

Linux on zSeries



Device Drivers and Installation Commands

March 23, 2005

Linux Kernel 2.4 - June 2003 stream

Linux on zSeries



Device Drivers and Installation Commands

March 23, 2005

Linux Kernel 2.4 - June 2003 stream

Note

Before using this document, be sure to read the information in “Notices” on page 325.

Sixth Edition (March 2005)

This edition applies to the Linux for zSeries and S/390 kernel 2.4 (June 2003 stream) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces LNX-1313-04. LNX-1313 is the Linux for zSeries and S/390 kernel 2.4 (June 2003 stream) equivalent to LNX-1303 which applies to Linux for zSeries and S/390 kernel 2.4, May 2002 stream.

© **Copyright International Business Machines Corporation 2000, 2005. All rights reserved.**

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

Contents

Figures	vii
--------------------------	------------

Tables	ix
-------------------------	-----------

Summary of changes.	xi
--------------------------------------	-----------

Edition 6 changes	xi
Edition 5 changes	xi
Edition 4 changes	xii

About this document	xiii
--------------------------------------	-------------

How this document is organized	xiii
Who should read this document	xiii
License conditions	xiv
Assumptions	xiv
Understanding syntax diagrams	xiv

Part 1. Device drivers overview	1
--	----------

Chapter 1. Common device support.	3
proc file system	5

Part 2. Block and character device drivers	7
---	----------

Chapter 2. DASD device driver	9
--	----------

DASD overview	9
DASD naming scheme using devfs	9
DASD naming scheme without devfs.	10
DASD structures.	11
DASD partitioning	14
DASD – Preparing for use	16
DASD features	17
DASD kernel parameter syntax.	17
DASD kernel example (using devfs)	18
DASD module parameter syntax	19
DASD module example	19
DASD dynamic attach and enable/disable	20
DASD reserve and release	21
Accessing boxed DASD	21
Accessing DASD by VOLSER	22
DASD API (ioctl interface)	23
DASD Channel Measurement Facility.	25

Chapter 3. XPRAM device driver.	29
--	-----------

XPRAM features.	29
Note on reusing XPRAM partitions	29
XPRAM kernel parameter syntax	30
XPRAM module parameter syntax.	31

Chapter 4. z/VM discontinuous saved segments device driver	33
---	-----------

Building a kernel with the DCSS device driver	33
---	----

Setting up the DCSS device driver.	33
Working with the DCSS device driver	35

Chapter 5. Console device drivers	39
--	-----------

Console features.	39
Console usage	39
Device names and nodes	41
Console kernel parameter syntax	41
Using the console	43
Console – Use of VInput	46

Chapter 6. Channel-attached tape device driver	49
---	-----------

Tape driver features	49
Tape disciplines as modules	50
Tape device front-ends.	50
Tape driver kernel parameter syntax	52
Tape driver kernel example	53
Tape driver module parameter syntax	53
Tape driver module example	54
Tape device driver API	54
/proc/tapedevices interface	55
Boxing tape devices	56
Tape display support	58
Tape driver examples	59
Tape driver further information.	61

Chapter 7. Generic cryptographic device driver	63
---	-----------

Features	63
What you should know about z90crypt	63
Building a kernel with the z90crypt device driver	65
Setting up the z90crypt device driver.	65
Working with the z90crypt device driver	67
External programming interfaces	69

Chapter 8. SCSI-over-Fibre Channel device driver	73
---	-----------

Configuring zfcp	73
devfs and zfcp mapping	81
Sample walkthrough	81
Installation considerations	83
zfcp HBA API (FC-HBA) support	83
Considerations for future distributions	87
References.	87

Chapter 9. z/VM recording device driver	89
--	-----------

Building a kernel with the z/VM recording device driver	89
Setting up the z/VM recording device driver	89
Working with the z/VM recording device driver	90

Part 3. Network device drivers	91
---	-----------

Chapter 10. Channel device layer 93

Description	93
Channel device layer options	94
See also	103
Files	103

Chapter 11. CTC/ESCON device driver 105

CTC/ESCON features	105
CTC/ESCON with the channel device layer	105
CTC/ESCON without the channel device layer	106
CTC/ESCON – Preparing the connection	108
CTC/ESCON – Recovery procedure after a crash	111

Chapter 12. CTCMPC device driver 113

Features	113
CTCMPC devices	113
Setting up the CTCMPC device driver	114
CTCMPC device configuration	114

Chapter 13. IUCV device driver 115

IUCV features	115
IUCV kernel parameter syntax.	115
IUCV kernel parameter example	116
IUCV module parameter syntax	116
IUCV module parameter example	116
IUCV – Preparing the connection.	117
IUCV – Further information	119
IUCV Application Programming Interface (API)	119

Chapter 14. LCS device driver 137

LCS supported functions	137
LCS channel device layer configuration.	137
LCS channel device layer configuration example	137
LCS warning	138

Chapter 15. QETH device driver for OSA-Express (QDIO) and HiperSockets 139

Introduction	139
Installing QETH	140
QETH supported functions.	141
Configuring QETH for OSA-Express and HiperSockets using the channel device layer	141
QETH parameter syntax.	144
OSA-Express CHPID in QDIO mode qeth module parameter syntax	147
OSA-Express CHPID in QDIO mode channel device layer configuration example	148
HiperSockets channel device layer configuration example	148
Examples: OSA-Express CHPID in QDIO mode	148
Examples: HiperSockets	149
OSA-Express CHPID in QDIO mode and HiperSockets – Preparing the connection	151
Duplicate IP addresses	151
IP address takeover	152
Proxy ARP	154
OSA-Express CHPID in QDIO mode – Virtual IP address (VIPA)	155

Priority queuing	155
Support for IP Version 6 (IPv6)	156
Querying and purging OSA and HiperSockets ARP data	157
Confirming that an IP address has been set	157
MTU considerations	157

Chapter 16. Linux for zSeries and S/390 CLAW device driver 159

CLAW features.	159
CLAW with the channel device layer	159
CLAW without the channel device layer	161

Part 4. Installation/configuration commands and parameters 165

Chapter 17. Useful Linux commands 167

dasdfmt - Format a DASD	168
dasdview - Display DASD structure.	172
fdasd - DASD partitioning tool	182
osasnmpd - Start OSA-Express SNMP subagent	189
qetharp - Query and purge OSA and HiperSockets ARP data.	197
qethconf - configure qeth devices.	199
snIPL - Simple network IPL (Linux image control for LPAR and VM)	201
zIPL - zSeries initial program loader	209
ifconfig - Configure a network interface	221
insmod - Load a module into the Linux kernel	225
modprobe - Load a module with dependencies into the Linux kernel	227
lsmod - List loaded modules	230
depmod - Create dependency descriptions for loadable kernel modules.	231
mke2fs - Create a file system on DASD.	233
vconfig - VLAN (802.1Q) configuration program	234

Chapter 18. VIPA – minimize outage due to adapter failure 237

Standard VIPA	237
Source VIPA.	239
Source VIPA 2	241

Chapter 19. VARY ON/OFF events and toggling CHPIDs online/offline 245

Response to VARY ON/OFF events	245
Toggling CHPIDs logically on and offline	245

Chapter 20. Multipathing support . . . 247

Supported storage devices	247
Multipath support in blockdevice managers	247

Chapter 21. Linux monitor stream support for z/VM 255

Building a kernel that is enabled for monitoring	255
Working with the monitoring stream	255
APPLDATA monitor record layout	257
Programming interfaces	260

Chapter 22. Virtual LAN (VLAN) support	261
Introduction to VLANs	261
Configuring VLAN devices.	262
Examples.	263
Further information	264
 Chapter 23. HiperSockets Network Concentrator	 265
Design	265
Setup	265
Availability setups.	266
Hints	267
Examples.	268
 Chapter 24. Enabling OSA-Express QDIO devices in Linux for DHCP and tcpdump	 271
How the OSA-Express QDIO microcode handles IPv4 packets.	272
Setting up for DHCP.	274
Setting up for tcpdump	276
The layer2 option	278
The fake_ll option.	279
DHCP and tcpdump for HiperSockets devices	280
 Chapter 25. Selected kernel parameters	 281
cio_ignore	282
cio_msg	284
cio_notoper_msg	285
ipldelay	286
maxcpus	287

mem	288
noinitrd	289
ramdisk_size	290
ro	291
root	292
use_pfix	293
vmhalt	294
vmpoff	295

Chapter 26. Overview of the parameter line file	297
--	------------

Appendix A. Building the kernel	299
Introduction	299
Kernel configuration options	303

Appendix B. Hotplug support	317
DASD hotplug events	317
Tape hotplug events	318
Crypto hotplug events	318
z/VM recording device driver hotplug events	319

Glossary	321
---------------------------	------------

Notices	325
Trademarks	326

International License Agreement for Non-Warranted Programs	327
---	------------

Index	333
------------------------	------------

Figures

1. Structure of VOL1, LNX and CMS labeled disks	11	15. OSA-Express SNMP agent flow	190
2. z90crypt device driver interfaces	64	16. Example of using Virtual IP Address (VIPA)	238
3. z90crypt kernel configuration menu option	65	17. Example of using source VIPA	240
4. General layout of zSeries FCP environment	73	18. Example of using source VIPA 2	243
5. zfcplib HBA API support modules.	84	19. Conventional routed network	261
6. Record structure	90	20. Switched VLAN network	262
7. Connection of two systems via CTC (kernel)	107	21. VLAN network organized for traffic flow	262
8. Connection of two systems via CTC (module)	108	22. Example 1 for HiperSockets network concentrator setup	268
9. I/O subchannel interface.	113	23. Example 2 for HiperSockets network concentrator setup	269
10. Connection of two systems using IUCV	116	24. IPv4 processing in non-mainframe environments	271
11. I/O subchannel interface.	140	25. IPv4 processing by OSA-Express	271
12. Example of Proxy ARP usage	154	26. qeth with fake_ll option for incoming packets	279
13. Connection of two systems via CLAW	160	27. qeth with fake_ll option for outgoing packets	280
14. RS/6000 machine connected to a Linux system.	162		

Tables

1. DASD naming convention	10	10. Basic zIPL functions	210
2. Supported DASD devices	17	11. APPLDATA_MEM_DATA record (Record ID X'01')	258
3. Supported consoles	41	12. APPLDATA_OS_DATA record (Record ID X'02')	259
4. Default console device driver	42	13. APPLDATA_NET_SUM_DATA record (Record ID X'03')	260
5. Control characters	44	14. Kernel configuration options specific to the Linux for zSeries and S/390 environment	304
6. Magic sysrequest commands	44		
7. zfcpx functions, parameters, and proc_fs entries	74		
8. SCSI module dependencies	83		
9. Supported devices	209		

Summary of changes

This revision reflects changes to the kernel 2.4 (June 2003 stream) until February 2005.

Edition 6 changes

New Information

- The qeth device driver now supports OSA-Express2.
- The cryptographic device driver now supports additional hardware, CEX2C (see Chapter 7, “Generic cryptographic device driver,” on page 63).

Changed Information

- The osasnmppd information has been rewritten (see “osasnmppd – Start OSA-Express SNMP subagent” on page 189).

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 prerequisites and restrictions is no longer provided in this book. You can find the prerequisites and restrictions for the device drivers, tools, and features described in this book, on the developerWorks kernel 2.4 (June 2003 stream) pages at:
ibm.com/developerworks/linux/linux390/june2003_recommended.shtml
If you are routed to the top-level page: On the left navigation bar, click **Kernel 2.4** . Then click the stream with which you want to work.

Edition 5 changes

New Information

- There is a new network device driver that supports CTC based SNA connections (see Chapter 12, “CTCMPC device driver,” on page 113).
- There is a new option, layer2, for qeth devices that allows MAC-based addressing for packets (see “The layer2 option” on page 278).
- The qeth device driver now supports 10 Gigabit Ethernet

Changed Information

- Chapter 7, “Generic cryptographic device driver,” on page 63 has been rewritten. The cryptographic device driver now distinguishes between different licensed internal code versions of PCIXCC.

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

Deleted Information

- None.

Edition 4 changes

New Information

- Chapter 4, “z/VM discontinuous saved segments device driver,” on page 33
- “zfcf HBA API (FC-HBA) support” on page 83
- z/VM[®] recording device driver information. See Chapter 9, “z/VM recording device driver,” on page 89 and “z/VM recording device driver hotplug events” on page 319.
- “snIPL – Simple network IPL (Linux image control for LPAR and VM)” on page 201
- Chapter 21, “Linux monitor stream support for z/VM,” on page 255

Changed Information

- The STONITH information has been moved from the Appendix to “snIPL – Simple network IPL (Linux image control for LPAR and VM)” on page 201.
- Cryptographic device driver. See Chapter 7, “Generic cryptographic device driver,” on page 63 and “Crypto hotplug events” on page 318.

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.

About this document

This document describes the drivers available to Linux™ for the control of zSeries® and S/390® devices and attachments with the kernel 2.4 (June 2003 stream). It also provides information on commands and parameters relevant to the installation process.

The drivers described herein have been developed with version 2.4.21 of the Linux kernel. If you are using a later version of the kernel, the kernel parameters may be different from those described in this document.

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

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

Note: For tools related to taking and analyzing system dumps, see *Linux for zSeries and S/390 Using the Dump Tools*, LNX-1318.

You can find the latest version of this document and of *Linux for zSeries and S/390 Using the Dump Tools* on in the “June 2003 stream” pages of the developerWorks® Web site at:

`ibm.com/developerworks/linux/linux390/overview24.shtml`

If you are routed to the top-level page: On the left navigation bar, click **Kernel 2.4 based**.

How this document is organized

The first part of this document contains general information relevant to all Linux for zSeries and S/390 device drivers.

Parts two and three consist of chapters specific to individual device drivers. (Part two describes the drivers for zSeries and S/390 hardware; part three describes the network device drivers.)

Part four contains information on the commands and parameters used in installing Linux for zSeries and S/390.

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

Who should read this document

This document is intended for system administrators who want to configure a Linux for zSeries or Linux for S/390 system.

License conditions

The 3590 tape discipline module (tape_3590.o) is subject to license conditions as reflected in: “International License Agreement for Non-Warranted Programs” on page 327.

Assumptions

The following general assumptions are made about your background knowledge:

- You have an understanding of Linux, zSeries, and S/390 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 and S/390 devices attached to your system.
-

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.

►► — repeat ————— ►►


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

►► — repeat ————— ►►


Defaults

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

►► ————— ►►


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:

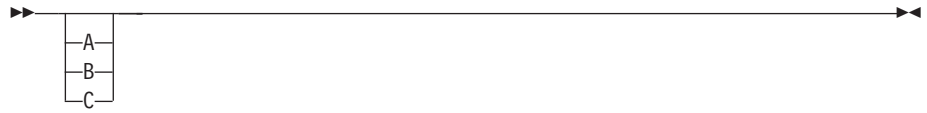
►► ————— ►►


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:

Understanding syntax diagrams



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

Part 1. Device drivers overview

This part describes principles common to different device drivers.

Chapter 1. Common device support

Before Linux for zSeries and S/390 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
3270 console	no	yes	no
DCSS	no	yes	yes
Tape	no	yes	yes
CTC / ESCON [®]	yes	yes	yes
CTCMPC	yes	yes	yes
IUCV	no	yes	yes
LCS	yes	yes	yes
OSA-Express / OSA-Express2 / HiperSockets [™]	yes	yes	yes
CLAW	yes	yes	yes
Cryptographic devices	no	yes	yes
SCSI-over-Fibre Channel	no	yes	yes
z/VM recording	no	no	yes

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

A description of how to build the kernel including device drivers is given in Appendix A, "Building the kernel," on page 299.

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 zip1 parameter file.
- For a VM installation, include the parameter in the PARM LINE A file.

For the format of this file see Chapter 26, "Overview of the parameter line file," on page 297.

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

Common device support

the Linux kernel” on page 225 or “modprobe - Load a module with dependencies into the Linux kernel” on page 227 for the syntax.

Because the zSeries and S/390 architecture differs from that used by the Intel™ PC and other machines, the I/O concepts used by zSeries and S/390 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.

Unlike other hardware architectures, zSeries and S/390 implement 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 and S/390 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 and S/390 architecture allows up to 65,535 devices to be attached.

To avoid the introduction of a new I/O concept to the common Linux code, Linux for zSeries and S/390 preserves the IRQ concept and systematically maps the zSeries and S/390 subchannels to Linux as IRQs. This allows Linux for zSeries and S/390 to support up to 65,535 different IRQs, each representing a unique device.

The unified I/O access method incorporated in Linux for zSeries and S/390 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 and S/390 specific:

get_dev_info_by_irq() / get_dev_info_by_devno()

Allows a device driver to find out what devices are attached (visible) to the system, and to determine their current status. **get_dev_info_by_irq()** takes the IRQ subchannel as the argument. **get_dev_info_by_devno()** takes the device number as the argument.

get_devno_by_irq() / get_irq_by_devno()

Get the device number from a given IRQ subchannel, and vice versa.

get_irq_first() / get_irq_next()

These routines can be used to loop through the list of all IRQ subchannels.

resume_IO()

Resumes suspended I/O operation.

clear_IO()

Reset to prepare for a new **do_IO()** request.

request_irq()

Assigns the ownership of a specific device to a device driver.

free_irq()

Releases the ownership of a specific device.

do_IO()

Initiates an I/O request.

halt_I0()

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_I0()`.

More information on these routines can be found in the Linux source directory (for example, `/usr/share/doc/kernel-doc-2.4.21/s390/cds.txt` for kernel 2.4.21) or the `Documentation/s390/cds.txt` file in the Linux kernel source tree.

proc file system

Some device drivers create working directories in the proc file system when they are loaded and delete them when they are unloaded. For example, the `zfc` driver creates a directory `/proc/scsi/zfc` where it maintains several files that can be used to interact with `zfc` (see “Configuring `zfc`” on page 73).

While a process uses the working directory of a device driver as present working directory (`pwd`), another process can unload the driver. All files in the directory are then deleted and the directory is flagged for deferred deletion. Reloading the device driver creates the files in a new working directory. The files are invisible in the copy of the working directory that is flagged for deferred deletion.

Part 2. Block and character device drivers

The zSeries and S/390 block and character device drivers are:

- Chapter 2, “DASD device driver”
- Chapter 3, “XPRAM device driver”
- Chapter 4, “z/VM discontinuous saved segments device driver”
- Chapter 5, “Console device drivers”
- Chapter 6, “Channel-attached tape device driver”
- Chapter 7, “Generic cryptographic device driver”
- Chapter 8, “SCSI-over-Fibre Channel device driver”
- Chapter 9, “z/VM recording device driver”

Note

For prerequisites and restrictions for these device drivers refer to the kernel 2.4 June 2003 stream pages on developerWorks at:

ibm.com/developerworks/linux/linux390/june2003_recommended.shtml

If you are routed to the top-level page: On the left navigation bar, click **Kernel 2.4** . Then click the stream with which you want to work.

Chapter 2. DASD device driver

DASD overview

The DASD device driver in Linux for zSeries and S/390 takes care of all real or emulated DASD (Direct Access Storage Device) that can be attached to a zSeries or S/390 via the channel subsystem.¹ 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 or S/390 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 IBM[®] TotalStorage[®] 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

The device file system (devfs) provides a comfortable and easy-to-use naming scheme for DASD. (The traditional naming scheme is described in "DASD naming scheme without devfs" on page 10.) 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.

1. SCSI disks attached via a zSeries FCP adapter are not classified as DASD. They are handled by the `zfc` driver (see Chapter 8, "SCSI-over-Fibre Channel device driver," on page 73).

DASD device driver

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 2.4 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.) This naming scheme provides more names than are needed to access the maximum number of DASD accessible.

Table 1. DASD naming convention

Names	Number of names	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.

DASD structures

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 into Linux to support platforms other than zSeries and S/390 will preserve zSeries and S/390 IPL, VTOC and label records.

To ensure compatibility with other zSeries and S/390 operating systems the IBM-label partitioning scheme has been added to Linux. This scheme currently supports VOL1 (zSeries and S/390), LNX1 (Linux) and CMS1 (z/VM) labeled disks, as well as unlabeled disks, which are treated equivalently to LNX1-labeled disks. The disk layout of the different types is shown in Figure 1.

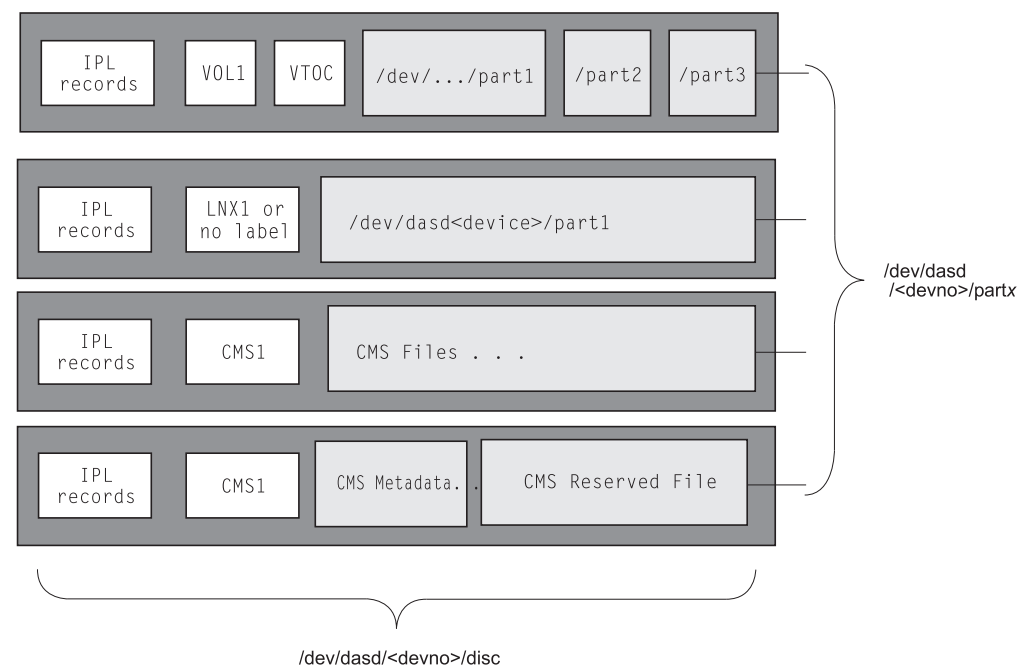


Figure 1. Structure of VOL1, LNX and CMS labeled disks

The first of these examples shows a DASD with the *compatible disk layout* (cdl). The second has been formatted with the Linux standard disk layout (ldl). The first two examples show disks in an LPAR or native mode, or a full-pack minidisk (dedicated DASD) in VM or a normal VM minidisk. These disk layouts apply to any kind of DASD. The third and fourth examples are VM specific.

VOL1 labeled disk:

This disk layout, also known as the compatible disk layout (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' or one format 7 label² depending on the disk size, and one to three 'format 1' labels 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 182).

For more information, see "zSeries and S/390 Compatible disk layout (CDL)" and "DASD partitioning" on page 14.

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 file system. If the volume contains a CMS file system it will be treated equivalently to a LNx labeled volume. If the volume is a CMS reserved volume³ the CMS reserved file is represented by the first and only partition. IPL and label records as well as the metadata of the CMS file system are reserved for access through the 'entire disk' device.

zSeries and S/390 Compatible disk layout (CDL)

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

For restrictions that apply when using the compatible disk layout visit:

ibm.com/developerworks/linux/linux390/june2003_restrictions.shtml

If you are routed to the top-level page: On the left navigation bar, click **Kernel 2.4**. Then click the stream with which you want to work. On the stream page, click **Restrictions**.

You cannot use DASD devices with compatible disk layout directly without partitions (for example, as a LVM physical volumes or for a file system). Directly using the whole device (for example, as /dev/dasdb or /dev/dasd/1234/device) results in data loss. Create a partition (for example, /dev/dasdb1 or /dev/dasd/1234/part1) with fdasd and use the partition instead. Alternatively use the Linux disk layout where this restriction does not apply.

2. The 'format 5' label is required by other operating systems but is unused by Linux and set to zeroes by fdasd.

3. CMS reserved volume means a volume that has been reserved by a 'CMS RESERVE fn ft fm' command.

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 80 byte data area. The contents are:

1. Key

This identifies the block as a volume label. It must contain the four EBCDIC⁴ characters 'VOL1'.

2. Label identifier

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

3. VOLSER (volume serial number)

This field identifies the volume on which the partition resides or will reside. A volume serial number is one to six characters in the EBCDIC code.

Notes:

- a. Entries shorter than 6 characters are padded with blanks.
- b. Volume serial numbers can be generated on Linux using the **dasdfmt** or **fdasd** commands. These commands ensure that volume serial numbers are alphanumeric and contain no special characters other than \$, #, @, and %. Embedded blanks are effectively relocated to the end of the number.
- c. The use of special characters is supported for compatibility reasons, but they should be avoided if possible.
- d. VOLSERS created by other operating systems can also contain other special characters or lowercase letters.
- e. A VOLSER containing special characters must normally be enclosed in apostrophes when specified as a command parameter, for example.
- f. The VOLSER values SCRTCH, PRIVAT, MIGRAT or Lnnnnn (L with five digits) are reserved in other zSeries and S/390 operating systems.

Refer to "Accessing DASD by VOLSER" on page 22 for a discussion of access to DASD volumes by VOLSER.

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).

Volume Table of Contents (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, z/VM or VSE/ESA.

4. The conversion to EBCDIC will be carried out by the **fdasd** tool.

DASD device driver

The VTOC can be displayed with standard zSeries and S/390 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 ***
LINUX.VLNX001.PART0001.NATIVE          0    0  1    0  1    1,1
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=RGF/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.

DASD partitioning

For the purpose of this chapter, the term *partitioning* refers to creating one or more partitions on a DASD with the compatible disk layout (CDL). A partition is a contiguous set of DASD blocks that is treated by Linux as an independent disk. The partition table defines the extents of partitions on a DASD.

On ECKD disks that have been formatted with CDL, you can have up to three partitions. If a DASD is formatted in the normal Linux disk layout (dasdfmt option -d ldl), it is not possible to create multiple partitions on it, and the entire DASD must be accessed as a single partition.

Why use partitions?

There are several reasons why you may want to partition a DASD containing 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.

The partition table

The partition table is an index of all the partitions on a DASD. In Linux for zSeries and S/390 the normal Linux partition table is not used. Rather, as in other zSeries and S/390 operating systems, a VTOC (Volume Table Of Contents) is used to store this index information (see “Volume Table of Contents (VTOC)” on page 13). The VTOC contains pointers to the location of every data set on the volume. In Linux for zSeries and S/390 these data sets form the Linux partitions.

Partitioning a DASD

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

1. Use the `dasdfmt` tool (see “`dasdfmt` - Format a DASD” on page 168) with the (default) `'-d cd1'` option to format the DASD with the IBM compatible disk layout.
2. Use the `fdasd` tool (see “`fdasd` - DASD partitioning tool” on page 182) 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 233) 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. When you have created a file system, you may mount the partition to the mount point of your choice in Linux.

See also “DASD - Preparing for use” on page 16.

Partitioning best practices

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

- You are advised to use `fdasd` to create or alter partitions, because 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, because `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.

DASD – Preparing for use

1) Low-level format

Before using an ECKD type DASD as a Linux for zSeries and S/390 disk, the device must be formatted. This should be done from Linux for zSeries and S/390 by issuing an `ioctl` called `BIODASDFMT` on the file descriptor of the opened volume `/dev/dasd/<devno>/<device>` or `/dev/dasd<device letter>` without `devfs`. 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 and S/390 system, forcing you to reinstall from scratch.

See the help given by `dasdfmt -help` and “`dasdfmt` - Format a DASD” on page 168 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

If a device has been formatted with the compatible disk layout (CDL) (which 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 “DASD structures” on page 11) 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 182 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 compatible disk layout, you cannot further partition the device. In this case only one partition per DASD is supported.

See also “DASD partitioning” on page 14.

3) Make a file system

Before using a DASD as a Linux for zSeries and S/390 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>`, or `/dev/dasd<device letter><partition number>`, for example, `/dev/dasda1`), rather than the whole volume.

Note: The blocksize of the file system must not be lower than that given to the `dasdfmt` command.

Recommendation: 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 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 specification.

The DASD device driver is also known to work with these devices:

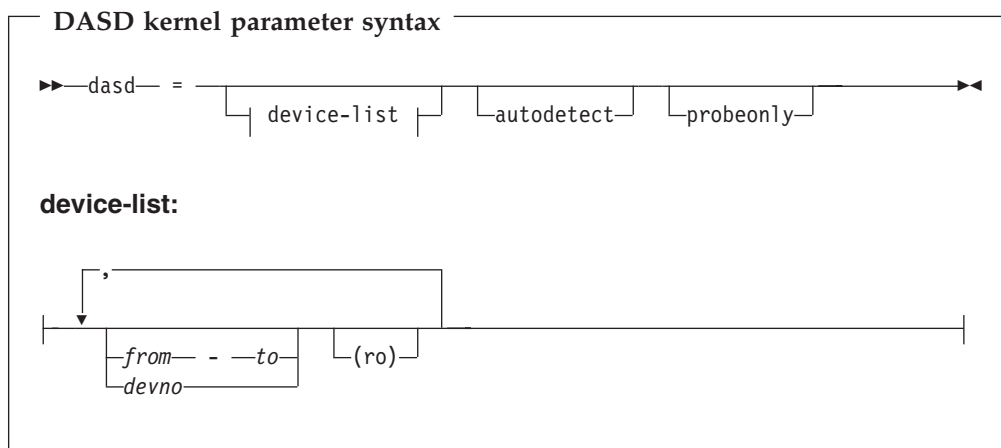
- Multiprise internal disks
- RAMAC
- RAMAC RVA
- IBM TotalStorage Enterprise Storage Server virtual ECKD-type disks

Linux for zSeries and S/390 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 11.

There is hotplug support for DASD (see Appendix B, “Hotplug support,” on page 317). If you have the required hotplug package and agent installed, the device nodes for all DASD devices present are created.

DASD kernel parameter syntax

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



DASD device driver

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.

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.⁵

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 yield confusing results if you change your I/O configuration between two IPLs, or if you are running as a guest operating system in VM 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)
```

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

```

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)

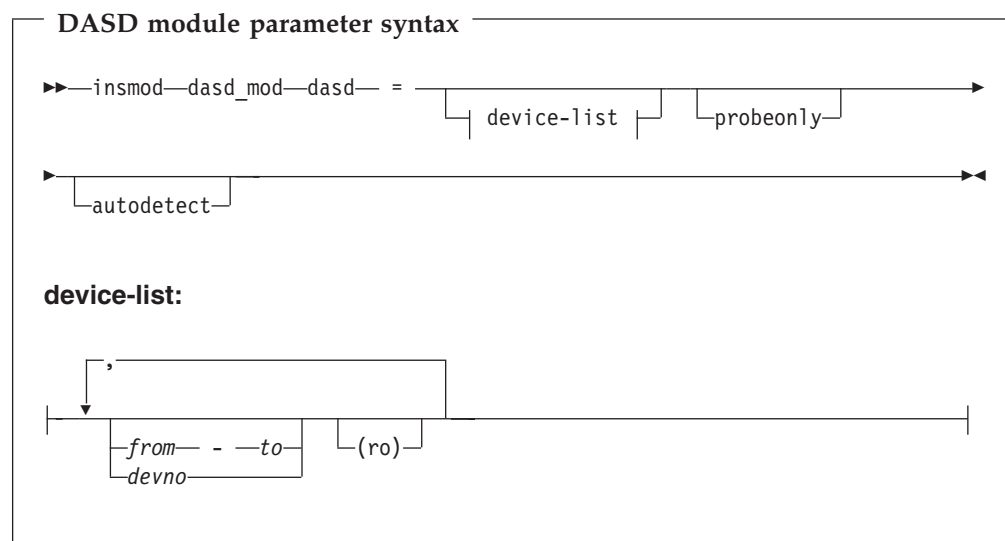
```

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

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

DASD module parameter syntax

The following are the DASD driver module parameters:



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 17.

DASD module example

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

The details are the same as "DASD kernel example (using devfs)" on page 18.

DASD dynamic attach and enable/disable

Dynamic device attach enables Linux for zSeries and S/390 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.

The /proc file system 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 accessed 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.

In these commands, *devno-range* follows this syntax:

devno-range:



Note: Option (ro) is applicable to **add** only.

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, for example, 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)

Note: Use these functions with care because you might block others. You can block entire users or even worse the entire operating system 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 operating system does not realize it.

Accessing boxed DASD

To use the DASD reserve and release (locking) functionality in a complex cluster environment, it is necessary to get access to a DASD even if it is locked by another system during startup. Such a DASD is referred to as "externally locked" or "boxed".

A method is implemented to break the external lock and get the device online afterwards. This means that the lock of the external system is broken with the "unconditional reserve" command, which might cause unpredictable situations in the system that previously had the lock.

The operator can do the following:

- Detect a DASD that was externally locked during IPL.
- Unconditionally reserve a DASD that is externally locked, even if it could not be analyzed (to do so, the operator must know the device number and discipline type).

Note: This functionality is implemented only for ECKD controllers (not FBA/DIAG).

User interfaces

The operator can monitor and control externally locked devices by reading and writing the `procfs` entry (`/proc/dasd/devices`). (It is not possible to use the `ioctl` interface because the device is not online and cannot be opened.)

DASD device driver

To detect externally locked devices, the status "boxed" is shown for the related device entry in `/proc/dasd/devices`. This indicates that the device was externally locked during startup.

Example:

```
> cat /proc/dasd/devices
7000(ECKD) at ( 94: 0) is dasda      : boxed
7001(ECKD) at ( 94: 4) is dasdb      : n/f
```

To alter the status of the boxed DASD, the command

```
echo "brk <devno> <discipline>" > /proc/dasd/devices
```

will force the device driver to unconditionally reserve the device even if it could not be analyzed before. If it is possible to break the reserved state, the device will be analyzed automatically when the unconditional reserve is successful. This means that no further action need be taken, and the device will be online, reserved, and fully accessible.

If it is not possible to break the reserve for any reason (such as a timeout), the device will remain in the boxed state.

Example:

```
> echo "brk 7000 eckd" > /proc/dasd/devices
> cat /proc/dasd/devices
7000(ECKD) at ( 94: 0) is dasda:active at blocksize: 4096, 601020 blocks, 2347 MB
7001(ECKD) at ( 94: 4) is dasdb      : n/f
```

The lock has been broken and the device is now enabled for normal access.

Accessing DASD 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 file name 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. A device node is created only if the VOLSER is alphanumeric or contains no special characters other than '\$', '@', '#', and '%'. **dasdfmt** and **fdasd** enforce this requirement and furthermore force letters to uppercase.
2. The specification of a device node in `/dev/labels` is case-sensitive.

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 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

BIODASDCMFENABLE

enables channel measurement and resets measurement data

BIODASDCMFDISABLE

disables channel measurement

BIODASDFMT

formats the device with a specified blocksize.

DASD device driver

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.

BIODASDREADALLCMB

reads channel measurement data

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 168) and `fdasd` ("`fdasd` - DASD partitioning tool" on page 182).

DASD Channel Measurement Facility

Description

The S/390 architecture provides a channel measurement facility for collecting statistical data about ongoing I/O-activity on the channel subsystem. This data can be accessed by the operating system kernel, which provides an interface to control the data collection and to read collected values for logical devices. This allows analysis of measurement data from user space applications.

For the Linux 2.4 kernel, this interface is provided only for DASD devices.

User Interfaces

The interface to user space applications consists of a set of *I/O control calls* into the DASD driver. These I/O calls are available only when channel measurement is enabled and the channel measurement facility driver is loaded.

The interface is defined in the `cmb.h` header file, which is part of the kernel and of any user space program using this interface:

```
struct cmbdata {
    __u64 size;
    __u64 elapsed_time;
    /* basic and extended format: */
    __u64 ssch_rsch_count;
    __u64 sample_count;
    __u64 device_connect_time;
    __u64 function_pending_time;
    __u64 device_disconnect_time;
    __u64 control_unit_queuing_time;
    __u64 device_active_only_time;
```

DASD device driver

```
/* extended format only: */
__u64 device_busy_time;
__u64 initial_command_response_time;
};

/* enable channel measurement */
#define BIODASDCMFENABLE _IO(DASD_IOCTL_LETTER,32)
/* disable channel measurement */
#define BIODASDCMFDISABLE _IO(DASD_IOCTL_LETTER,33)
/* read channel measurement data */
#define BIODASDREADALLCMB _IOWR(DASD_IOCTL_LETTER,33,struct cmbdata)
```

Each ioctl call is performed on a device node for a DASD device.

Return values

ENODEV

The DASD device is not accessible.

EINVAL

Invalid parameter: The channel measurement facility driver is not loaded or attempts to access measurement blocks while measurement is not activated for that device.

EBUSY/O

I/O activity prevents activation or deactivation of channel measurement.

ENOMEM

Out of memory. Possibly the maximum number of measured devices needs to be adjusted.

cmbdata argument

The argument of the BIODASDREADALLCMB function is a pointer to a struct cmbdata memory block. The format of this structure corresponds loosely to the layout of the hardware channel measurement blocks (for details, see *z/Architecture™ Principles of Operation*, SA22-7832-02, section “Measurement Block” in chapter 17). All values are extended to 64-bit in order to simplify compatibility among different CMB layouts in hardware and between 31- and 64-bit Linux kernels.

Two fields are added at the beginning of the cmb structure:

size The size of the valid data of struct cmbdata in bytes: This can either have the value 72 (basic format) or 88 (extended format).

elapsed_time

Time interval between activating channel measurement for the subchannel and reading the measurement block. All time values are in units of nanoseconds, not milliseconds as used by the measurement facility.

A value overflow may occur in the measurement facility. The kernel will not protect you from reading incorrect data after an overflow. As ssch_rsch_count and sample_count only have a resolution of 16-bit when using the basic format, applications should call BIODASDCMFENABLE at a reasonable interval to avoid this.

Kernel/Module Builder's Interface

When the DASD driver is selected, the option Support for Channel Measurement on DASD devices can be selected. It is possible to build the driver either as loadable module, or as built-in module when the DASD driver is integrated into the kernel.

Selecting the latter option has no effects other than a slight increase of the kernel image size as all measurements are disabled by default.

The two modules created are `cmf.o` for the low-level code, and `dasd_cmb.o` for the interface code. The `cmf.o` module has two parameters:

format

A numeric value, which can either be 0 or 1, indicating the basic or extended format for hardware channel measurement blocks. The default value is -1, indicating autodetection. In autodetection mode, the extended format for channel measurement blocks is chosen when a z900 or newer machine is found. In most cases, autodetection is sufficient.

maxchannels

Maximum number of devices that can be measured with the basic format. Using the basic CMB format, no more than `maxchannels` device can be enabled. The default value is 1024, and the kernel restricts memory allocation to a maximum of 4096 channels. Using the extended CMB format, this parameter is ignored and there is no restriction of the number of concurrently measured devices.

If the `cmf.o` is built into the kernel, the parameters can be specified via the kernel command line using the `cmf_format` and `cmf_maxchannels`.

Chapter 3. XPRAM device driver

The zSeries architecture in 31-bit mode and the S/390 architecture support 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 and S/390 architectures allow applications to access up to 18 EB (exabytes) and 16 TB (terabytes) of expanded storage, respectively (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 and S/390 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 and S/390 to access the expanded storage. Thus XPRAM can be used as a basis for fast swap devices and/or fast file systems.

XPRAM features

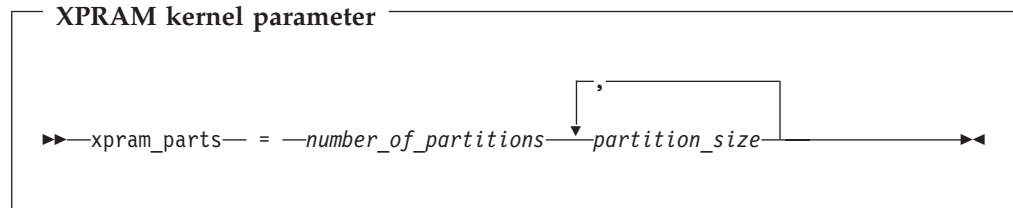
- Automatic detection of expanded storage.
(If expanded storage is not available, XPRAM fails with a log message reporting the lack of expanded storage.)
- Storage can be subdivided into up to 32 partitions.
- Device driver major number: 35.
- Partition minor numbers: 0 through 31.
- Hard sector size: 4096 bytes.
- The device file system (devfs) is supported. If devfs is switched on during kernel build XPRAM automatically generates the device nodes `/dev/s1ram/0` through `/dev/s1ram/31`.

Note on reusing XPRAM partitions

It is possible to reuse the file system 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 file system (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:



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.

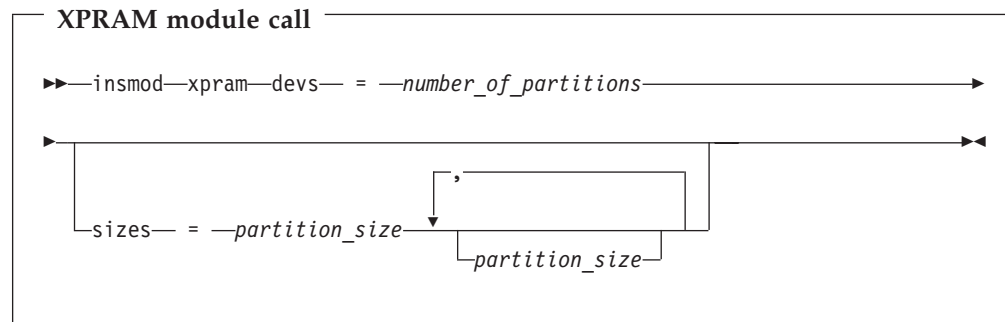
XPRAM kernel example

```
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:



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 4. z/VM discontinuous saved segments device driver

The z/VM discontinuous saved segments (DCSS) device driver provides disk-like fixed block access to z/VM discontinuous saved segments.

The device driver facilitates:

- Initializing and updating ext2 compatible file system images in z/VM saved segments.
- Implementing a shared read-write RAM disk for Linux guests, for example, for a file system that can be shared among multiple Linux images that run as guest systems under the same z/VM.

