On a native zSeries system the initialization function of the 3215 driver returns without registering the driver to the system.

Entering (Y) to this option switches on the compilation of parts 1 and 2 of the 3215 terminal driver. The option makes it possible to use "Support for console output on 3215 line mode terminal".

Support for console output on 3215 line mode terminal

Configuration option
CONFIG_TN3215_CONSOLE

Capable of being a module? -- (Module Name)
No

Value Required by Linux for zSeries
No

Description
This option is subordinate to "Support for 3215 line mode terminal" on page 222.

This option enables console output on the first 3215 console in the system. It prints kernel errors and kernel warnings to the 3215 terminal in addition to the normal output on the TTY device.

Support for 3270 console

Configuration option
CONFIG_TN3270

Capable of being a module? -- (Module Name)
No

Value Required by Linux for zSeries
No

Description
The 3270 console driver is used to read and write to a 3270 console.

Entering (Y) to this option switches on the compilation of the 3270 terminal driver.

Support for hardware console

Configuration option
CONFIG_HWC

Capable of being a module? -- (Module Name)
No

Value Required by Linux for zSeries
See Description

Description
The hardware console is an alternative terminal, usually required for a native Linux for zSeries installation although it is also run under VM/ESA.

You would normally enter (Y) for this option in a native installation if your hardware configuration includes a hardware console. In a VM/ESA installation, without a hardware console, you would normally enter (N).

Read the Device driver description Chapter 5, “Linux for zSeries Console device drivers” on page 29 for more information.

Console output on hardware console

Configuration option
CONFIG_HWC_CONSOLE

Capable of being a module? -- (Module Name)
No

Value Required by Linux for zSeries
No

Description
This option is subordinate to “Support for hardware console” on page 223.

This option enables console output on the first hardware console in the system. It prints kernel errors and kernel warnings to the hardware console in addition to the normal output on the TTY device.

Tape device support

Configuration option
CONFIG_S390_TAPE

Capable of being a module? -- (Module Name)
tape390.o

Value Required by Linux for zSeries
No

Description
If you want to use the tape device driver with Linux enter (m) here.

SCSI device support via Fibre Channel

Configuration option
CONFIG_ZFCP

Capable of being a module? -- (Module Name)
zfcp.o

Value Required by Linux for zSeries
Yes

Description
To configure access to SCSI devices over Fibre Channel interfaces; enter (Y) here.

Configuration option
CONFIG_QDIO

Capable of being a module? -- (Module Name)
qdio.o

Value Required by Linux for zSeries
Yes

Description
To configure Queued Direct I/O for OSA-Express, enter (Y) here. This option is required for SCSI access via Fibre Channel.

Configuration option
CONFIG_QDIO_PERF_STATS

Capable of being a module? -- (Module Name)
qdio.o

Value Required by Linux for zSeries
No

Description
To get performance statistics in /proc/qdioi_perf, enter (Y) here.

Glossary

This glossary includes IBM product terminology as well as selected other terms and definitions. Additional information can be obtained in:

- The American National Standard Dictionary for Information Systems , ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036.
- The ANSI/EIA Standard–440-A, Fiber Optic Terminology. Copies may be purchased from the Electronic Industries Association, 2001 Pennsylvania Avenue, N.W., Washington, DC 20006.
- The Information Technology Vocabulary developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1).
- The IBM Dictionary of Computing , New York: McGraw-Hill, 1994.
- Internet Request for Comments: 1208, Glossary of Networking Terms
- Internet Request for Comments: 1392, Internet Users' Glossary
- The Object-Oriented Interface Design: IBM Common User Access Guidelines , Carmel, Indiana: Que, 1992.

A

asynchronous transfer mode (ATM). A transfer mode in which the information is organized into cells; it is asynchronous in the sense that the recurrence of cells containing information from an individual user is not necessarily periodic. ATM is specified in international standards such as ATM Forum UNI 3.1.

auto-detection. Listing the addresses of devices attached to a card by issuing a query command to the card.

C

cdl. compatible disk layout. A disk structure for Linux for zSeries which allows access from other zSeries operating systems. This replaces the older **ldl**.

CEC. (Central Electronics Complex). A synonym for *CPC*.

chandev. channel device layer. A unified programming interface to devices attached to the zSeries via the channel subsystem.

channel subsystem. The programmable input/output processors of the zSeries, 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.

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 zSeries devices using device numbers.

D

DASD. direct access storage device. A mass storage medium on which a computer stores data.

device driver. (1) A file that contains the code needed to use an attached device. (2) A program that enables a computer to communicate with a specific peripheral device; for example, a printer, a videodisc player, or a CD-ROM drive. (3) A collection of subroutines that control the interface between I/O device adapters and the processor.