Related information:

- For z/VM information see *CP Command and Utility Reference*, SC24-6008.
- For information on DCSS see *z/VM Saved Segments Planning and Administration*, SC24-6056

Building a kernel with the DCSS device driver

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

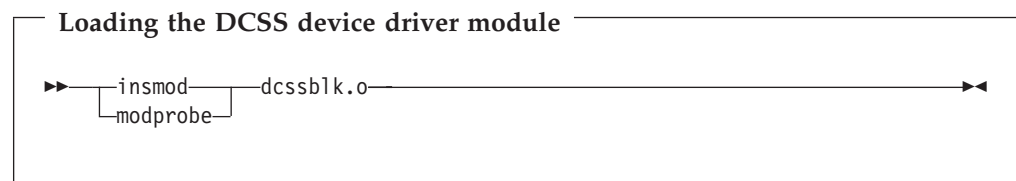
```
Block device drivers
--> z/VM discontinuous saved segments (DCSS) device driver
```

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

Setting up the DCSS device driver

The DCSS device driver does not have parameters in the parameter line file. Accordingly, there are no parameters for loading it as a module.

If your DCSS device driver has been compiled as a module, you can load it with **insmod** or **modprobe**.



You configure the DCSS device driver through a /proc interface. You can add and remove DCSS devices. If you are not using the device file system, you need to create device nodes for your DCSS devices.

Adding a DCSS

Before you start:

- You need to have set up a DCSS on z/VM and know the name assigned to the DCSS on z/VM.

- You must have set the mem kernel parameter to cover the upper limit of the DCSS.
- You cannot concurrently access overlapping DCSSs.
- You cannot access a DCSS that overlaps with your guest virtual storage.

To add a DCSS device write the name of the DCSS to `/proc/dcssblk/add`.

Example: To add a DCSS called “MYDCSS” issue:

```
echo "MYDCSS" > /proc/dcssblk/add
```

If you are using the device file system, a corresponding entry of the form `/dev/dcssblk/<dcss name>` is created for each device you add. You can then use these entries as block devices. For example, the device node for a DCSS named “MYDCSS” would be `/dev/dcssblk/MYDCSS`.

If you are not using the device file system, you need to create a node for the device (see “Creating DCSS device nodes”).

Querying the major and minor device number of a DCSS

All DCSS devices on a particular Linux system have the same major device number. You can find the major number in `/proc/devices` preceeding the driver name “dcssblk”.

You can find the names and the corresponding minor numbers of the currently loaded DCSS in `/proc/dcssblk/list`.

Example:

```
# cat /proc/dcssblk/list
0 MYDCSS
```

The format of the output is `<minor>TAB<name>`.

Creating DCSS device nodes

If the device file system is not used, you need to create the device nodes with the **mknod** command. Issue a command like this:

```
mknod /dev/<nodename> b <major> <minor>
```

Where `<major>` and `<minor>` are the major and minor device numbers (see “Querying the major and minor device number of a DCSS”), `<nodename>` is the name you assign to the node, and `b` specifies that the node represents a block device.

Example: to create a device node `/dev/dcssblk0` with major number 253 for a segment that has the minor number 0, issue:

```
mknod /dev/dcssblk0 b 253 0
```

Removing a DCSS

To remove a DCSS device write the name of the DCSS to `/proc/dcssblk/remove`.

Example: To remove a DCSS called “MYDCSS” issue:

```
echo "MYDCSS" > /proc/dcssblk/remove
```

If you are using the device file system, the corresponding device file system entry is removed for you.

If you are not using the device file system, you can keep nodes that you have created. If you are keeping nodes for reuse, be aware that the major number of the device might change when you unload and reload the DCSS device driver. When the major number of your device has changed existing nodes become unusable.

Note: DCSSs can only be removed when they are not in use.

Removing DCSS device nodes

Use the **rmnod** command to remove a device node.

Example: to remove a device node `/dev/dcssblk0` issue:

```
rmnod /dev/dcssblk0
```

Note: DCSS device nodes can only be removed when the corresponding DCSS is not in use.

Working with the DCSS device driver

The following sections describe how you can:

- Toggle between two different write access modes
- Save changes to a DCSS

Setting the access mode

There are two possible write access modes to the DCSS:

shared

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

exclusive-writable

In the exclusive-writable mode you write to a private copy of the DCSS. The private copy is writable, even if the original DCSS is read-only. The changes in your private copy are invisible to other guests until you save the changes (see “Saving an updated DCSS” on page 36).

After saving the changes all guests that open the DCSS access the changed copy. z/VM retains a copy of the original DCSS for those guests that continue accessing it, until the last guest has stopped using it.

For either access mode the changes are volatile until they are saved.

For each DCSS that you have added, there is a /proc file entry through which you control the access mode. The entry is called /proc/dcssblk/<name>/shared where <name> is the name of the DCSS.

To find out the current access mode for a DCSS read the contents of the corresponding /proc entry.

Example: To find out the access mode for a DCSS “MYDCSS” issue:

```
cat /proc/dcssblk/MYDCSS/shared
```

If the command returns “1” the access mode is shared; if the command returns “0”, the access mode is exclusive-writable.

To change from the shared to the exclusive-writable mode write the value 0 to the /proc entry.

Example: To access a DCSS “MYDCSS” in exclusive-writable mode issue:

```
echo 0 > /proc/dcssblk/MYDCSS/shared
```

To change from the exclusive-writable to the shared mode write the value “1” to the /proc entry.

Example: To access a DCSS “MYDCSS” in shared mode issue:

```
echo 1 > /proc/dcssblk/MYDCSS/shared
```

Saving an updated DCSS

For each DCSS that you have added, there is a /proc entry through which you can place a save request for saving the changes permanently on the spool disk.

To save an updated DCSS write “1” to /proc/dcssblk/<name>/save where <name> is the name of the DCSS.

Example: To save changes to a DCSS “MYDCSS” issue:

```
echo 1 > /proc/dcssblk/MYDCSS/save
```

Saving is delayed until you close the device.

You can check if a save request is waiting to be performed by reading the contents of the /proc entry.

Example: To check if a save request exists for a DCSS “MYDCSS” issue:

```
cat /proc/dcssblk/MYDCSS/save
```

If the entry contains “1”, a save request is waiting to be performed. You can purge a save request by writing “0” to the corresponding /proc entry.

Example: To purge a save request for a DCSS “MYDCSS” issue:

```
echo 0 > /proc/dcscblk/MYDCSS/save
```

Chapter 5. Console device drivers

The Linux for zSeries and S/390 console device drivers allow you to use a zSeries- or S/390-machine's hardware console for basic Linux control, for example, for booting Linux. The device drivers accesses the hardware console through the SCLP (service-call logical processor) interface.

If Linux is running under VM or on a P/390, you can use a 3215 or a 3270 terminal instead of the hardware console.

Note that “terminal” and “console” have special meanings in Linux, which should not be confused with the zSeries and S/390 usage.

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

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

The Linux console and the Linux terminals are different applications which both run on zSeries and S/390 *terminals*. Linux terminals are devices through which users interact with Linux and Linux applications. The Linux console is a device that handles user interactions with the Linux kernel. The Linux console traffic is directed to one of the Linux terminals.

In this chapter, we use console and terminal to refer to a Linux console or terminal, unless indicated otherwise.

Console features

- Provides a line mode or full-screen mode typewriter terminal.
- Provides Linux console output on a designated terminal.
- Supports a 3215 terminal for the United States code page (037)

Console usage

The possible uses of the console device drivers depends on the mode of the terminal, line mode or full-screen mode. The 3215 and 3270 console devices driver always provides a line-mode terminal. The hardware console device driver can provide a line-mode terminal or a VT220-like full-screen mode terminal.

For line-mode terminals

The line-mode terminals are primarily intended for booting Linux. The recommended user access to a running Linux for zSeries and S/390 is through a networked terminal emulation such as telnet or ssh.

Tip: If the terminal does not provide the expected output, ensure that `dumb` is assigned to the `TERM` environment variable. For example, issue the following command on the bash shell: `export TERM=dumb`.

For full-screen mode terminals

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

Tip: If the terminal does not provide the expected output, ensure that `linux` is assigned to the `TERM` environment variable. For example, issue the following command on the bash shell: `export TERM=linux`.

To set `TERM=linux` at startup add a line like this to the `/etc/inittab` file:

```
<id>:2345:respawn:/sbin/agetty -L 9600 ttyS1 linux
```

where:

`<id>`

is a unique identifier for the entry in the `inittab` file.

Be sure not to provide multiple entries for `ttyS1`. For more details see the man page for the `inittab` file.

Enabling a terminal for user log-ins

To allow user log-ins from a terminal, add a line like this to the `/etc/inittab` file:

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

where:

`<id>`

is a unique identifier for the entry in the `inittab` file.

`<dev>`

is the device node name.

Your Linux system's `/etc/inittab` file might already have an entry for a terminal. Be sure not to provide multiple entries for the same device. See Table 3 on page 41 for the device node names. If an existing entry uses a different name and you are not sure how it maps to the names of Table 3, you can comment it out and replace it. When referring to the device node in a command or parameter, always use the names of Table 3.

For more details see the man page for the `inittab` file.

Example: To enable a device `ttyS0` for user log-ins specify, for example:

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

Device names and nodes

Table 3 summarizes the supported devices, with their internal names, and device numbers:

Table 3. Supported consoles

Device driver	Internal name/ Node name	Major	Minor
Line-mode hardware console	ttyS0	4	64
Full-screen mode hardware console	ttyS1	4	65
3215 terminal	ttyS0	4	64
3270 terminal	tty3270	227	0
Note: The specifications for the line-mode hardware console and the 3215 device are identical. The device name and numbers are assigned to whatever device is present or to the device you specify with the conmode parameter (“conmode parameter” on page 42). You cannot have both devices simultaneously.			

Note that there are different options that must be selected during the kernel configuration to enable the devices of Table 3 as Linux terminals or consoles (see the character device section in “Kernel configuration options” on page 303).

To make a 3270 device driver available to other programs you have to create a device node. Unless you are using devfs, you also have to create a device node for the 3215 and hardware console drivers.

To create a device node issue a Linux shell command like this:

```
mknod /dev/<device node name> c <major> <minor>
```

where:

<device node name>

specifies the internal name of the device driver.

c specifies that the device is a character device.

<major> <minor>

specifies the major and minor number to be used for the device.

Example: The following Linux shell command creates a device node at /dev/tty3270:

```
mknod /dev/tty3270 c 227 0
```

Console kernel parameter syntax

There are three kernel parameters for the console:

conmode to override the default console of an environment.

console to activate a console and designate a *preferred* console.

condev to define a 3215 or 3270 device to a P/390.

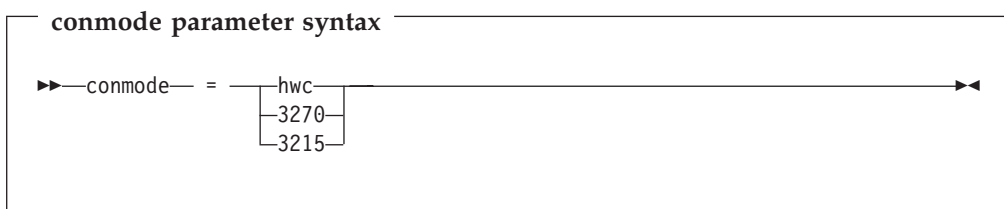
conmode parameter

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

Table 4. Default console device driver

LPAR or native	hardware console device driver
VM	3215 or 3270 console device driver, depending on the guest's console settings (the CONMODE field in the output of #CP QUERY TERMINAL).
P/390	3215 console device driver

Use the conmode parameter to override the default.



conmode parameter example

To use the 3215 terminal in a z/VM environment specify:

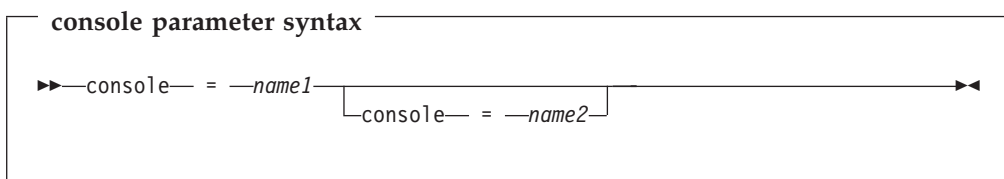
```
conmode=3215
```

console parameter

The console parameter applies to the hardware console only.

Only active consoles receive Linux operating system messages and only one console can be the *preferred* console. The preferred console is used as initial input and output device, beginning at the stage of the boot process when the 'init'-script is called. Messages issued by programs that are run at this stage are therefore only displayed on the preferred console. On default, ttyS0 is active and used as the preferred console.

Use the console parameter to activate ttyS1 instead of ttyS0, or to activate both ttyS0 and ttyS1 and designate one of them as the preferred console.



Where:

name1

is the internal name of the active console (ttyS0 | ttyS1).

name2

is the internal name of the second console if both are to be active (ttyS0 | ttyS1). If both consoles are specified, this is the preferred console.

console parameter examples

To activate the full-screen console device driver instead of the line-mode console driver, add the following line to the kernel command line:

```
console=ttyS1
```

To activate both consoles, provide a specification for both drivers, for example:

```
console=ttyS0 console=ttyS1
```

The last statement determines the preferred console, ttyS1 in the example.

condev parameter

This kernel parameter applies only to 3215 and 3270 console device drivers if used on a P/390. This supplies the device driver with the device number of the 3215 or 3270 device. The reason that this parameter is needed is that there is no guaranteed method of recognizing a 3215 or 3270 device attached to a P/390.

condev parameter 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 do not accept this parameter in hexadecimal form.

condev parameter examples

```
condev=0x001f
```

or

```
condev=31
```

Both of these formats tell the device driver to use device number hex 1F for the 3215 or 3270 terminal.

Using the console

The console can be a line-mode terminal or a full-screen mode terminal.

On a full-screen mode terminal, pressing any key immediately results in data being sent to the TTY routines. Also, terminal output can be positioned anywhere on the screen. This allows for advanced interactivity when using terminal based applications.

On a line-mode terminal, the user first types a full line and then presses Enter to let the system know that a line has been completed. The device driver then issues a read to get the completed line, adds a new line and hands over the input to the generic TTY routines.

Console special characters on line-mode terminals

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

Table 5. Control characters

For the key combination	Type this	Usage
Ctrl+C	^c	Cancel the process that is currently running in the foreground of the terminal.
Ctrl+D	^d	Generate an end of file (EOF) indication.
Ctrl+Z	^z	Stop a process.
n/a	^n	Suppresses the automatic generation of a new line. This makes it possible to enter single characters, for example those characters that are needed for yes/no answers in the ext2 file system utilities.

If you are running under VM without a 3215 console you will have to use the CP VINPUT command to simulate the Enter and Spacebar keys.

The Enter key is simulated by entering:

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

The Spacebar key is simulated by entering:

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

Magic sysrequest function

The two characters ^~ followed by a third character invoke the so called “magic sysrequest” function on systems where the respective kernel option CONFIG_MAGIC_SYSRQ has been activated (see “CONFIG_MAGIC_SYSRQ parameter” on page 315). 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 possible sequences are:

Table 6. Magic sysrequest commands

Enter	To
^~b	Re-IPL immediately.
^~s	Emergency sync all file systems.
^~u	Emergency remount all mounted file systems read-only.
^~t	Show task info.
^~m	Show memory.
^~ followed by a digit (0 to 9)	Set the console log level.
^~e	Terminate all tasks.
^~i	Kill all tasks except init.

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 vmpoff="MSG OP REBOOT"#IPL 290"
```

the actual commands received by CP will be:

```
CP HALT
CP IPL 290 PARM vmpoff="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
```

Hardware console

The following applies to both the line-mode terminal and the full-screen mode terminal on the hardware console:

- There can only be one active terminal session on an HMC.
- Security hint: Always end a terminal session by explicitly logging off (for example, type "exit" and press Enter). Simply closing the terminal window leaves the session active and the next user opening the window resumes the existing session without a logon.

Console device drivers

- Slow performance of the hardware console is often due to a busy console or increased network traffic.

The following applies to the full-screen mode terminal only:

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

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 (as a line-mode terminal, full-screen mode is not supported), it is important to consider how the input is handled. Instead of writing into the suitable field within the graphical user interface at the service element or HMC you have to use the VInput command provided by VM. The following examples are written at the input line of a 3270 terminal or terminal emulator (for example, x3270).

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`

Chapter 6. Channel-attached tape device driver

The 3590 tape discipline module ('tape_3590.o') is subject to license conditions as reflected in: "International License Agreement for Non-Warranted Programs" on page 327.

The Linux for zSeries and S/390 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 zSeries and S/390 tape devices. The driver allocates major numbers dynamically and lists them on initialization. Major number 254 is allocated for both the character and the block device front-ends, unless 254 is already occupied by another device, for example by a DASD.

```
# cat /proc/devices
Character devices
...
254 tape
...
Block devices
...
254 tBLK
```

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. There is also hotplug support for channel attached tapes (see Appendix B, "Hotplug support," on page 317). With the support enabled, your system adds tape devices and removes them as they are connected or disconnected by VM.

If Linux 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_34xx.o`
- `tape_3590.o`

Tape driver as a kernel module

When the tape driver (`tape390`) is built as a kernel module, discipline `tape_34xx` is also built as a kernel module. When the `tape390` kernel module initializes, it tries to load all known disciplines. Therefore, the user need not load the modules for each discipline separately.

```
> modprobe tape390
> lsmod
Module                Size  Used by
tape_3590              12384  0 (autoclean)
tape_34xx              1472  0 (autoclean) (unused)
tape390                46384  0 [tape_3590 tape_34xx]
```

Tape driver built into the kernel

The tape driver (`tape390`) and discipline module `tape_34xx` can also be built directly into the kernel. Any disciplines built or supplied as modules must be loaded separately:

```
> modprobe tape_3590
> modprobe tape_34xx
```

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 to access the tape devices sequentially (the traditional way) without any caching done in the kernel. There are two device nodes for each tape device. The nodes differ in what is done when the device is closed. Either the tape is rewound (rewind drive) or left at the current position (non-rewind drive).

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 four names given to the first two devices are:

- `/dev/rtibm0`
- `/dev/ntibm0`
- `/dev/rtibm1`
- `/dev/ntibm1`

Here `ntibm1`, for example, is the non-rewind device (prefix `n`) for the physical device with device number 1(suffix 1).

The minor number for the non-rewind device is the tape device number of `/proc/tapedevices` multiplied with 2. The minor number for the rewind device is the non-rewind number +1. If the device nodes do not exist, they can be created with `mknod`.

For example, if the output of `cat /proc/tapedevices` is:

```
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

Assuming that the major number for tape devices is 254, the corresponding device nodes can be created like this:

```
mknod /dev/ntibm0 c 254 0
mknod /dev/rtibm0 c 254 1
mknod /dev/ntibm1 c 254 2
mknod /dev/rtibm1 c 254 3
```

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 and S/390 platforms. (This is similar to the way most Linux software distributions are shipped on CD using the ISO9660 file system.)

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.

Channel-attached tape device driver

The device nodes have the same minor as the matching non-rewinding character device.

It is advisable to use only the ISO9660 file system on Linux for zSeries and S/390 tapes, since this file system is optimized for CD-ROM devices, which – just like 3480, 3490, or 3590 tape devices – cannot perform fast seeks.

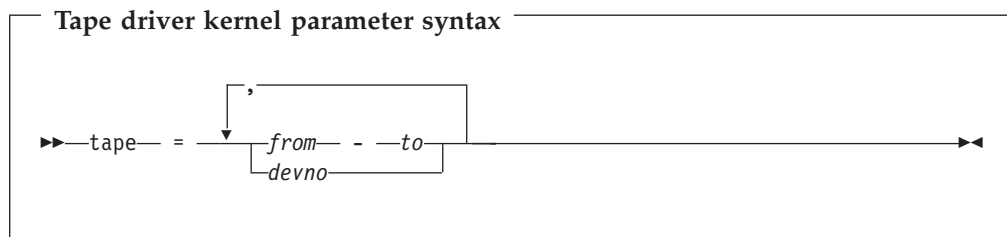
The ISO9660 file system image file need not be the first file on the tape but can start at any position. The tape must be positioned at the start of the image file before the mount command is issued to the tape block device.

Tape driver kernel parameter syntax

You do not need to give the tape device driver parameters if:

- You are using devfs
- You are using auto-detection

Auto-detection without devfs does not guarantee the same device number for a particular device when Linux is rebooted. If you are not using devfs and you want to be sure that your tape devices are assigned the same device numbers each time you boot Linux, specify the physical tape devices to be used you must configure the tape driver by adding a parameter to the kernel parameter line:



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.

Do not specify more than 128 devices in the parameter line as this is the maximum number of devices manageable by the driver.⁶

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

Starting from 0, the tape driver assigns tape numbers sequentially in the order in which the devices are specified in the command line. Device specifications beyond 128 devices are ignored. The assigned numbers are retained even if device is not available so that the numbers of the visible tape devices are not necessarily contiguous and do not necessarily start with tape number 0.

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, 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:

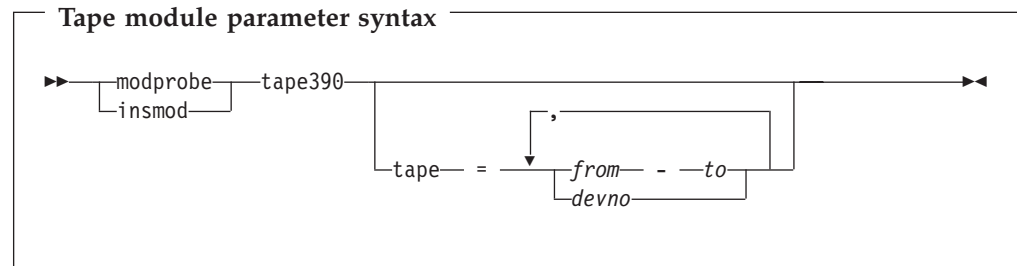
0181	<i>will be</i>	/dev/ntibm0	/dev/rtibm0	/dev/btibm0
0182	<i>will be</i>	/dev/ntibm1	/dev/rtibm1	/dev/btibm1
0183	<i>will be</i>	/dev/ntibm2	/dev/rtibm2	/dev/btibm2
0184	<i>will be</i>	/dev/ntibm3	/dev/rtibm3	/dev/btibm3
019f	<i>will be</i>	/dev/ntibm4	/dev/rtibm4	/dev/btibm4

If devfs is used the device names will be:

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

Tape driver module parameter syntax

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



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.

Channel-attached tape device driver

There is hotplug support for channel attached tape devices (see Appendix B, “Hotplug support,” on page 317). If you have the required hotplug package and agent installed, the driver creates the device nodes for all present tape devices when the device driver is loaded.

Tape driver module example

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

The details are the same as “Tape driver kernel example” on page 53.

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 expels the tape from the drive. Although the drives do not support a load command, MTUNLOAD inserts the next tape in the stacker if the stacker mode of operation is automatic and another tape is present. The stacker mode can be “automatic” to always attempt to load a new tape on unload, “system” to let the operating system handle loading, or “manual” to always let an operator handle tape loads.
- 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
```

where <tape> is the device name, for example, /dev/ntibm0

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.
- MTIOCGGET 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 58).

/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 and also enables selected changes. 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 not in use and is currently available (to any operating system image in a shared environment)

IN_USE

Device is being used by a process on this Linux image

ASSIGN

Device is not in use but has been exclusively reserved by this Linux image

BOXED

Device has been exclusively reserved by another operating system image (in a shared environment) and is unavailable to this Linux image

NOT_OP

Device is not operational

Op Tape operation which is currently active, for example:

Channel-attached tape device driver

--- 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

Boxing tape devices

The tape driver assigns a tape device to a Linux image on open. When it is closed, other operating system images in a shared environment might access the data. Boxing enables user space applications to protect their data from access by other images by reserving a tape device after closing it.

The `/proc/tapedevices` interface can be used to box (assign) and release (unassign) a tape drive. The generic format is:

```
echo "command device" >/proc/tapedevices
```

where *command* can be one of the following:

assign causes the specified tape device to be assigned to this Linux image. As seen by this Linux image:

- The state changes from 'UNUSED' to 'ASSIGN' for an operational device that is currently not being used by this Linux image.
- The state remains 'IN_USE' for a device that is currently being used by this Linux image. However, instead of changing to 'UNUSED' the state changes to 'ASSIGN' when the device is closed.
- The states of devices that are 'BOXED' or 'NOT_OP' remain unchanged.

A device with one of the states 'IN_USE' or 'ASSIGN' on one Linux image has the state 'BOXED' on all other Linux images that share it.

unassign

is effective only for a tape device that has previously been assigned to this Linux image with **assign**. As seen by this Linux image:

- The state changes from 'ASSIGN' to 'UNUSED' for a device that is currently not being used.
- The state remains 'IN_USE' for a device that is currently being used. However, instead of changing to 'ASSIGN' the state changes to 'UNUSED' when the device is closed.

While a tape device is assigned to one operating system image, other images receive an 'assigned elsewhere' unit check if they attempt to assign the device to themselves. This is reflected in the `/proc/tapedevices` output by a 'BOXED' state. The 'BOXED' state is cleared when the operation is retried and the other image has

released the device. There is no notification from the other image, so any application has to manually retry a failed operation.

If an image crashes or reboots while it has assigned a device, the device remains boxed for all other images. An image that had a device assigned across a reboot can release the device by:

- Opening and closing the device, for example by issuing:

```
mt -f <tape> status
```

where <tape> is the device name, for example, /dev/ntibm0

- Repeating the assign followed by an unassign.

opening and closing it. Resetting the tape unit itself also releases the device from the image it is assigned to.

Examples:

```
> 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

```
> echo "assign 1A28" >/proc/tapedevices
> 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	ASSIGN	---	UNKNOWN
2	1A32	3590	50	3590	11	auto	IN_USE	---	LOADED
3	1A33	3590	50	3590	11	1024	UNUSED	---	LOADED

After refreshing its state information on device 1A28, for example by attempting to load it, another system with the identical tape configuration would then report the following status:

```
> 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	BOXED	---	UNKNOWN
2	1A32	3590	50	3590	11	auto	IN_USE	---	LOADED
3	1A33	3590	50	3590	11	1024	UNUSED	---	LOADED

The assigning system can later release the drive as follows:

```
> echo "unassign 1A28" >/proc/tapedevices
> 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

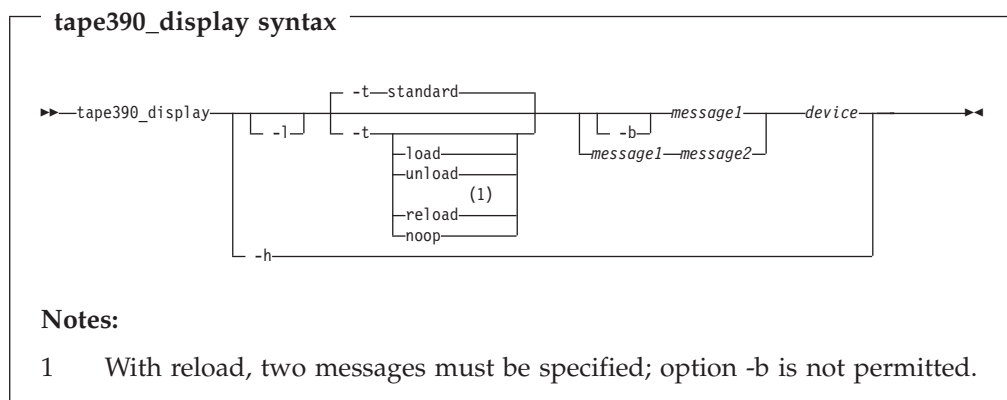
Tape display support

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.

tape390_display tool syntax



where:

-l or --load

instructs the tape unit to load the next indexed tape from the automatic tape loader (if installed); ignored if there is no loader installed or if the loader is not in "system" mode. The loader "system" mode allows the operating system to handle tape loads.

-t or --type

The possible values have the following meanings:

standard

displays the message or messages until the tape drive processes the next tape movement command

load

displays the message or messages until a tape is loaded; if a tape is already loaded, the message is ignored

unload

displays the message or messages while a tape is loaded; if no tape is loaded, the message is ignored

reload

displays the first message while a tape is loaded and the second message when the tape is removed; if no tape is loaded, the first message is ignored and the second message is displayed immediately; the second message is displayed until the next tape is loaded

noop

this option is intended for test purposes; it accesses the tape device but does not display the message or messages

-b or --blink

causes *message1* to be displayed repeatedly for 2 seconds with a half-second pause in between

message1

is the first or only message to be displayed

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 are converted to uppercase.

Numeric characters:

0 1 2 3 4 5 6 7 8 9

Special characters:

@ \$ # , . / ' () * & + - = % : _ < > ? ;

The following are included in the 3490 hardware reference but might not display on all devices: | ¢

2. If only one message is defined, it remains displayed until the tape driver next starts to move or the message is updated.
3. If the messages contain spaces or shell-sensitive characters, they must be enclosed in quotation marks.

Examples

The following examples assume that you are not using devfs:

- Alternately display “BACKUP” and “COMPLETE” at two second intervals until device /dev/ntibm0 processes the next tape movement command:

```
tape390_display BACKUP COMPLETE /dev/ntibm0
```

- Display the message “REM TAPE” while a tape is in the tape drive followed by the message “NEW TAPE” until a new tape is loaded:

```
tape390_display --type reload "REM TAPE" "NEW TAPE" /dev/ntibm0
```

- Attempts to unload the tape and load a new tape automatically, the messages are the same as in the previous example:

```
tape390_display -l -t reload "REM TAPE" "NEW TAPE" /dev/ntibm0
```

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 file system is created using the first tape device. For this the ISO9660 file system support must be built into your system kernel.

Channel-attached tape device driver

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

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

```
mkdir somedir
cp contents somedir
```

2. Insert a tape
3. Ensure the tape is positioned at the beginning of the tape file that is to contain the file system image. Assuming that it is the second file on the tape, issue:

```
mt -f /dev/ntibm0 rewind
mt -f /dev/ntibm0 fsf 1
```

`fsf` skips a specified number of files, for example, `fsf 2` skips two files.

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

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

5. Write the file system to the character device driver:

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

6. Rewind the tape again:

```
mt -f /dev/ntibm0 rewind
mt -f /dev/ntibm0 fsf 1
```

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

```
mount -t iso9660 -o ro,block=2048 /dev/btibt0 /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`.

1. Insert a tape medium in the device (here: `/dev/tape/019f/char/nonrewinding`).
2. 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
```

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

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

4. 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
```

5. 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
```

6. Insert a new tape manually (if not using a tape unit magazine with autoload).
7. 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.

8. Repeat this process with any remaining tapes until the backup is complete.

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 devices. They do apply to tape devices accessed via the `zfc` driver, however.

Chapter 7. Generic cryptographic device driver

z90crypt is a generic character device driver for a cryptographic device. The device driver routes work to any of the supported physical cryptographic devices installed on the system.

The z90crypt device driver controls PCICCs, PCICAs, PCIXCCs (MCL2 and MCL3), or CEX2C in a Linux environment. VM hides PCICCs and PCIXCCs from its guests if a PCICA is also available. For VM guests, PCIXCC cards are supported as of z/VM 5.1.

Features

The cryptographic device driver supports the following devices and functions:

- PCI Cryptographic Coprocessor (PCICC)
- PCI Cryptographic Accelerator (PCICA)
- PCI-X Cryptographic Coprocessor (PCIXCC)

The cryptographic device driver distinguishes PCIXCCs according to the licensed internal code (LIC) level:

- PCIXCC (MCL3) as of LIC EC J12220 level 29
- PCIXCC (MCL2) with a LIC prior to EC J12220 level 29

- Crypto Express2 Coprocessor (CEX2C)

The cryptographic device driver supports these cryptographic operations:

- Encryption, with a public or private key, by arithmetic operations involving very large numbers

RSA exponentiation operation using either a modulus-exponent (Mod-Expo) or Chinese-Remainder Theorem (CRT) key.

- Decryption, closely related to encryption, with the counterpart of the key used for encryption

What you should know about z90crypt

This section provides information on the software you need to use z90crypt and when z90crypt uses cryptographic hardware.

Required software components

To run programs that use the z90crypt device driver, you need:

- The device driver module (z90crypt)
- The libica library (except for programs that circumvent libica)

To support applications that use the PKCS #11 API you also need:

- The openCryptoki library

The openCryptoki library requires the libica library. Applications can either directly use the libica library or use libica indirectly through the openCryptoki PKCS #11 API. Applications can also interface directly with the z90crypt driver module.

Figure 2 on page 64 illustrates the software relationships:

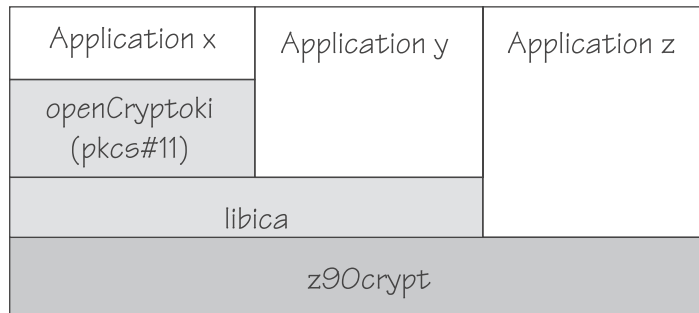


Figure 2. z90crypt device driver interfaces

In Figure 2 “Application x” is an application that uses the PKCS #11 API while “Application y” directly uses the libica library. “Application z” directly uses the z90crypt interfaces (see “External programming interfaces” on page 69).

See “Setting up the z90crypt device driver” on page 65 for information on how to set up the different components.

Hardware offload

The libica library can perform the generation of public/private key pairs, encryption and decryption, signing and signature verification, through software.

If cryptographic hardware is available, the encryption and decryption operations, which may include signing and signature verification, are performed in hardware, with enhanced performance, provided that the padding requirements are handled correctly (see “Padding”). If the padding is not done correctly, the cryptographic operations are performed in software.

Although libica can perform the cryptographic operations in software if no cryptographic hardware is available, it does not work without the z90crypt device driver.

Padding

If you have a PCICC only, or are attempting to use a CRT key on a system with PCIXCC (MCL2) only, you need to ensure that your data is PKCS-1.2 padded. In this case, the z90crypt device driver or the cryptographic hardware might change the padding.

If you have at least one PCICA, PCIXCC (MCL3), or CEX2C, or if you are only using Mod-Expo keys with a PCIXCC (MCL2), you do not need to ensure PKCS-1.2 padding. In this case the padding remains unchanged.

Load balancing

The z90crypt device driver assigns work to cryptographic devices according to device type in the following order of preference:

1. CEX2C, PCICA, and PCIXCC (MCL3)
2. PCIXCC (MCL2)
3. PCICC

For small work loads, a particular job is assigned to the first device that offers the required capabilities. The device order is according to the physical arrangement of the cryptographic cards (see “Checking hardware status” on page 67).

Further information

- For information on RSA-PKCS 1.2-padding visit <http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1/>.
- For information on how to set up cryptographic hardware on your mainframe refer to *zSeries Crypto Guide Update*, SG24-6870.
- For CP commands related to cryptographic devices refer to *z/VM CP Command and Utility Reference*, SC24-6008.

Building a kernel with the z90crypt device driver

You need to select the option `CONFIG_Z90CRYPT` if you want to use a PCI-attached cryptographic adapter.

Cryptographic devices	
Support for PCI-attached cryptographic adapters	(<code>CONFIG_Z90CRYPT</code>)

Figure 3. z90crypt kernel configuration menu option

The `CONFIG_Z90CRYPT` option can be compiled into the kernel or as a separate module, `z90crypt`.

See also “Setting up for the 31-bit compatibility mode” on page 67 if you want to build a 64-bit system with 31-bit compatibility.

Setting up the z90crypt device driver

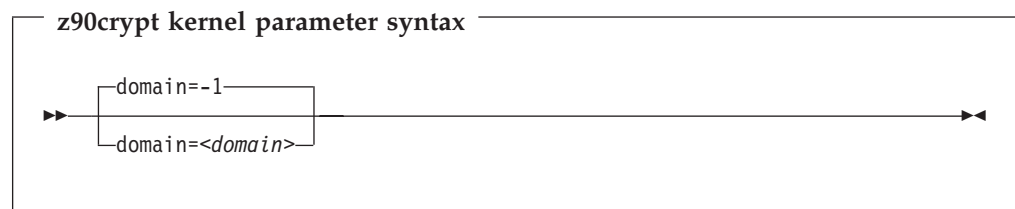
This section describes the `z90crypt` module and kernel parameters, and how to install additional components required by the device driver.

This section also describes how to create the required device node. It further describes how to set up the cryptographic devices on your LPAR, and where applicable, VM, to make it available to your Linux instance.

Kernel parameters

This section describes how to configure the `z90crypt` device driver if `z90crypt` has been compiled into the kernel. You can configure the device driver by adding the domain parameter to the kernel parameter line.

You need to specify the domain parameter only if you are running Linux in an LPAR for which multiple cryptographic domains have been defined.



where `<domain>` is an integer in the range from 0 to 15 that identifies the cryptographic domain for the Linux instance.

The specification “domain=-1” causes the device driver to attempt to autodetect the domain to use. This is the default.

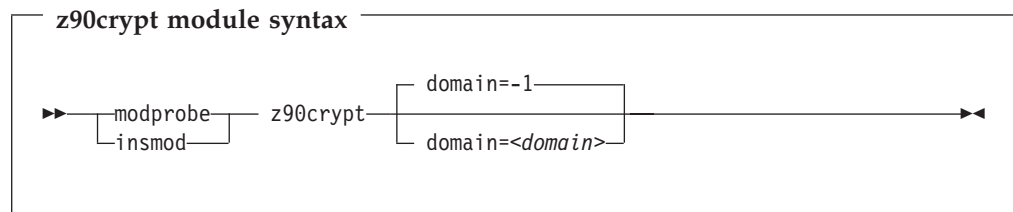
Example

The following kernel parameter line specification makes the z90crypt device driver assume that it operates within the cryptographic domain “1”:

```
domain=1
```

Module parameters

This section describes how to load and configure the z90crypt device driver if it has been compiled as a separate module.



where *<domain>* is an integer in the range from 0 to 15 that identifies the cryptographic domain for the Linux instance. Omit the domain parameter if only one cryptographic domain is defined for the LPAR where your Linux instance runs.

The specification “domain=-1” causes the device driver to attempt to autodetect the domain to use. This is the default.

Refer to the respective man page for details on **modprobe** or **insmod**.

Examples

- This example loads the z90crypt device driver module if Linux runs in an LPAR with only one cryptographic domain:

```
# modprobe z90crypt
```

- This example loads the z90crypt device driver module and makes the z90crypt device driver assume that it operates within the cryptographic domain “1”:

```
# modprobe z90crypt domain=1
```

The libica library

You can obtain the libica library from the SourceForge Web site at:

<http://sourceforge.net/projects/opencryptoki>

Both a 31-bit and a 64-bit version are available. The 64-bit version includes the 31-bit compatibility code.

openCryptoki

You can obtain the openCryptoki library from the SourceForge Web site at:

<http://sourceforge.net/projects/opencryptoki>

You can find the release details with the module on the **Downloads** tab. To access the details click on the link with the release date, just above the module download.

To be able to configure openCryptoki (with pkcsconf) user root must be a member of group pkcs11.

Setting up for the 31-bit compatibility mode

31-bit applications can access the 64-bit z90crypt driver by using the 31-bit compatibility mode.

Assuring that you have a device node

User space programs address cryptographic devices through a single device node. Depending on your distribution, this device node might be created for you. Issue the following command to search for an existing device node:

```
# find / -name z90crypt -type c
```

If there is not already a device node, ensure that option `Z90CRYPT_USE_HOTPLUG` in `z90common` is enabled. This option causes a dynamic major number to be assigned to the `z90crypt` device driver. The minor number to be used with the dynamic major number is “0”.

After booting, and if applicable, loading the device driver you can find out which major number has been assigned. The major number is the value for the entry “`z90crypt`” in `/proc/devices`.

You can then create the device node by issuing a command like this:

```
# mknod /dev/z90crypt c <dynamic_major> 0
```

Tip: You can also use hotplug events to generate or remove the device node for `z90crypt` (see “Crypto hotplug events” on page 318).

Working with the z90crypt device driver

Typically, cryptographic devices are not directly accessed by users but through user programs. This section describes the following tasks:

- Checking hardware status
- Deactivating devices under `z90crypt`

Checking hardware status

You can get the hardware status using the `cat` function, for example, if you issue:

```
# cat /proc/driver/z90crypt
```

you should have an output similar to the following:

```

z90crypt version: 1.3.2
Cryptographic domain: 6
Total device count: 5
PCICA count: 2
PCICC count: 0
PCIXCC MCL2 count: 0
PCIXCC MCL3 count: 1
CEX2C count: 0
requestq count: 0
pendingq count: 0
Total open handles: 0

Online devices: 1: PCICA, 2: PCICC, 3: PCIXCC (MCL2), 4: PCIXCC (MCL3), 5: CEX2C
1140000000000000 0000000000000000 0000000000000000 0000000000000000

Waiting work element counts
1000000000000000 0000000000000000 0000000000000000 0000000000000000

Per-device successfully completed request counts
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
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

```

Mask “Online devices” represents the physical arrangements of the cryptographic cards. In the example, there are PCICA cards in the first and in the second position, and a PCIXCC (MCL3) card in the third position.

Mask “Waiting work element counts” represents the same arrangement of physical cards. In this mask, the values represent units of outstanding work. In the example, there is one work element for the card in the first position and no work element for the cards in the second and third position.

Mask “Per-device successfully completed request counts” represents the same arrangement of physical cards as before. In this mask, the values represent units of successfully completed work. If a request fails for any reason, it is not counted.

Deactivating devices under z90crypt

For test or error analysis purposes, you might want to deactivate a cryptographic device. You can do this by editing the `/proc/driver/z90crypt` file with your preferred text editor. Proceed like this to deactivate a cryptographic device:

1. Open `/proc/driver/z90crypt` with your editor. You will see several lines including two lines like this:

```

Online devices: 1: PCICA, 2: PCICC, 3: PCIXCC (MCL2), 4: PCIXCC (MCL3), 5: CEX2C
1140000000000000 0000000000000000 0000000000000000 0000000000000000

```

The lower line represents the physical arrangement of the cryptographic devices with digits 1, 2, 3, 4, and 5 representing PCICA, PCICC, PCIXCC (MCL2), PCIXCC (MCL3), and CEX2C cards, respectively.

2. Overwrite the digit that represents the card you want to deactivate with a character d. To deactivate the card in the second position, of our example overwrite the second 1:

```
Online devices: 1: PCICA, 2: PCICC, 3: PCIXCC (MCL2), 4: PCIXCC (MCL3), 5: CEX2C
1d40000000000000 0000000000000000 0000000000000000 0000000000000000
```

3. Close and save `/proc/driver/z90crypt`. Confirm that you want to save your changes even if the content of the file has changed since you opened it.

To enable a deactivated device proceed like this:

1. Open `/proc/driver/z90crypt` with your editor. You will see two lines like this:

```
Online devices: 1: PCICA, 2: PCICC, 3: PCIXCC (MCL2), 4: PCIXCC (MCL3), 5: CEX2C
1d40000000000000 0000000000000000 0000000000000000 0000000000000000
```

Each `d` in the second line represents a deactivated device. In our example, the device in the second position has been deactivated.

2. Overwrite the `d` that represents the device you want to enable with an `e`:

```
Online devices: 1: PCICA, 2: PCICC, 3: PCIXCC (MCL2), 4: PCIXCC (MCL3), 5: CEX2C
1e40000000000000 0000000000000000 0000000000000000 0000000000000000
```

3. Close and save `/proc/driver/z90crypt`. Confirm that you want to save your changes even if the content of the file has changed since you opened it. The device driver replaces the `e` with the digit for the actual device.

External programming interfaces

Refer to `drivers/s390/crypto/z90crypt.h` in the Linux kernel source tree, for different structures that you might want to use.

Outline of decryption program

The following steps are required:

1. Get a device handle for `z90crypt`.

Example:

```
dh= open("/dev/z90crypt", 0_RDWR)
```

2. Create and load one of the following structures

- `ica_rsa_modexpo` (see “The `ica_rsa_modexpo` structure” on page 70)
- `ica_rsa_modexpo_crt` (see “The `ica_rsa_modexpo_crt` structure” on page 70)

Both structures are defined in `z90crypt.h`. In the structure, you define the private key to be used and set the input buffer pointer to the data you want to decrypt and the output buffer pointer for the decrypted data.

3. Invoke `ioctl` to activate `z90crypt`:

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

where:

`<function code>`

is `ICARSAMODEXPO` for structure type `ica_rsa_modexpo` or `ICARSACRT` for structure type `ica_rsa_modexpo_crt`

`<structure name>`

is the variable for the structure you created, of the type `ica_rsa_modexpo` or `ica_rsa_modexpo_crt`

`<rc>`

is the variable for the return code

Example:

```
myrc = ioctl(dh, ICARSAMODEXPO, &mycryptmex);
```

4. Obtain the decrypted and decoded data from the output buffer in the structure.
See “Padding” on page 64 about when the original data must have been PKCS 1.2 encoded – that is, when the decrypted data must have the correct padding.

The ica_rsa_modexpo structure

The ica_rsa_modexpo structure is defined in the z90crypt header file, z90crypt.h.

The (private) key consists of the exponent in *b_key and the modulus in *n_modulus. Both of these are hexadecimal representations of large numbers. The length L of *n_modulus must be in the range 64 – 256.

Both the input data and the exponent b_key must be of the same length L as the modulus. If they are shorter than the modulus, they must be padded on the left with zeroes. The output data length must be at least L.

The ica_rsa_modexpo_crt structure

The ica_rsa_modexpo_crt structure is defined in the z90crypt header file, z90crypt.h.

The ica_rsa_modexpo_crt structure is similar to the ica_rsa_modexpo structure but has been defined so that the Chinese Remainder Theorem (CRT) can be used in decryption. z90crypt performs better if the CRT definition is used. The key-definition fields are all in hexadecimal representation. They have these meanings and limitations:

- *bp_key and *bq_key are the prime factors of the modulus. In 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 and *nq_prime are exponents used for *bp_key and *bq_key respectively.
- *u_mult_inv is a coefficient used in the z90crypt implementation of decryption by CRT.
- *bp_key, *np_prime, and *u_mult_inv must all be of length 8 + L/2
- *bq_key and *nq_prime must both be of length L/2

The input data length must be L, and the output data length must be at least L

Querying the hardware status

There is an ioctl interface for checking on underlying hardware in z90crypt. There are a number of ioctls for each status needed. These ioctls are defined in the header file z90crypt.h. When control returns, you will have the information you requested.

Example:

```
<rc> = ioctl(<dh>, Z90STAT_TOTALCOUNT, &<int_variable>);
```

where:

<rc> is the variable for the ioctl return code

<dh> is the variable for the z90crypt device handle

`<int_variable>` is the variable you want to use for the returned information, in the example, the total count of installed cards that are recognized by `z90crypt`.

Returns from `ioctl`

0 means everything went well and the result is in your output buffer.

A return code of -1 indicates an error code. Error codes greater than 128, are described in `drivers/s390/crypto/z90crypt.h`. For all other error codes, refer to `/usr/include/asm/errno.h`

Chapter 8. SCSI-over-Fibre Channel device driver

The SCSI-over-Fibre Channel driver (zfc) provides support for Fibre Channel attached SCSI devices on Linux for zSeries and S/390. This chapter details the usage of the SCSI driver (zfc) for the QDIO-based zSeries FCP (zSeries SCSI over Fibre Channel) channel. In the following, we will use the term “zSeries FCP adapter” to address a single virtual instance of the zSeries FCP channel.

The zfc 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 and S/390 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 zfc requires an IBM @server zSeries 900 (z900), IBM @server zSeries 800 (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.

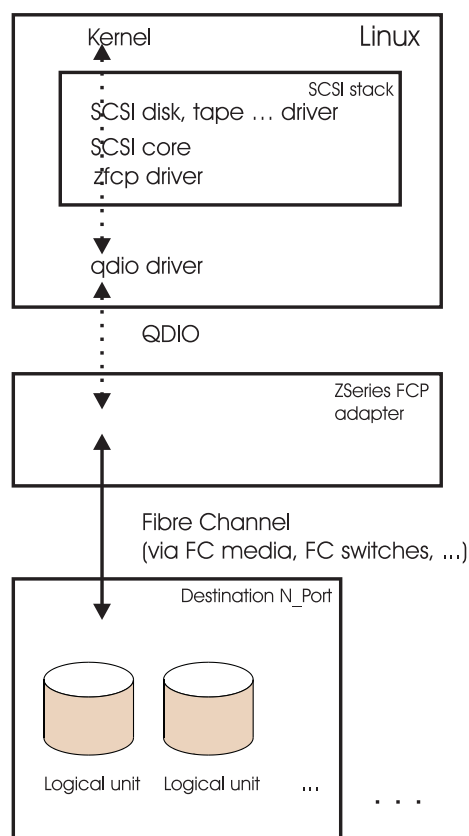


Figure 4. General layout of zSeries FCP environment

Configuring zfc

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

- address mappings between Linux SCSI and FCP schemes (mandatory for each SCSI target device)

SCSI-over-Fibre Channel device driver

- logging level to determine the verbosity of the zfcplib device driver (optional, default value is used if not supplied)

For these purposes, the zfcplib driver provides different means of configuration:

- kernel parameters
- module parameters (such as for use in modules.conf)
- various proc-fs entries in /proc/scsi/zfcplib

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

Table 7. zfcplib functions, parameters, and proc-fs entries

Function	Module parameter	Kernel parameter	proc-fs entry
Set logging level	loglevel	zfcplib_loglevel	/proc/scsi/zfcplib/mod_parm
Get logging level (and other global module information)	(not applicable)	(not applicable)	/proc/scsi/zfcplib/mod_parm
Add address mapping(s)	map	zfcplib_map	/proc/scsi/zfcplib/add_map
Get all existing address mappings	(not applicable)	(not applicable)	/proc/scsi/zfcplib/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 "zfcplib_map" kernel parameter:

Module:

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

Kernel:

```
zfcplib_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 WWPNN)

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 87))

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).

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

fcplun

Logical Unit Number as defined by FCP (64 bits, according to the FCP/SCSI-3 specifications (see “References” on page 87).

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 zfcpl on address mapping processing. That is why user comments are not available via /proc/scsi/zfcpl/map.
- **Number formats:** Numbers can be in the decimal, octal, or hexadecimal systems following the C programming language conventions regarding prefixes.
Recommendation: For better readability of FCP LUNs, WWPNs, and device numbers of FCP channels use the hexadecimal notation with prefix “0x”.
- **Allowed values:** The zfcpl 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 zfcpl.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.

SCSI-over-Fibre Channel device driver

Kernel:

```
zfc_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/zfc/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/zfc/add_map"` proc-fs entry:

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

or

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

where the content of `my_mapping.txt` could be:

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

Usage of the `"/proc/scsi/zfc/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/zfc/add_map"` (note: only line-breaks are used as separators for this proc-fs entry).

Example for the output of `"/proc/scsi/zfc/map"`:

```
cat /proc/scsi/zfc/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/zfc/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 zfc)
- other Linux systems (sharing environment)

How to release access to SAN devices and HBAs

You can release access to an adapter, target port, or LUN by writing “set offline” to the respective status file in `procs`.

Example: the command:

```
# echo "set offline" >/proc/scsi/zfc/devno0x4200/id0x1/lun0x0/status
```

triggers a shutdown procedure on LUN 0 of target port 1 of adapter 0x4200. The error recovery subsystem then attempts to close the LUN if it is still open.

Ensure that the adapter, port, or LUN is unused before writing “set offline” to its status file. In particular, unmount all affected devices and deregister them with the SCSI subsystem.

This feature is particularly useful with regard to LUNs that can only be accessed in exclusive mode through shared FCP Channels. In conjunction with the complementing feature described in “How to regain access to SAN devices and HBAs,” it allows SAN devices to be reassigned to other systems sharing the same FCP CHPID. See “How to regain access to SAN devices and HBAs” for an example of such a LUN takeover.

Note: This action does not discard corresponding entries from `/proc/scsi/zfc/map`, but merely changes device connection. Therefore, it is not necessary to add these map-entries anew after releasing access to devices.

How to regain access to SAN devices and HBAs

You can regain access to an adapter, target port, or LUN by writing “set online” to the respective status file in `procs`.

Example: the command:

```
# echo "set online" >/proc/scsi/zfc/devno0x4200/id0x1/lun0x0/status
```

triggers a reopen procedure on LUN 0 of target port 1 of adapter 0x4200. The error recovery then attempts to reopen the LUN.

You need to register the LUN to the SCSI subsystem, to make the reclaimed device visible in Linux.

Example for a LUN takeover and return:

Assumptions: The `zfc-map` of Linux A contains the entry “0x4200 0x1:0x2422067800001000 0x0:0x5101000000000000” and corresponds to SCSI device 0:0:1:0, or `/dev/sdb`, respectively. The SCSI disk is used and contains one mounted file system on `/dev/sdb1`. The same device is not yet configured in Linux B. Linux B connects to the SAN through the FCP subchannel with device number 0x4206. Linux B already uses several other

SCSI-over-Fibre Channel device driver

HBAs and SCSI devices. In Linux B, the same device appears as SCSI device 3:0:1:0, or /dev/sdg, respectively.

1. Linux A releases the device:

```
# umount /dev/sdb1
# echo "scsi remove-single-device 0 0 1 0" > /proc/scsi/scsi
# echo "set offline" > /proc/scsi/zfc/devno0x4200/id0x1/lun0x0/status
```

2. Linux B gains access to the device and later releases it:

```
# echo "0x4206 0x1:0x2422067800001000 0x0:0x5101000000000000" > /proc/scsi/zfc/add_map
# echo "scsi add-single-device 3 0 1 0" > /proc/scsi/scsi
# mount -t ext3 /dev/sdg1 /mnt/timeshared-disk
...
# umount /dev/sdg1
# echo "scsi remove-single-device 3 0 1 0" > /proc/scsi/scsi
# echo "set offline" > /proc/scsi/zfc/devno0x4206/id0x1/lun0x0/status
```

3. Linux A regains access to the device:

```
echo "set online" /proc/scsi/zfc/devno0x4200/id0x1/lun0x0/status
echo "scsi add-single-device 0 0 1 0" > /proc/scsi/scsi
mount -t ext3 /dev/sdb1 /mnt/timeshared-disk
```

Usage of the "loglevel" module parameter and "zfc_loglevel" kernel parameter:

These parameters allow specifying the level of verbosity of the zfc 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)
 - zfc version
 - syntax/semantic errors in address mappings
 - detected external conditions which restrict or prevent zfc operation (e.g., another OS instance owns exclusive access to a configured logical unit via same Fibre Channel link)
 - unexpected hardware behavior
 - unexpected zfc 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:
zfc_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 zfcpl.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".

```
zfcpl_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 "zSeries FCP" and "SCSI".

Usage of the "/proc/scsi/zfcpl/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 zfcpl driver configurable via "/proc/scsi/zfcpl/mod_parm".

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

```
echo "loglevel=0x00123001" > /proc/scsi/zfcpl/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/zfcpl/mod_parm" proc-fs entry:

SCSI-over-Fibre Channel device driver

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

Module Information:

```
Module Version $Revision: 3.157.6.9 $ running in mode: FULL FEATURED
Debug proc output enabled: YES
```

Full log-level is: 0x01111111 which means:

ERP log-level:	1		
QDIO log-level:	1	Dynamic IO log-level:	1
Configuration log-level:	1	FSF log-level:	1
SCSI log-level:	1	Other log-level:	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 is always 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 0x0:0x01f0000000000000"
```

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

Before you start:

- 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. Load the modules required by zfcp, e.g.:

SCSI-over-Fibre Channel device driver

```
modprobe qdio
modprobe scsi_mod
```

3. Load zfc, specifying the initial disk and/or loglevel, if desired (both optional), e.g.:

```
insmod zfc.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.

4. Optionally, specify additional devices via add_map and make them known to the SCSI stack.

Example:

```
echo "0x4202 0x1:0x2422067800001000 0x1:0x5112000000000000;\
0x4202 0x1:0x2422067800001000 0x2:0x5103000000000000"\
>/proc/scsi/zfc/add_map
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: add-single-device is required only for devices that have been added manually (via /proc/scsi/zfc/add_map). The numbers that follow add-single-device represent host, bus, target, and LUN in the same manner and order as explained for devfs in “devfs and zfc mapping” on page 81.

For more information on the use of the Linux SCSI stack, see the SCSI-stack HOWTO.

5. 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
```

6. Create and mount a file system on a SCSI disk:

- Partition a disk (see “man fdisk”):

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

- Create a file system on the first partition:

```
mke2fs /dev/scsi/host0/bus0/target1/lun0/part1
```

- Mount it:

```
mount /dev/scsi/host0/bus0/target1/lun0/part1 /mnt
```

For a multipathing setup, file systems would be created and mounted on the appropriate device node, for example, as created by the LVM multipathing tools. Device nodes of SCSI devices associated with a meta-device capable of multipathing must not be accessed directly, specifically for creating or mounting file systems. See Chapter 20, “Multipathing support,” on page 247 for details.

Installation considerations

Notes:

1. The zfcplib driver requires version 2 of QDIO for Linux and is not compatible with any older version.
2. Relevant higher-level, common-code SCSI drivers must be present either as modules or compiled into the kernel:

Table 8. SCSI module dependencies

Functionality	Kernel config option	Module name
zfcplib	CONFIG_ZFCPLIB (mandatory)	zfcplib.o
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 file systems (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.

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

zfcplib HBA API (FC-HBA) support

The zfcplib host bus adapter API (HBA API) provides an interface for SAN management clients that run on z/Series Linux.

As shown in Figure 5 on page 84, the zfcplib HBA API support includes a kernel module and a user space library.

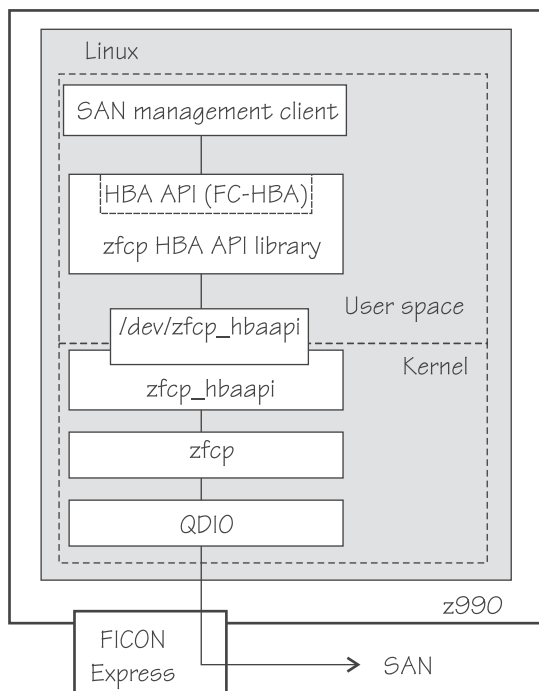


Figure 5. zfc HBA API support modules

The library provides the zfc HBA API to SAN management applications and uses a misc device file to communicate with the kernel module. The kernel module uses the SCSI-over-Fibre Channel device driver (zfc) to communicate with the FICON® or FICON Express adapter card and the SAN.

Building a kernel with the zfc HBA API support

To build a kernel that includes the zfc HBA API support you need to select the configuration menu option:

```

SCSI support
--> SCSI low-level drivers
    --> FCP host bus adapter driver for IBM z800, z900, z990 (GA2)
        --> HBA API support for FCP host bus adapter driver for IBM z990 (GA2)
    
```

You can compile the zfc HBA API support into the kernel or compile it as a module, zfc_hbaapi.o.

Setting up the zfc HBA API support

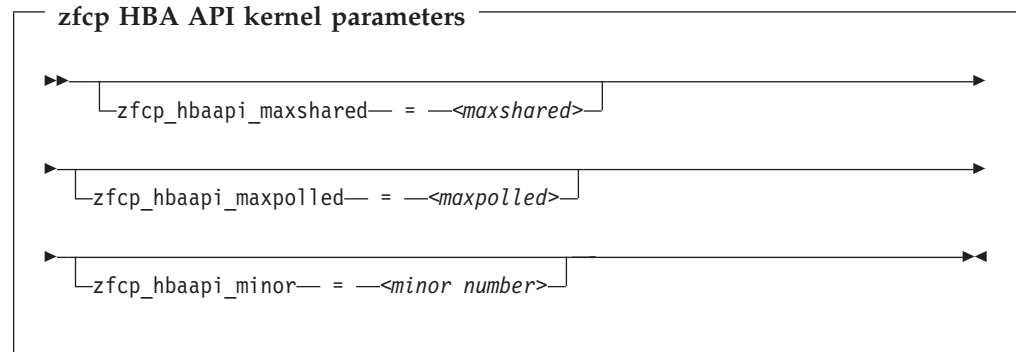
To set up the zfc HBA API support you need to:

1. Configure the zfc HBA API support and ensure that the module is loaded if the support has been compiled as a module.
2. Install the zfc HBA API library.
3. Create a device node.

Configuring the zfc HBA API support

The following sections describe the parameters that you can use to configure the zfc HBA API support if the support has been compiled into the kernel and if the support has been compiled as a module.

Kernel parameters: If the HBA API kernel module has been compiled into the kernel, you can optionally configure it by adding parameters to the kernel parameter line:



where:

<maxshared>

is the maximum number of events in the shared event queue.

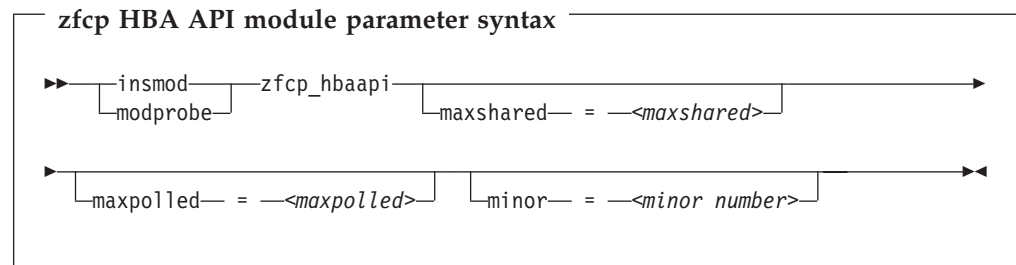
<maxpolled>

is the maximum number of events in the polled event queue.

<minor number>

is the minor number for the misc device that is registered. By default, the minor number is allocated dynamically.

Loading the module: If your zfc HBA API support has been compiled as a module, you can load the module with **insmod** or **modprobe**. If you use **insmod**, be sure that the required zfc module is already loaded, or that the zfc support has been compiled into the kernel.



where:

<maxshared>

is the maximum number of events in the shared event queue.

<maxpolled>

is the maximum number of events in the polled event queue.

<minor number>

is the minor number for the misc device that is registered. By default, the minor number is allocated dynamically.

Installing the zfcph HBA API library

You can find the library as a source package lib-zfcph-hbaapi-1.0.tar.gz at developerWorks:

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

Perform the following steps to install the library:

1. Download the source package lib-zfcph-hbaapi-1.0.tar.gz from developerWorks.
2. Compile and install the package:

```
#> tar xzf lib-zfcph-hbaapi-1.0.tar.gz
#> cd lib-zfcph-hbaapi-1.0
#> ./configure
#> make
#> make install
```

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

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

Result: You have installed:

- Shared and static versions of libzfcphbaapi at /usr/local/lib.
- The header file hbaapi.h at /usr/local/include.
- Optionally, the documentation package at /usr/local/share/doc/zfcph-hbaapi-1.0.

Creating a device node

The module zfcph_hbaapi provides a misc device. This misc device is used for kernel-user-space communication. The major number for the device is 10.

You can specify a minor number as a kernel or module parameter (see “Configuring the zfcph HBA API support” on page 84) or use a dynamic minor number. You can find the assigned dynamic minor number in the entry for zfcph_hbaapi in /proc/misc. If you are using the zfcph HBA API support as a module, the dynamic minor number is assigned at load time.

After the minor number has been assigned, the device node, /dev/zfcph_hbaapi, can be created using the following commands:

```
#> minor=$(cat /proc/misc | awk "\$2==\"zfcph_hbaapi\" {print \$1}")
#> mknod /dev/zfcph_hbaapi c 10 $minor
```

Functions provided

The zfcph HBA API is defined in the Fibre Channel - HBA API (FC-HBA) specification (see <http://www.t11.org>).

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

- HBA_GetVersion()
- HBA_LoadLibrary()
- HBA_FreeLibrary()
- HBA_GetWrapperLibraryAttributes()
- HBA_GetVendorLibraryAttributes()
- HBA_GetNumberOfAdapters()
- HBA_GetAdapterName()
- HBA_OpenAdapter()

- HBA_CloseAdapter()
 - HBA_RefreshInformation()
 - HBA_RefreshAdapterConfiguration()
 - HBA_GetAdapterAttributes()
 - HBA_GetAdapterPortAttributes()
 - HBA_GetDiscoveredPortAttributes()
 - HBA_SendScsiInquiry()
 - HBA_SendReadCapacity()
 - HBA_SendReportLUNs()
 - HBA_GetFcpTargetMapping()
 - HBA_SendCTPassThru() (supports requests to the Name Server Directory Service only)
 - HBA_GetRNIDMgmtInfo()
 - HBA_GetEventBuffer()
 - HBA_GetPortStatistics()
 - HBA_ResetStatistics()
- In case of a Vendor Library the functions are also provided:
- HBA_RegisterLibrary()
 - HBA_RegisterLibraryV2()

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

The detailed description of ZFCP HBA API-relevant functions and structures of the modules `zfcplib` and `zfcplib_hbaapi` can be found in the kernel sources. A template `Documentation/DocBook/zfcplib-hba-api.tmpl` exists. With this template, documentation is generated which contains information about all interfaces and some internals of the kernel modules `zfcplib` and `zfcplib_hbaapi`.

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 host bus adapter (HBA) identification (`/dev/scsi/host0/bus0/target1/lun0`). The issue of HBA identification is resolved in the scope of `zfcplib`.

'sd_many' patch

There are so-called `sd_many` patches, which raise this limit to a considerably higher number (for example, thousands) of SCSI disks. Such external patches are not supported by IBM, but you can use them at your own risk.

Persistent configuration

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

References

The following are sources of additional information for zSeries FCP:

SCSI-over-Fibre Channel device driver

FCP/SCSI-3 specifications

Describes SCSI-3, the Fibre Channel Protocol, and related information.

<http://www.t10.org> and <http://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.tldp.org/HOWTO/SCSI-2.4-HOWTO/>

Getting Started with zSeries Fibre Channel Protocol

Introduces the concepts of zSeries Fibre Channel Protocol support, and shows how various SCSI devices can be configured to build a zSeries FCP environment.

[http://publib-](http://publib-b.boulder.ibm.com/Redbooks.nsf/RedpaperAbstracts/redp0205.html?Open)

[b.boulder.ibm.com/Redbooks.nsf/RedpaperAbstracts/redp0205.html?Open](http://publib-b.boulder.ibm.com/Redbooks.nsf/RedpaperAbstracts/redp0205.html?Open)

Supported FCP connectivity options

Lists supported SCSI devices and provides links to further documentation on FCP and SCSI.

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

Chapter 9. z/VM recording device driver

The z/VM recording device driver can be used by Linux systems that run as z/VM guests. The device driver enables the Linux guest to read from the CP Error Logging System Service (*LOGREC) and, thus, act as a z/VM wide control point.

The driver uses the z/VM RECORDING command to collect records and IUCV to transmit them to the Linux guest.

Building a kernel with the z/VM recording device driver

You need a kernel that also includes the IUCV device driver (see Chapter 13, “IUCV device driver,” on page 115).

The module for the z/VM recording device driver is called `vmlogrdr.o`. To build a Linux kernel that supports the z/VM recording device driver select the configuration menu option:

```
Character device drivers
--> Support for the z/VM recording system services
```

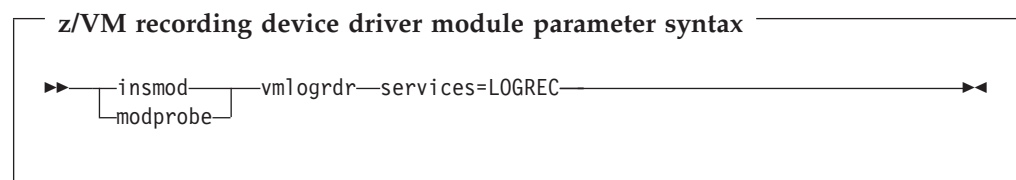
Compile the device driver as a module. If the driver is compiled directly into the kernel it cannot be activated to collect records.

Setting up the z/VM recording device driver

The major device number for the z/VM recording device driver is assigned dynamically. Check `/proc/devices` to find out which major number has been assigned. The minor number is 0.

Loading the module

Before you can use the z/VM recording device driver, you must load the `vmlogrdr` module with the `insmod` or `modprobe` command.



If you use `insmod`, be sure that the IUCV device driver is available.

Creating a device node for the z/VM recording device driver

You access the *LOGREC data through a device node. Use a command like this to create a node:

```
mknod -m 440 /dev/<file> c <major> 0
```

where:

<file>

is the file name that you assign to the device node.

<major>

is the major number that has been dynamically assigned to the z/VM recording device driver. You can find the major number in /proc/devices as the number for the entry “vmlogrdr”.

Example: If the major number is 254, and you want to name the device node as /dev/vmlogrdr issue:

```
mknod -m 440 /dev/vmlogrdr c 254 0
```

Working with the z/VM recording device driver

Before you start: The Linux guest must be authorized to use the z/VM RECORDING command. Depending on the z/VM environment, this could be either of the following authorization classes: A, B, C, E, or F.

Opening and closing

You can open, read, and release the device. You cannot open the device multiple times. Each time the device is opened it must be released before it can be opened again. Each time the device is opened or closed, all existing records are deleted from the record queue in z/VM.

Reading

The read function returns one record at a time. If there is no record, the read function waits until a record becomes available.

Each record begins with a 4 byte field containing the length of the remaining record. The remaining record contains the binary z/VM data followed by the four bytes X'454f5200' to mark the end of the record. These bytes build the zero terminated ASCII string “EOR”, which is useful as an eye catcher.

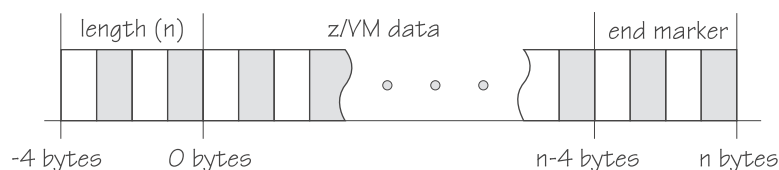


Figure 6. Record structure

Figure 6 illustrates the structure of a complete record as returned by the device. If the buffer assigned to the read function is smaller than the overall record size, multiple reads are required to obtain the complete record.

The format of the z/VM data (*LOGREC) depends on the record type described in the common header for error records HDRREC.

For more information on the z/VM record layout, refer to the *CMS and CP Data Areas and Control Blocks* documentation at ibm.com/vm/pubs/ctlblk.html.

For general information about CP system services and the error logging system service refer to *z/VM CP Programming Services*, SC24-6001.

Part 3. Network device drivers

These chapters describe the channel device layer and the device drivers available to connect zSeries and S/390 systems to your network.

The drivers described are:

- Chapter 11, “CTC/ESCON device driver”
- Chapter 12, “CTCMPC device driver”
- Chapter 13, “IUCV device driver”
- Chapter 14, “LCS device driver”
- Chapter 15, “QETH device driver for OSA-Express (QDIO) and HiperSockets”
- Chapter 16, “Linux for zSeries and S/390 CLAW device driver”

Note

For prerequisites and restrictions for these device drivers refer to the kernel 2.4 June 2003 stream pages on developerWorks at:

ibm.com/developerworks/linux/linux390/june2003_recommended.shtml

If you are routed to the top-level page: On the left navigation bar, click **Kernel 2.4** . Then click the stream with which you want to work.

Chapter 10. Channel device layer

The channel device layer provides a common interface to Linux for zSeries and S/390 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 the following OSA-Express features in non-QDIO mode: Fast Ethernet, 1000Base-T Ethernet (z890 and z990), and Token Ring.
2. **CTC / ESCON** – high speed serial link
3. **CTCMPC** – used by IBM Communications Server for Linux for CTC-based connections to VTAM[®]
4. **QETH** – supports OSA-Express and OSA-Express2 features in QDIO mode and HiperSockets.
5. **CLAW** – used to talk to CISCO routers

The channel device layer draws together the configuration of the drivers and resolves conflicts. These could, for example, result in the LCS, CTC, and CTCMPC 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 102. You can also add comments to the configuration file. Comment lines must be prefixed with a `#` character.

If `chandev.conf` contains a statement with a syntax error, all statements following the error are ignored.

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.

Channel device layer

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:

```
/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

claw 0x20

ctcmpc

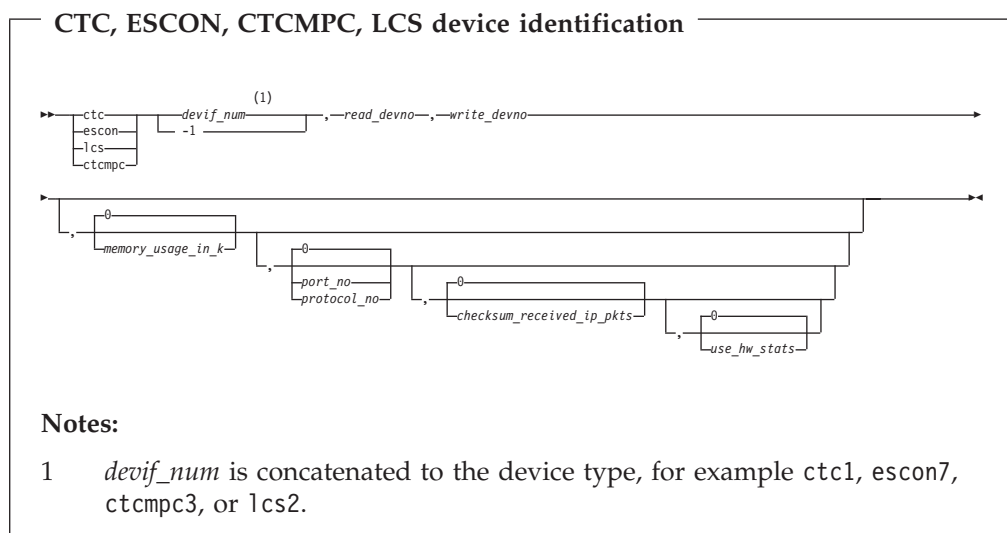
0x40

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, CTCMPC, and LCS)

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

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



Where:

ctc | escon | lcs | ctcmpc
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.
- 4 for CTCMPC devices.

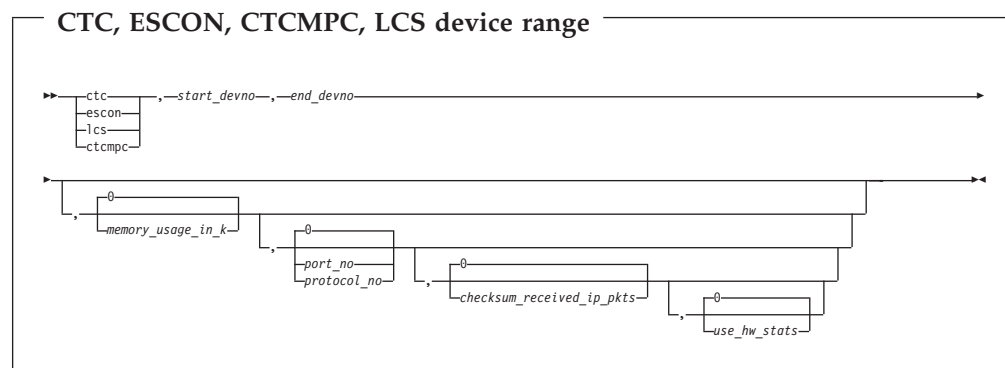
checksum_received_ip_pkts
is a flag: '1' = true; '0' (default) = false.

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

For examples of device identification see:

- CTC, ESCON: “Configuration examples” on page 105
- LCS: “LCS channel device layer configuration example” on page 137
- CLAW: “Configuration example” on page 160

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



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 105
- LCS: “LCS channel device layer configuration example” on page 137
- CLAW: “Configuration example” on page 160

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 141.

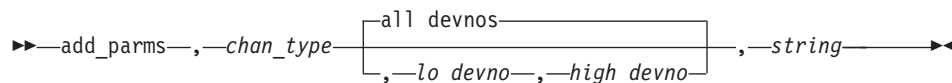
Device identification (CLAW)

For the syntax of the CLAW device driver with the channel device layer see “Channel device layer configuration” on page 159.

Commonly used options

These options are used to set up the system.

add parameters

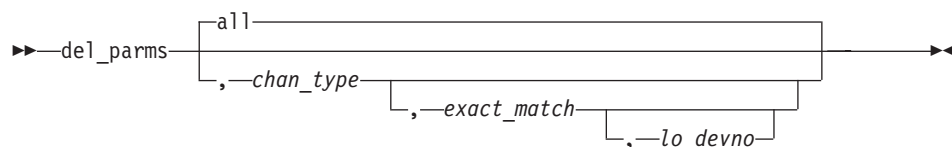


`chan_type` is defined in *chan_type bitfield* in the terminology on page 95.

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).

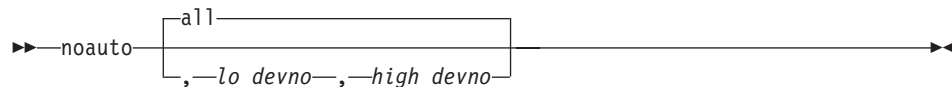
delete parameters



`chan_type` is defined in the terminology on page 95.

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



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.

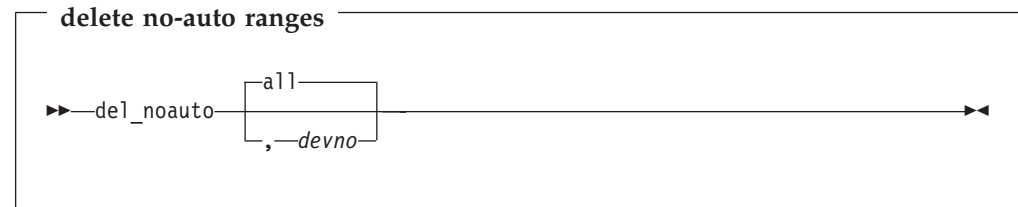
use device names



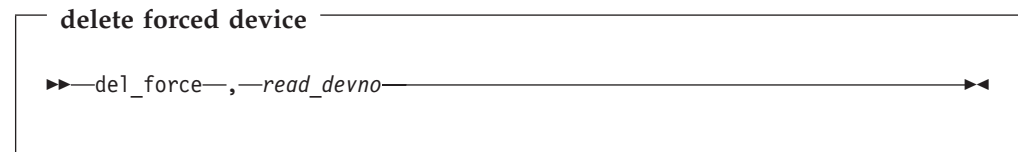
This instructs the channel device layer to assign device names based on the cuu number of the read channel. For example a Token Ring read channel with cuu number 0x7c00 would be assigned an interface name of tr0x7c00. This may be used to avoid device name conflicts. The default is to generate device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

Options for power users

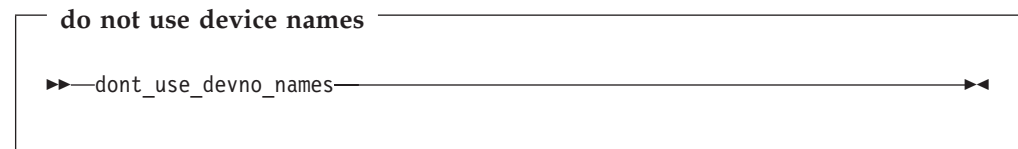
These options are used for maintenance or fine-tuning.



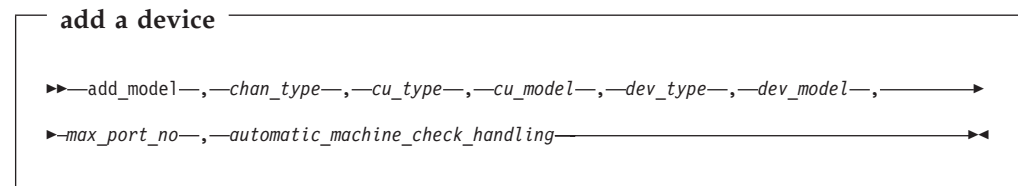
Delete the range containing devno, or all noauto ranges if devno is not given.



Remove a forced channel device from the force list.



Cancel a **use_devno_names** command.



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 95.

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, CTCMPC, 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

```
►►auto_msck—

|                          |
|--------------------------|
| all devnos               |
| ,—lo_devno—,—high_devno— |

,—auto_msck_recovery—►►
```

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 95.

delete machine check recovery

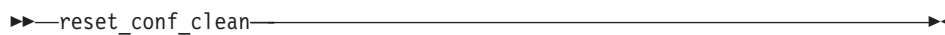
Delete machine check recovery for the range of devices including *devno*, or all machine check recovery if *devno* is not specified.

null model information

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

default model information

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

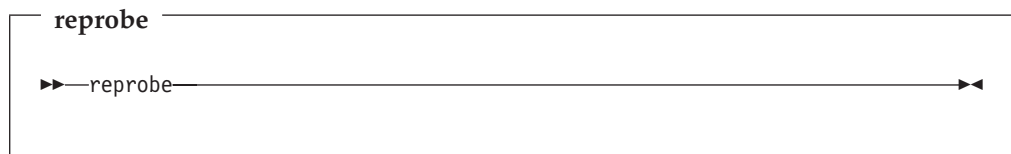
empty model information

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

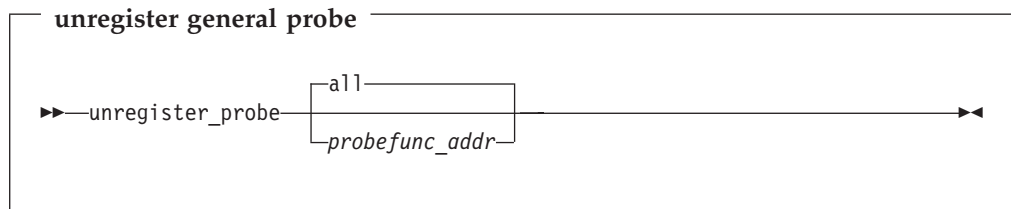
shutdown device

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.

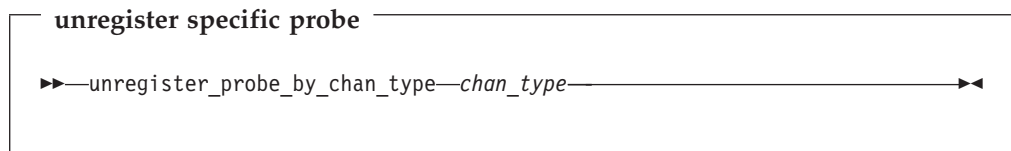
Channel device layer



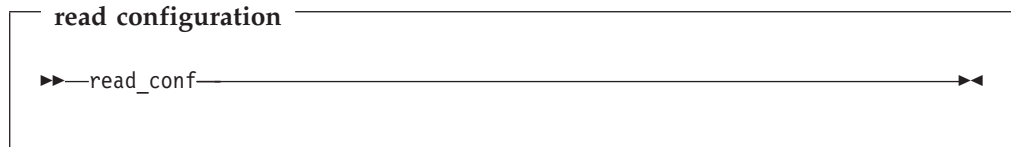
Call probe method for channels whose interrupts are not owned.



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

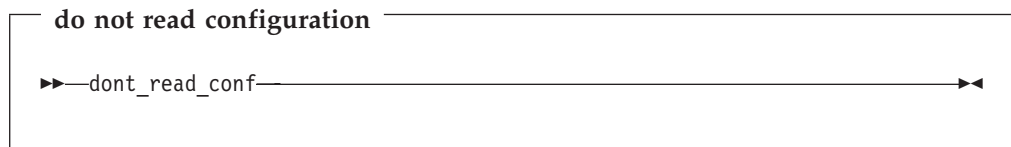


Unregister all probe functions which match the `chan_type` bitfield exactly. This is useful if you want a configuration to survive a kernel upgrade.



Read 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 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').

Channel device layer

Chapter 11. 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 than a physical connection.

The Linux for zSeries and S/390 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 and S/390 systems or between a Linux for zSeries and S/390 system and another operating system such as z/VM, VSE/ESA, 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, CTCMPC, and LCS)" on page 95.

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.)

CTC/ESCON device driver

Or for two network devices (CTC + ESCON):

```
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.

Note: “ctc” and “escon” are used as synonyms in the channel device layer. Be sure that the suffix numbers for CTC- and ESCON-devices are all different.