E

ECKD. extended count-key-data device. A disk storage device that has a data transfer rate faster than some processors can utilize and that is connected to the processor through use of a speed matching buffer. A specialized channel program is needed to communicate with such a device.

ESCON. enterprise systems connection. A set of IBM products and services that provide a dynamically connected environment within an enterprise.

Ethernet. A 10-Mbps baseband local area network that allows multiple stations to access the transmission medium at will without prior coordination, avoids contention by using carrier sense and deference, and resolves contention by using collision detection and delayed retransmission. Ethernet uses CSMA/CD.

F

Fast Ethernet (FENET). Ethernet network with a bandwidth of 100 Mbps

FBA. fixed block architecture. A type of DASD on Multiprise 3000 or P/390 or emulated by VM.

FDDI. fiber distributed data interface. An American National Standards Institute (ANSI) standard for a 100-Mbps LAN using optical fiber cables.

FTP. file transfer protocol. In the Internet suite of protocols, an application layer protocol that uses TCP and Telnet services to transfer bulk-data files between machines or hosts.

G

Gigabit Ethernet (GbE). An Ethernet network with a bandwidth of 1000-Mbps

G3, G4, G5 and G6. The generation names of the S/390 CMOS based product family.

H

hardware console. A service-call logical processor that is the communication feature between the main processor and the service processor.

HBA. ???

HMC. hardware management console. A console used to monitor and control hardware such as the zSeries microprocessors.

HFS. hierarchical file system. A system of arranging files into a tree structure of directories.

I

IOCS. input / output channel subsystem. See channel subsystem.

IP. internet protocol. In the Internet suite of protocols, a connectionless protocol that routes data through a network or interconnected networks and acts as an intermediary between the higher protocol layers and the physical network.

IP address.. The unique 32-bit address that specifies the location of each device or workstation on the Internet. For example, 9.67.97.103 is an IP address.

IPIP. IPv4 in IPv4 tunnel, used to transport IPv4 packets in other IPv4 packets.

IPL. initial program load (or boot). (1) The initialization procedure that causes an operating system to commence operation. (2) The process by which a configuration image is loaded into storage at the beginning of a work day or after a system malfunction. (3) The process of loading system programs and preparing a system to run jobs.

IPv6. IP version 6. The next generation of the Internet Protocol.

IPX. Internetwork Packet Exchange. (1) The network protocol used to connect Novell servers, or any workstation or router that implements IPX, with other workstations. Although similar to the Internet Protocol (IP), IPX uses different packet formats and terminology.

IPX address. The 10-byte address, consisting of a 4-byte network number and a 6-byte node address, that is used to identify nodes in the IPX network. The node address is usually identical to the medium access control (MAC) address of the associated LAN adapter.

IUCV. inter-user communication vehicle. A VM facility for passing data between virtual machines and VM components.

K

kernel. The part of an operating system that performs basic functions such as allocating hardware resources.

kernel module. A dynamically loadable part of the kernel, such as a device driver or a file system.

kernel image. The kernel when loaded into memory.

L

LAN. local area network.

LCS. LAN channel station. A protocol used by OSA.

ldl. Linux disk layout. A basic disk structure for Linux for zSeries. Now replaced by `cdl`.

LDP. Linux Documentation Project. An attempt to provide a centralized location containing the source material for all open source Linux documentation. Includes user and reference guides, HOW TOs, and FAQs. The homepage of the Linux Documentation Project is **http://www.linuxdoc.org**

Linux . a version of UNIX which runs on a wide range of machines from wristwatches through personal and small business machines to enterprise systems.

Linux for zSeries. the port of Linux to the IBM zSeries architecture.

LPAR. logical partition of a zSeries.

LVS (Linux virtual server). Network sprayer software used to dispatch, for example, http requests to a set of Web servers to balance system load.

M

MAC. medium access control. In a LAN this is the sub-layer of the data link control layer that supports medium-dependent functions and uses the services of the physical layer to provide services to the logical link control (LLC) sub-layer. The MAC sub-layer includes the method of determining when a device has access to the transmission medium.

Mbps. million bits per second.

MIB (Management Information Base). (1) A collection of objects that can be accessed by means of a network management protocol. (2) A definition for management information that specifies the information available from a host or gateway and the operations allowed.

MTU. maximum transmission unit. The largest block which may be transmitted as a single unit.

Multicast. A protocol for the simultaneous distribution of data to a number of recipients, for example live video transmissions.