To scan a range of devices:

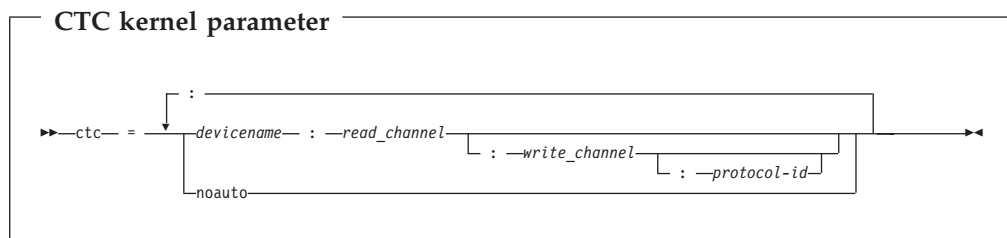
```
ctc,0x700,0x7ff,100
```

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.



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.

Note: To use IPv6 with CTC you have to set the protocol-id to 1.

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):

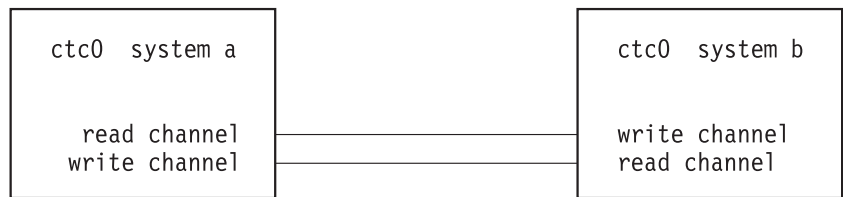


Figure 7. Connection of two systems via CTC (kernel)

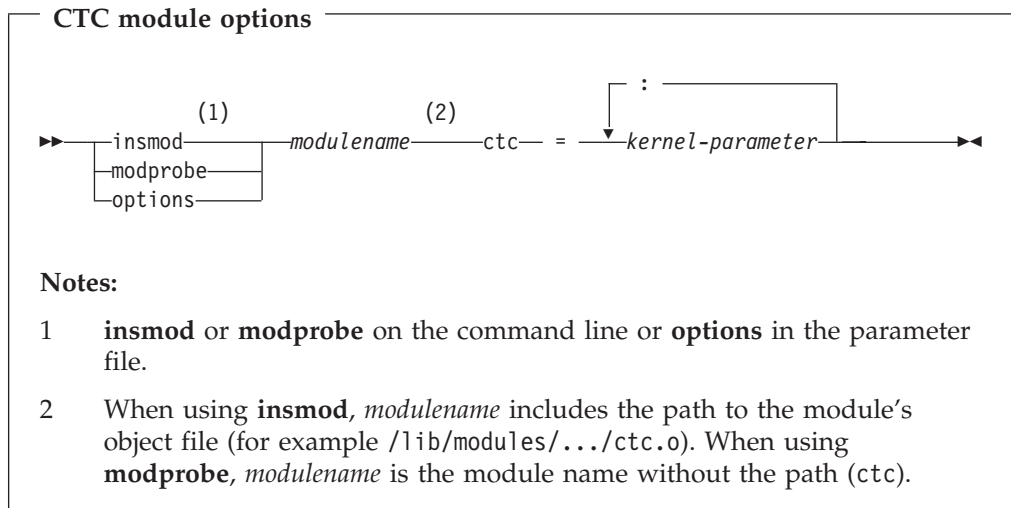
```
ctc=ctc0:0x600
```

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).



Where:

kernel-parameter

is as defined above in “Kernel parameter syntax” on page 106

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):

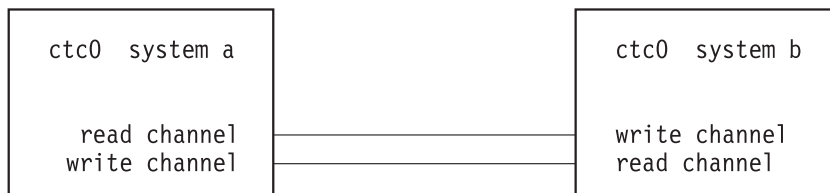


Figure 8. Connection of two systems via CTC (module)

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 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 TCP/IP 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 and S/390 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 and S/390, 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 and S/390 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

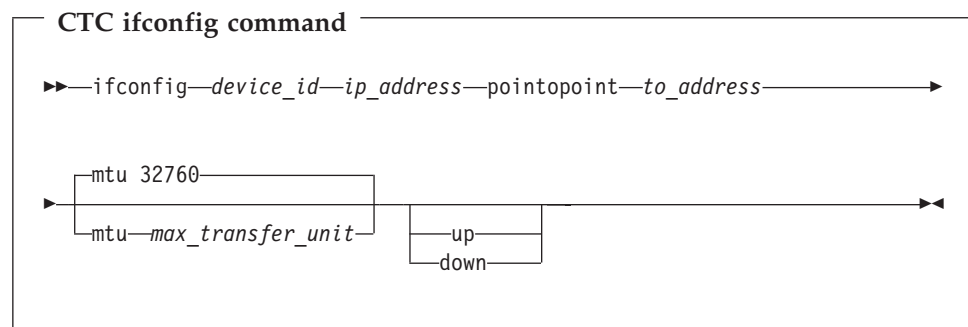
Activate the channels on the remote side. This again will vary depending on the operating system used on the remote side.

4. Activation on the Linux for zSeries and S/390 side

CTC/ESCON device driver

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:



Where:

device_id

identifies the device. (`ctc0` to `ctcn` or `escon0` to `esconn`)⁷

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.

Note: Device major number 43 is reserved on PC architecture for `/dev/isdn`. This number has been allocated to CTC/ESCON on Linux for zSeries and S/390 because there is no ISDN support on zSeries and S/390. 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 close

7. 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.

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 and S/390 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 and S/390 side using (for instance):

```
ifconfig escon0 down
```

2. Activate the channels on the remote side.
3. Activate the channels on the Linux for zSeries and S/390 side, for example:

```
ifconfig escon0 10.0.0.1 pointopoint 10.0.50.1 mtu 32760
```

Chapter 12. CTCMPC device driver

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

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

Features

The CTCMPC device driver allows Communications Server for Linux to provide:

- ESCON or real CTC connections between mainframes in basic mode, LPARs or VM guests.
- Virtual CTC/A connections between VM guests of the same VM system.
- Connections to VTAM on traditional mainframe operating systems.

CTCMPC devices

The CTCMPC device driver requires two I/O subchannels for each network device, a read subchannel and a write subchannel. The device numbers that correspond to the two subchannels must be configured for control unit type 3088.

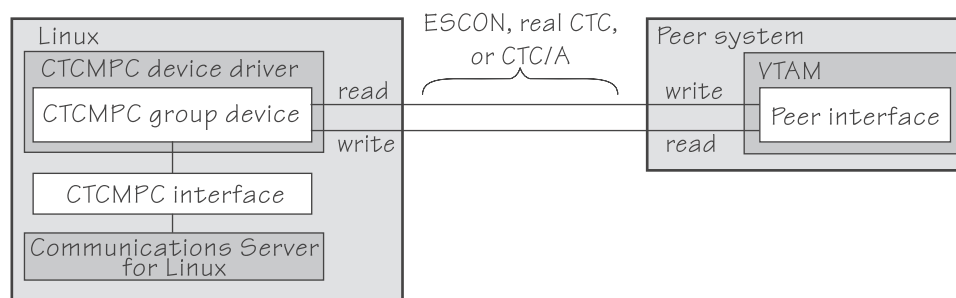


Figure 9. I/O subchannel interface

For communication with traditional mainframe operating systems, the following rules apply to the device numbers:

read must be even.

write must be the device number of the read subchannel plus one.

On the communication-peer operating system instance, read and write subchannels are reversed. That is, the write subchannel of the local interface is connected to the read subchannel of the remote interface and vice-versa.

Setting up the CTCMPC device driver

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

```
# modprobe ctcmpc
```

CTCMPC device configuration

For the syntax of CTCMPC configuration with the channel device layer see “Device identification (CTC/ESCON, CTCMPC, and LCS)” on page 95.

Refer to the Communications Server for Linux documentation for information on how to set up, activate, and work with CTCMPC interfaces.

For more information on configuring and using CTCMPC devices, refer to the Communications Server for Linux documentation.

Chapter 13. IUCV device driver

The Inter-User Communication Vehicle (IUCV) is a VM 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 and S/390 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.

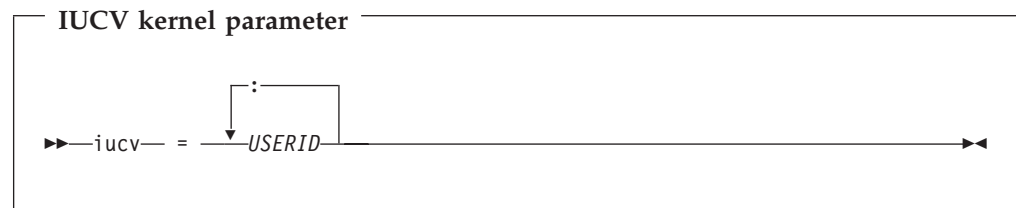
IUCV features

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
- Internet Protocol, Version 4 (IPv4) only

IUCV kernel parameter syntax

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



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 and S/390 machines:

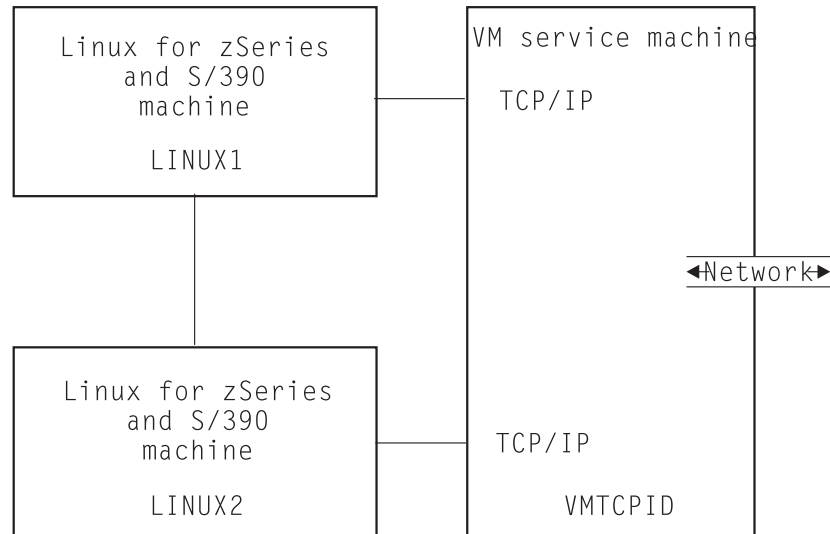


Figure 10. Connection of two systems using IUCV

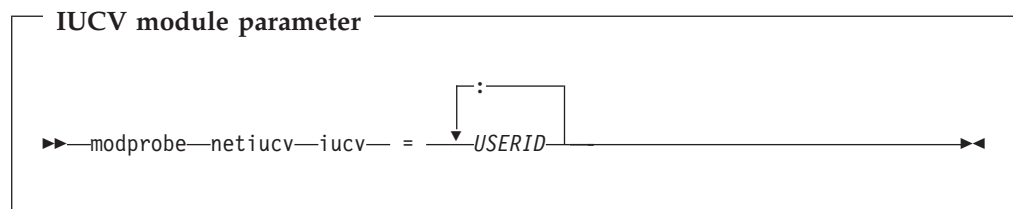
The command

```
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:



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

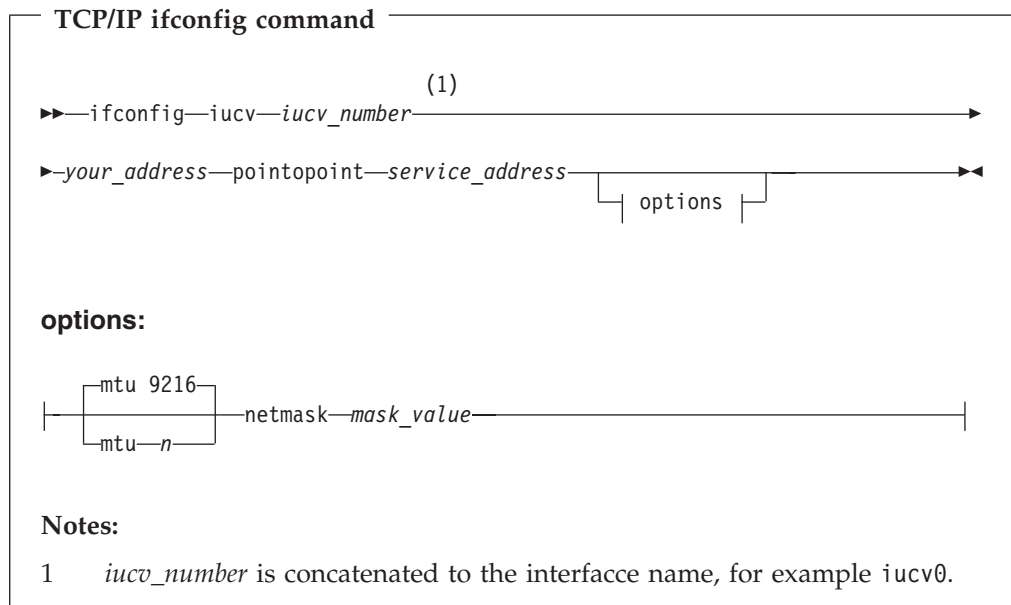
Start device_name1
Start device_name2
```

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:



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

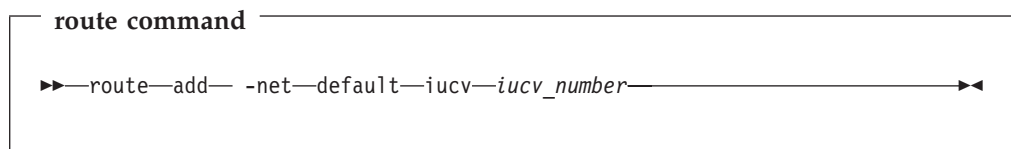
Address of the TCP/IP service machine to connect to

n

maximum transfer unit size. The default is 9216, which is suitable for use with the zSeries and S/390 Virtual Image Facility for Linux (VIF). The maximum value is 32764.

mask_value

Mask to identify addresses served by this connection



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: *z/VM TCP/IP Planning and Customization*, SC24-6019.

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 SC24-6001 at ibm.com/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	output	number of outstanding messages

Return value: type int

IUCV device driver

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	output	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 <code>msgid</code> is specified then <code>pathid</code> and <code>srccls</code> 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:

IUCV device driver

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 <code>b2f0_result = 5</code>.

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`.

IUCV device driver

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:
IPPTY	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
------	------	--------------	-------------

IUCV device driver

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 discontinuous 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
------	------	--------------	-------------

IUCV device driver

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.
trgc1s	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:
IPPTY	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_MessagePendingInterruptsFlag	0x40		
Priority_MessagePendingInterruptsFlag	0x80		
Nonpriority_MessageCompletionInterruptsFlag	0x20		
Priority_MessageCompletionInterruptsFlag	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,
```

IUCV device driver

```
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 14. LCS device driver

The LAN channel station device driver (LCS device driver) supports these Open Systems Adapter (OSA) features in non-QDIO mode:

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

To configure this device driver, use the channel device layer. For details on how to use the channel device layer, see Chapter 10, “Channel device layer,” on page 93.

The LAN Channel Station (LCS) network interface has two channels, one read channel and one write channel. This is very similar to the zSeries and S/390 CTC interface (see Chapter 11, “CTC/ESCON device driver,” on page 105). 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 the CHPID is connected to Token Ring or Ethernet
- Internet Protocol, Version 4 (IPv4) only

LCS channel device layer configuration

For the syntax of LCS configuration with the channel device layer see “Device identification (CTC/ESCON, CTCMPC, and LCS)” on page 95.

LCS channel device layer configuration example

```
chandev=noauto,lcs0,0x7c00,0x7c01,0,1,1,1
```

LCS device driver

Entered in the boot command line, this will make LCS device 0 use addresses 7c00 and 7c01 for input and output, let the driver decide on memory usage, use relative adapter 1, set IP checksumming on, and collect network statistics from the hardware.

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.

Chapter 15. QETH device driver for OSA-Express (QDIO) and HiperSockets

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

- OSA-Express:
 - Fast Ethernet
 - 1000Base-T Ethernet (z890 and z990)
 - Gigabit Ethernet
 - Token Ring
 - ATM (running Ethernet LAN emulation)
- OSA-Express2 (z890 and z990):
 - Gigabit Ethernet
 - 10 Gigabit Ethernet
- HiperSockets:
 - Virtual network devices that provide virtual networks within a zSeries mainframe.

A HiperSockets CHPID is controlled by the same device driver as an OSA-Express CHPID in QDIO mode. Most of the device driver parameters are common to the two types of CHPIDs.

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

The OSA-Express in QDIO mode is described in detail in *OSA-Express Customer's Guide and Reference*, SA22-7935.

This driver has been developed for the zSeries 64-bit and 31-bit architectures and for the S/390 31-bit architecture with version 2.4 of the Linux kernel.

Note: z/VM 4.2 and later releases can virtualize HiperSockets devices via "guest LANs" that can interconnect virtual machines. That is, z/VM simulates the HiperSockets function for communication among virtual machines that reside within the same z/VM system, without the need for IQD channels. In addition, z/VM 4.3 allows the virtualization of a QDIO LAN environment. For more information, refer to:

- <http://www.linuxvm.org/Info/HOWTOs/guestlan.html>
- z/VM documentation
- The redbook *zSeries HiperSockets*, SG24-6816

Introduction

You need four modules to configure OSA-Express CHPIDs in QDIO mode and HiperSockets:

- The `ipv6` module provides IPv6 support in the kernel and is usually compiled as a module.

QETH device driver

- The 8021q module provides IEEE 802.1Q VLAN support in the kernel and is also usually compiled as a module.
- The QDIO protocol governs the interface between the zSeries or S/390 and an OSA-Express CHPID. You need only load the qdio module, or build it into the kernel; no configuration is necessary.
- The QETH module controls the features themselves. Configuring the QETH module is described in this chapter.

The four modules can either be compiled into the kernel or installed as described in “Installing QETH.”

For HiperSockets or an OSA-Express CHPID 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.

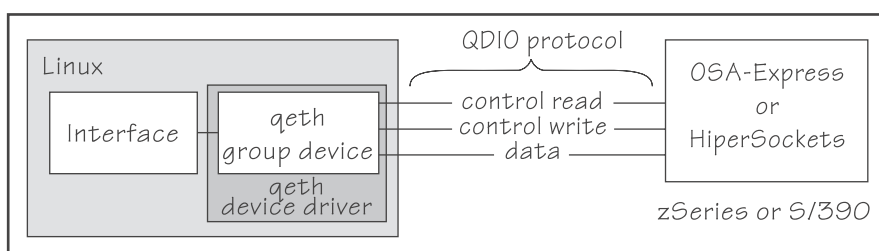


Figure 11. I/O subchannel interface

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 `ipv6.o` module is not compiled into the kernel, issue the **insmod** command:

```
insmod ipv6
```
4. If the `8021q.o` module is not compiled into the kernel, issue the **insmod** command:

```
insmod 8021q
```
5. If the QDIO driver is not compiled into the kernel, issue the **insmod** command:

```
insmod qdio
```
6. If the QETH driver is not compiled into the kernel, issue the **insmod** command for the `qeth.o` module:

```
insmod qeth
```
7. Issue the **ip** or **ifconfig** command with the desired parameters for any QDIO OSA or HiperSockets interfaces.

See also “Confirming that an IP address has been set” on page 157.

The interfaces to the devices are now set up.

QETH supported functions

The following functions are supported:

- Auto-detection of devices
- Primary and secondary routers (does not apply to HiperSockets)
- Multicast routing (does not apply to OSA-Express CHPID in QDIO mode)
- Priority queueing (does not apply to HiperSockets)
- Broadcast.

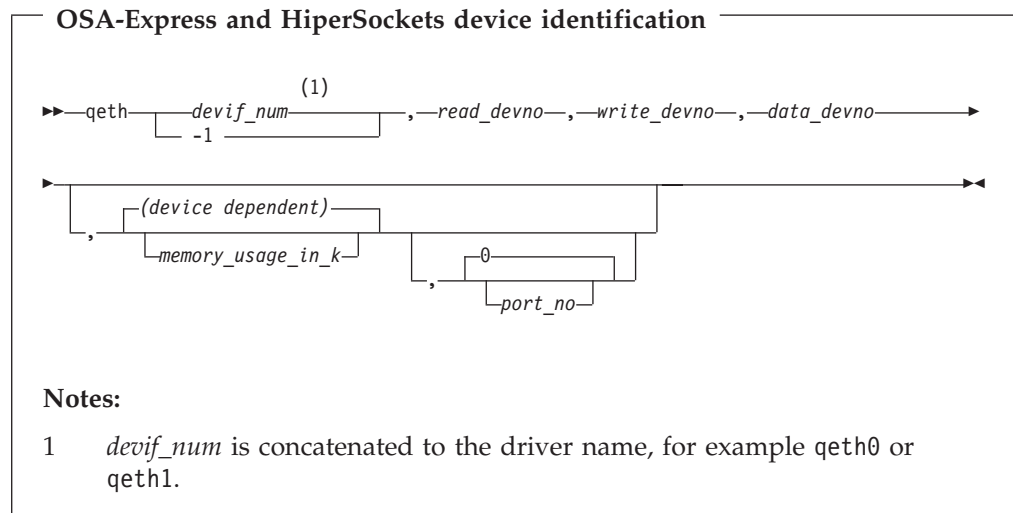
If a device supports broadcast traffic, this will be enabled automatically and reflected in the interface flags (seen in `ifconfig`). Broadcast traffic can then flow in and out of the device.

- 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 152.
- Virtual IP addresses (VIPA) , including Source VIPA (does not apply to HiperSockets). See Chapter 18, “VIPA – minimize outage due to adapter failure,” on page 237.
- HiperSockets. See “Examples: HiperSockets” on page 149 and “Example: HiperSockets” on page 153.
- VLAN. See Chapter 22, “Virtual LAN (VLAN) support,” on page 261.
- Query and purge of ARP data. See “qetharp - Query and purge OSA and HiperSockets ARP data” on page 197.
- Internet Protocol Version 6 (IPv6) (Ethernet interfaces only). See “Support for IP Version 6 (IPv6)” on page 156.
- SNMP (via the OSA-Express feature). See “osasnmpd – Start OSA-Express SNMP subagent” on page 189.
- Fast path router support (PFIX). See “use_pfix” on page 293 for usage information and security considerations.
- DHCP (does not apply to HiperSockets). See Chapter 24, “Enabling OSA-Express QDIO devices in Linux for DHCP and tcpdump,” on page 271.
- MAC-based addressing for qeth devices (see “The layer2 option” on page 278).
- Hardware checksumming (does not apply to HiperSockets).

Configuring QETH for OSA-Express and HiperSockets using the channel device layer

This section describes how to configure QETH for an OSA-Express CHPID in QDIO mode and HiperSockets with the channel device layer. Only the most common options are given here to illustrate the syntax; see Chapter 10, “Channel device layer,” on page 93 for full details of all channel device options.

The driver will normally use auto-detection to find all OSA-Express QDIO 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** parameter.



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 subchannel address (in hexadecimal preceded by 0x)

For OSA-Express devices, this address must be an even number.

write_devno

is the write subchannel address (in hexadecimal preceded by 0x)

For OSA-Express devices, this address must be one greater than the read channel address.

data_devno

is the data subchannel 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.) '0' specifies the default.

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 221). For more information, see "MTU considerations" on page 157.

port_no

is the relative port number of the CHPID. Except for OSA-Express ATM in QDIO mode set up for Ethernet LAN emulation, which allows the

configuration of two emulated ports through OSA/SF, HiperSockets and OSA-Express CHPIDs in QDIO mode use only port 0, the default port. A value of '-1' invokes the default. An OSA-Express ATM CHPID in QDIO mode set up for Ethernet LAN emulation allows configuration of two emulated ports through OSA/SF.

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 an OSA-Express ATM device 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 device, `hsi7` for HiperSockets, and `tr7` for a Token Ring device.

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 141 for the description of `devif_num`.

- If `chandev` is instructed to use device number names (`use_devno_names`), the interface number will be the device number of the read subchannel. For example, if the read subchannel has the device number `0xfd0c`, the interface name would be `eth0xfd0c` for an Ethernet device, `hsi0xfd0c` for a HiperSockets device, and `tr0xfd0c` for a Token Ring device. See "Commonly used options" on page 97 for the description of `use_devno_names`.

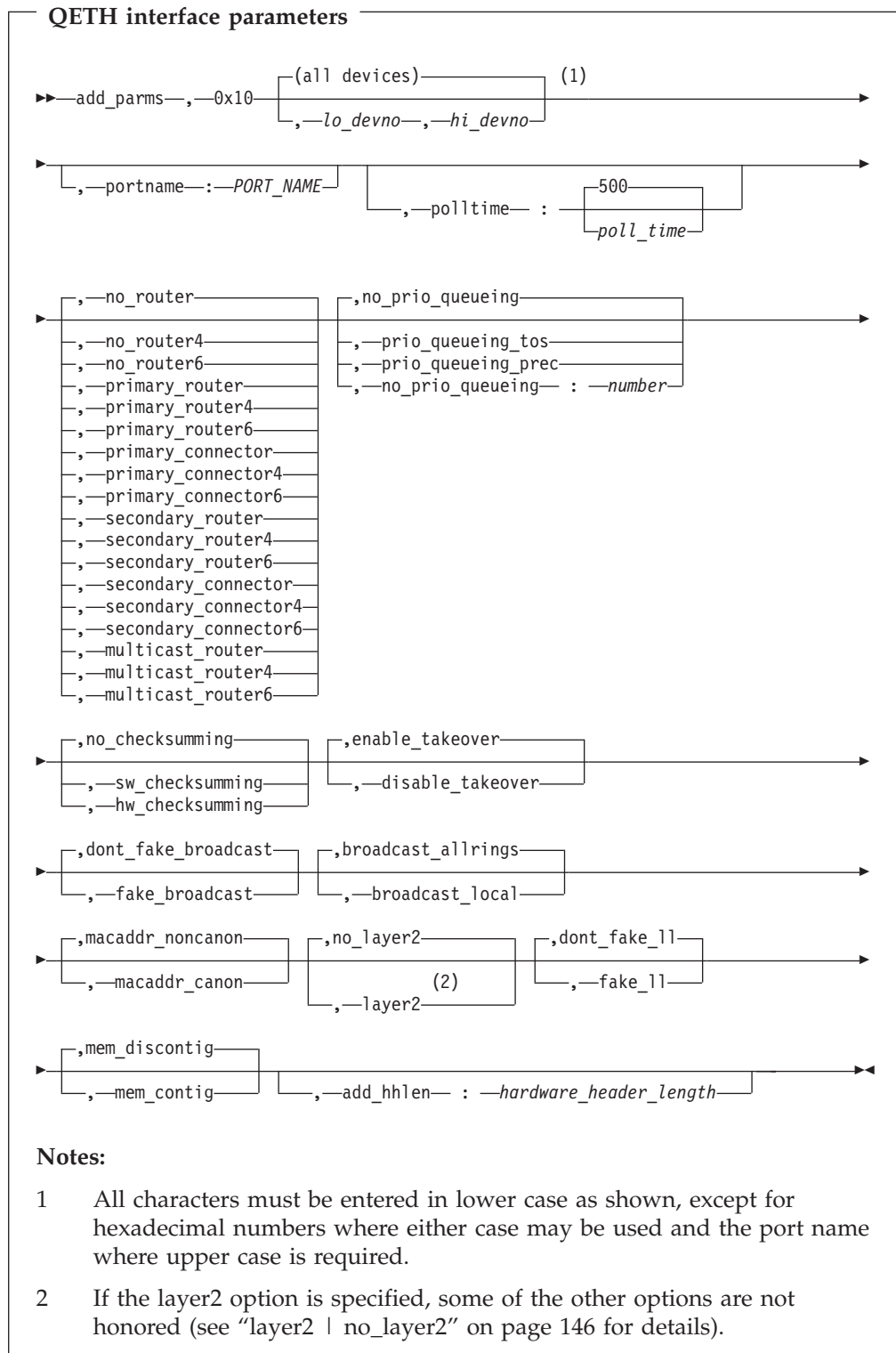
Port identification

For S/390 mainframes and z900 or z800 and with a microcode level earlier than Driver 3G - EC stream J11204, MCL032 (OSA level 3.33), you must associate each OSA-Express CHPID in QDIO mode with a port name. To do this, use the channel device layer **add_parms** command with the portname option.

See "QETH parameter syntax" on page 144 for the **add_parms** syntax and "1: Basic configuration" on page 148 for a usage example.

HiperSockets devices do not require a port name.

QETH parameter syntax



Where:

add_parms

Used to pass additional parameters to the driver.

0x10 Identifies the device as an OSA-Express CHPID in QDIO mode or a HiperSockets CHPID.

lo_devno,hi_devno

Specifies the desired device address range.

PORT_NAME

Must be specified for each OSA-Express CHPID in QDIO mode. Identifies the port for sharing by other operating system images. On z900, z800, and on S/390 mainframes the *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.

Note: This parameter is required only for z900, z800 and S/390 mainframes. For z890, z990 and later mainframes you are advised to omit it. See also “Port identification” on page 143

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.

no_router | no_router4 | no_router6 | primary_router | primary_router4 | primary_router6 | primary_connector | primary_connector4 | primary_connector6 | secondary_router | secondary_router4 | secondary_router6 | secondary_connector | secondary_connector4 | secondary_connector6 | multicast_router | multicast_router4 | multicast_router6

Specifies the router options. Options that end with a 4 or a 6 only apply to IPv4 or IPv6, respectively. The default is **no_router**.

For HiperSockets: The connector options are used for the HiperSockets Concentrator. The **multicast_router** is assigned to the OSA to allow multicast and Broadcast packets to be passed up by the OSA to the Concentrator xcec-bridge code. For details, see Chapter 23, “HiperSockets Network Concentrator,” on page 265.

For an OSA-Express CHPID 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.

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.

On zSeries mainframes, you can reset the status by issuing:

```
echo no_router ifname > /proc/qeth
```

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.

See "use_pfix" on page 293 for a configuration option that you might want to consider for your router.

prio_queueing_tos | prio_queueing_prec | no_prio_queueing |

no_prio_queueing: *number*

For HyperSockets: This parameter is ignored.

Specifies the type of priority queuing to be used. See "Priority queuing" on page 155 for details. **no_prio_queueing** is equivalent to **no_prio_queueing: 2** (the default queue).

hw_checksumming | sw_checksumming | no_checksumming

Specifies whether error detection is to be performed by the OSA adapter hardware, the TCP/IP stack, or is not required. If you specify **hw_checksumming** for a card that does not support it, a kernel message is issued and the TCP/IP stack performs the checksumming instead of the card. Currently, hardware checksumming is provided only by OSA adapter hardware for a z990 for the 1000Base-T Ethernet, 10 Gigabit Ethernet, and Gigabit Ethernet CHPIDs running in QDIO mode. Checksumming is restricted to incoming packets.

/proc/qeth provides information about the checksumming method of the device.

enable_takeover | disable_takeover

Allow/do not allow IP address takeover.

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.

layer2 | no_layer2

Use MAC based routing mechanism (see "The layer2 option" on page 278).

This option applies to OSA-Express only.

If the layer2 option is specified, the following are implied:

- **no_router**
- **disable_takeover**
- **dont_fake_broadcast**
- **broadcast_allrings**
- **macaddr_noncanon**

- dont_fake_ll

Any alternatives to these options are ignored, if specified. Specifying hw_checksumming is also ignored.

You can specify portname, poll time, priority queueing, the contiguous I/O buffers setting, and the additional hardware-header space as usual.

fake_ll | dont_fake_ll

Insert a fake LLC header in all incoming packets (see “The fake_ll option” on page 279).

mem_contig | mem_discontig

mem_contig tries to allocate contiguous I/O buffers of up to 64 K size which can lead to a slight performance improvement. However, it can also cause problems during device initialization, for example, after a file system check during the boot process.

Recommendation: Do not use mem_contig.

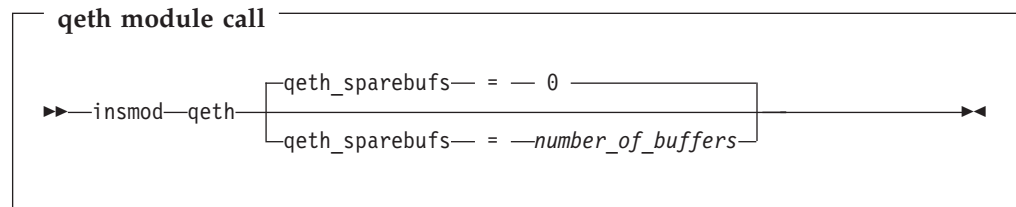
add_hhlen

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 IP/IP tunnels, in general for software that makes use of free space in front of packets.

OSA-Express CHPID in QDIO mode qeth module parameter syntax

If qeth is loaded as a module, you can optionally use the qeth_sparebufs module parameter to preallocate spare 64 K buffers. These buffers are for low-memory high-traffic situations when the queue memory cannot be reused fast enough. However, the buffers are to be allocated contiguously and it is not guaranteed that all buffers are allocated successfully. If this parameter is omitted, no spare buffers are preallocated.

Recommendation: Omit qeth_sparebufs unless you are sure that you need spare buffers.



where:

number_of_buffers

the number of preallocated spare buffers.

qeth_sparebufs can also be specified in a single line in /etc/modules.conf with the following format:

options qeth qeth_sparebufs=<number_of_buffers>

OSA-Express CHPID 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
```

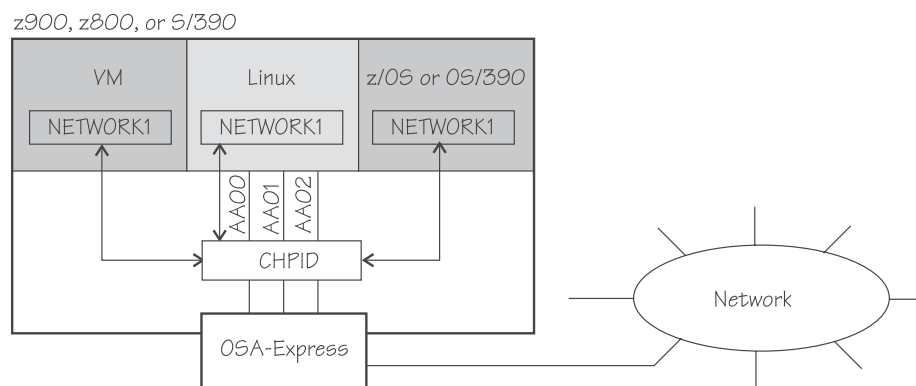
This tells the channel device layer to force qeth1 (if detected) to use device addresses 7c00, 7c01 and 7c02, allocate four megabytes of buffer space, and use the default port.

Examples: OSA-Express CHPID in QDIO mode

1: Basic configuration

In this example a single OSA-Express CHPID is being used to connect a Linux for zSeries system or a Linux for S/390 system to a network.

Hardware configuration – OSA-Express connecting Linux for zSeries and S/390 to a network



Software configuration – OSA-Express connecting Linux for zSeries and S/390 to a network

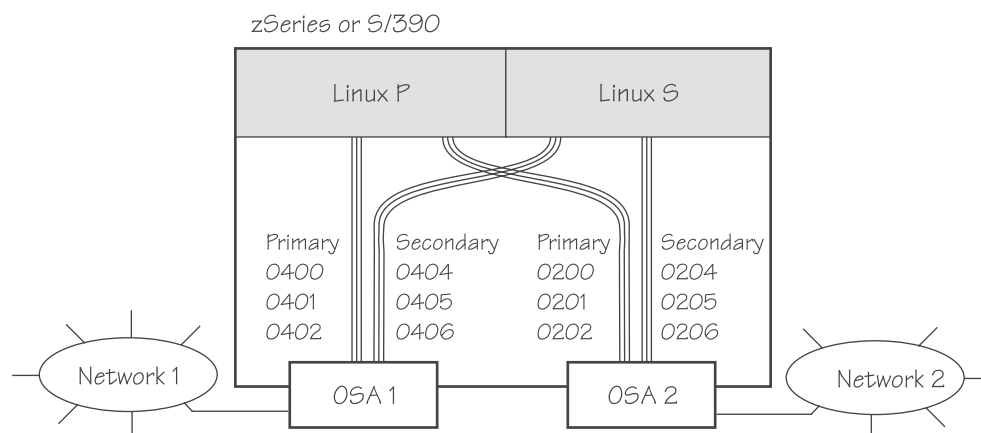
With the channel device layer the load commands for this configuration are:

```
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 or S/390 can 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 and S/390 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 and S/390 as a router

LPAR 1 – This LPAR is configured to be the primary routing LPAR:

```
add_parms,0x10,0x400,0x402,primary_router,portname:OSACARD1
qeth0,0x400,0x401,0x402
add_parms,0x10,0x200,0x202,primary_router,portname:OSACARD2
qeth1,0x200,0x201,0x202
```

LPAR 2 – This LPAR is configured to be the secondary routing LPAR:

```
add_parms,0x10,0x404,0x406,secondary_router,portname:OSACARD1
qeth0,0x404,0x405,0x406
add_parms,0x10,0x204,0x206,secondary_router,portname:OSACARD2
qeth1,0x204,0x205,0x206
```

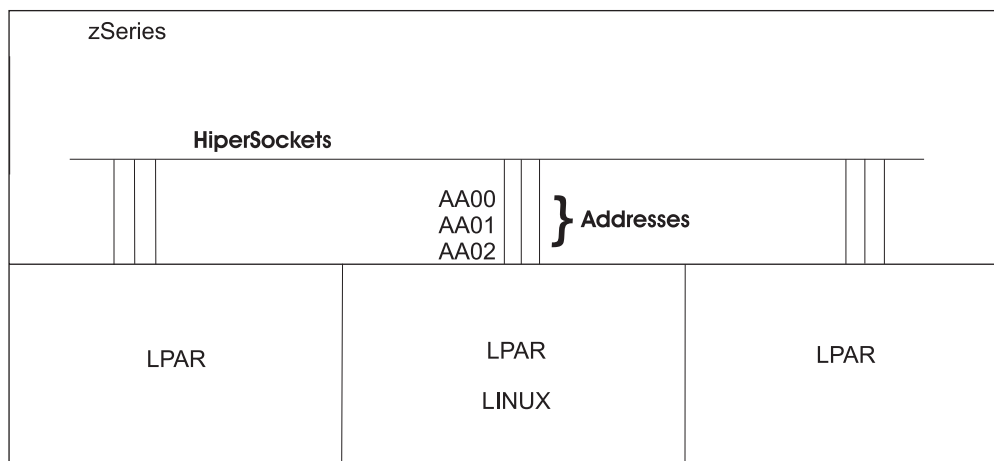
Note: In conjunction with the layer2 option (see “The layer2 option” on page 278), you set up your Linux routers as you would on discrete hardware. For example, the `primary_router` and `secondary_router` options are meaningless in conjunction with the layer2 option.

Examples: HiperSockets

1: Basic configuration

In this example a single HiperSockets CHPID is being used to connect a Linux for zSeries and S/390 system to a network.

Hardware configuration – HiperSockets connecting Linux for zSeries and S/390 to a network



Software configuration – HiperSockets connecting Linux for zSeries and S/390 to a network

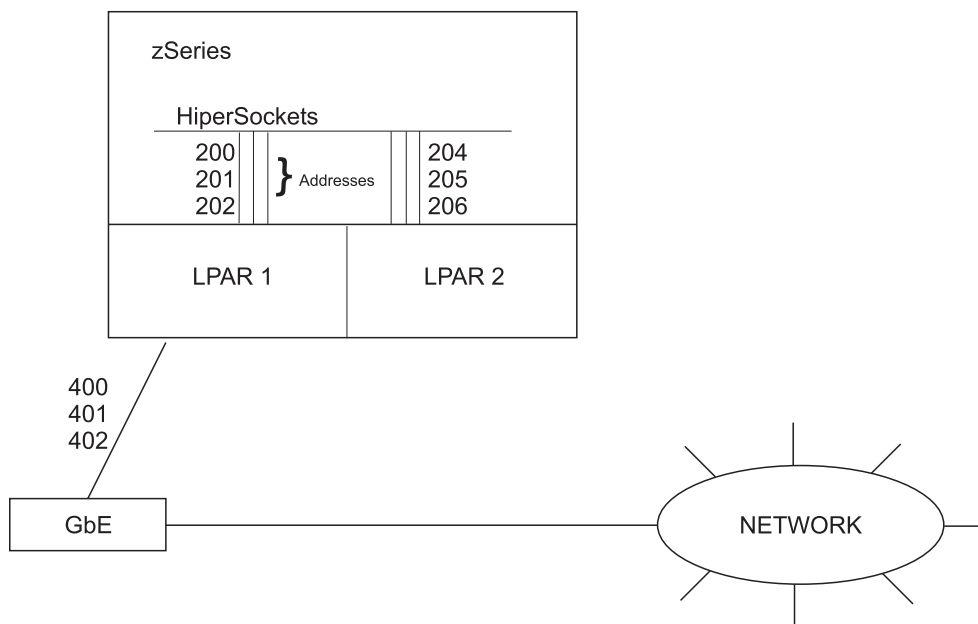
With the channel device layer the load commands for this configuration are:

```
qeth-1,0xAA00,0xAA01,0xAA02
```

2: Router configuration

This example shows how Linux systems running on different LPARs in a zSeries or S/390 system can use HiperSockets to communicate with a network or to act as a router between networks.

Hardware configuration – HiperSockets, an OSA-Express CHPID, and Linux for zSeries and S/390 as a router



In this example it is assumed that Linux is configured as a router in LPAR 1.

Software configuration – HiperSockets, an OSA-Express CHPID, and Linux for zSeries and S/390 as a router

LPAR 1 – uses subchannels 200 - 202 for connecting to the HiperSockets CHPID and 400 - 402 to connect to the OSA-Express CHPID. It is then able to route packets in and out.

```
add_parms,0x10,0x400,0x402,primary_router,portname:OSACARD1
qeth0,0x400,0x401,0x402
qeth1,0x200,0x201,0x202
```

LPAR 2 – uses subchannels 204 - 206 as a network client:

```
qeth-1,0x204,0x205,0x206
```

OSA-Express CHPID in QDIO mode and HiperSockets – Preparing the connection

Activating an OSA-Express CHPID 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. For HiperSockets, you must use the same MTU size as that which is defined on the remote side.

The correct 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 CHPID 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
```

Duplicate IP addresses

The OSA-Express adapter card in QDIO mode recognizes duplicate IP addresses on the same OSA-Express card or in the network using ARP. Without any special action, duplicate addresses cannot be set. They are displayed by **ifconfig** (as there is no means for a network device driver to report such failures). However, the system log in `/var/log/messages` or via command **dmesg** will indicate that an IP address could not be set.

This behavior can be overridden using `/proc/qeth_ipa_takeover`. In this `/proc`-file, addresses can be specified, that will be set regardless of any duplicates on the same OSA-Express card or in the LAN. Gratuitous ARP is used to provide a kind of takeover functionality in other hosts' ARP caches. See the section "IP address takeover" on page 152 for further information on IP takeover.

IP address takeover

You cannot configure for IP address takeover as described in this section if you are using the layer2 option. The information in this section assumes that you have not set the layer2 option (see “layer2 | no_layer2” on page 146).

It is possible to add and remove ranges of IP addresses for an OSA-Express CHPID in QDIO mode or HiperSockets CHPID 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 CHPID. If another LPAR on the same CHPID has already registered for that IP address, this association is 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.

IP address takeover is enabled by default and can be switched off at the granularity of individual devices. To switch off IP address takeover, use the “disable_takeover” parameter in the `add_parms` statement of the `qeth chandev` parameters (see “QETH parameter syntax” on page 144). For a successful takeover, takeover must be enabled on the image taking over the address, and for HiperSockets additionally on the image that gives up its address.

The registered addresses are held in the `/proc/qeth_ipa_takeover` 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>]`

Tip: Use the `qethconf` utility to issue these commands (see “`qethconf` - configure qeth devices” on page 199).

`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 “Confirming that an IP address has been set” on page 157.

Example: OSA-Express CHPID 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 eth0 interface, and removes it from other OS images if necessary.

The IP address must be different from that previously set on the device or no action will be taken. For example, if there are two Linux images with the same IP address defined for the same CHPID, and image 1 actually “owns” the address, image 2 can take over that IP address from image 1. However, since image 2 also has that same address already defined for the CHPID, a different address needs to be assigned to the CHPID on image 2 first. Then, the original address can be reassigned to the CHPID, and image 2 will take over the IP address from image 1. As an example, if the IP address assigned to the CHPID on both images is 192.168.10.5, the following two commands could be issued on image 2 in order for it to take over the address from image 1:

```
ifconfig eth0 0.0.0.0
ifconfig eth0 192.168.10.5
```

The second line alone will not take over the IP address from another OS image if the IP address is the same as that set by the other OS image.

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 from that previously set on the device or no action will be taken. For example, if there are two Linux images with the same IP address defined for the same CHPID and image 1 actually “owns” the address, image 2 can take over that IP address from image 1. However, since image 2 also has that same address already defined for the CHPID, a different address needs to be assigned to the CHPID on image 2 first. Then, the original address can be reassigned to the CHPID, and image 2 will take over the IP address from image 1. As an example, if the IP address assigned to the CHPID on both images is 192.168.10.5, the following two commands could be issued on image 2 in order for it to take over the address from image 1:

```
ifconfig hsi0 0.0.0.0
ifconfig hsi0 192.168.10.5
```

The second line alone will not take over the IP address from another OS image if the IP address is the same as that set by the other OS image.

Proxy ARP

Note: You cannot configure for proxy ARP as described in this section if you are using the layer2 option. The information in this section assumes that you have not set the layer2 option (see “layer2 | no_layer2” on page 146).

The following commands implement a proxy ARP capability:

- `add_rxip4 <addr>:<interface>`
- `add_rxip6 <addr>:<interface>`
- `del_rxip4 <addr>:<interface>`
- `del_rxip6 <addr>:<interface>`

Tip: Use the `qethconf` utility to issue these commands (see “`qethconf - configure qeth devices`” on page 199).

The `add_rxip4` and `add_rxip6` commands cause gratuitous ARP packets to be sent out of the specified interface for the specified address. The `del_rxip4` and `del_rxip6` commands disable this function for the specified interface and address.

Example:

G1, G2, and G3 are Linux guests (connected, for example, via IUCV to R), reached

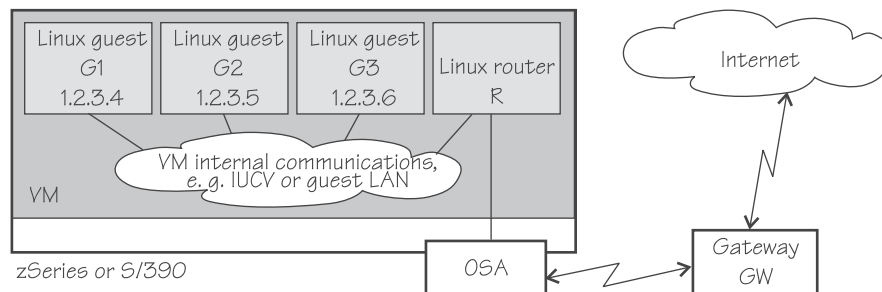


Figure 12. Example of Proxy ARP usage

from GW (or the outside world) via R. R is the ARP proxy for G1, G2, and 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, and 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 also “Confirming that an IP address has been set” on page 157.

More information on Proxy ARP is available from

<http://www.sjdwjweis.com/linux/proxyarp/>

OSA-Express CHPID in QDIO mode – Virtual IP address (VIPA)

Notes:

1. This section does not apply to HiperSockets.
2. You cannot configure for VIPA as described in this section if you are using the layer2 option. The information in this section assumes that you have not set the layer2 option (see “layer2 | no_layer2” on page 146).

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:eth0 > /proc/qeth_ipa_takeover
```

Tip: Use the qethconf utility to issue these commands (see “qethconf - configure qeth devices” on page 199).

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> specifies the interface name to which the address range is bound.

For an example of how to use VIPA, see Chapter 18, “VIPA – minimize outage due to adapter failure,” on page 237.

Note: See also “Confirming that an IP address has been set” on page 157.

Priority queuing

Note: This does not apply to HiperSockets.

An OSA-Express CHPID 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 the IP type of service and the number of the queue that is used 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 <code>qeth</code> parameter.)

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 only to OSA-Express Ethernet adapters running in QDIO mode.

There are noticeable differences between the IP stacks for versions 4 and 6. Some concepts in IPv6 are different from IPv4, such as neighbor discovery, broadcast, and IPSec. IPv6 uses a 16-byte address field, while the addresses under IPv4 are 4 bytes in length.

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 152, “OSA-Express CHPID in QDIO mode – Virtual IP address (VIPA)” on page 155, and “Proxy ARP” on page 154, for example).

Note: See also “Confirming that an IP address has been set” on page 157.

Stateless autoconfiguration in IPv6

Note: The information in this section does not apply if you have set the `layer2` option (see “`layer2` | `no_layer2`” on page 146).

For IPv6, stateless autoconfiguration generates a unique IP addresses for an operating system, based on the MAC address of the network adapter the operating system is using. This results in conflicts in mainframe environments where multiple Linux instances share an OSA-Express adapter card. The generated IP addresses would be identical for all Linux instances that share a card.

This problem is resolved by a kernel patch included in the S/390-specific kernel patches on developerWorks. The patch affects `struct net_device` and thus all networking device drivers. After you have applied the patch, there is a new kernel option (see “Prepare `net_device` struct for shared IPv6 cards” on page 315). If you

select this recommended kernel option, stateless autoconfiguration will generate a unique IP addresses for the Linux system, even if it shares an OSA-Express adapter card with other operating systems.

Querying and purging OSA and HiperSockets ARP data

This function is implemented by the `qetharp` utility. See “`qetharp` - Query and purge OSA and HiperSockets ARP data” on page 197 for details.

Confirming that an IP address has been set

The following applies if you have not set the `layer2` option (see “`layer2` | `no_layer2`” on page 146):

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

There are usually failure messages of the form “could not set IP address” or “duplicate IP address” in the kernel messages. You can display these messages with `dmesg`. For most distributions you can also find the messages in `/var/log/messages`.

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

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

MTU considerations

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

On HiperSockets, the maximum MTU size is restricted by the maximum frame size as announced by the licensed internal code (LIC). The maximum MTU is equal to the frame size minus 8 KB. Hence, the possible frame sizes of 16 KB, 24 KB, 40 KB or 64 KB result in maximum MTU sizes of 8 KB, 16 KB, 32 KB or 56 KB, respectively.

The MTU size defaults to the correct settings for both HiperSockets and OSA-Express CHPIDs in QDIO mode. As a result, you need not specify the MTU size when activating the interface.

Note that, on heavily loaded systems, MTU sizes exceeding 8 KB can lead to memory allocation failures for packets due to memory fragmentation. A symptom of this problem are messages of the form “order-N allocation failed” in the system

QETH device driver

log; in addition, network connections will drop packets, in extreme cases to the extent that the network is no longer usable.

As a workaround, use MTU sizes at most of 8 KB (minus header size), even if the network hardware allows larger sizes (for example, HiperSockets or 10 Gigabit Ethernet).

Chapter 16. Linux for zSeries and S/390 CLAW device driver

Common Link Access to Workstation (CLAW) is a point-to-point protocol. A CLAW device is a channel connected device that supports CLAW protocol. These devices may be used to connect your Linux for zSeries and S/390 system to another system, for example a RISC System/6000[®] or a Cisco CIP.

You can use CLAW with or without the channel device layer. For the channel device layer description, see “CLAW with the channel device layer” below.

Without the channel device layer you can compile the CLAW driver into the kernel or start CLAW as a module. See “CLAW without the channel device layer” on page 161.

CLAW features

The CLAW driver supports up to 256 devices. Use any valid device address for a CLAW device.

CLAW supports a default MTU of 4096 bytes. You can change the MTU by altering the code and recompiling. You can specify MTU values up to 36K. Keep in mind only one I/O is done per buffer and only one packet of size MSS⁸ is written per I/O.

You can check the status of a CLAW device with `cat /proc/claw` or `ifconfig` and also by `cat /proc/chandev` if the channel device layer is used.

CLAW with the channel device layer

This section describes how to use the CLAW driver with the channel device layer. Only the most common options are given here to illustrate the syntax; see Chapter 10, “Channel device layer,” on page 93 for full details of all channel device options.

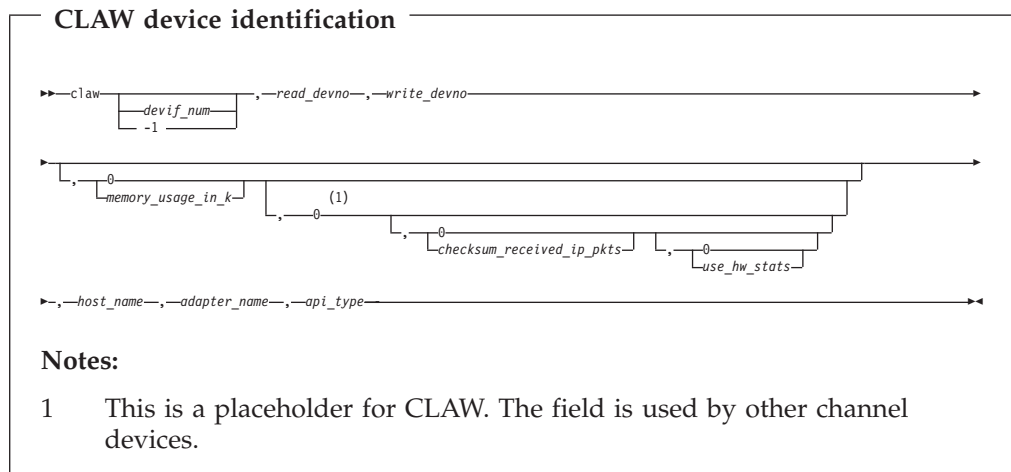
Channel device layer configuration

Note: There is a difference between the format of the CLAW parameters using the channel device layer and the format using a module or kernel implementation, in that the sets of parameters are separated by a comma ‘,’ instead of a colon ‘:’.

Note: The CLAW device driver cannot at present handle a range of devices in a single call (as other device drivers do) because of the need to specify host name, adjacent host and API type for each device.

The syntax for using CLAW with the channel device layer follows below.

8. maximum segment size – for CLAW this is effectively the same as MTU (maximum transmission unit)



Where:

claw specifies the channel device type

host_name

is the name of this host (1 to 8 characters).

adapter_name

is the name of the adjacent host (1 to 8 characters).

api_type

is the type of API expected. This can be any name of 1 to 8 characters (common values are API and TCPIP).

and the other parameters are as described in "Device identification (CTC/ESCON, CTCMPC, and LCS)" on page 95.

Configuration example

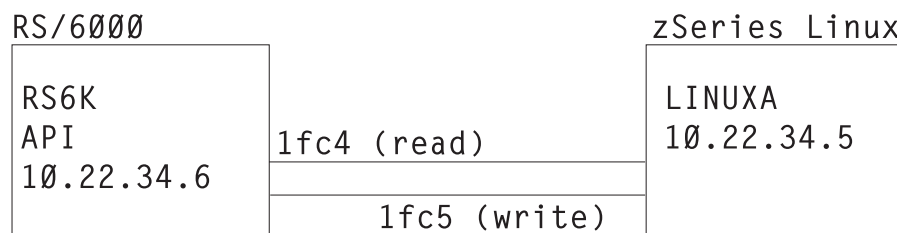


Figure 13. Connection of two systems via CLAW

You can add and delete devices on the Linux boot command line, or you can edit `chandev.conf`. Changes to `chandev.conf` will only take effect after you reboot or after issuing: `echo reset_conf;read_conf > /proc/chandev`. Assuming the configuration shown in Figure 13, a sample partial `chandev.conf` could be:

```

use_devno_names
claw0,0x1fc4,0x1fc5,400,0,0,LINUXA,RS6K,API

```

which allocates 400 kilobytes for buffers (typically 50 read and 50 write buffers), and the two zeroes ('0,0,') mean 'do not checksum received ip packets' and 'do not use hardware statistics'.

Alternatively, the parameter file could be coded to start the device at boot time:

```
chandev=use_devno_names,claw0,0x1fc4,0x1fc5,400,0,0,LINUXA,RS6K,API
```

This parameter file should be used together with an ifconfig statement such as:

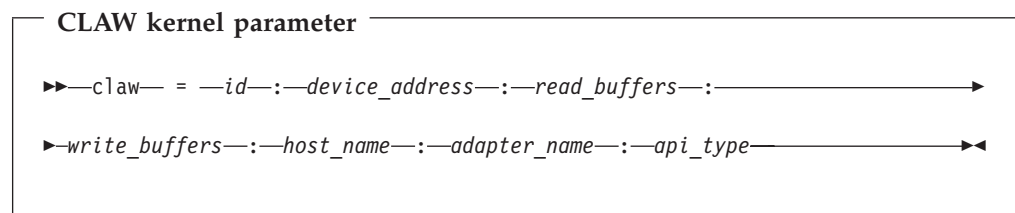
```
ifconfig claw1fc4 10.22.34.5 netmask 255.255.255.248 dstaddr 10.22.34.6
```

CLAW without the channel device layer

You can compile the CLAW driver into the kernel or start it as a module.

Kernel parameter syntax

There are no defaults for the CLAW driver. The syntax of the kernel parameter for CLAW is as follows:



This parameter may be repeated (including the 'claw=' and separated by commas) for additional CLAW devices.

The meaning of the variables are:

id The number (0 through 255) which identifies the CLAW device (claw0 to claw255). For example 3 would denote device claw3.

device_address

The address of the read channel (which must be an even-numbered hexadecimal device address). The write channel address is automatically allocated, and is numerically one greater than the read channel. The device address must be in the form 0xnnnn, for example 0x1cd2. This would give a read channel of 1cd2 and a write channel of 1cd3.

read_buffers, write_buffers

The number of buffers, each currently 4096 bytes in size, to be used for read and write operations. Increasing the number of buffers used can improve performance up to a point.

host_name

The name of this host. This name is used in CLAW negotiations with the adjacent CLAW host and must match the definition coded for *adapter_name* in that machine.

adapter_name

The host name of the adjacent host. This name is used in CLAW negotiations with the adjacent CLAW host and must match the definition coded for *host_name* in that machine. This is the name expected from the remote side during CLAW negotiation.

api_type

The type of API expected. It can be any name of 1 to 8 characters (common values are API and TCPIP).

Kernel example

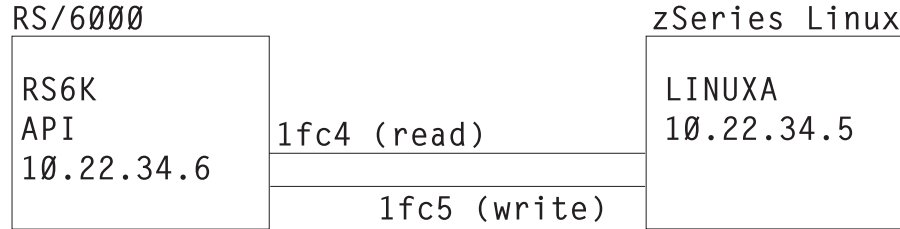


Figure 14. RS/6000 machine connected to a Linux system

Figure 14 shows an example of an RS/6000[®] machine connected to a Linux system. On the Linux system use a command line such as:

```
claw=0:0x1fc4:15:25:LINUXA:RS6K:API
```

and an ifconfig statement such as:

```
ifconfig claw0 10.22.34.5 netmask 255.255.255.248 dstaddr 10.22.34.6
```

To add a second CLAW device (at addresses 023c and 023d, connecting to system Z123 using TCPIP):

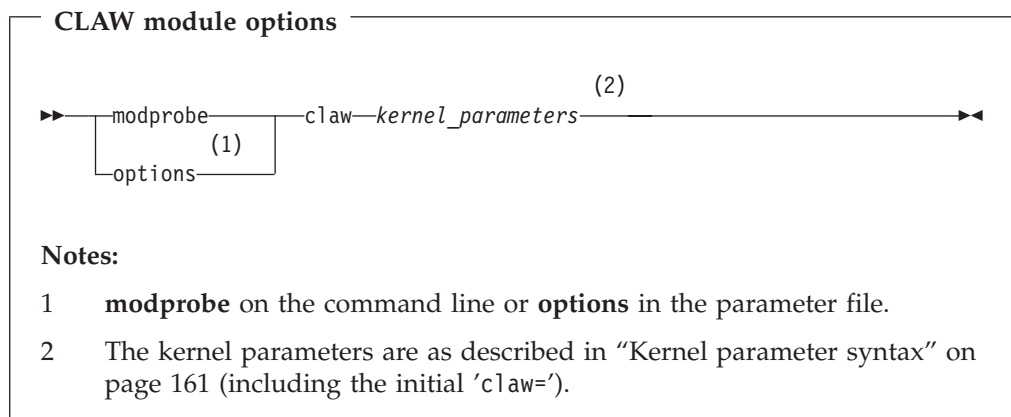
```
claw=0:0x1fc4:15:25:LINUXA:RS6K:API,claw=1:0x23c:20:20:LINUXA:Z123:TCPIP
```

with a second ifconfig statement such as:

```
ifconfig claw1 10.22.34.5 netmask 255.255.255.248 dstaddr 10.22.34.7
```

Module parameter syntax

If it is not included in the kernel, CLAW can be loaded as a module. You can either pass parameters using modprobe or in the parameter file "/etc/modules.conf" or "/etc/conf.modules" (the name depends on the Linux distribution). The syntax of the module parameters passed to modprobe is as follows:



Module example

For the same hardware configuration as in "Kernel example," the module install command would be:

```
modprobe claw claw=0:0x1Fc4:15:25:LINUXA:RS6K:API
```

Alternatively the /etc/modules config line would be:

```
options claw claw=0:0x1fc4:15:25:LINUXA:RS6K:API
```

Both these require an `ifconfig` statement such as:

```
ifconfig claw0 10.22.34.5 netmask 255.255.255.248 dstaddr 10.22.34.6
```

For two devices:

```
modprobe claw claw=0:0x1fc4:15:25:LINUXA:RS6K:API,claw=1:0x23c:20:20:LINUXA:Z123:TCPIP
```

or

```
options claw claw=0:0x1fc4:15:25:LINUXA:RS6K:API,claw=1:0x23c:20:20:LINUXA:Z123:TCPIP
```

with

```
ifconfig claw0 10.22.34.5 netmask 255.255.255.248 dstaddr 10.22.34.6  
ifconfig claw1 10.22.34.5 netmask 255.255.255.248 dstaddr 10.22.34.7
```

Part 4. Installation/configuration commands and parameters

This section describes configuration parameters for Linux for zSeries and S/390 and tools available for configuration and IPL. It also describes how to implement selected technologies in a Linux for zSeries and S/390 environment.

These are described in the chapters:

- Useful Linux commands:
 - dasdfmt - Format a DASD
 - dasdview - Display DASD characteristics
 - fdasd - Partition a DASD
 - osasnmppd - Start OSA-Express SNMP subagent
 - qetharp - Query and purge OSA and HiperSockets ARP data
 - qethconf - qeth configuration tool
 - 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.
- VARY ON/OFF events - notifications provided to device drivers and toggling CHPIDs logically on- or offline under Linux.
- Multipathing support
- Monitor stream support for z/VM
- Virtual LAN (VLAN) support
- HiperSockets network concentrator
- DHCP and tcpdump support for OSA-Express QDIO devices in Linux
- Kernel parameters:
 - cio_ignore
 - cio_msg
 - cio_notoper_msg
 - ipldelay
 - maxcpus
 - mem
 - noinitrd
 - ramdisk_size
 - ro
 - root
 - use_pfix

- vmhalt
- vmpoff
- Overview of the parameter line file.

Note: For tools related to taking and analyzing system dumps, see *Linux for zSeries and S/390 Using the Dump Tools*, LNX-1318.

Note

For prerequisites and restrictions refer to the kernel 2.4 June 2003 stream pages on developerWorks at:

ibm.com/developerworks/linux/linux390/june2003_recommended.shtml

If you are routed to the top-level page: On the left navigation bar, click **Kernel 2.4** . Then click the stream with which you want to work.

Chapter 17. Useful Linux commands

This chapter describes commands which have been created to install and configure Linux for zSeries and S/390:

- dasdfmt
- dasdview
- fdasd
- osasnmppd
- qetharp
- qethconf
- snIPL
- zIPL

It also summarizes standard Linux commands used during the installation, configuration and initial startup of Linux for zSeries and S/390. These are:

- ifconfig
- insmod
- modprobe
- lsmod
- depmod
- mke2fs
- vconfig

Note: For tools related to taking and analyzing system dumps, see *Linux for zSeries and S/390 Using the Dump Tools*, LNUX-1318.

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 or S/390 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 “DASD partitioning” on page 14 for further information about partitioning DASD.

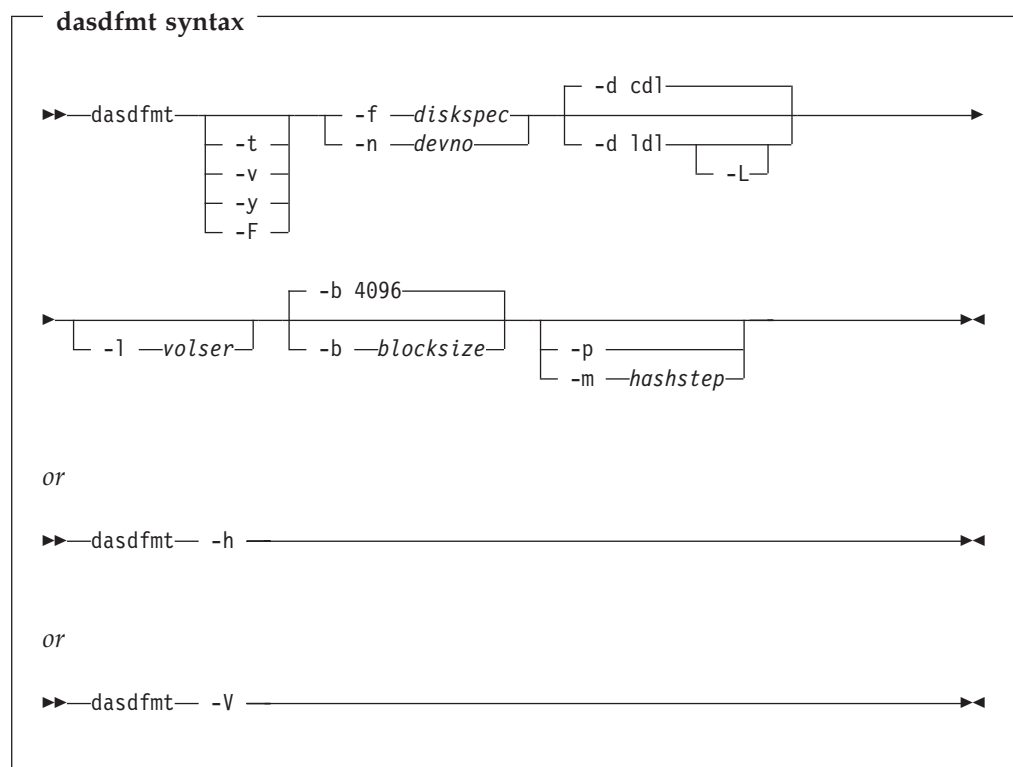
Usage

Before you start:

(see “DASD partitioning” on page 14 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



The parameters are:

-f diskspec or **--device=diskspec**

Specifies the device to be formatted. *diskspec* takes one of the following forms:

- /dev/dasd/xxxx/device, where *xxxx* is the four-character hexadecimal device address of the DASD.
- /dev/labels/volser
- /dev/dasdx

The first two options apply if the device file system has been enabled.

-n devno or --devno devno

Specifies the four-character hexadecimal device address of the disk to format, for example -n 01a3. The -n option can only be used in conjunction with devfs.

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 (possible values are therefore 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 the compatible disk layout (cdl) or the 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 volser or --label=volser

Specifies the volume serial number (see 3 on page 13) to be written to the disk. If the VOLSER contains special characters, it must be enclosed in single quotes. In addition, any '\$' character in the VOLSER must be preceded by a backslash ('\').

-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.

Escaping from the accepted state

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 with device node /dev/dasdc:

```

bash-2.04# dasdfmt -b 4096 -d ldl -p -f /dev/dasdc
Drive Geometry: 100 Cylinders * 15 Heads = 1500 Tracks

I am going to format the device /dev/dasdc in the following way:
Device number of device : 0x192
Labelling device       : yes
Disk label             : LNX1
Disk identifier        : 0X0192
Extent start (trk no)  : 0
Extent end (trk no)    : 1499
Compatible Disk Layout : no
Blocksize              : 4096

---->> ATTENTION! <----
All data of that device will be lost.
Type "yes" to continue, no will leave the disk untouched: yes
Formatting the device. This may take a while (get yourself a coffee).

cyl   100 of   100 |#####| 100%

Finished formatting the device.
Rereading the partition table... ok
bash-2.04#

```

Example 2

To format the same disk with the compatible disk layout (using the default value of the `-d` option).

```

bash-2.04# dasdfmt -b 4096 -p -f /dev/dasdc
Drive Geometry: 100 Cylinders * 15 Heads = 1500 Tracks

I am going to format the device /dev/dasdc in the following way:
Device number of device : 0x192
Labelling device       : yes
Disk label             : VOL1
Disk identifier        : 0X0192
Extent start (trk no)  : 0
Extent end (trk no)    : 1499
Compatible Disk Layout : yes
Blocksize              : 4096

---->> ATTENTION! <----
All data of that device will be lost.
Type "yes" to continue, no will leave the disk untouched: yes
Formatting the device. This may take a while (get yourself a coffee).

cyl   100 of   100 |#####| 100%

Finished formatting the device.
Rereading the partition table... ok
bash-2.04#

```

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

Before you start:

(See “DASD partitioning” on page 14 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
    [-0] [-1] [-2] [-n devno] [-f node]
    [-b begin] [-s size]
    [-i]
    [-x]
    [-j]
    [-l]
    [-t spec]
  
```

The parameters are:

-h or **--help** or **-?**

Display short usage text on console. To view the man page, 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.

-j Print volume serial number (volume identifier).

-l or --label

Display the volume label.

-t spec or --vtoc=spec

Display the VTOC's table-of-contents, or a single VTOC entry, on the console. The variable *spec* can take these values:

- info** Display overview information about the VTOC, such as a list of the 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 with VOLSER 'DASD01' that has a device node */dev/dasda* or */dev/dasd/0193/device*. You could then issue one of the following commands:

```
dasdview -i -f /dev/dasda
dasdview -i -f /dev/dasd/0193/device
dasdview -i -f /dev/labels/DASD01
```

The first form must be used if devfs is not enabled.

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
chang_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 displays:

[illegible]

To display partition information:

```
bash-2.04# dasdview -t info -n 193
```

This displays:

```
--- VTOC info -----
The VTOC contains:
 3 format 1 label(s)
 1 format 4 label(s)
 1 format 5 label(s)
 0 format 7 label(s)
Other S/390 and zSeries operating systems would see the following data sets:
```

data set	start	end
LINUX.V0X0193.PART0001.NATIVE	trk	trk
data set serial number : '0X0193'	2	500
system code : 'IBM LINUX '	cyl/trk	cyl/trk
creation date : year 2001, day 317	0/ 2	33/ 5
LINUX.V0X0193.PART0002.NATIVE	trk	trk
data set serial number : '0X0193'	501	900
system code : 'IBM LINUX '	cyl/trk	cyl/trk
creation date : year 2001, day 317	33/ 6	60/ 0
LINUX.V0X0193.PART0003.NATIVE	trk	trk
data set serial number : '0X0193'	901	1499
system code : 'IBM LINUX '	cyl/trk	cyl/trk
creation date : year 2001, day 317	60/ 1	99/ 14

```
bash-2.04#
```

To display VTOC information:

```
bash-2.04# dasdview -t f4 -n 193
```

This displays:

```
--- VTOC format 4 label -----
DS4KEYCD   : 04040404040404040404040404040404040404040404040404040404...
DS4IDFMT   : dec 244, hex f4
DS4HPCHR   : 0000000105 (cyl 0, trk 1, blk 5)
DS4DSREC   : dec 7, hex 0007
DS4HCCHH   : 00000000 (cyl 0, trk 0)
DS4NOATK   : dec 0, hex 0000
DS4VTOCI   : dec 0, hex 00
DS4NOEXT   : dec 1, hex 01
DS4SMSFG   : dec 0, hex 00
DS4DEVAC   : dec 0, hex 00
DS4DSCYL   : dec 100, hex 0064
DS4DSTRK   : dec 15, hex 000f
DS4DEVTK   : dec 58786, hex e5a2
DS4DEVI    : dec 0, hex 00
DS4DEVL    : dec 0, hex 00
DS4DEVK    : dec 0, hex 00
DS4DEVFG   : dec 48, hex 30
DS4DEVTL   : dec 0, hex 0000
DS4DEVDT   : dec 12, hex 0c
DS4DEVDB   : dec 0, hex 00
DS4AMTIM   : hex 0000000000000000
DS4AMCAT   : hex 000000
DS4R2TIM   : hex 0000000000000000
res1       : hex 0000000000
DS4F6PTR   : hex 0000000000
DS4VTOCE   : hex 01000000000100000001
              typeind   : dec 1, hex 01
              seqno     : dec 0, hex 00
              llimit    : hex 00000001 (cyl 0, trk 1)
              ulimit    : hex 00000001 (cyl 0, trk 1)
res2       : hex 00000000000000000000
DS4EFLVL   : dec 0, hex 00
DS4EFPTR   : hex 0000000000 (cyl 0, trk 0, blk 0)
res3       : hex 00000000000000000000
bash-2.04#
```

dasdview

To print the contents of a disk to the console starting at block 2 (volume label):

```
bash-2.04# dasdview -b 2b -s 128 -n 193
```

This displays:

```
+-----+-----+-----+
| HEXADECIMAL | EBCDIC | ASCII |
| 01....04 05....08 09....12 13....16 | 1.....16 | 1.....16 |
+-----+-----+-----+
| E5D6D3F1 E5D6D3F1 F0E7F0F1 F9F34000 | VOL1VOL10X0193?. | ???????????????. |
| 00000101 40404040 40404040 40404040 | ..... | ..... |
| 40404040 40404040 40404040 40404040 | ???????????????? | @@@@@@@@@@@@@@@@@@ |
| 40404040 40404040 40404040 40404040 | ???????????????? | @@@@@@@@@@@@@@@@@@ |
| 40404040 40404040 40404040 40404040 | ???????????????? | @@@@@@@@@@@@@@@@@@ |
| 40404040 88001000 10000000 00808000 | ???h..... | @@@@?..... |
| 00000000 00000000 00010000 00000200 | ..... | ..... |
| 21000500 00000000 00000000 00000000 | ?. | !..... |
+-----+-----+-----+
bash-2.04#
```

To display the contents 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 displays:

```
+-----+-----+-----+-----+-----+
| BYTE | BYTE | HEXADECIMAL | EBCDIC | ASCII |
| DECIMAL | HEXADECIMAL | 1 2 3 4 5 6 7 8 | 12345678 | 12345678 |
+-----+-----+-----+-----+-----+
| 57344 | E000 | D3C9D5E4 | E74BE5F0 | LINUX.V0 | ?????K?? |
| 57352 | E008 | E7F0F1F9 | F34BD7C1 | X0193.PA | ?????K?? |
| 57360 | E010 | D9E3F0F0 | F0F14BD5 | RT0001.N | ?????K?? |
| 57368 | E018 | C1E3C9E5 | C5404040 | ATIVE??? | ?????@?? |
| 57376 | E020 | 40404040 | 40404040 | ???????? | @@@@@@@@@@ |
| 57384 | E028 | 40404040 | F1F0E7F0 | ???10X0 | @@@@???? |
| 57392 | E030 | F1F9F300 | 0165013D | 193.??? | ????.?e?= |
| 57400 | E038 | 63016D01 | 0000C9C2 | ??_?..IB | c?m?...?? |
| 57408 | E040 | D440D3C9 | D5E4E740 | M?LINUX? | ?@?????@ |
| 57416 | E048 | 40404065 | 013D0000 | ??????.. | @@@e?=. |
| 57424 | E050 | 00000000 | 88001000 | ...h.? | ...?.?. |
| 57432 | E058 | 10000000 | 00808000 | ?...??. | ?...??. |
| 57440 | E060 | 00000000 | 00000000 | ..... | ..... |
| 57448 | E068 | 00010000 | 00000200 | .?....? | .?....? |
| 57456 | E070 | 21000500 | 00000000 | ?.?.... | !.?. |
| 57464 | E078 | 00000000 | 00000000 | ..... | ..... |
+-----+-----+-----+-----+-----+
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 displays:

```
+-----+-----+-----+
| HEXADEDECIMAL | 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 displays:

```
+-----+-----+-----+
| HEXADEDECIMAL | 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 compatible disk layout ('cdl') 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 serial number.

Note: To partition a SCSI disk, use **fdisk** rather than **fdasd**.

Usage

Before you start:

(see "DASD partitioning" on page 14 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: Careless use of `fdasd` can result in loss of data.

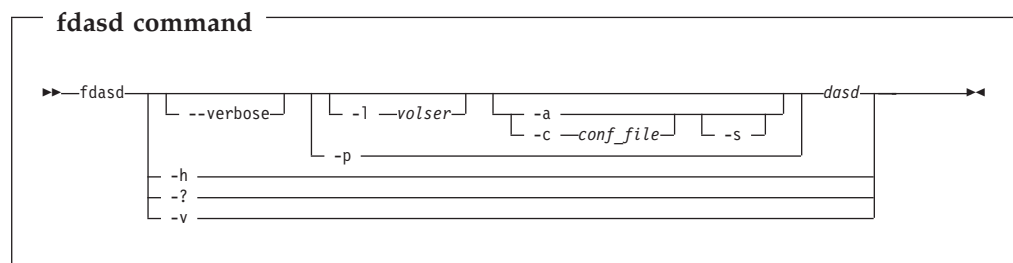
`fdasd` is a menu-driven tool that you call with the command `fdasd` followed by the device node of the DASD you want to partition.

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. In interactive mode, 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, sometimes a 'format 7' DSCB depending on the DASD size, and one to three 'format 1' DSCBs.

Format

Command line syntax



Where:

--verbose

Prints additional messages that are normally suppressed.

-l volser

Is the volume serial number (see 3 on page 13). If the VOLSER contains

special characters, it must be enclosed in single quotes. In addition, any '\$' character in the VOLSER must be preceded by a backslash ('\').

-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 sample 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).

-p or --table

Prints the partition table and exits.

dasd Is the device node of the DASD you want to partition. If the device file system is enabled, this has the form /dev/dasd/xxxx/device, where xxxx is the device number (devno) of the DASD, or /dev/labels/volser. If the device file system is not enabled, it has the form /dev/dasdxxx where xxx is one to three letters. See "DASD overview" on page 9 for more information.

-h or -? or --help

Displays help on command line arguments.

-v or --version

Displays the version of fdasd.