Multiprise. An enterprise server of the S/390 family.

N

NIC. network interface card. The physical interface between the zSeries and the network.

O

OS. operating system. (1) Software that controls the execution of programs. An operating system may provide services such as resource allocation, scheduling, input/output control, and data management. (2) A set of programs that control how the system works. (3) The software that deals with the most basic operations that a computer performs.

OSA-2. Open Systems Adapter-2. A common zSeries network interface feature

OSA-Express. Abbreviation for S/390 and zSeries Open Systems Adapter-Express networking features. These include Gigabit Ethernet, Fast Ethernet, Token Ring, and ATM.

OSPF. open shortest path first. A function used in route optimization in networks.

P

POR. power-on reset

POSIX. Portable Operating System Interface for Computer Environments. An IEEE operating system standard closely related to the UNIX system.

R

router. A device or process which allows messages to pass between different networks.

S

S/390. The predecessor of the zSeries.

SA/SE. stand alone support element. See SE.

SE. support element. (1) An internal control element of a processor that assists in many of the processor operational functions. (2) A hardware unit that provides communications, monitoring, and diagnostic functions to a central processor complex.

SNA. systems network architecture. The IBM architecture that defines the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of, networks. The layered structure of SNA allows the ultimate origins and destinations of information (the users) to be independent of and unaffected by the specific SNA network services and facilities that are used for information exchange.

SNMP (Simple Network Management Protocol). In the Internet suite of protocols, a network management protocol that is used to monitor routers and attached networks. SNMP is an application layer protocol. Information on devices managed is defined and stored in the application's Management Information Base (MIB).

Sysctl. system control programming manual control (frame). A means of dynamically changing certain Linux kernel parameters during operation.

T

TCP. transmission control protocol. A communications protocol used in the Internet and in any network that follows the Internet Engineering Task Force (IETF) standards for internetwork protocol. TCP provides a reliable host-to-host protocol between hosts in packet-switched communications networks and in interconnected systems of such networks. It uses the Internet Protocol (IP) as the underlying protocol.

TCP/IP. transmission control protocol/internet protocol. (1) The Transmission Control Protocol and the Internet Protocol, which together provide reliable end-to-end connections between applications over interconnected networks of different types. (2) The suite of transport and application protocols that run over the Internet Protocol.

Telnet. A member of the Internet suite of protocols which provides a remote terminal connection service. It allows users of one host to log on to a remote host and interact as if they were using a terminal directly attached to that host.

Token Ring. (1) According to IEEE 802.5, network technology that controls media access by passing a token (special packet or frame) between media-attached stations. (2) A FDDI or IEEE 802.5 network with a ring topology that passes tokens from one attaching ring station (node) to another.

U

UNIX. An operating system developed by Bell Laboratories that features multiprogramming in a multiuser environment. The UNIX operating system was originally developed for use on minicomputers but has been adapted for mainframes and microcomputers.

V

V=R. In VM, a guest whose real memory (virtual from a VM perspective) corresponds to the real memory of VM.

V=V. In VM, a guest whose real memory (virtual from a VM perspective) corresponds to virtual memory of VM.

Virtual LAN (VLAN).. A group of devices on one ore more LANs that are configured (using management software) so that they can communicate as if they were attached to the same wire, when in fact they are located on a number of different LAN segments. Because VLANs are based on logical rather than physical connections, they are extremely flexible.

volume. A data carrier that is usually mounted and demounted as a unit, for example a tape cartridge or a disk pack. If a storage unit has no demountable packs the volume is the portion available to a single read/write mechanism.

Z

zSeries. The family of IBM enterprise servers that demonstrate outstanding reliability, availability, scalability, security, and capacity in today's network computing environments.

Numbers

3215. IBM console printer-keyboard.

3270. IBM information display system.

3370, 3380 or 3390. IBM direct access storage device (disk).

3480 or 3490. IBM magnetic tape subsystem.

9336 or 9345. IBM direct access storage device (disk).

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information about the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

- AIX
- Common User Access
- ECKD
- Enterprise Storage Server
- ESCON
- IBM
- Multiprise
- MVS
- OS/2
- OS/390
- PR/SM
- RAMAC
- S/390
- Seascape
- System/390
- VM/ESA
- VSE/ESA
- z/OS
- z/VM
- zSeries

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds and others.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

International License Agreement for Non-Warranted Programs

Part 1 - General Terms

PLEASE READ THIS AGREEMENT CAREFULLY BEFORE USING THE PROGRAM. IBM WILL LICENSE THE PROGRAM TO YOU ONLY IF YOU FIRST ACCEPT THE TERMS OF THIS AGREEMENT. BY USING THE PROGRAM YOU AGREE TO THESE TERMS. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, PROMPTLY RETURN THE UNUSED PROGRAM TO THE PARTY (EITHER IBM OR ITS RESELLER) FROM WHOM YOU ACQUIRED IT TO RECEIVE A REFUND OF THE AMOUNT YOU PAID.