Setting up osasnmpd

fdasd menu

When you have called fdasd using the syntax described in "Command line syntax" on page 182, 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 182 for the format.
- t** Changes the partition type. You will be asked to identify the partition to be changed. You will then be asked for the new partition type (Linux native or swap). Note that this type is a guideline; the actual use Linux makes of the partition depends on how it is defined with the mkswap or mkxxfs tools. The main function of the partition type is to describe the partition to other operating systems so that, for example, swap partitions can be skipped by backup programs.
- r** 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
```

	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

osasnmppd – Start OSA-Express SNMP subagent

Purpose

The **osasnmppd** command is used to start the OSA-Express Simple Network Management Protocol (SNMP) subagent (osasnmppd). When you start osasnmppd, a daemon called osasnmppd-2.4 starts.

The subagent supports management information bases (MIBs) provided by the following OSA features in QDIO mode only:

- OSA-Express:
 - Fast Ethernet
 - 1000Base-T Ethernet (z890 and z990)
 - Gigabit Ethernet
 - Token Ring
 - ATM (running Ethernet LAN emulation)
- OSA-Express2 (z890 and z990):
 - Gigabit Ethernet
 - 10 Gigabit Ethernet

This subagent capability through the OSA-Express features is also called *Direct SNMP* to distinguish it from another method of accessing OSA SNMP data through OSA/SF, a package for monitoring and managing OSA features that does not run on Linux.

To use the osasnmppd subagent you need:

- OSA-Express feature running in QDIO mode with the latest OSA-Express LIC level (for the latest MIB structure)
- qeth device driver for OSA-Express (QDIO) and HiperSockets
- osasnmppd subagent from s390-tools
- One of:
 - net-snmp package 5.1.x or higher
 - ucd-snmp package 4.2.x (recommended 4.2.3 or higher)

What you need to know about osasnmppd

The osasnmppd subagent requires a master agent to be installed on a Linux system. You get the master agent from either the net-snmp or the ucd-snmp package. The subagent uses the Agent eXtensibility (AgentX) protocol to communicate with the master agent.

net-snmp/ucd-snmp is an Open Source project that is owned by the Open Source Development Network, Inc. (OSDN). For more information on net-snmp/ucd-snmp visit:

<http://net-snmp.sourceforge.net/>

When the master agent (snmpd) is started on a Linux system, it loads the available subagents, binds to a port (default 161) and awaits requests from SNMP management software. When the osasnmppd subagent is started, it retrieves the MIB objects of the OSA-Express features currently present on the Linux system. It then registers with the master agent the object IDs (OIDs) for which it can provide information.

An OID is a unique sequence of dot-separated numbers (for example, .1.3.6.1.4.1.2) that represents a particular information. OIDs form a hierarchical structure. The longer the OID, that is the more numbers it is made up of, the more specific is the information that is represented by the OID. For example, .1.3.6.1.4.1.2 represents all IBM-related network information while ..1.3.6.1.4.1.2.6.188 represents all OSA-Express-related information.

A MIB corresponds to a number of OIDs. MIBs provide information on their OIDs including textual representations the OIDs. For example, the textual representation of .1.3.6.1.4.1.2 is .iso.org.dod.internet.private.enterprises.ibm.

The structure of the MIBs might change when updating the OSA-Express licensed internal code (LIC) to a newer level. If new MIBs are introduced by a new LIC level, you need to download them (see “Downloading the IBM OSA-Express MIB” on page 191), but you do not need to update the subagent.

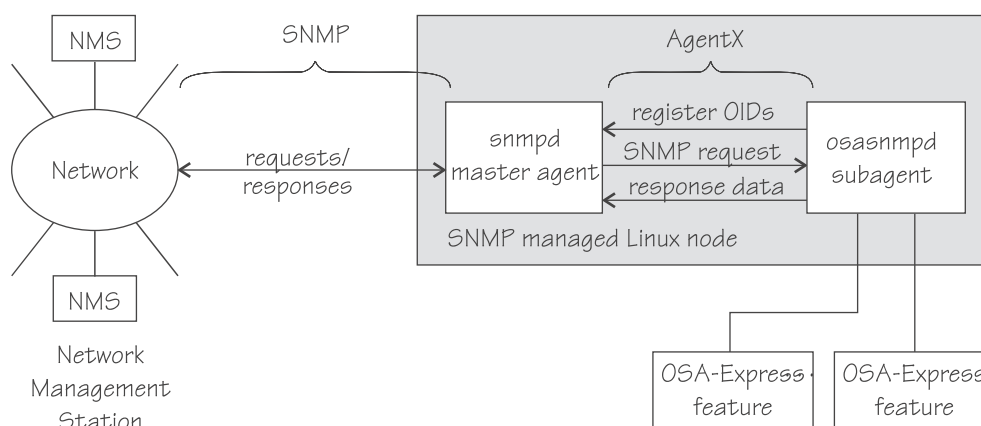


Figure 15. OSA-Express SNMP agent flow

Example: This example shows the processes running after the snmpd master agent and the osasnmppd subagent have been started. When you start osasnmppd, a daemon called osasnmppd-2.4 starts. In the example, PID 687 is the SNMP master agent and PID 729 is the OSA-Express SNMP subagent process:

```
ps -ef | grep snmp
```

USER	PID						
root	687	1	0	11:57	pts/1	00:00:00	snmpd
root	729	659	0	13:22	pts/1	00:00:00	osasnmppd-2.4

When the master agent receives an SNMP request for an OID that has been registered by a subagent, the master agent uses the subagent to collect any requested information and to perform any requested operations. The subagent returns any requested information to the master agent. Finally, the master agent returns the information to the originator of the request.

Setting up osasnmppd

This section describes the following setup tasks you need to perform if you want to use the osasnmppd subagent:

- Downloading the IBM OSA-Express MIB
- Configuring access control

Downloading the IBM OSA-Express MIB

Perform the following steps to download the IBM OSA-Express MIB. The MIB file is valid only for hardware that supports the OSA-Express adapter card.

1. Go to ibm.com/servers/resource/links/
A user ID and password are required. You can apply for a user ID if you do not yet have one.
2. Sign in.
3. Select "Library" from the left-hand navigation area.
4. Under "Library shortcuts", select "Open Systems Adapter (OSA) Library".
5. Follow the link for "OSA-Express Direct SNMP MIB module".
6. Select and download the MIB for your LIC level.
7. Rename the MIB file to the name specified in the MIBs definition line and use the extension .txt.

Example: If the definition line in the MIB looks like this:

```
==>IBM-OSA-MIB DEFINITIONS ::= BEGIN
```

Rename the MIB to IBM-OSA-MIB.txt.

8. Place the MIB into /usr/share/snmp/mibs.

If you want to use a different directory, be sure to specify the directory in the snmp.conf configuration file (see step 10 on page 193).

Result: You can now make the OID information from the MIB file available to the master agent. This allows you to use textual OIDs instead of numeric OIDs when using master agent commands.

See also the FAQ (How do I add a MIB to the tools?) for the master agent package at

<http://net-snmp.sourceforge.net/FAQ.html>

Configuring access control

During subagent startup or when network interfaces are added or removed, the subagent has to query OIDs from the interfaces group of the standard MIB-II. To start successfully, the subagent requires at least read access to the standard MIB-II on the local node.

This section gives an example of how you can use the snmpd.conf and snmp.conf configuration files to assign access rights using the View-Based Access Control Mechanism (VACM). The following access rights are assigned on the local node:

- General read access for the scope of the standard MIB-II
- Write access for the scope of the OSA-Express MIB
- Public local read access for the scope of the interfaces MIB

The example is intended for illustration purposes only. Depending on the security requirements of your installation, you might need to define your access differently. Refer to the snmpd man page for a more information on how you can assign access rights to snmpd.

1. Refer to your distribution documentation to find out where you can find a template for snmpd.conf and where you need to place it. Some of the possible locations are:
 - /usr/local/share/snmp
 - /etc/snmp
 - /usr/share/snmp

2. Open snmpd.conf with your preferred text editor.
3. Find the security name section and include a line of this form to map a community name to a security name:

```
com2sec <security-name> <source> <community-name>
```

where:

<security-name>

is given access rights through further specifications within snmpd.conf.

<source>

is the IP-address or DNS-name of the accessing system, typically a Network Management Station.

<community-name>

is the community string used for basic SNMP password protection.

Example:

```
#      sec.name      source      community
com2sec osasec      default    osacom
com2sec pubsec      localhost  public
```

4. Find the group section. Use the security name to define a group with different versions of the master agent for which you want to grant access rights. Include a line of this form for each master agent version:

```
group <group-name> <security-model> <security-name>
```

where:

<group-name>

is a group name of your choice.

<security-model>

is the security model of the SNMP version.

<security-name>

is the same as in step 3.

Example:

```
#      groupName      securityModel  securityName
group  osagroup       v1           osasec
group  osagroup       v2c          osasec
group  osagroup       usm          osasec
group  osasnmpd       v2c          pubsec
```

Group “osasnmpd” with community “public” is required by osasnmpd to determine the number of network interfaces.

5. Find the view section and define your views. A view is a subset of all OIDs. Include lines of this form:

```
view <view-name> <included|excluded> <scope>
```

where:

<view-name>

is a view name of your choice.

<included|excluded>

indicates whether the following scope in an inclusion or an exclusion statement.

<scope>

specifies a subtree in the OID tree.

Example:

```
#      name      incl/excl  subtree      mask(optional)
view allview    included    .1
view osaview    included    .1.3.6.1.4.1.2
view ifmibview  included    interfaces
```

View “allview” encompasses all OIDs while “osaview” is limited to IBM OIDs. The numeric OID provided for the subtree is equivalent to the textual OID “.iso.org.dod.internet.private.enterprises.ibm”. View “ifmibview” is required by osasnmpd to determine the number of network interfaces.

Tip: Specifying the subtree with a numeric OID leads to better performance than using the corresponding textual OID.

6. Find the access section and define access rights. Include lines of this form:

```
access <group-name> "" any noauth exact <read-view> <write-view> none
```

where:

<group-name>

is the group you defined in step 4 on page 192.

<read-view>

is a view for which you want to assign read-only rights.

<write-view>

is a view for which you want to assign read-write rights.

Example:

```
#      group      context sec.model sec.level prefix read  write  notif
access osagroup  ""      any      noauth  exact  allview osaview none
access osasnmpd ""      v2c      noauth  exact  ifmibview none  none
```

The access line of the example gives read access to the “allview” view and write access to the “osaview”. The second access line gives read access to the “ifmibview”.

7. Also include the following line to enable the AgentX support:

```
master agentx
```

By default, AgentX support is compiled into the net-snmp master agent 5.1.x and, as of version 4.2.2, also into the ucd-snmp master agent.

8. Save and close snmpd.conf.
9. Open snmp.conf with your preferred text editor.
10. Include a line of this form to specify the directory to be searched for MIBs:

```
mibdirs +<mib-path>
```

Example:

```
mibdirs +/usr/share/snmp/mibs
```

11. Include a line of this form to make the OSA-Express MIB available to the master agent:

```
mibs +<mib-name>
```

where <mib-name> is the stem of the MIB file name you assigned in “Downloading the IBM OSA-Express MIB” on page 191.

Example:

```
mibs +IBM-OSA-MIB
```

12. Define defaults for the version and community to be used by the snmp commands. Add lines of this form:

```
defVersion <version>
defCommunity <community-name>
```

where *<version>* is the SNMP version and *<community-name>* is the community you defined in step 3 on page 192.

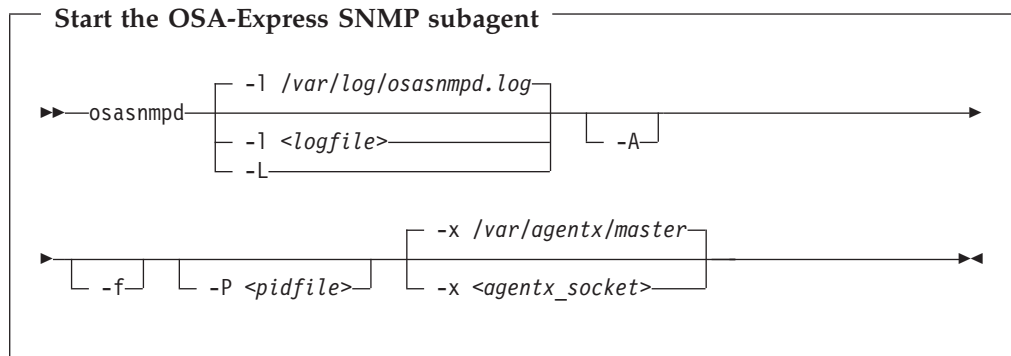
Example:

```
defVersion 2c
defCommunity osacom
```

These default specifications simplify issuing master agent commands.

13. Save and close snmp.conf.

Format - starting osasnmpd



-l <logfile>

specifies a file for logging all subagent messages and warnings, including stdout and stderr. If no path is specified, the log file is created in the current directory. The default log file is /var/log/osasnmpd.log.

-L print messages and warnings to stdout/stderr.

-A appends to an existing log file rather than replacing it.

-f prevents forking from the calling shell.

-P <pidfile>

saves the process ID of the subagent in a file *<pidfile>*. If a path is not specified, the current directory is used.

-x <agentx_socket>

specifies the socket to be used for the AgentX connection. The default socket is /var/agentx/master.

The socket can either be a UNIX[®] domain socket path, or the address of a network interface. If a network address of the form *inet-addr:port* is specified, the subagent uses the specified port. If a net address of the form *inet-addr* is specified, the subagent uses the default AgentX port, 705.

-h or --help

displays help information for the command.

-v or --version

displays version information for the command.

Examples

To start the osasnmpd subagent with all default settings:

```
# osasnmpd
```

Working with the osasnmppd support

This section describes the following tasks:

- Checking the log file
- Issuing queries
- Stopping osasnmppd

Checking the log file

Warnings and messages are written to the log file of either the master agent or the OSA-Express subagent. It is good practise to check these files at regular intervals.

Example: This example assumes that the default subagent log file is used. The lines in the log file show the messages after a successful OSA-Express subagent initialization.

```
# cat /var/log/osasnmppd.log
IBM OSA-E NET-SNMP 5.1.x subagent version 1.3.0
Jul 14 09:28:41 registered Toplevel OID .1.3.6.1.2.1.10.7.2.
Jul 14 09:28:41 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.1.
Jul 14 09:28:41 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.3.
Jul 14 09:28:41 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.4.
Jul 14 09:28:41 registered Toplevel OID .1.3.6.1.4.1.2.6.188.1.8.
OSA-E microcode level is 611 for interface eth0
Initialization of OSA-E subagent successful...
```

Issuing queries

This section provides some examples of what SNMP queries might look like. For more comprehensive information on the master agent commands refer to the snmpcmd man page.

The commands can use either numeric or textual OIDs. While the numeric OIDs might provide better performance, the textual OIDs are more meaningful and give a hint on which information is requested.

The query examples in this section gather information on an interface, eth0, for which the procs information looks like this:

```
# cat /proc/qeth
devices          CHPID interface cardtype      port checksum prio-q'ing rtr4 rtr6 fsz  cnt
-----
0.0.f200/0.0.f201/0.0.f202 x6B  eth0      OSD_100    0   sw      always_q_0 no   no   64k  16
0.0.fa04/0.0.fa05/0.0.fa03 x7A  eth1      OSD_1000   0   no      always_q_0 no   no   64k  128
0.0.f600/0.0.f601/0.0.f602 x7D  eth2      OSD_1000   0   no      always_q_0 no   no   64k  128
0.0.8200/0.0.8201/0.0.8202 xFD  hsi0      HiperSockets 0   no      always_q_0 no   no   40k  128
```

The CHPID for the eth0 of our example is 0x6B.

- To find the CHPID numbers for your OSA devices :

```
# snmpwalk -OS -v 2c -c osacom localhost .1.3.6.1.4.1.2.6.188.1.1.1.1
IBM-OSA-MIB::ibmOSAExpChannelNumber.6 = Hex-STRING: 00 6B
IBM-OSA-MIB::ibmOSAExpChannelNumber.7 = Hex-STRING: 00 7A
IBM-OSA-MIB::ibmOSAExpChannelNumber.8 = Hex-STRING: 00 7D
```

The first line, with index number 6, corresponds to CHPID 0x6B of our eth0 example. The example assumes that the community osacom has been authorized as described in “Configuring access control” on page 191.

If you have provided defaults for the SNMP version and the community (see step 12 on page 193), you can omit the -v and -c options:

```
# snmpwalk -OS localhost .1.3.6.1.4.1.2.6.188.1.1.1.1
IBM-OSA-MIB::ibmOSAExpChannelNumber.6 = Hex-STRING: 00 6B
IBM-OSA-MIB::ibmOSAExpChannelNumber.7 = Hex-STRING: 00 7A
IBM-OSA-MIB::ibmOSAExpChannelNumber.8 = Hex-STRING: 00 7D
```

You can obtain the same output by substituting the numeric .1.3.6.1.4.1.2.6.188.1.1.1 with its textual equivalent:

.iso.org.dod.internet.private.enterprises.ibm.ibmProd.ibmOSAMib.ibmOSAMibObjects.ibmOSAExpChannelTable.ibmOSAExpChannelEntry.ibmOSAExpChannelNumber

You can shorten this somewhat unwieldy OID to the last element, `ibmOsaExpChannelNumber`:

```
# snmpwalk -OS localhost ibmOsaExpChannelNumber
IBM-OSA-MIB::ibmOSAExpChannelNumber.6 = Hex-STRING: 00 6B
IBM-OSA-MIB::ibmOSAExpChannelNumber.7 = Hex-STRING: 00 7A
IBM-OSA-MIB::ibmOSAExpChannelNumber.8 = Hex-STRING: 00 7D
```

- To find the port type for the interface with index number 6:

```
# snmpwalk -OS localhost .1.3.6.1.4.1.2.6.188.1.4.1.2.6
IBM-OSA-MIB::ibmOsaExpEthPortType.6 = INTEGER: fastEthernet(81)
```

`fastEthernet(81)` corresponds to card type `OSD_100`.

Using the short form of the textual OID:

```
# snmpwalk -OS localhost ibmOsaExpEthPortType.6
IBM-OSA-MIB::ibmOsaExpEthPortType.6 = INTEGER: fastEthernet(81)
```

specifying the index, 6 in the example, limits the output to the interface of interest.

Stopping osasnmpd

If you restart the master agent, you must also restart the subagent. When the master agent is started, it does not look for already running subagents. Any running subagents must also be restarted to be register with the master agent.

The subagent can be stopped by sending either a `SIGINT` or `SIGTERM` signal to the thread. Avoid stopping the subagent with `kill -9` or with `kill -SIGKILL`. These commands do not allow the subagent to unregister the OSA-Express MIB objects from the SNMP master agent. This can cause problems when restarting the subagent.

If you have saved the subagent PID to a file when you started it, you can consult this file for the PID (see 194). Otherwise you can issue a `ps` command to find it out.

Example:The `osasnmpd` subagent starts a daemon called `osasnmpd-2.4`. To stop `osasnmpd`, issue the `kill` command for either the daemon or its PID.

```
# ps -ef | grep snmp

USER      PID      1  0 11:57 pts/1    00:00:00 snmpd
root      729    659  0 13:22 pts/1    00:00:00 osasnmpd-2.4
# killall osasnmpd-2.4
# kill 729
```

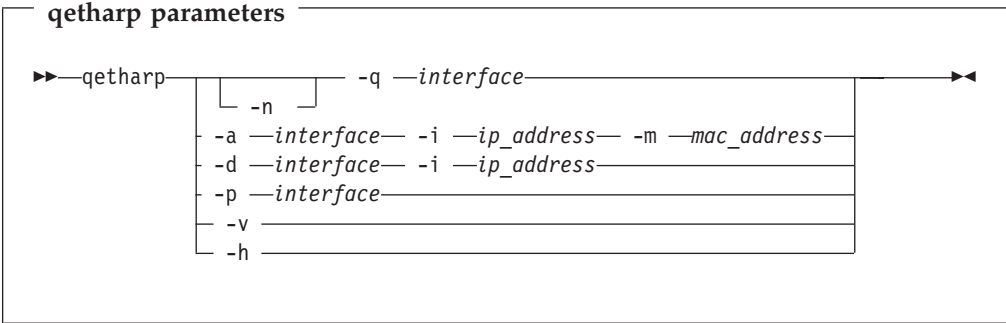
qetharp - Query and purge OSA and HiperSockets ARP data

Purpose

Note: The information in this section does not apply if you have set the layer2 option (see “layer2 | no_layer2” on page 146).

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



The meanings of the parameters of this command are as follows:

-q or --query
Shows the address resolution protocol (ARP) information found in the ARP cache of the OSA or HiperSockets, which depends on *interface*. If it is an OSA device, it shows the ARP entries stored in the OSA feature’s ARP cache, otherwise, the ones from the HiperSockets ARP cache. If the IP address is an IPv4 address, qetharp tries to determine the symbolic host name. If it fails, the IP address will be shown. In case of IPv6, there is currently no attempt to determine host names, so that the IP address will be shown directly.

-n or --numeric
Shows numeric addresses instead of trying to determine symbolic host names. This option can only be used in conjunction with the -q option.

interface
The qeth interface to which the command applies.

-a or --add
Adds a static ARP entry to the OSA adapter card.

ip_address
IP address to be added to the OSA adapter card.

-d or --delete
Deletes a static ARP entry from the OSA adapter card.

mac_address
MAC address to be added to the OSA adapter card.

-p or --purge

Flushes the ARP cache of the OSA, causing the hardware to regenerate the addresses. This option works only with OSA devices. qetharp returns immediately.

-v or --verbose

Shows version information and exits

-h or --help

Shows usage information and exits

Examples

- Show all ARP entries of the OSA defined as eth0:

```
qetharp -q eth0
```

- Show all ARP entries of the OSA defined as eth0, without resolving host names:

```
qetharp -nq eth0
```

- Flush the OSA's ARP cache for eth0:

```
qetharp -p eth0
```

- Add a static entry for eth0 and IP address 1.2.3.4 to the OSA's ARP cache, using MAC address aa:bb:cc:dd:ee:ff:

```
qetharp -a eth0 -i 1.2.3.4 -m aa:bb:cc:dd:ee:ff
```

- Delete the static entry for eth0 and IP address 1.2.3.4 from the OSA's ARP cache, using MAC address aa:bb:cc:dd:ee:ff:

```
qetharp -d eth0 -i 1.2.3.4
```


qethconf - configure qeth devices

Purpose

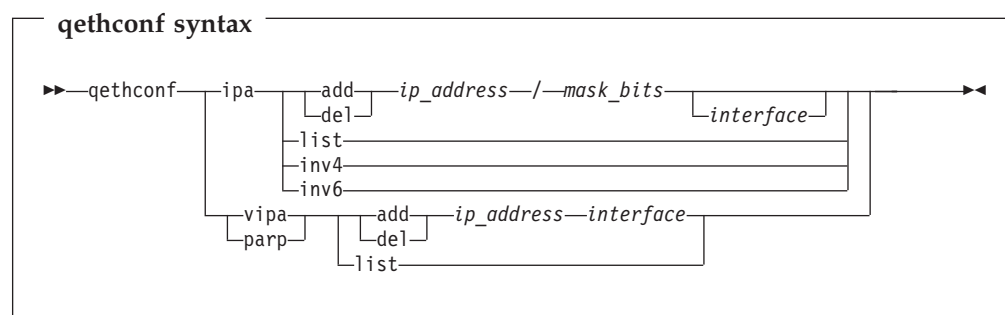
Note: The information in this section does not apply if you have set the `layer2` option (see “`layer2 | no_layer2`” on page 146).

The `qethconf` configuration tool is a bash shell script that simplifies configuring qeth devices (see Chapter 15, “QETH device driver for OSA-Express (QDIO) and HiperSockets,” on page 139) for:

- IP address takeover
- VIPA (virtual IP address)
- Proxy ARP

From the arguments that are specified, `qethconf` assembles the corresponding function command and redirects it to the respective qeth proc file, `/proc/qeth/ipa_takeover`. You can also use `qethconf` to list the already defined entries per function.

Format



The meanings of the parameters of this command are as follows:

ipa

Configure qeth for IP address takeover (IPA).

vipa

Configure qeth for virtual IP address (VIPA).

parp

Configure qeth for proxy ARP.

add

Add an IP address or address range.

del

Delete an IP address or address range.

inv4

Invert the selection of address ranges for IPv4 address takeover. This makes the list of IP addresses that has been specified with `qethconf add` and `qethconf del` an exclusion list.

inv6

Invert the selection of address ranges for IPv6 address takeover. This makes the list of IP addresses that has been specified with `qethconf add` and `qethconf del` an exclusion list.

list

List existing definitions for specified qeth function.

ip_address

IP address. Can be specified in one of these formats:

- IP version 4 format, for example, 192.168.10.38
- IP version 6 format, for example, FE80::1:800:23e7:f5db
- 8- or 32-character hexadecimals prefixed with -x, for example, -xc0a80a26

mask_bits

Number of bits that are set in the network mask. Allows you to specify an address range.

Example: A *mask_bits* of 24 corresponds to a network mask of 255.255.255.0.

interface

Interface associated with the specified address or address range.

Examples

- List existing proxy ARP definitions for eth0:

```
[root]# qethconf parp list
parp add 1.2.3.4 eth0
```

- Assume responsibility for packages destined for 1.2.3.5:

```
[root]# qethconf parp add 1.2.3.5 eth0
qethconf: parp add command successful.
```

Confirm the new proxy ARP definitions:

```
[root]# qethconf parp list
parp add 1.2.3.4 eth0
parp add 1.2.3.5 eth0
```

- Configure eth0 for IP address takeover for all addresses that start with 192.168.10:

```
[root]# qethconf ipa add 192.168.10.0/24 eth0
qethconf: ipa add command successful.
```

Display the new IP address takeover definitions:

```
[root]# qethconf ipa list
ipa add 192.168.10.0/24 eth0
```

- Configure VIPA for eth1:

```
[root]# qethconf vipa add 10.99.3.3 eth1
qethconf: vipa add command successful.
```

Display the new VIPA definitions:

```
[root]# qethconf vipa list
vipa add 10.99.3.3 eth1
```

snIPL – Simple network IPL (Linux image control for LPAR and VM)

Purpose

snIPL (simple **network IPL**) is a command line tool for *remotely controlling Linux images* using either

- basic zSeries and S/390 support element (SE) functions for systems running in **LPAR mode**, or
- basic z/VM system management functions for systems running as a **z/VM guest** (z/VM 4.4 or higher).

Note: Be aware that incautious use of snIPL can result in loss of data.

LPAR mode

In LPAR mode, snIPL allows you to:

- *Load* an LPAR.
- *Send* and *retrieve* operating system messages.
- *Activate, reset, or deactivate* an LPAR for I/O-fencing purposes.

Using snIPL in LPAR mode allows you to overcome the limitations of the SE graphical interface when snIPL is used for I/O-fencing from within a clustered environment of Linux systems that run in LPAR mode.

snIPL uses the network management *application programming interfaces* (API) provided by the SE, which establishes an SNMP network connection and uses the SNMP protocol to send and retrieve data. The API is called "hwmcaapi". It has to be available as shared library.

To establish a connection (using a valid community), the IP address of the initiating system and the community has to be configured in the *SNMP configuration* task of the SE. Also, SNMP support must be configured in the *SE settings* task. If snIPL in LPAR mode repeatedly reports a timeout, the target SE is most likely inaccessible or not configured properly. For details on how to configure the SE, refer to the *Application Programming Interfaces* book.

For further details, refer to *zSeries Application Programming Interfaces*, SB10-7030, or *S/390 Application Programming Interfaces*, SC28-8141, which is obtainable from the following Web site:

ibm.com/servers/resourceLink/

z/VM mode

In z/VM mode, snIPL allows you to remotely control basic z/VM system management functions. You can:

- *Activate, reset, or deactivate* an image for I/O-fencing purposes.

snIPL in z/VM mode uses the *system management application programming interfaces* (APIs) of z/VM (version 4.4 or higher). To communicate with the z/VM host, snIPL establishes a network connection and uses the RPC protocol to send and retrieve data.

To establish a connection to the VM host, the VSMSEVER server must be configured and the vmsapi service must be registered on the target VM host. Also, there has to be an account for the specified user ID on the host. If snIPL in VM mode repeatedly reports "RPC: Port mapper failure - RPC timed out", it is most

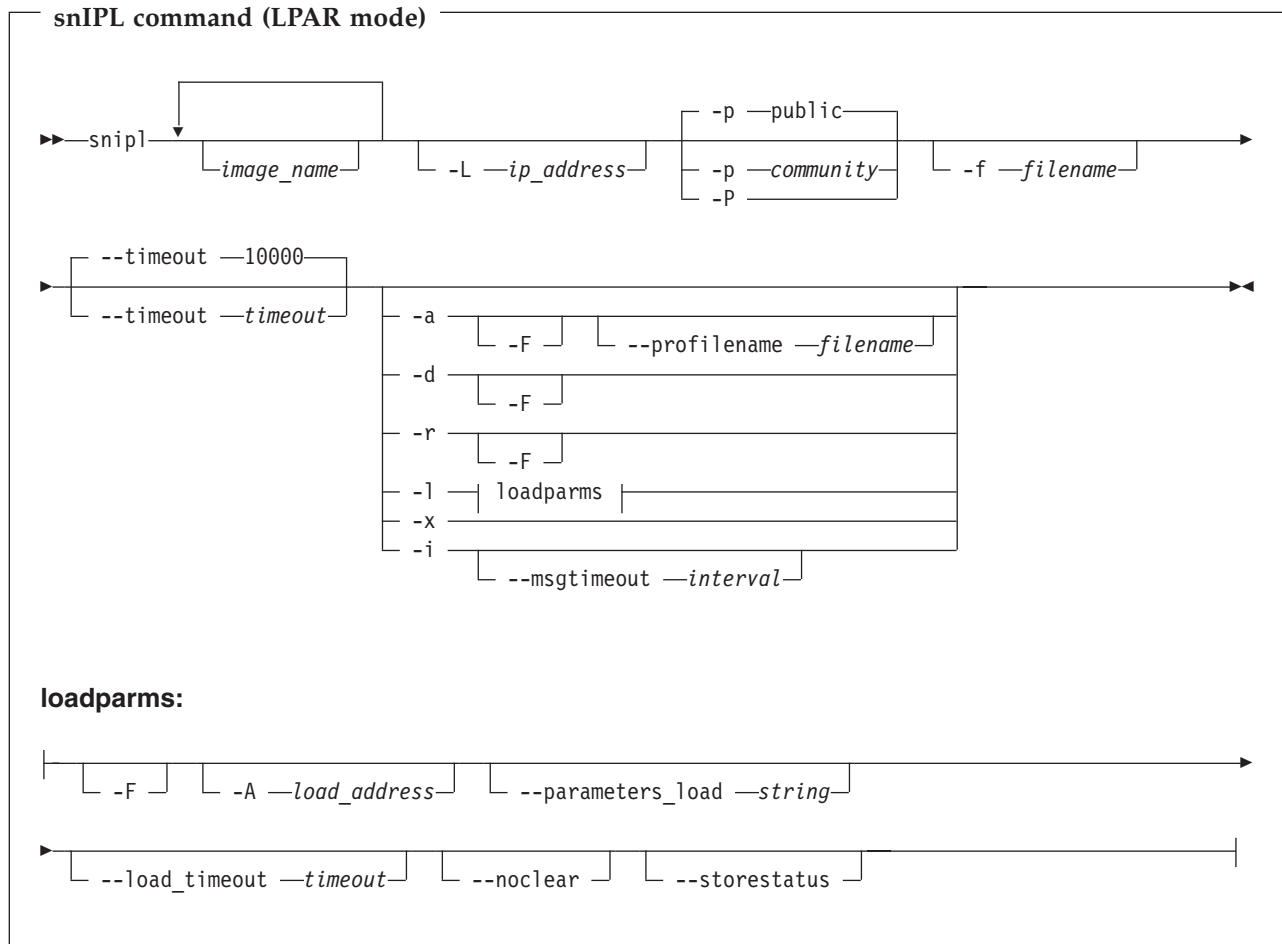
likely that the target z/VM host is inaccessible, or the service is not registered, or the configuration of the VSMSEVER server is not correct.

For details about configuration of the VSMSEVER server on z/VM refer to *z/VM: Systems Management Application Programming*, SC24-6063 obtainable from the following Web site:

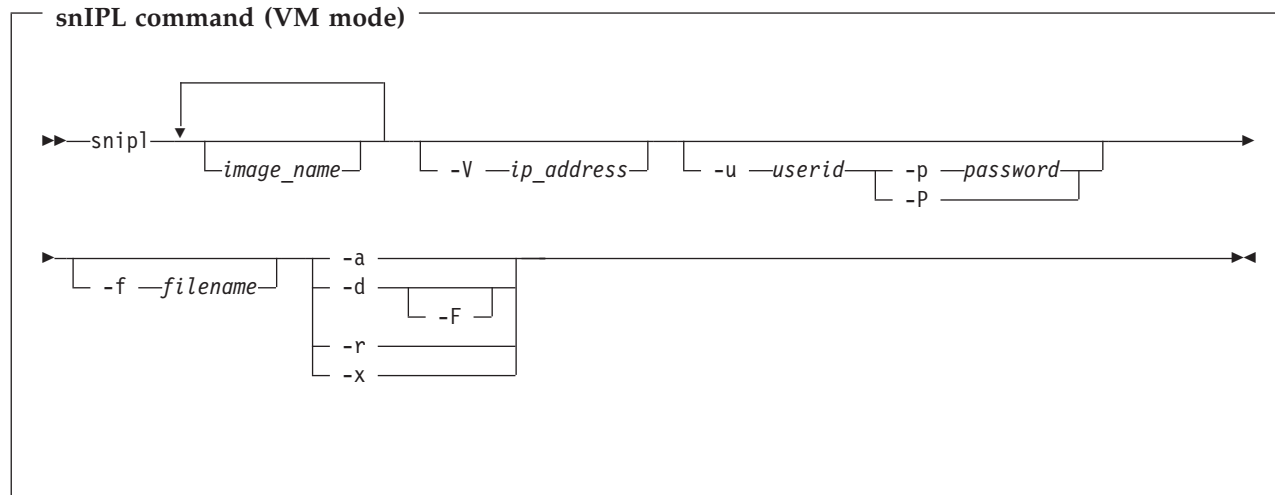
ibm.com/vm/

Usage

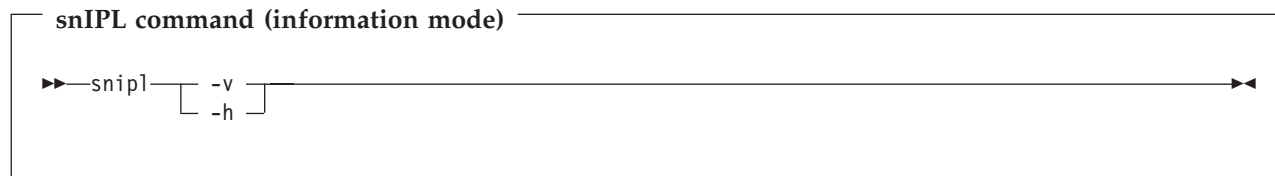
Command line syntax (LPAR mode)



Command line syntax (VM mode)



Command line syntax (information mode)



Options and Parameters

image_name

Specifies the name of the targeted LPAR or z/VM guest. This parameter is required for *--activate*, *--deactivate*, *--reset*, *--load*, and *--dialog*. If the same command is to be performed on more than one image of a given server, more than one **image_name** can be specified. Exception: A *--dialog* can only be started with one image.

-V, --vmserver *ip_address*

Specifies the server to be of type VM. Use this option if the system is running in VM mode. Also specifies the IP-address/host-name of targeted VM-host. This option can also be defined in the configuration file and thus may also be omitted.

-L, --lparserver *ip_address*

Specifies the server to be of type LPAR. Use this option if the system is running in LPAR mode. Specifies the IP-address/hostname of targeted SE. This option can also be defined in the configuration file and thus may also be omitted.

-u, --userid *userid*

VM only: Specifies the userid used to access the VM-host. If none is given, the configuration file can be used to determine the userid for a given IP-address or VM-guest-name.

-p, --password *community* | *password*

- For LPAR mode, the option specifies the *community* (HMC term for password) of the initiating host system. The default for *community* is

public. The value entered here must match the entry contained in the SNMP configuration settings on the SE.

- For VM mode, specifies the password for the given user ID.

If no password is given, the configuration file can be used to determine the password for a given IP address, LPAR, or VM guest name.

-P, --promptpassword

Lets snIPL prompt for a password in protected entry mode.

-f, --configfilename *filename*

Specifies the name of a configuration file containing HMC/SE IP-addresses together with their community (=password) and VM IP-address together with their userid and password followed by a list of controlled LPARnames or VM-guest-names. Default *user-specific filename* is `$HOME/.snipl.conf` and *default system-wide filename* is `/etc/snip1.conf`. Without available configuration file all required options have to be specified with the command. The structure of the configuration file is described below.

-x, --listimages

Lists all available images for the specified server.

- For VM this may be specified with *image*, *server*, *server+user* or *image+user* according to the uniqueness in the configuration file. In case of VM the returned list is retrieved from the configuration file only.
- For LPAR just the *server name* is used to retrieve the actual images. The information is directly retrieved from the SE.

-a, --activate

Issues an activate command for the targeted LPAR or VM guest.

-d, --deactivate

Issues a deactivate command for the target LPAR or VM guest.

-r, --reset

Issues a reset command for the targeted LPAR(s) or VM guest(s).

-l, --load

LPAR only: Issues a load command for the target LPAR.

-i, --dialog

LPAR only: This option starts an operating system message dialog with the targeted LPAR. It allows the user to enter arbitrary commands, which are sent to the targeted LPAR. In addition, dialog starts a background process, which continuously retrieves operating system messages. The output of this polling process is sent to stdout. The operating system messages dialog is aborted by pressing CTRL-D. This also kills the polling process. After the dialog is terminated, snIPL exists.

-t, --timeout *timeout*

LPAR only: Specifies the timeout in milliseconds for general management API calls. The default is 10000 ms.

-m, --msgtimeout *interval*

LPAR only: Specifies – in conjunction with `--dialog` – the interval in milliseconds for management API calls that retrieve operating system messages. The default value is set to 5000 ms.

-F, --force

Forces the imageoperation.

- VM: in conjunction with `--deactivate` non graceful deactivation of the image.

- LPAR: In conjunction with `--activate`, `--deactivate`, `--reset` and `--load` allows unconditional execution of the command regardless of the state of the image.

--profilename *filename*

LPAR only: In conjunction with `--activate` the option specifies the profile name used on the activate command for LPAR mode. If none is provided, the HMC/SE default profile name for the given image is used.

-A, --address_load *hexaddress*

LPAR only: In conjunction with `--load` specifies the load address in 4 hexadecimal digits. If none is provided, the address of the previous load is used as load address.

--parameters_load *parameter string*

LPAR only: In conjunction with `--load` specifies a parameter string for loading. If none is given, the parameter string of the previous load is used. This parameter is used for instance for IPL of z/OS and z/VM.

--noclear

LPAR only: In conjunction with `--load` denies memory clearing before loading. The memory is cleared by default.

--load_timeout *timeout*

LPAR only: In conjunction with `--load` specifies the maximum time for load completion, in seconds. The value must be between 60 and 600 seconds. The default value is 60 seconds.

--storestatus

LPAR only: In conjunction with `--load` requests status before loading. The status is not stored by default.

-v --version

Prints version of snIPL and exits.

-h --help

Prints usage and exits.

Structure of the configuration file

A configuration file contains a list of addresses (IP-addresses of an SE or a z/VM host), and the host type (LPAR vs. VM). The configuration file also contains a list of image names available for control on the subswitch.

- For LPAR, the list of image names can also be retrieved from the SE.
- For z/VM the list can only be retrieved by users with appropriate z/VM access rights. Therefore, a local list must be available.

The following is an example for the structure of the snIPL configuration file:

```
Server = <IP-address>
type = <host-type>
password = <password>
image = <imagename>
image = <imagename>
image = <imagename>
Server = <IP-address>
type = <host-type>
user = <username>
password = <password>
image = <imagename>
image = <imagename>
image = <imagename>
image = <imagename>
```

Blanks and n/ are separators. The keywords are not case-sensitive.

snIPL command examples

LPAR mode: Activate:

```
[root@mylinux root]# snipl LPARLNX1 -L 9.164.70.100 -a -P
Enter password: Warning : No default configuration file could be found/opened.
processing.....
LPARLNX1: acknowledged.
```

LPAR mode: Load: Load using configuration file:

```
[root@mylinux root]# snipl LPARLNX1 -f xcfg -l -A 5119
processing.....
LPARLNX1: acknowledged.
```

z/VM mode: Activate using configuration file:

```
[root@mylinux root]# snipl -f xcfg -a vmlnx2 vmlnx1
* ImageActivate : Image vmlnx1 Request Successful
* ImageActivate : Image vmlnx2 Image Already Active
```

Connection errors and exit codes

If a connection error occurs (e.g. *timeout*, or *communication failure*), snIPL sends an *error code* of the management API and a *message* to stderr. For

- `snipl --vmserver` the shell exit code is set to "1000 + error code"
- `snipl --lparserver` the shell exit code is set to "2000 + error code"

Return codes like

```
LPARLNX1: not acknowledged - command was not successful - rc is 135921664
```

are described in *Appendix B* of the HWMCAAPI document "zSeries Application Programming Interfaces" SB10-7030, You can obtain this publication from the following Web site: ibm.com/servers/resource/links/

Additionally, the following snIPL *error codes* exist. They are accompanied by a short message on stderr:

- | | |
|----|--|
| 1 | An unknown option is specified. |
| 2 | An option with an invalid value is specified. |
| 3 | An option is specified more than once. |
| 4 | Conflicting options are specified. |
| 5 | No command option is specified. |
| 6 | Server is not specified and cannot be determined. |
| 7 | No image is specified. |
| 8 | User-ID is not specified and cannot be determined. |
| 9 | Password is not specified and cannot be determined. |
| 10 | A specified image name does not exist on the server used. |
| 20 | An error occurred while processing the configuration file. |
| 22 | Operation --dialog: More than one image name is specified. |
| 30 | An error occurred while loading one of the libraries <i>libhwmcaapi.so</i> or <i>libvmsmapi.so</i> |

- 40 Operation --dialog encounters a problem while starting another process.
- 41 Operation --dialog encounters a problem with stdin attribute setting.
- 50 Response from HMC/SE is cannot be interpreted.
- 60 Response buffer is too small for HMC/SE response.
- 90 A storage allocation failure occurred.

If no error occurs, a shell exit code of 0 is returned upon completion of snipl.

Recovery

Currently, snIPL does not

- recover connection failures.
- recover errors in API call execution.

In these cases, it is sufficient to *restart* the tool. Should the problem persist, a networking failure is most likely. In this case, increase the time-out values for snipl --lparserver.

STONITH support (snIPL for STONITH)

The STONITH implementation is part of the Heartbeat framework of the High Availability Project (<http://linux-ha.org/>) and STONITH is generally used as part of this framework. It can also be used independently, however. A general description of the STONITH technology can be found at: <http://linux-ha.org/stonith.html>.

The STONITH support for snIPL can be regarded as a driver for one or more virtual power switches controlling a set of Linux images located on LPARs or z/VM instances as z/VM guests. A single LPAR or z/VM host can be seen as a VPS subswitch. STONITH requires the availability of a list of the controllable images by a switch. For this Linux Image Control VPS, the set of controlled images is retrieved from different locations depending on access rights and configuration.

The format of the snIPL for STONITH configuration file corresponds with the configuration file format of snIPL, see “Structure of the configuration file” on page 205.

Environment setup: The setup for using the STONITH plug-in depends on the environment in which you want to implement it.

- snIPL for STONITH in LPAR mode:

The SE must be configured to allow the initiating host system to access the network management API. Direct communication with the HMC is not supported.

For details, refer to either of these publications, as applicable:

zSeries Application Programming Interfaces, SB10-7030

S/390 Application Programming Interfaces, SC28-8141

You can obtain these publications from the following Web site:

ibm.com/servers/resource1ink/

- snIPL for STONITH in VM mode:

To communicate with the z/VM host, snIPL establishes a network connection and uses the Remote Procedure Call (RPC) protocol to send and retrieve data.

Communication with z/VM requires prior configuration of the VSMSEVERE server on z/VM. For details, refer to:

z/VM: Systems Management Application Programming, SC24-6063

You can obtain this publication from the following Web site: ibm.com/vm/

zIPL – zSeries initial program loader

Purpose

zIPL can be used on both zSeries and S/390 systems to prepare a device for the following purposes:

- booting Linux (as a Linux program loader)
- dumping
- loading a data file to initialize named saved segments

Notes:

1. In this description, the terms ‘boot’ and ‘IPL’ are used synonymously.
2. For more information on the dump tools that zIPL installs and on using the dump functions, refer to *Linux for zSeries and S/390 Using the Dump Tools*, LNX-1318.

Usage

zIPL supports devices as shown in Table 9.

Table 9. Supported devices

As a boot loader	As a dump tool installer
ECKD ¹ DASDs with fixed block, AIX®-compliant Linux Disk Layout (LDL)	ECKD DASDs with fixed block, AIX-compliant Linux Disk Layout (LDL)
ECKD DASDs with z/OS-compliant Compatible Disk Layout (CDL)	ECKD DASDs with z/OS-compliant Compatible Disk Layout (CDL)
Fixed Block Access (FBA) DASDs	Magnetic tape subsystems compatible with IBM3480, IBM3490, or IBM3590
SCSI with PC-BIOS disk layout	SCSI with PC-BIOS disk layout
¹ Enhanced Count Key Data	

In preparing a device, zIPL operates in one of two modes:

Configuration-file mode

A configuration file is accessed if the parameter for the basic zIPL function is not given on the command line (or if the `--menu` parameter is specified). The default configuration file is `/etc/zipl.conf`. You can also use the environment variable `ZIPLCONF` or the command-line parameter `--config` to specify the location of the configuration file.

In configuration-file mode, no command-line option is permitted that has a corresponding option line in a configuration section. Therefore, command-line options cannot be used to override or supplement configuration file sections.

Exception: The only exception to the above rule is the `-P` (or `--parameters=`) option. Parameters defined in this manner are combined with (but do not override) any found in the configuration section.

Command-line mode

If the basic zIPL function (other than `--menu`) is defined via a command-line option, no configuration file is accessed. In this case, the entire configuration must be defined using command-line parameters.

The basic zIPL functions are each invoked with one of the following parameters in the configuration file or on the command line:

Table 10. Basic zIPL functions

Parameter	Command line option	Configuration file option	Function
image	-i, --image	image=	Install a boot loader
dumpto	-d, --dumpto	dumpto=	Prepare a dump device (DASD or tape)
dumptofs	-D, --dumptofs	dumptofs=	Prepare a dump device (SCSI disk)
segment	-s, --segment	segment=	Prepare a device to load a file to initialize named saved segments
menu	-m, --menu	(None)	Install a multiboot configuration

One of these options is required for each zIPL call. If none is defined on the command line, the configuration file will be searched for a default configuration section. If none is found, or if the parameters are used in combination, zIPL will issue an error message and exit.

Except for dumpto, a *target directory* must be specified. The target directory is used to store boot-relevant files. The actual boot loader is installed onto the device containing the target directory. (The dumpto option does not require a target directory because the dump device is defined by the dumpto parameter itself.)

Installing a boot loader

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, must reside on the same device as the target directory; otherwise an error is reported. See also the --add-files option.
2. The target parameter.
This specifies the target directory, in which boot-relevant data will be stored. The boot loader will be installed on the device containing the target directory.
3. Boot parameters and/or the location of the parmfile. If you are defining the configuration via the command line, you can define parameters using the -P or --parameters and/or -p or --parmfile options. In configuration-file mode, you can set parameters in an IPL or SCSI dump section in the configuration file by using the parameters= and/or parmfile= option lines, and you can also use the -P or --parameters option (but not -p or --parmfile) on the command line as well.

Note: Parameters specified on the command line in configuration-file mode do not override corresponding entries in the configuration file. All parameters encountered, whether in the command line or the configuration file, are combined and passed to the kernel in one string. This string is loaded to the default address (0x1000) at IPL time unless

an alternate address was provided via the 'parmfile' option. Parameters passed to the SCSI System Dumper tool are always loaded to the default address, and any address specified with 'parmfile' is ignored.

Creating a dump device on DASD or tape

Before you start: For DASD, 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 `dump` parameter, either on the command line or in the configuration file. This parameter specifies the dump device (DASD partition or tape drive) to be prepared. zIPL will delete all data on the dump device and install the zIPL boot loader there.

Notes:

1. If the dump device is an ECKD disk with fixed-block layout (LDL), the dump utility will be overwritten by the dump, and you must re-install it in order to take another dump.
2. If the dump device is a disk with z/OS-compliant layout (CDL), you do not need to re-install the dump utility after every dump.
3. A target directory is not required because the dump device is defined by the `dump` specification.

Dumps to a DASD partition or tape are written in raw format and can be converted to a processable form using the `zgetdump` utility.

Creating a dump device on SCSI disk

At least one partition, the *target partition*, must be available to zIPL. This is the partition containing the target directory and is the partition that is accessed to load the SCSI System Dumper tool at IPL time. The dumps are written as files to the *dump partition*. The target partition can also be the dump partition, but it is possible (and recommended) to configure a separate dump partition. The target and dump partitions must be formatted with a file system supported by the SCSI Linux System Dumper tool. Unlike DASD and tape, creating a dump device on SCSI disk does not destroy the contents of the target partition.

To create a SCSI dump device with zIPL, you must specify the following options, either on the command line or in the configuration file:

1. The `dump` option, which specifies the dump partition, i.e., the partition to which the dump files will be written.
2. The `target` option, which specifies the target directory to which the SCSI System Dumper components will be written. zIPL uses the target directory to determine the dump device (target partition).

Installing a loader to initialize named saved segments (NSS)

To load a data file to initialize named saved segments, you must use the 'segment' parameter to specify the file to load at IPL time and the memory address. After the segment has been loaded, the system is put into the "disabled wait state". No Linux system will be started.

Format

Command line syntax

Prepare an IPL device (command-line mode)

The diagram illustrates the syntax of the `zip` command. It shows a sequence of options and their arguments, with memory addresses indicated for some options:

- `zip` (command name)
- `-i` (option) followed by `image` (argument) and a memory address `0x10000`.
- `-r` (option) followed by `ramdisk` (argument) and a memory address `0x80000`.
- `-p` (option) followed by `parmfile` (argument) and a memory address `0x1000`.
- `-P` (option) followed by `"parameters"` (argument).
- `-t` (option) followed by `directory` (argument).
- `-v` (option).
- `-a` (option).

Prepare a dump device (command-line mode)