The Program is owned by International Business Machines Corporation or one of its subsidiaries (IBM) or an IBM supplier, and is copyrighted and licensed, not sold.

The term ″Program″ means the original program and all whole or partial copies of it. A Program consists of machine-readable instructions, its components, data, audio-visual content (such as images, text, recordings, or pictures), and related licensed materials.

This Agreement includes Part 1 - General Terms and Part 2 - Country-unique Terms and is the complete agreement regarding the use of this Program, and replaces any prior oral or written communications between you and IBM. The terms of Part 2 may replace or modify those of Part 1.

1. License

   Use of the Program

   IBM grants you a nonexclusive license to use the Program.

   You may 1) use the Program to the extent of authorizations you have acquired and 2) make and install copies to support the level of use authorized, providing you reproduce the copyright notice and any other legends of ownership on each copy, or partial copy, of the Program.

   If you acquire this Program as a program upgrade, your authorization to use the Program from which you upgraded is terminated.

   You will ensure that anyone who uses the Program does so only in compliance with the terms of this Agreement.

   You may not 1) use, copy, modify, or distribute the Program except as provided in this Agreement; 2) reverse assemble, reverse compile, or otherwise translate the Program except as specifically permitted by law without the possibility of contractual waiver; or 3) sublicense, rent, or lease the Program.

   Transfer of Rights and Obligations

   You may transfer all your license rights and obligations under a Proof of Entitlement for the Program to another party by transferring the Proof of Entitlement and a copy of this Agreement and all documentation. The transfer of your license rights and obligations terminates your authorization to use the Program under the Proof of Entitlement.

2. Proof of Entitlement

   The Proof of Entitlement for this Program is evidence of your authorization to use this Program and of your eligibility for future upgrade program prices (if announced) and potential special or promotional opportunities.

3. Charges and Taxes

   IBM defines use for the Program for charging purposes and specifies it in the Proof of Entitlement. Charges are based on extent of use authorized. If you wish to increase the extent of use, notify IBM or its reseller and pay any applicable charges. IBM does not give refunds or credits for charges already due or paid.

   If any authority imposes a duty, tax, levy or fee, excluding those based on IBM's net income, upon the Program supplied by IBM under this Agreement, then you agree to pay that amount as IBM specifies or supply exemption documentation.

4. No Warranty

   SUBJECT TO ANY STATUTORY WARRANTIES WHICH CAN NOT BE EXCLUDED, IBM MAKES NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, THE WARRANTY OF NON-INFRINGEMENT AND THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY. IBM MAKES NO WARRANTY REGARDING THE CAPABILITY OF THE PROGRAM TO CORRECTLY PROCESS, PROVIDE AND/OR RECEIVE DATE DATA WITHIN AND BETWEEN THE 20TH AND 21ST CENTURIES.

   The exclusion also applies to any of IBM's subcontractors, suppliers, or program developers (collectively called "Suppliers").

   Manufacturers, suppliers, or publishers of non-IBM Programs may provide their own warranties.

5. Limitation of Liability

   NEITHER IBM NOR ITS SUPPLIERS WILL BE LIABLE FOR ANY DIRECT OR INDIRECT DAMAGES, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST SAVINGS, OR ANY INCIDENTAL, SPECIAL, OR OTHER ECONOMIC CONSEQUENTIAL DAMAGES, EVEN IF IBM IS INFORMED OF THEIR POSSIBILITY. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE EXCLUSION OR LIMITATION MAY NOT APPLY TO YOU.

6. General

   Nothing in this Agreement affects any statutory rights of consumers that cannot be waived or limited by contract.

   IBM may terminate your license if you fail to comply with the terms of this Agreement. If IBM does so, you must immediately destroy the Program and all copies you made of it.

   You agree to comply with applicable export laws and regulations.

   Neither you nor IBM will bring a legal action under this Agreement more than two years after the cause of action arose unless otherwise provided by local law without the possibility of contractual waiver or limitation.

   Neither you nor IBM is responsible for failure to fulfill any obligations due to causes beyond its control.

   IBM does not provide program services or technical support, unless IBM specifies otherwise.