```

>>> zip1
> -d dump_device [-n]
> [-D dump_partition -t directory [-p "parameters"] [-p parmfile]
> [-v]

```

Load an NSS data file (command-line mode)

```

▶▶—zipl— -s —segment_file,addr— -t —directory—
                                     └─v─┘

```

Install a multiboot (menu) configuration

```

>> zipl -m menu_name [ -c /etc/zipl.conf (1) ] [ -c config_file ] [ -v ]
[ -a ]

```

Notes:

- 1 The ZIPLCONF environment variable is used if it is defined. Explicit specification of a configuration file with this option overrides both the default file and ZIPLCONF.

Perform a zIPL function in configuration-file mode

```

>> zipl [ -c /etc/zipl.conf (1) ] [ -c config_file ] [ -v ] [ configuration (2) ]

```

Notes:

- 1 The ZIPLCONF environment variable is used if it is defined. Explicit specification of a configuration file with this option overrides both the default file and ZIPLCONF.
- 2 If no configuration name is given zIPL will use the configuration specified in the [defaultboot] section of the configuration file.

Get zIPL information

```

>> zipl [ -v ] [ -h ]

```

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 to be read from the configuration file.

-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 and an optional alternate address for the combined parameter list presented to the kernel (see item 3 on page 210). The default address is 0x1000.

Unlike **--parameters=**, this option cannot be specified in configuration-file mode.

Note: This option can also specify parameters for the SCSI System Dumper (SCSI dump device). Refer to *Linux for zSeries and S/390 Using the Dump Tools*, LNUX-1318 for a discussion of these parameters. The address field is ignored in this case.

-P *'parameters'* or **--parameters=***'parameters'*

Specifies kernel parameters. The string *parameters* is enclosed in single or double quotation marks. This option corresponds to the "parameters" line in the configuration file. The parameters have the form *key=value*, without spaces. Multiple *key=value* pairs are separated by spaces.

See item 3 on page 210 for a discussion of how parameters are combined and presented to the kernel. This option can also be specified on the command line when processing a single configuration in configuration-file mode.

Note: This option can also specify parameters for the SCSI System Dumper (SCSI dump device). Refer to *Linux for zSeries and S/390 Using the Dump Tools*, LNUX-1318 for a discussion of these parameters.

-d *dump_device* or **--dumpto=***dump_device*

Specifies the DASD partition or tape drive on which the dump will be located after IPL.

-D *dump_partition* or **--dumptofs=***dump_partition*

Specifies the partition to which the SCSI dump file will be written. This partition must be formatted with a file system supported by the SCSI Linux System Dumper tool (currently ext2 and ext3). It need not be the partition that contains the target directory (target partition), but it must be on the same physical SCSI disk.

The SCSI System Dumper image and the ramdisk are written to the target directory. The location and names of both files are defined in a shared config file in the s390-tools package. After zIPL has copied the files, it installs a boot loader configuration for the SCSI System Dumper and defines an implicit kernel parameter (*dump_part*) specifying the dump partition. For example, if **'-D /dev/sdb2'** is specified, zIPL inserts the implicit kernel parameter **'dump_part=2'**.

The **-D** option can install only a single dump configuration. Multiple dump and/or IPL configurations must be defined using a configuration file.

-s *segment_file,addr* or **--segment=***segment_file,addr*

Specifies the segment file to load and the memory location for the segment

-t *directory* or **--target=***directory*

Specifies the target directory where zIPL creates files. The actual boot loader will be installed on the disk containing the target directory. In the case of a SCSI dump device, this partition must have been formatted with

a file system supported by the SCSI System Dumper (currently ext2 or ext3). This parameter is not required in conjunction with `dump` to because the dump device is implied by the value of the `dump` parameter.

-m *menu_name* or --menu *menu_name*

Specifies the name of the menu defining a multiboot configuration to be installed on the disk defined by the target directory given in the menu section. If supported by the hardware, an interactive choice of configurations will be available at boot time. Otherwise, the default entry of the multiboot configuration will be used.

-V or --verbose

Provides more detailed information regarding the current zIPL operation.

-n or --noninteractive

Specifies that all confirmation questions (such as those warning against data loss when preparing a DASD disk for dumping) are automatically answered 'yes'. The questions themselves are suppressed, but all other output is shown. This option is useful when zIPL is called in a shell script or when interaction is not possible.

-a or --add-files

Specifies that all files (such as image, parmfile, ramdisk) are added to the bootmap file in the target directory rather than being referenced from this file. Use this option if you want to use files on one disk for IPL from another one.

Note: Specifying this option will significantly increase the size of the bootmap file created in the target directory.

-v or --version

Prints version information and then exits.

-h or --help

Displays help on command line arguments and then exits.

Configuration file syntax

Location: By default, zIPL retrieves the configuration file from `/etc/zipl.conf`. You can set environment variable `ZIPLCONF` to override 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.

Note: Option lines are valid only as part of a configuration-file section. Blank lines are permitted, and lines beginning with '#' are treated as comments and ignored.

Configuration file sections:

defaultboot

The configuration file can contain a `[defaultboot]` section with a **default=** option line (which is the only option permitted in this section). If no configuration is specified on the command line, the `[defaultboot]` section is accessed and the default configuration section, which can be of any type, is activated.

default=*configuration_section*

Indicates which configuration is used if none is specified on the command line.

Example:

```
[defaultboot]
default=myconfig
```

configuration

A configuration section defines an IPL or dump configuration. It begins with a line with the section name (other than [defaultboot]) in square brackets. Configuration section names are specified in the [defaultboot] section or on the command line.

Example:

```
[myconfig]
(options)
```

A configuration section can contain one or more of the following option lines:

target=*directory*

Specifies the directory that zIPL uses to store boot-relevant files. The actual boot loader (or the SCSI System Dumper and associated components) is installed on the device containing the target directory. This option is not required in conjunction with the *dump*to parameter, which defines the dump device in this case.

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 the 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 and an optional alternate address for the combined parameter list presented to the kernel (see item 3 on page 210). The default address is 0x1000.

This option line can also be defined in a SCSI dump configuration to provide parameters to the SCSI disk dump tool. See *Using the Dump Tools* for a description of these parameters. The address entry is ignored in this case.

parameters=*'parameters'*

Specifies the parameters for the kernel (in conjunction with an IPL configuration). 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.

See item 3 on page 210 for a discussion of how parameters are combined and presented to the kernel.

This option line can also be defined in a SCSI dump configuration to provide parameters to the SCSI disk dump tool. See *Using the Dump Tools* for a description of these parameters.

dumpto=*dump_device*

Specifies the DASD partition or tape drive on which the dump will be located after IPL.

dumptofs=*dump_partition*

Specifies the formatted SCSI disk partition to which the dump file will be written.

segment=*segment_file,address*

Specifies a file (and load address) to be loaded at IPL to define named saved segments.

menu

Multiple boot configurations can be defined using a menu section. If supported by the hardware, an interactive choice of these configurations will be available at boot time. Otherwise, the menu's default entry is booted.

A menu section begins with a line consisting of a colon followed by an alphanumeric menu name. This line is followed by one or more option lines as follows:

menu_position=section_name

For each configuration to be included in the menu, a line consisting of *menu_position=section name* is present. The section name identifies the respective configuration defined elsewhere in the file. The menu position can be a number between 1 and 62 (30 in case of SCSI devices). This number specifies the menu position of the respective configuration. The number must be unique within the menu, i.e., specifying the same position twice in one menu will be rejected by zIPL as an error. Also, only standard IPL (image) and SCSI dump (dumptofs) sections are valid in a menu—sections containing the dumpto or segment keyword are invalid and will result in an error message.

target=directory

In the menu section, specifies the directory which zIPL uses to identify the device on which to install the configurations specified by the menu. If this option is included in a configuration that is part of a menu, it is ignored when the menu is installed. All configurations in a menu section are installed on the same device.

default=menu_position

Indicates which menu position to use if none is selected at boot time. If the position specified by the default entry is not defined in the menu, zIPL will report an error. If this line is not present, the lowest defined menu position is the default. The default position is assigned an internal menu number of '0'.

Multiple menus can be defined in the same configuration file. However, only one menu can be written to a particular target directory.

There are no restrictions on positioning the menu section with respect to the configuration sections it includes. In particular, it does not have to follow those sections. Also, the menu-position lines within a menu section can be in any order.

A multiboot configuration is installed by calling zIPL with the --menu command-line option, which takes the menu name as an argument.

Example:

```
:menu1
1=config1
2=config2
target=/boot
default=1
```

Examples

Installing an IPL configuration using the zIPL command line

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=/dev/dasda1 ro dasd=fd00
```

Result: zIPL will install the boot loader on the device of the target directory.

Installing a SCSI dump configuration using the zIPL command line

The following example shows how to use zIPL to define a single dump configuration on a SCSI disk. Assume that the partition /dev/sdb1 has been formatted with the PC-BIOS disk layout and a file system (ext2 or ext3) compatible with the SCSI System Dumper. This configuration can be invoked at IPL time using configuration number 0.

```
mount /dev/sdb1 /mnt
zipl -D /dev/sdb1 -t /mnt -P "dump_compress=gzip"
umount /dev/sdb1
```

Result: zIPL will install the SCSI System Dumper tool in the partition's root directory. Dump files generated when this partition is IPLed will also be written to this directory, with gzip compression.

Loading a file to initialize named saved segments

The following example shows how to use zIPL to install a segment loader for named saved segments:

```
zipl --segment=/boot/segment,0x80000 --target=/boot
```

Result: zIPL will install a segment loader that will load the file specified by /boot/segment to address 0x80000 at IPL time and then place the CPU in the "disabled wait" state.

Using a configuration file

This sample configuration file (assumed to be /etc/zipl.conf) defines multiple configuration sections and a menu. The SCSI disk has the PC-BIOS disk layout, and the target and dump partitions are formatted with a file system supported by the SCSI System Dumper.

```
[defaultboot]
default=ip12

# First boot configuration
[ip11]
ramdisk=/boot/initrd
parameters='root=/dev/ram0 ro'
image=/boot/image-1
target=/boot

# Second boot configuration
[ip12]
image=/boot/image-2
ramdisk=/boot/initrd
parmfile=/boot/parmfile-2
target=/boot

# Configuration for dumping to tape
[dumptape]
```

```

dumppto=/dev/rtibm0

# Configuration for dumping to DASD
[dumpdasd]
dumppto=/dev/dasdc1

# Configuration for dumping to SCSI disk
# Separate IPL and dump partitions
[dumpscsi]
target=/boot
dumptrfs=/dev/sda2
parameters="dump_dir=/mydumps dump_compress=none dump_mode=auto"

# Menu containing the IPL and SCSI dump configurations
:menu
1=ipl1
2=ipl2
3=dumpscsi
default=2
target=/boot

# Configuration for initializing named saved segments
[segment]
segment=/boot/segment,0x80000
target=/boot

```

Given this configuration file, the following commands could be issued:

- Call zIPL to use the default configuration file settings:

```
zipl
```

Result: zIPL reads the default option from the [defaultboot] section and selects the [ipl2] section. It then installs a boot loader that will load /boot/image-2, /boot/initrd, and /boot/parmfile-2

- Call zIPL to create a tape that can be IPLed for a tape dump::

```
zipl dumptape
```

Result: zIPL selects the [dumptape] section and prepares a dump tape on /dev/rtibm0.

- Call zIPL to create a DASD dump device:

```
zipl -n dumpdasd
```

Result: zIPL selects the [dumpdasd] section and prepares the dump device /dev/dasdc1. Questions normally issued during the installation process are automatically answered 'yes'.

- Call zIPL to create a SCSI dump device:

```

mount /dev/sda1 /boot
mount /dev/sda2 /dumps
mkdir /dumps/mydumps
zipl dumpscsi
umount /dev/sda1
umount /dev/sda2

```

Result: zIPL selects the [dumpscsi] section and prepares the dump device /dev/sda1. The associated dump file will be created uncompressed in directory

zIPL

/mydumps on the dump partition. If space is required, the lowest-numbered dump file in the directory will be deleted.

- Call zIPL to create a multiboot configuration:

```
zipl --menu=menu
```

Result: zIPL will define a multiboot configuration, written to the device implied by /boot, with 3 options. The default configuration is 2.

- Call zIPL to install a loader to initialize named saved segments:

```
zipl segment
```

Result: zIPL will install a segment loader that will load the contents of file /boot/segment to address 0x80000 at IPL time and then put the processor into the disabled wait state.

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 “Confirming that an IP address has been set” on page 157 for information on potential failures when setting IP addresses on interfaces associated with OSA-Express in QDIO mode (QETH driver).

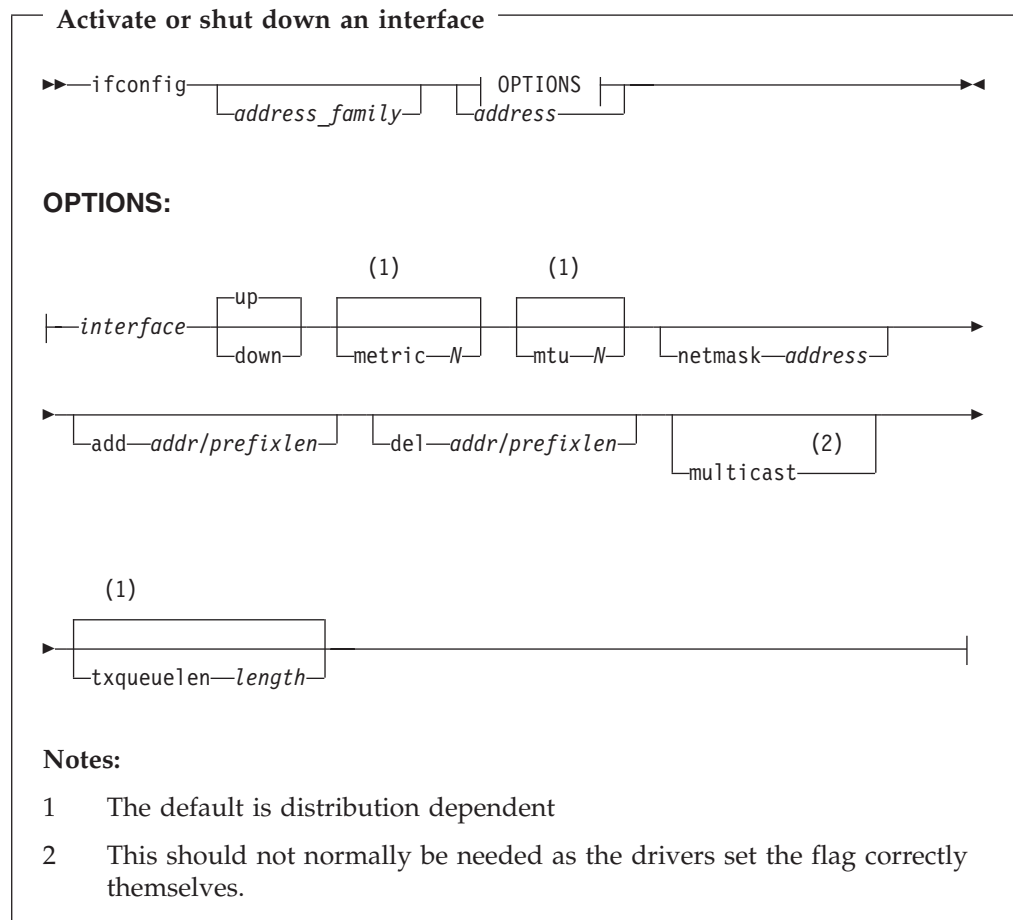
Format

Display active interface status

►► ifconfig ◀◀

Display status of given interface

►► ifconfig *interface* ◀◀



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 and S/390, 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:

- *ctci* (*i* = 0 to 255)
- *esconi* (*i* = 0 to 255)
- *ethi* (*i* = 0 to 255)
- *hssi* (*i* = 0 to 255)
- *iucv0*
- *lo*
- *tri* (*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.⁹

mtu N This parameter sets the maximum transfer unit (MTU) of an interface. See “MTU considerations” on page 157 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

9. 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:

$$\text{cost} = \text{ref-bandwidth} / \text{bandwidth}$$

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 or OS/390)

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=0,0xdd,0xxyy,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:

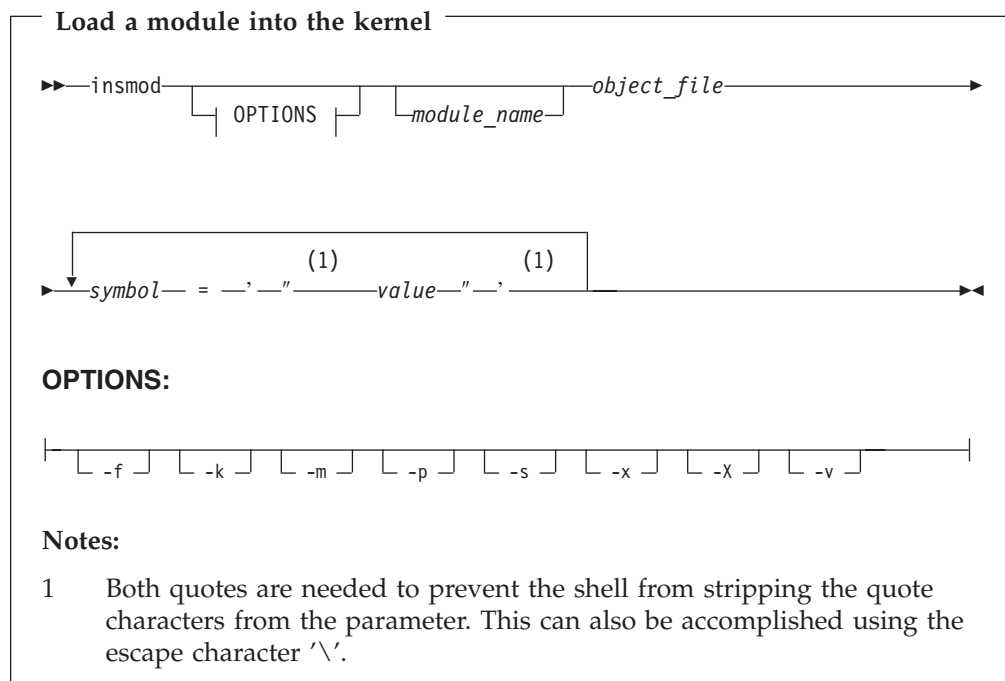
```
ifconfig eth0 192.168.100.11
```

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



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.

insmod

- 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
```

QDIO module

```
insmod qdio
```

OSA Express module

```
insmod qeth
```

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 231). 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

```
modprobe [OPTIONS] [-C —/etc/modules.conf | -C —configfile] module_name
```

```
symbol = '(1) value(1)'
```

OPTIONS:

```
-a -d -k -n -q -s -v
```

Notes:

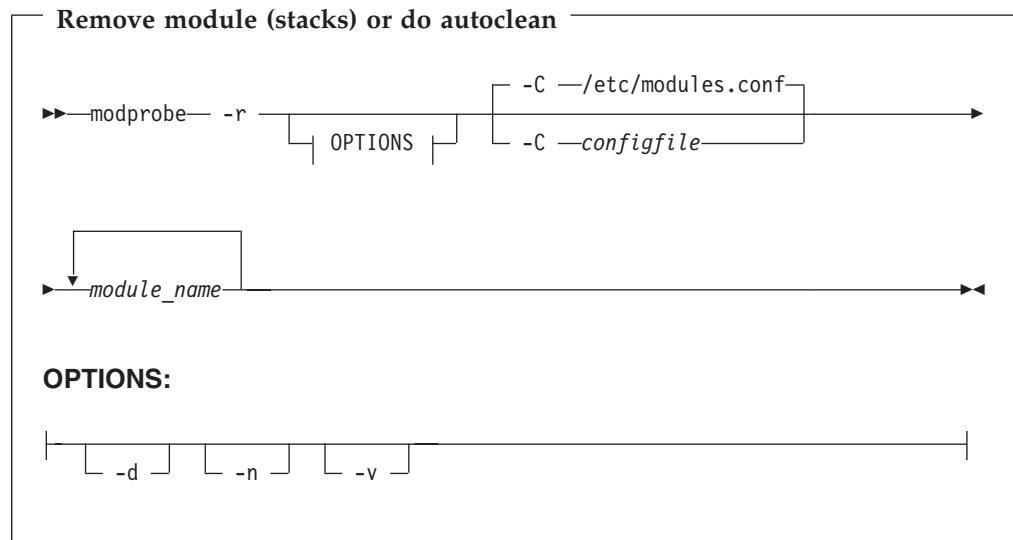
- Both quotes are needed to prevent the shell from stripping the quote characters from the parameter. This can also be accomplished using the escape character `'\'`.

List matching modules

```
modprobe -l [-C —/etc/modules.conf | -C —configfile] [-t —type] pattern
```

Show configuration

```
modprobe -c [-C —/etc/modules.conf | -C —configfile]
```



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.

Working with the osasmpd support

Note that this list is not comprehensive. See `man modprobe` for information on the full set of parameters.

Modules not matching the kernel version can be forced by adding the line `insmod_opt=-f` to `/etc/modules.conf`. This has the same effect as the `insmod` parameter `-f`.

Examples

CLAW module

```
modprobe claw
```

This loads CLAW passing it the parameter string
"0:0x1Fc4:15:25:LINUXA:RS6K:API".

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

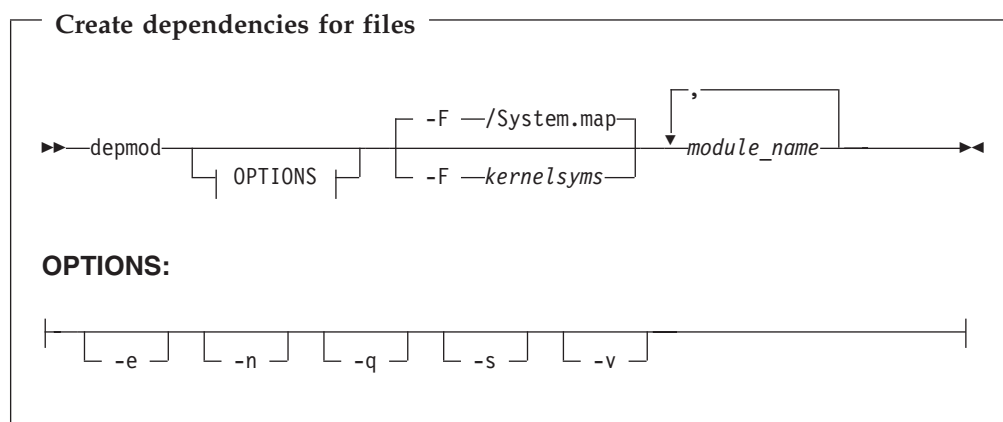
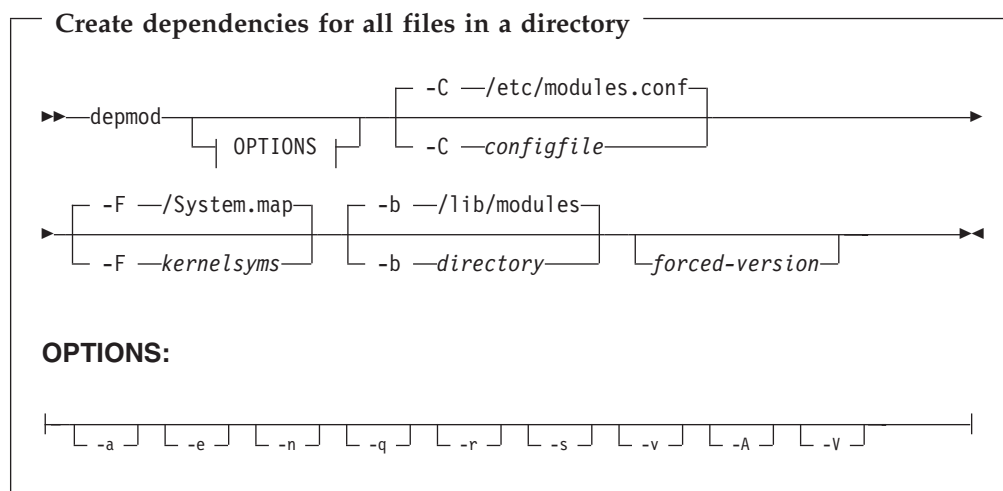
```
# lsmod
Module                Size  Used by
qeth                  135680  1 (autoclean)
qdio                   22992   1 (autoclean) [qeth]
#
```


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



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.

depmod

- 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
```

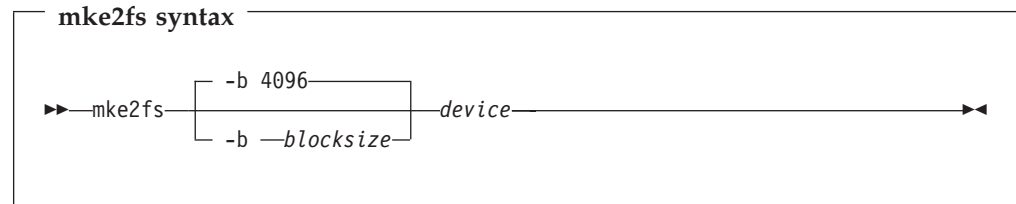
displays the unresolved references in *mymodule* on stdout.

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



-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

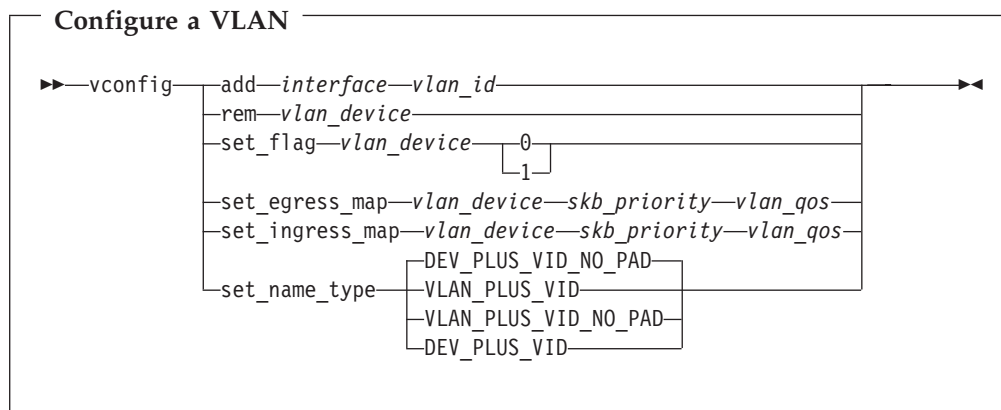
```
mke2fs -b 4096 /dev/dasd/<devno>/part<x>
```

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



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 22, "Virtual LAN (VLAN) support," on page 261.

vconfig

Chapter 18. VIPA – minimize outage due to adapter failure

This chapter describes how to use

- Standard VIPA
- Source VIPA (versions 1.x)
- Source VIPA 2 (version 2.0.0)

VIPA allows you to assign IP addresses to a *system*, instead of to *individual adapters*. This minimizes outage caused by adapter failure. Standard VIPA is usually sufficient for applications, such as Web Server, that do *not* open connections to other nodes. Source VIPA is used for applications that open connections to other nodes. Source VIPA Extensions enable you to work with multiple VIPAs per destination in order to achieve multipath load balancing.

Notes:

1. The VIPA functionality requires a kernel built with the CONFIG_DUMMY option.
2. See the information in “Confirming that an IP address has been set” on page 157 concerning possible failure when setting IP addresses for OSA-Express features in QDIO mode (QETH driver).
3. The configuration file layout for Source VIPA has changed since the 1.x versions. In the 2.0.0 version a *policy* is included. For details see the README and the man pages provided with the package.

Standard VIPA

Purpose

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

Usage

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

1. Create a dummy device 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 16 on page 238).
 - 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.

Virtual IP addressing (VIPA)

- 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 16 shows the network adapter configuration used in the example.

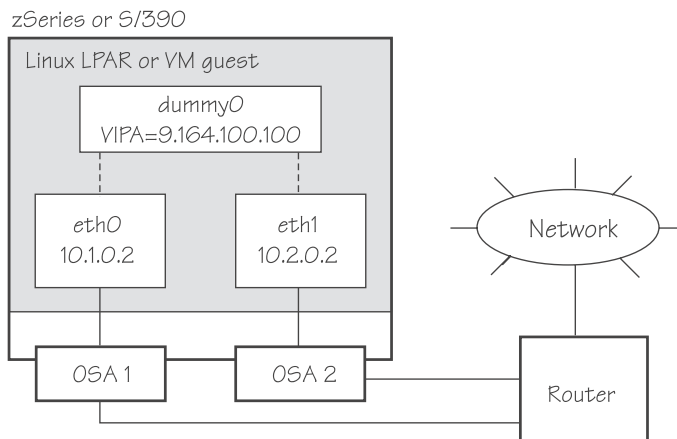


Figure 16. Example of using Virtual IP Address (VIPA)

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.

```
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:

```
echo add_vipa6 200200000000000000000000000012345678:eth0 >/proc/qeth_ipa_takeover  
echo add_vipa6 20020000000000000000000000001235678:eth1 >/proc/qeth_ipa_takeover
```

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
```


Source VIPA

This version of Source VIPA has been superseded in October 2003 by Source VIPA 2 (version 2.0.0). If you use Source VIPA 2 read “Source VIPA 2” on page 241.

Purpose

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

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

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

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

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

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

Usage

Installation

```
make
make starter
make install
```

Paths can be changed in the Makefile. Defaults are:

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

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

Configuration

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

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

`D1.D2.D3.D4/MASK` specifies a range of destination addresses and the number of bits set in the subnet mask (MASK). As soon as a socket is opened and connected to these destination addresses and the application does not do an explicit bind to a source address, `src_vipa` does a bind to `S1.S2.S3.S4`. Instead of IP addresses in dotted notation, hostnames can be used and will be resolved using DNS.

Virtual IP addressing (VIPA)

.INADDR_ANY P1-P2 S1.S2.S3.S4, or the same command with only one port P, will associate sockets with IP addresses causing bind calls with INADDR_ANY as a local address to be intercepted if the port the socket is bound to is between P1 and P2 (inclusive). In this case, INADDR_ANY will be replaced by S1.S2.S3.S4 (which can be 0.0.0.0).

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

The default behavior for all ports is that the kind of bind calls will not be modified. If an application first does a bind call with INADDR_ANY and an INADDR_ANY rule is defined with a matching port range, the .INADDR_ANY rule is used. In this case any D1.D2.D3.D4/MASK rules are ignored. If no matching .INADDR_ANY rule exists, or the application does not call bind at all, the D1.D2.D3.D4/MASK rule is used, if the target address of the program's connect call matches.

Enabling an application

The command:

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

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

Example

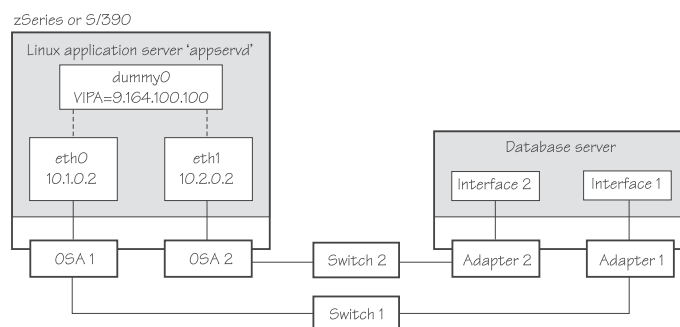


Figure 17. Example of using source VIPA

The command:

```
src_vipa.sh appservd start
```

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

Source VIPA 2

Purpose

Source VIPA 2 is particularly suitable for high-performance environments. It selects one source address out of a range of source addresses when it replaces the source address of a socket. The reason for using several source addresses lies in the inability of some operating system kernels to do load balancing among several connections with the same source and destination address over several interfaces.

To achieve load balancing, a *policy* has to be selected in the *policy* section of the configuration file of Source VIPA 2 (`/etc/src_vipa.conf`). This *policy* section also allows to specify several source addresses used for one destination. Source VIPA then applies the source address selection according to the rules of the *policy* selected in the configuration file.

This Source VIPA solution does not affect kernel stability. Source VIPA is controlled by a configuration file containing flexible rules for when to use Source VIPA based on destination IP address ranges.

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

Usage

Installation

```
make
make starter
make install
```

Paths can be changed in the Makefile. Defaults are:

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

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

Migration

If you migrate from an earlier version of Source VIPA and do not need multiple VIPAs, the *onevipa policy* followed by your VIPA is the recommended change (see “Policies” on page 242). Please check your syslog (usually in `/var/log/messages`) for problems the first time you use the new version.

Configuration

With Source VIPA 2 the configuration file has changed: the *policy* section was added. The default configuration file is `/etc/src_vipa.conf`.

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

```
# comment
D1.D2.D3.D4/MASK POLICY S1.S2.S3.S4 [T1.T2.T3.T4 [...]]
.INADDR_ANY P1-P2 POLICY S1.S2.S3.S4 [T1.T2.T3.T4 [...]]
.INADDR_ANY P POLICY S1.S2.S3.S4 [T1.T2.T3.T4 [...]]
```

`D1.D2.D3.D4/MASK` specifies a range of destination addresses and the number of bits set in the subnet mask (MASK). As soon as a socket is opened and connected to these destination addresses and the application does not do an explicit bind to a source address, Source VIPA does a bind to one of the source addresses specified

Virtual IP addressing (VIPA)

(S, T, [...]) using the *policy* selected in the configuration file to distribute the source addresses. See the *policy* section below for available load distribution policies. Instead of IP addresses in dotted notation, hostnames can also be used and will be resolved using DNS.

`.INADDR_ANY P1-P2 POLICY S1.S2.S3.S4` or `.INADDR_ANY P POLICY S1.S2.S3.S4` causes bind calls with `.INADDR_ANY` as a local address to be intercepted if the port the socket is bound to is between P1 and P2 (inclusive). In this case, `.INADDR_ANY` will be replaced by one of the source addresses specified (S, T, [...]), which can be 0.0.0.0.

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

The default behavior for all ports is that the kind of bind calls will not be modified. If an application first does a bind call with `INADDR_ANY` and an `INADDR_ANY` rule is defined with a matching port range, the `.INADDR_ANY` rule is used. In this case any `D1.D2.D3.D4/MASK` rules are ignored. If no matching `.INADDR_ANY` rule exists, or the application does not call bind at all, the `D1.D2.D3.D4/MASK` rule is used, if the target address of the program's connect call matches.

Policies

With Source VIPA Extensions you provide a range of dummy source addresses for replacing the source addresses of a socket. The policy selected determines which method is used for selecting the source addresses from the range of dummy addresses..

onevipa

Only the first address of all source addresses specified is used as source address.

random

The source address used is selected randomly from all the specified source addresses.

llr (local round robin)

The source address used is selected in a round robin manner from all the specified source addresses. The round robin takes place on a per-invocation base: each process is assigned the source addresses round robin independently from other processes.

rr:ABC

Stands for round robin and implements a global round robin over all Source VIPA instances sharing the same configuration file. All processes using Source VIPA access an IPC shared memory segment to fulfil a global round robin algorithm. This shared memory segment is destroyed when the last running Source VIPA ends. However, if this process does not end gracefully (for example, is ended by a `kill` command), the shared memory segment (size: 4 bytes) can stay in the memory until it is removed by `ipcrm`. The tool `ipcs` can be used to display all IPC resources and to get the key or id used for `ipcrm`. ABC are UNIX permissions in octal writing (for example, 700) that are used to create the shared memory segment. This

permission mask should be as restrictive as possible. A process having access to this mask can cause an imbalance of the round robin distribution in the worst case.

- 1c Attempts to balance the number of connections per source address. This policy always associates the socket with the VIPA that is least in use. If the policy cannot be parsed correctly, the policy is set to round robin per default.

Enabling an application

The command:

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

enables the Source VIPA functionality for the application. The configuration file is read once the application is started. It is also possible to change the starter script and run multiple applications using different Source VIPA settings in separate files. For this, a SRC_VIPA_CONFIG_FILE environment variable pointing to the separate files has to be defined and exported prior to invoking the respective application.

Example

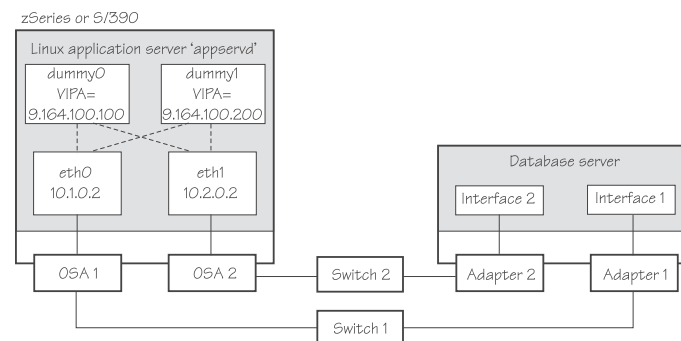


Figure 18. Example of using source VIPA 2

The entry in the Source VIPA configuration file::

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

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

Chapter 19. 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, logically offline or n/a (for CHPIDs on standby). 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

VARY ON/OFF events

- 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 20. Multipathing support

Some storage devices allow operating systems to access device storage through more than one physical path, thus extending availability and sometimes allowing better performance by permitting use of all paths in parallel. To make use of these features, an operating system must be able to detect multipathed devices and manage all access in a way that avoids data corruption.

Currently, Linux for zSeries and S/390 supports only the Enterprise Storage Server (ESS) through extensions to the Linux Logical Volume Manager (LVM). To make use of the LVM extensions, any device driver must provide one device node for every path that can be accessed independently (the device driver handles the path switching if the device architecture needs anything special for that purpose).

Supported storage devices

IBM Enterprise Storage Server (ESS)

At the present time, only SCSI-attached ESS units are supported. SCSI-attached ESS units accessed by `zfc` can have additional paths created in Linux by defining additional mappings between the SCSI host adapter to ESS WWPN to the same LUN. The only mapping that will not work is an additional mapping through the same virtual host adapter and the same ESS port to the same device. Since that would neither give more reliability nor increase performance, this limitation is not a serious one.

Multipath support in blockdevice managers

Block-device managers (such as LVM, MD, or EVMS) take block devices and provide new logical ones after some transformation. This just requires the lower device drivers to provide a separate device for every path so that the manager knows about the paths.

Logical Volume Manager (LVM)

The Logical Volume Manager for Linux enables you to concatenate several physical volumes into a so-called volume group (VG) and form a storage pool, like a virtual disk.

The first step is to create LVM metadata on either the disk or one of its aliases. This is done with the `pvcreate` command exactly the same way it is done for "normal" disks.

This disk is then used either for a new volume group (VG) or to extend an existing one. Again, it does not matter whether the main disk or one of its aliases is used. LVM compares the UUIDs, stored in the metadata, of all known disks (partitions). Whenever it finds two identical UUIDs, it assumes this is just another path to the previously known disk.

Note: Since only the UUID on the disk is currently used for path detection, the raw disk must *never* be copied to another disk. Otherwise, LVM will use the backup as an additional path.

Multipathing support

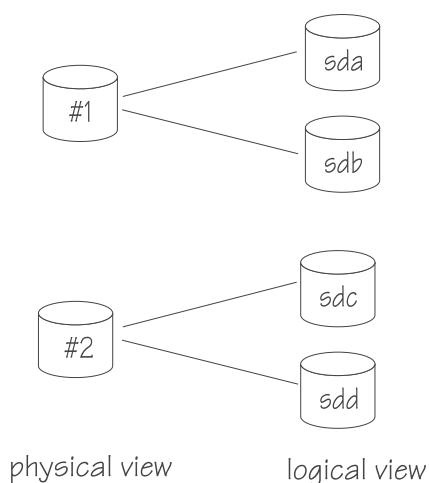
As soon as the new physical volume (PV) is integrated into the VG, it can be used for logical volumes (LV). Since the multiple paths are an attribute of a PV, only I/O to those physical extents (PE) that belong to a multipathed PV can be done in parallel.