   The laws of the country in which you acquire the Program govern this Agreement, except 1) in Australia, the laws of the State or Territory in which the transaction is performed govern this Agreement; 2) in Albania, Armenia, Belarus, Bosnia/Herzegovina, Bulgaria, Croatia, Czech Republic, Georgia, Hungary, Kazakhstan, Kirghizia, Former Yugoslav Republic of Macedonia

(FYROM), Moldova, Poland, Romania, Russia, Slovak Republic, Slovenia, Ukraine, and Federal Republic of Yugoslavia, the laws of Austria govern this Agreement; 3) in the United Kingdom, all disputes relating to this Agreement will be governed by English Law and will be submitted to the exclusive jurisdiction of the English courts; 4) in Canada, the laws in the Province of Ontario govern this Agreement; and 5) in the United States and Puerto Rico, and People's Republic of China, the laws of the State of New York govern this Agreement.

Part 2 - Country-unique Terms

AUSTRALIA:

No Warranty (Section 4):

The following paragraph is added to this Section:

Although IBM specifies that there are no warranties, you may have certain rights under the Trade Practices Act 1974 or other legislation and are only limited to the extent permitted by the applicable legislation.

Limitation of Liability (Section 3):

The following paragraph is added to this Section:

Where IBM is in breach of a condition or warranty implied by the Trade Practices Act 1974, IBM's liability is limited to the repair or replacement of the goods, or the supply of equivalent goods. Where that condition or warranty relates to right to sell, quiet possession or clear title, or the goods are of a kind ordinarily acquired for personal, domestic or household use or consumption, then none of the limitations in this paragraph apply.

GERMANY:

No Warranty (Section 4):

The following paragraphs are added to this Section:

The minimum warranty period for Programs is six months.

In case a Program is delivered without Specifications, we will only warrant that the Program information correctly describes the Program and that the Program can be used according to the Program information. You have to check the usability according to the Program information within the ″money-back guaranty″ period.

Limitation of Liability (Section 5):

The following paragraph is added to this Section:

The limitations and exclusions specified in the Agreement will not apply to damages caused by IBM with fraud or gross negligence, and for express warranty.

INDIA:

General (Section 6):

International license agreement

The following replaces the fourth paragraph of this Section:

If no suit or other legal action is brought, within two years after the cause of action arose, in respect of any claim that either party may have against the other, the rights of the concerned party in respect of such claim will be forfeited and the other party will stand released from its obligations in respect of such claim.

IRELAND:

No Warranty (Section 4):

The following paragraph is added to this Section:

Except as expressly provided in these terms and conditions, all statutory conditions, including all warranties implied, but without prejudice to the generality of the foregoing, all warranties implied by the Sale of Goods Act 1893 or the Sale of Goods and Supply of Services Act 1980 are hereby excluded.

ITALY:

Limitation of Liability (Section 5):

This Section is replaced by the following:

Unless otherwise provided by mandatory law, IBM is not liable for any damages which might arise.

NEW ZEALAND:

No Warranty (Section 4):

The following paragraph is added to this Section:

Although IBM specifies that there are no warranties, you may have certain rights under the Consumer Guarantees Act 1993 or other legislation which cannot be excluded or limited. The Consumer Guarantees Act 1993 will not apply in respect of any goods or services which IBM provides, if you require the goods and services for the purposes of a business as defined in that Act.

Limitation of Liability (Section 5):

The following paragraph is added to this Section:

Where Programs are not acquired for the purposes of a business as defined in the Consumer Guarantees Act 1993, the limitations in this Section are subject to the limitations in that Act.

PEOPLE'S REPUBLIC OF CHINA:

Charges (Section 3):

The following paragraph is added to the Section:

All banking charges incurred in the People's Republic of China will be borne by you and those incurred outside the People's Republic of China will be borne by IBM.

UNITED KINGDOM:

Limitation of Liability (Section 5):

The following paragraph is added to this Section at the end of the first paragraph:

The limitation of liability will not apply to any breach of IBM's obligations implied by Section 12 of the Sales of Goods Act 1979 or Section 2 of the Supply of Goods and Services Act 1982.

Index

Readers' Comments — We'd Like to Hear from You

Linux for zSeries
Device Drivers and Installation Commands
April 30, 2002 (Draft)
Linux Kernel 2.4

Publication No. LNUX-1103-08

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name	Address
Company or Organization	
Phone No.	

Readers' Comments — We'd Like to Hear from You
LNUX-1103-08

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Deutschland Entwicklung GmbH
Information Development
Department 3248
Schoenaicher Strasse 220
71032 Boeblingen
Germany

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

LNUX-1103-08