After a VG with multipathed PVs is active, some characteristics can be changed by the `pvpath` command.

1) Creating the devices and assigning them to a volume group

Use `pvcreate` and `vgcreate` as you would for a single path, that is, always specify the physical device, not the paths.

Example: Let `sda`, `sdb`, `sdc`, `sdd`, be paths to devices with one partition each. `sda` and `sdb` are alternative paths to the same physical device (#1) and `sdc` and `sdd` are alternative paths to the same physical device (#2).



To create a device use `pvcreate` and specify only one of the alternative paths for each physical device.

```
#> pvcreate /dev/sda1
pvcreate -- physical volume "/dev/sda1" successfully created

#> pvcreate /dev/sdd1
pvcreate -- physical volume "/dev/sdd1" successfully created
```

Tip: You can use `#> pvcreate /dev/sd[ad]1` as a shorthand that creates both devices.

To create a volume group `vgcreate` and specify only one of the alternative paths for each physical device you want to add.

```
#> vgcreate vg01 /dev/sd[ad]1
vgcreate -- INFO: using default physical extent size 4 MB
vgcreate -- INFO: maximum logical volume size is 255.99 Gigabyte
vgcreate -- doing automatic backup of volume group "vg01"
vgcreate -- volume group "vg01" successfully created and activated
```

Tip: You can use `vgextend` to add devices to an existing volume group.

2) Obtaining path information

You can display path information by:

- Displaying corresponding files in the `/proc/lvm` directory
- Using the `pvpath` command. In addition to the information that can be obtained with `cat`, `pvpath` also gives information on the weight setting

You can display generic path information using the `cat` command to read out general path information from the global file.

```
#> cat /proc/lvm/global
LVM module LVM version 1.0.5(mp-v7)(15/07/2002)

Total: 1 VG 2 PVs 0 LVs (0 LVs open)
Global: 7298 bytes malloced IOP version: 10 0:01:23 active

VG: vg01 [2 PV, 0 LV/0 open] PE Size: 4096 KB
Usage [KB/PE]: 966656 /236 total 0 /0 used 966656 /236 free
PVs: [AA] sda1 483328 /118 0 /0 483328 /118
      +-- sdb1
      [AA] sdc1 483328 /118 0 /0 483328 /118
      +-- sdd1
LVs: none
```

Note: Regardless of which path was used when creating the physical device the main device nodename is always the lowest major-minor

To display detailed path information for a specific path use the `cat` command to read out the corresponding file:

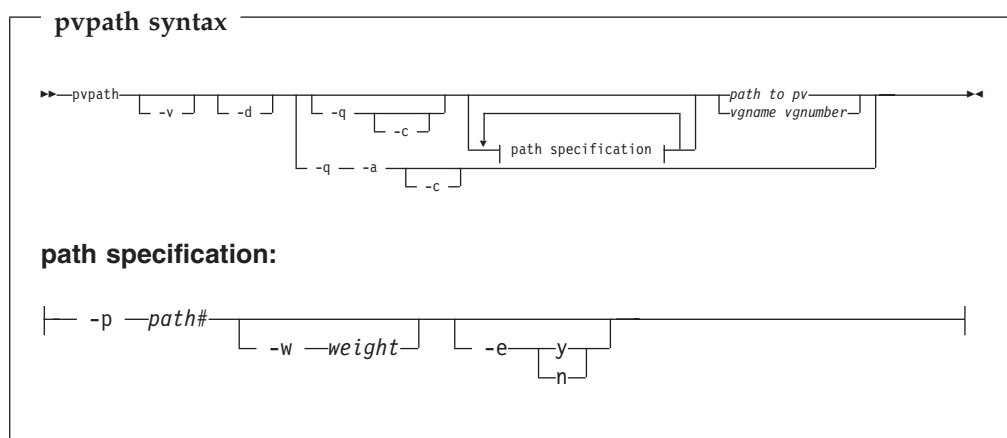
```
#> cat /proc/lvm/VGs/vg01/PVs/sda1
name: /dev/sda1
size: 976260
status: 1
number: 1
allocatable: 2
LV current: 0
PE size: 4096
PE total: 118
PE allocated: 0
device: 08:01
uuid: whFb-mLOL-diN9-UpTy-I0TI-Sd3C-s6eh-iCrS

Path pending state device
# 0: 0 enabled /dev/sda1
# 1: 0 disabled /dev/sdb1
```

In the output, the “Pending” column contains the number of I/O blocks that were passed to the device driver and are still being processed. If there is no I/O taking place on the device, the number is 0. If a single application is reading, then this may be about `<read ahead sectors>/<number of paths>`. Since writes are done asynchronously, the numbers can be quite high (eg. 3000-5000), depending on memory and speed of the device.

In the output, the two numbers separated by a colon (device 08:01), are the major and minor of the device that is used as a path.

You can also use the `pvpath` command to display path information.



The meanings of the parameters of this command are as follows:

- v** Turns on verbose output
- d** Turns on debugging messages
- q** Queries then lists information about the paths
- c** Formats the output into columns. This option is effective only when specified in conjunction with **-q**.
- a** This parameter is permitted only with **-q**, as in **-qa**. It lists all multi path PVs. When used in this way, you can not make any changes to a path.
- p path#** Sets the path for the following **-w** and **-e** options.
- w** Sets the weight. Applies to the previously specified path (**-p**) only.
- e y|n** Enables (y) or disables (n) the previously specified path (**-p**).

path to PV

Path to main device node (not alias), for example, `/dev/sda1`.

Notes:

1. The blanks between parameter and parameter value are optional (for example, **-p 0** and **-p0** are equivalent)
2. The blanks between parameters without variables are optional (for example, **-q -a** and **-qa** are equivalent)

The following example queries the information for all paths.

```
#> pvpath -qa
Physical volume /dev/sda1 of vg01 has 2 paths:
  Device Weight Failed Pending State
# 0: 8:1 0 0 0 enabled
# 1: 8:17 0 0 0 disabled
Physical volume /dev/sdc1 of vg01 has 2 paths:
  Device Weight Failed Pending State
# 0: 8:33 0 0 0 enabled
# 1: 8:49 0 0 0 disabled
```

The following example displays the same information as the previous one, but in a different format.

```
#> pvpath -qac
VG:vg01
PV:1:2:/dev/sda1
PATH:0:8-1:0:0:0:enabled
PATH:1:8-17:0:0:0:disabled
VG:vg01
PV:2:2:/dev/sdc1
PATH:0:8-33:0:0:0:enabled
PATH:1:8-49:0:0:0:disabled
```

Changing the path status

Because of security considerations, by default only one of the alternative paths to a physical device are enabled. Use `pvpath` to change the path status.

The following example uses the volume group specification to enable the second path.

```
#> pvpath -q -p1 -ey vg01 1
vg01: setting state of path #1 of PV#1 to enabled
Physical volume /dev/sda1 of vg01 has 2 paths:
      Device Weight Failed Pending State
#  0:   8:1      0      0      0 enabled
#  1:   8:17     0      0      0 enabled
```

The following example uses the device path specification to enable the second path. Note that the main device nodename for the physical device is required. `/dev/sdd1` would not work.

```
#> pvpath -qc -p1 -ey /dev/sdc1
vg01: setting state of path #1 of PV#2 to enabled
PV:2:2:/dev/sdc1
PATH:0:8-33:0:0:0:enabled
PATH:1:8-49:0:0:0:enabled           done by the hotplug agent
```

Setting the weight value

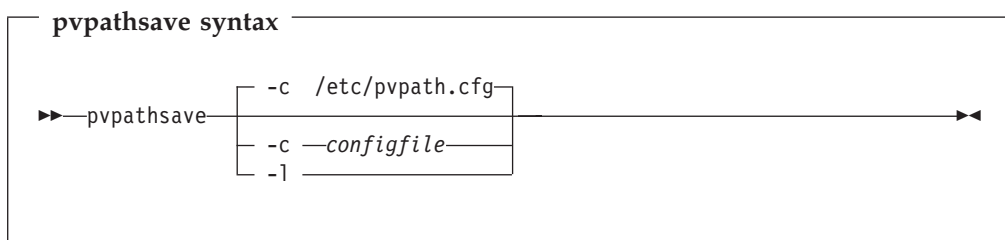
The *weight* value changes the way LVM uses the available paths. If all paths have the same weight, LVM uses all paths while trying to balance the load across the number of active I/O operations between all paths. In the case of different weights, LVM only uses the paths that have the lowest weight, as long as one active path remains. Should all paths of that weight fail, the path(s) with the next higher weight are used. If all paths have a different weight, this is failover-only support.

The following example sets the weight value of path #0 to 2 and disables it. At the same time it sets the weight value of path #1 to 1.

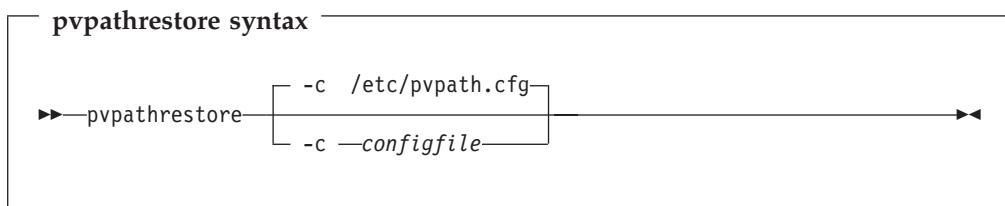
```
pvpath -p0 -en -w2 -p1 -w1 vg01 1
Physical volume /dev/sda1 of vg01 has 2 paths:
      Device Weight Failed Pending State
#  0:   8:1      2      0      0 disabled
#  1:   8:17     1      0      0 enabled
```

Saving and restoring path states

`pvpathsaves` stores the current state and weight of all paths into the specified configuration file. The default for the configuration file is `/etc/pvpath.cfg`. Option `-l` prevents `pvpathsaves` from storing to a file and prints to stdout instead (List data).



`pvpathrestore` sets state and weights according to the specified configuration file. The default for the configuration file is `/etc/pvpath.cfg`.



The layout of the file consists of three types of entries:

1. VG:<vgname>:<default path state (enabled|disabled)>

This starts a new section for a volume group. The PV and PATH statements that follow it refer to this VG. The default path state is used whenever there is no matching PV/PATH to set a specific state and weight.

2. PV:<pvnumber>:<number of paths>:<pvpath>

This starts a new PV section. The PATH statements that follow it refer to this path. The `pvpath` is the device path for the first path of the PV (the path with the lowest major/minor number)

3. PATH:<pathnumber>:<major>-<minor>:<weight>:<state>

This specifies the weight and state for a path. The major-minor number is set to make sure that on `pvpathrestore`, only matching major-minors are set to a specific state.

To enable all paths, you have to create `/etc/pvpath.cfg` with one line for each VG as shown below:

```
VG:<vgname>:enabled
```

Since there are no specific path lines, all paths will be enabled with weight unchanged (0).

Setting the read-ahead value

To increase performance, the number of read-ahead sectors for the LV has to be set according to the number of parallel paths. (This currently requires another patch to the LVM kernel code.) The read-ahead value is set by

```
blockdevice --setra <#>
```

Another, possibly better way to set the read-ahead is to use the

```
lvchange -r<ra sectors> <lvpath>
```

command of LVM, since this stores the read-ahead value permanently in the LVM metadata.

Chapter 21. Linux monitor stream support for z/VM

z/VM is a convenient point for collecting VM guest performance data and statistics for an entire server farm. Linux guests can export such data to z/VM by means of “APPLDATA monitor records”. z/VM regularly collects these records. The records are then available to z/VM performance monitoring tools.

A virtual CPU timer on the Linux guest to be monitored controls when data is collected. The timer only accounts for busy time to avoid unnecessarily waking up an idle guest. The monitoring stream support comprises several modules. A base module provides an intra-kernel interface and the timer function. The intra-kernel interface is used by *data gathering modules* that collect actual data and determine the layout of a corresponding APPLDATA monitor record (see “APPLDATA monitor record layout” on page 257).

Building a kernel that is enabled for monitoring

This section describes the kernel options you must select to enable the Linux monitor stream support.

The Virtual CPU Timer Interface must be installed on your Linux guest.

Kernel configuration menu option	Required for	Compilation	Module name
Linux - VM Monitor Stream, base infrastructure	All data gathering	Must be compiled into the kernel	n/a
Monitor memory management statistics	Memory data	Can be compiled into the kernel or as separate module	appldata_mem.o
Monitor OS statistics	Processor data		appldata_os.o
Monitor overall network statistics	Networking		appldata_net_sum.o

These options can be found in the config menu under “General setup”.

Data gathering modules that are not compiled into the kernel can be loaded with **insmod** or **modprobe**. There are no kernel or module parameters for the base module or for the data gathering modules.

Working with the monitoring stream

Before you start: The Linux guest directory must include the option APPLMON.

You control the monitor stream support through the /proc file system. You can set the timer interval and switch on or off data collection. APPLDATA monitor records are produced if both a particular data gathering module and the monitoring support in general are switched on.

Switching on or off the monitoring support

You switch on or off the monitoring support by writing “1” (on) or “0” (off) to `/proc/sys/appldata/timer`.

To read the current setting issue:

```
cat /proc/sys/appldata/timer
```

To switch on the monitoring support issue:

```
echo "1" > /proc/sys/appldata/timer
```

To switch off the monitoring support issue:

```
echo "0" > /proc/sys/appldata/timer
```

Activating or deactivating individual data gathering modules

You can activate or deactivate the data gathering modules individually. Each data gathering module has a `/proc` file system entry that contains a value “1” if the module is active and “0” if the module is inactive. The entries are:

`/proc/sys/appldata/mem` for the memory data gathering module

`/proc/sys/appldata/os` for the CPU data gathering module

`/proc/sys/appldata/net_sum` for the net data gathering module

To check if a module is active look at the content of the corresponding `/proc` entry.

Example: To check if the memory data gathering module is active issue:

```
cat /proc/sys/appldata/mem
```

Note: An active data gathering module produces APPLDATA monitor records only if the monitoring support is switched on (see “Switching on or off the monitoring support”).

To activate a data gathering module write “1” to the corresponding `/proc` entry.

Example: To activate the memory data gathering module issue:

```
echo "1" > /proc/sys/appldata/mem
```

To deactivate a data gathering module write “0” to the corresponding `/proc` entry.

Example: To deactivate the memory data gathering module issue:

```
echo "0" > /proc/sys/appldata/mem
```

Setting the sampling interval

You can set the time that lapses between consecutive data samples. The time you set is measured by the virtual CPU timer. Because the virtual timer slows down as the guest idles, the time sampling interval in real time can be longer than the value you set.

The sample interval in seconds is the value in `/proc/sys/appldata/interval`. The default sample interval is 60 s. To read the current value issue:

```
cat /proc/sys/appldata/interval
```

To set the sample interval to a different value write the new value (in seconds) to `/proc/sys/appldata/interval`.

Example: To set the sampling interval to 30 s issue:

```
echo "30" > /proc/sys/appldata/interval
```

APPLDATA monitor record layout

This section describes the layout of the APPLDATA monitor records that can be provided to z/VM. Each of the modules that can be installed with the base module corresponds to a type of record:

- Memory data (see Table 11 on page 258)
- Processor data (see Table 12 on page 259)
- Networking (see Table 13 on page 260)

z/VM can identify the records by their unique product ID. The product ID is a string like this: "LINUXKRNL<record ID>240100". The <record ID> is treated as a byte value, not a string. For example, for APPLDATA_MEM_DATA (see Table 11 on page 258) with a record ID of X'01' <record ID> is 1, not "01").

The records contain data of the following types:

- u32** unsigned 4 byte integer
- u64** unsigned 8 byte integer

Important

On 31-bit Linux systems, the u64 values are actually only 32-bit values. That is, the lower 32 bit wrap around like 32-bit counters and the upper 32 bit are always zero.

Table 11. APPLDATA_MEM_DATA record (Record ID X'01')

Offset		Type	Name	Description
Decimal	Hex			
0	X'0'	u64	timestamp	TOD timestamp generated on the Linux side after record update
8	X'8'	u32	sync_count_1	After VM collected the record data, sync_count_1 and sync_count_2 should be the same. Otherwise, the record has been updated on the Linux side while VM was collecting the data. As a result, the data might be inconsistent.
12	X'C'	u32	sync_count_2	
16	X'10'	u64	pgpgin	Data read from disk (in KB)
24	X'18'	u64	pgpgout	Data written to disk (in KB)
32	X'20'	u64	pswpin	Pages swapped in
40	X'28'	u64	pswpout	Pages swapped out
48	X'30'	u64	sharedram	Shared RAM in KB, currently set to 0 by Linux kernel (2.4 and 2.6)
56	X'38'	u64	totalram	Total usable main memory size in KB
64	X'40'	u64	freeram	Available memory size in KB
72	X'48'	u64	totalhigh	Total high memory size in KB
80	X'50'	u64	freehigh	Available high memory size in KB
88	X'58'	u64	bufferram	Memory reserved for buffers, free cache in KB
96	X'60'	u64	cached	Size of used cache, without buffers in KB
104	X'68'	u64	totalswap	Total swap space size in KB
112	X'70'	u64	freeswap	Free swap space in KB

Table 12. APPLDATA_OS_DATA record (Record ID X'02')

Offset		Type	Name	Description
Decimal	Hex			
0	X'0'	u64	timestamp	TOD timestamp generated on the Linux side after record update
8	X'8'	u32	sync_count_1	After VM collected the record data, sync_count_1 and sync_count_2 should be the same. Otherwise, the record has been updated on the Linux side while VM was collecting the data. As a result, the data might be inconsistent.
12	X'C'	u32	sync_count_2	
16	X'10'	u32	nr_cpus	Number of virtual CPUs
20	X'14'	u32	per_cpu_size	Size of the data struct (= 16) for each CPU
24	X'18'	u32	cpu_offset	Offset of the first CPU data struct (= 48)
28	X'1C'	u32	nr_running	Number of runnable threads
32	X'20'	u32	nr_threads	Number of threads
36	X'24'	3 × u32	avenrun[3]	Average number of running processes during the last 1 (1st value), 5 (2nd value) and 15 (3rd value) minutes. These values are "fake fix-point", each composed of 10 bits integer and 11 bits fractional part. See note 1 at the end of this table.
48	X'30'	See note 2.	per_cpu_size	Time spent in user/kernel/idle/nice mode for every CPU. See note 3 at the end of this table.
48	X'30'	u32	per_cpu_user	Timer ticks spent in user mode (CPU0)
52	X'34'	u32	per_cpu_nice	Timer ticks spent with modified priority (CPU0)
56	X'38'	u32	per_cpu_system	Timer ticks spent in kernel mode (CPU0)
60	X'3C'	u32	per_cpu_idle	Timer ticks spent in idle mode (CPU0)
Notes: 1. The following C-Macros are used inside Linux to transform these into values with 2 decimal places: <pre>#define LOAD_INT(x) ((x) >> 11) #define LOAD_FRAC(x) LOAD_INT(((x) & ((1 << 11) - 1)) * 100)</pre> 2. nr_cpus * per_cpu_data 3. per_cpu_size through per_cpu_idle are repeated for each CPU.				

Table 13. APPLDATA_NET_SUM_DATA record (Record ID X'03')

Offset		Type	Name	Description
Decimal	Hex			
0	X'0'	u64	timestamp	TOD timestamp generated on the Linux side after record update
8	X'8'	u32	sync_count_1	After VM collected the record data, sync_count_1 and sync_count_2 should be the same. Otherwise, the record has been updated on the Linux side while VM was collecting the data. As a result, the data might be inconsistent.
12	X'C'	u32	sync_count_2	
16	X'10'	u32	nr_interfaces	Number of interfaces being monitored
20	X'14'	u32	padding	Unused. The next value is 64-bit aligned, so these 4 byte would be padded out by compiler
24	X'18'	u64	rx_packets	Total packets received
32	X'20'	u64	tx_packets	Total packets transmitted
40	X'28'	u64	rx_bytes	Total bytes received
48	X'30'	u64	tx_bytes	Total bytes transmitted
56	X'38'	u64	rx_errors	Number of bad packets received
64	X'40'	u64	tx_errors	Number of packet transmit problems
72	X'48'	u64	rx_dropped	Number of incoming packets dropped because of insufficient space in Linux buffers
80	X'50'	u64	tx_dropped	Number of outgoing packets dropped because of insufficient space in Linux buffers
88	X'58'	u64	collisions	Number of collisions while transmitting

Programming interfaces

The monitor stream support base module exports two functions:

- `appldata_register_ops()` to register data gathering modules
- `appldata_unregister_ops()` to undo the registration of data gathering modules

Both functions receive a pointer to a struct `appldata_ops` as parameter. Additional data gathering modules that want to plug into the base module must provide this data structure. You can find the definition of the structure and the functions in `arch/s390/appldata/appldata.h` in the Linux source tree.

See “APPLDATA monitor record layout” on page 257 for an example of APPLDATA data records that are to be sent to z/VM.

Recommendation: include the `timestamp`, `sync_count_1`, and `sync_count_2` fields at the beginning of the record as shown for the existing APPLDATA record formats.

Chapter 22. 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.

You need a Linux kernel with VLAN and OSA-Express support to use VLAN under Linux.

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.

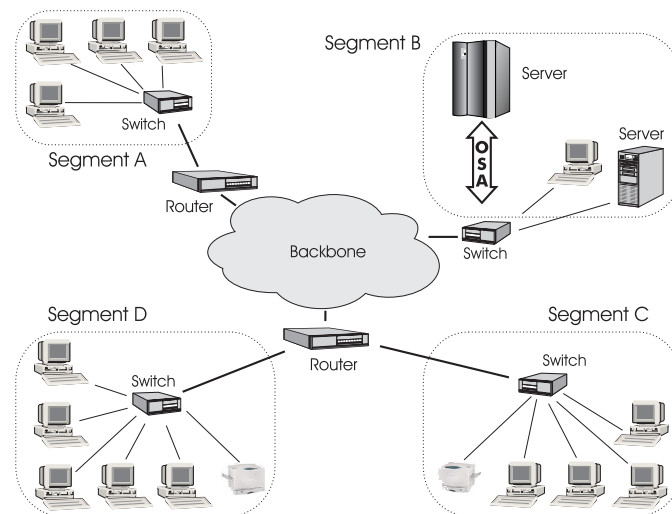


Figure 19. Conventional routed network

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.

Virtual LAN (VLAN) support

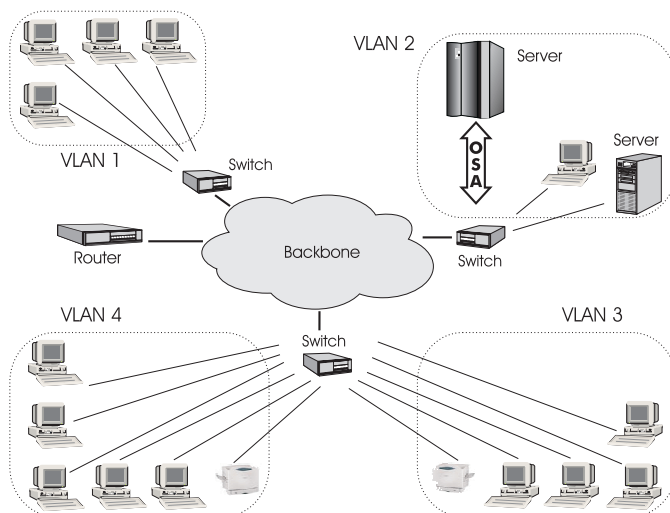


Figure 20. Switched VLAN network

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.

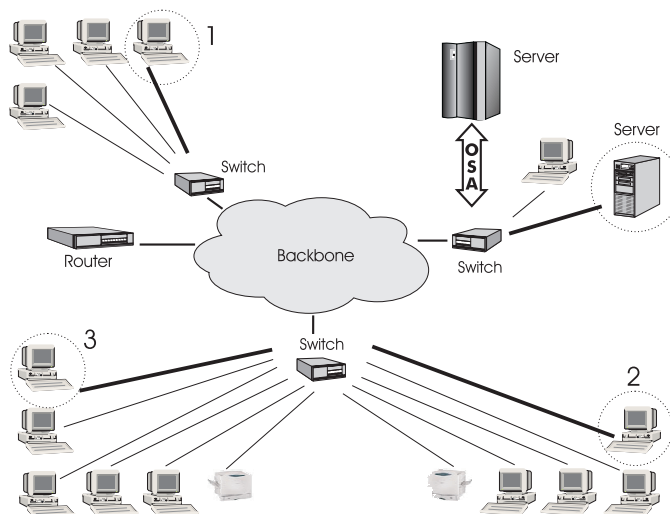


Figure 21. VLAN network organized for traffic flow

Configuring VLAN devices

VLANs are configured using the 'vconfig' command (see "vconfig - VLAN (802.1Q) configuration program" on page 234).

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

```

Virtual LAN (VLAN) support

```
(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 23. HiperSockets Network Concentrator

You cannot configure a HiperSockets Network Concentrator as described in this section if you are using the layer2 option. The information in this section assumes that you have not set the layer2 option (see “layer2 | no_layer2” on page 146).

The HiperSockets Network Concentrator connects systems to an external LAN within one IP subnet using HiperSockets. HiperSockets Network Concentrator connected systems appear as if they were directly connected to the LAN. This helps to reduce the complexity of network topologies resulting from server consolidation. HiperSockets Network Concentrator allows to migrate systems from the LAN into a zSeries Server environment, or systems connected by a different HiperSockets Network Concentrator into a zSeries Server environment, without changing the network setup. Thus, HiperSockets Network Concentrator helps to simplify network configuration and administration.

Design

A connector Linux system forwards traffic between the external OSA interface and one or more internal HiperSockets interfaces. This is done via IPv4 forwarding for unicast traffic and via a particular bridging code (xcec_bridge) for multicast traffic.

A script named ip_watcher.pl observes all IP addresses registered in the HiperSockets network and sets them as Proxy ARP entries (see “Proxy ARP” on page 154) on the OSA interfaces. The script also establishes routes for all internal systems to enable IP forwarding between the interfaces.

All unicast packets that cannot be delivered in the HiperSockets network are handed over to the connector by HiperSockets. The connector also receives all multicast packets to bridge them.

Setup

The setup principles for configuring the HiperSockets Network Concentrator on a zSeries Linux system are as follows:

leaf nodes

The leaf nodes do not require a special setup. To attach them to the HiperSockets network, their setup should be as if they were directly attached to the LAN. They do not have to be Linux systems.

connector systems

In the following, HiperSockets Network Concentrator IP refers to the subnet of the LAN that is extended into the HiperSockets net.

- The OSA interface for traffic into the LAN must be set as multicast router to support forwarding of all packet types: use the option `multicast_router` for the corresponding devices. If only unicast packages are to be forwarded, there is also the possibility not to identify the OSA interface as multicast router: add the interface name to the `start_hsync` script and only unicast packets will be forwarded.
- All HiperSockets interfaces involved must be set up as connectors: use the options `primary_connector` or `secondary_connector` for the corresponding devices. Alternatively, you can add the OSA interface

HiperSockets Network Concentrator

name to the start script as a parameter. This option results in HiperSockets Network Concentrator ignoring multicast packets, which are then not forwarded to the HiperSockets interfaces.

- IP forwarding must be enabled for the connector partition. This can be achieved either manually with the command

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

Alternatively, distribution-dependent configuration files can be used to activate IP forwarding for the connector partition automatically after booting.

- The network routes for the HiperSockets interface must be removed, a network route for the HiperSockets Network Concentrator IP subnet has to be established via the OSA interface. To achieve this, the IP address 0.0.0.0 can be assigned to the HiperSockets interface while an address used in the HiperSockets Network Concentrator IP subnet is to be assigned to the OSA interface. This sets the network routes up correctly for HiperSockets Network Concentrator.
- To *start* HiperSockets Network Concentrator, run the script `start_hsync.sh`. You can specify an interface name as optional parameter. This makes HiperSockets Network Concentrator use the specified interface to access the LAN. There is no multicast forwarding in that case.
- To *stop* HiperSockets Network Concentrator, use the command `killall ip_watcher.pl` to remove changes caused by running HiperSockets Network Concentrator.

Availability setups

If a connector system fails during operation, it can simply be restarted. If all the startup commands are executed automatically, it will instantaneously be operational again after booting. Two common availability setups are mentioned here:

One connector partition and one monitoring system

As soon as the monitoring system cannot reach the connector for a specific timeout (for example, 5 seconds), it restarts the connector. The connector itself monitors the monitoring system. If it detects (with a longer timeout than the monitoring system, for example 15 seconds) a monitor system failure, it restarts the monitoring system.

Two connector systems monitoring each other

In this setup, there is an active and a passive system. As soon as the passive system detects a failure of the active connector, it takes over operation. In order to do this it needs to reset the other system to release all OSA resources for the multicast_router operation. The failed system can then be restarted manually or automatically, depending on the configuration. The passive backup HiperSockets interface can either switch into `primary_connector` mode during the failover, or it can be setup as `secondary_connector`. A `secondary_connector` takes over the connecting functionality, as soon as there is no active `primary_connector`. This setup has a faster failover time than the first one.

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

Hints

- The MTU of the OSA and HiperSocket link should be of the same size. Otherwise multicast packets not fitting in the link's MTU are discarded as there is no IP fragmentation for multicast bridging. Warnings are printed to `/var/log/messages` or a corresponding syslog destination.
- The script `ip_watcher.pl` prints error messages to the standard error descriptor of the process.
- `xcec-bridge` logs messages and errors to syslog. On most distributions creates entries in `/var/log/messages`.
- Registering all internal addresses with the OSA card can take some seconds for each address.
- To shut down the HiperSockets Network Concentrator functionality, simply issue `killall ip_watcher.pl`. This removes all routing table and Proxy ARP entries added while using HiperSockets Network Concentrator.

Examples

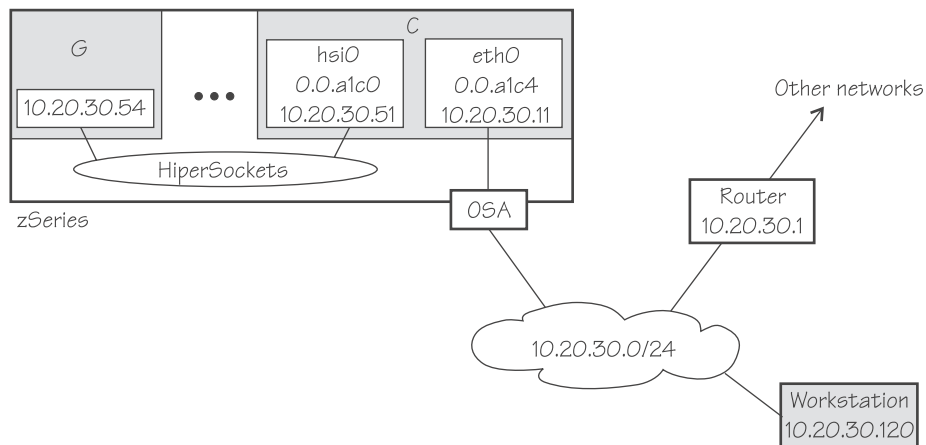


Figure 22. Example 1 for HiperSockets network concentrator setup

setup for G:

No special setup required. The HiperSockets interface has IP 10.20.30.54, and netmask is 255.255.255.0

The default gateway is 10.20.30.1

setup for C:

The HiperSockets interface has IP 10.20.30.51, and the netmask is 255.255.255.0.

The default gateway is 10.20.30.1.

In /etc/chandev.conf add:

```
add_parms, 0x10, <HiperSocket-Device-Range>, primary_connector
```

```
add_parms, 0x10, <OSA-Device-Range>, multicast_router
```

Issue `sysctl -w net.ipv4.ip_forward=1` (or make it permanent in /etc/sysctl.conf or /etc/sysconfig/sysctl depending on the distribution in use).

To start it:

```
route del -net 10.20.30.0 netmask 255.255.255.0 dev <HiperSockets interface name>
```

```
start_hsync.sh &
```

setup for workstations:

No special setup required. The network interface IP 10.20.30.120, and the netmask is 255.255.255.0.

The default gateway is 10.20.30.1

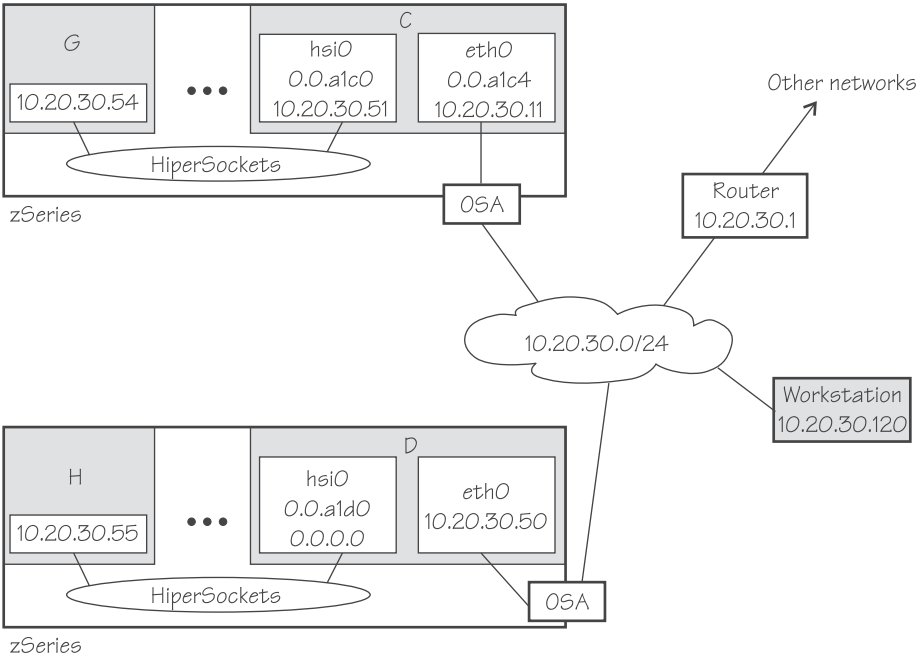


Figure 23. Example 2 for HiperSockets network concentrator setup

additional systems:

setup for H:
No special setup required.
The HiperSockets interface has IP 10.20.30.55, netmask is 255.255.255.0
The default gateway is 10.20.30.1

setup for D:
The HiperSockets interface has 0.0.0.0
The OSA interface has IP 10.20.30.50, and the netmask is 255.255.255.0
The default gateway is 10.20.30.1
in /etc/chandev.conf add:add_parms,0x10,<HiperSocket-Device-Range>,primary_connector
Issue <code>sysctl -w net.ipv4.ip_forward=1</code> (or make it permanent in /etc/sysconfig/sysctl, depending on the distribution in use)
To start it:
<code>start_hsync.sh <OSA interface name> &</code>
Note the difference between C and D: these show two slightly different ways to setup the connector systems. The second setup forwards only unicast packets.

Chapter 24. Enabling OSA-Express QDIO devices in Linux for DHCP and tcpdump

This section describes consideration when running OSA-Express QDIO devices on zSeries and S/390 in conjunction with IPv4.

In LAN environments, data packets find their destination through Media Access Control (MAC) addresses in their Logical Link Control (LLC) header (see Figure 24).

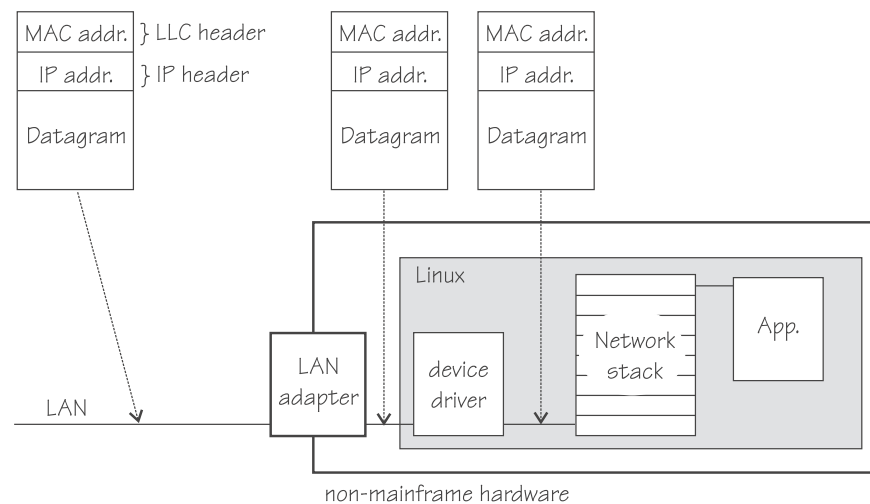


Figure 24. IPv4 processing in non-mainframe environments

By default, an OSA-Express adapter card in QDIO mode removes the LLC header from incoming packets. Packets are then passed to the network stacks of the recipient Linux instances (see Figure 25) based on the receiver's IP address (layer 3 information).

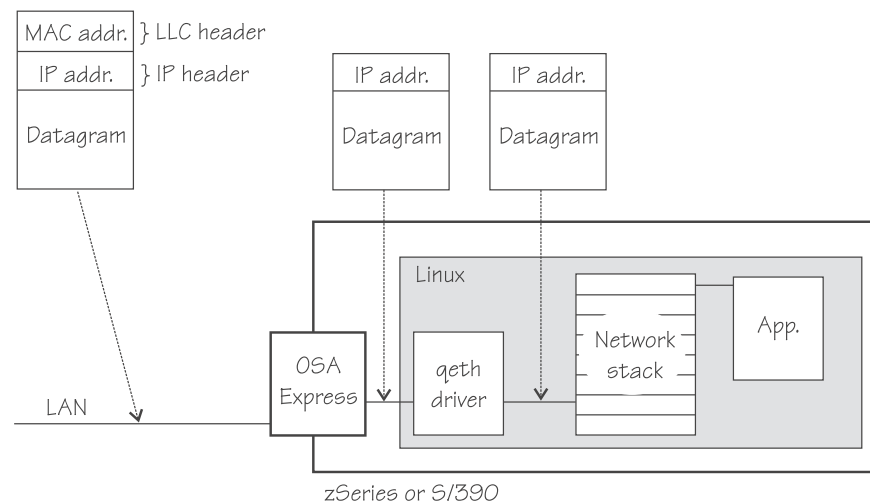


Figure 25. IPv4 processing by OSA-Express

Conversely, the OSA-Express adapter card in QDIO mode adds the LLC header to IPv4 packets with the destination MAC address for outgoing packets.

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

With IPv6, both incoming and outgoing packets are complete packets with LLC headers. The OSA-Express adapter card in QDIO mode passes complete packets to the Linux image and the driver lets the network stack compose packets with an LLC header.

See “Stateless autoconfiguration in IPv6” on page 156 for how to enable OSA-Express adapter card sharing for IPv6.

The following subsections provide some more detail on how an OSA-Express adapter card in QDIO mode handles IPv4 packets. They also describe solutions to problems that this handling for IPv4 packets can result in for:

- tcpdump
- DHCP

See “The layer2 option” on page 278 for an option that allows you to change the way IPv4 packets are handled by the OSA-Express QDIO microcode.

How the OSA-Express QDIO microcode handles IPv4 packets

This section applies to interfaces for which the layer2 option (see “The layer2 option” on page 278) is not enabled.

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

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

- ARP processing
- Handling MAC addresses and LLC headers

ARP

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

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

If there is an entry for the destination IP address, the corresponding MAC address is copied into the LLC header and the packet is added to the appropriate interface’s output queue. If the entry is not found, the ARP functions retain the IP

packet, and broadcast an ARP request asking the destination host for its MAC address. When a reply is received, the packet is sent to its destination.

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

LLC headers

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

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

Setting up for DHCP

Dynamic Host Configuration Protocol (DHCP) is a TCP/IP protocol that allows clients to obtain IP network configuration information (including an IP address) from a central DHCP server. The DHCP server controls whether the address it provides to a client is allocated permanently or is leased temporarily. DHCP specifications are described by RFC 2131 “Dynamic Host Configuration Protocol” and RFC 2132 “DHCP options and BOOTP Vendor Extensions”, which are available on the Internet at: <http://www.ietf.org/>.

Two types of DHCP environments have to be taken into account:

- DHCP via OSA-Express cards in QDIO mode
- DHCP in a z/VM guest LAN

For information on setting up DHCP for Linux for zSeries in a z/VM guest LAN environment, refer to Redpaper *Linux on IBM eServer zSeries and S/390: TCP/IP Broadcast on z/VM Guest LAN*, REDP3596 at: ibm.com/redbooks/.

There are three possibilities to get the DHCP client *dhcpcd* and server *dhcp* working properly via OSA-Express adapter cards in QDIO mode:

- Enabling the *layer2* option (see “The layer2 option” on page 278). This is the preferred method.
- Enabling the *qeth fake_ll* option (see “The fake_ll option” on page 279). This is the preferred method for environments that do not support the *layer2* option.
- Patching the DHCP client and server packages.

This section describes how to patch the DHCP client and server packages. Note that the sample patches are for illustration purposes only and not intended for you to use directly. You need to write your own patches that are suited to your environment.

The sample patches assume, that all incoming packets are missing layer 2 Ethernet header information. These sample patches might change the application behavior when handling packets that contain layer 2 Ethernet header information.

Sample DHCP patch

The following patch shows an example of how to make the DHCP server *dhcp* (version 3.0rc12) work properly on Linux for zSeries.

The sample patch is for illustration purposes only and not intended for you to use directly. You need to write your own patch that is suited to your environment.

DHCP server sample patch for dhcp 3.0rc12

```
diff -ru dhcp-3.0rc12/common/bpf.c dhcp-3.0rc12-qeth/common/bpf.c
--- dhcp-3.0rc12/common/bpf.c Mon Apr 9 06:12:49 2001
+++ dhcp-3.0rc12-qeth/common/bpf.c Wed Feb 26 23:01:10 2003
@@ -184,23 +184,19 @@
     offsets used in if_register_send to patch the BPF program! XXX */

    struct bpf_insn dhcp_bpf_filter [] = {
- /* Make sure this is an IP packet... */
- BPF_STMT (BPF_LD + BPF_H + BPF_ABS, 12),
- BPF_JUMP (BPF_JMP + BPF_JEQ + BPF_K, ETHERTYPE_IP, 0, 8),
-
- /* Make sure it's a UDP packet... */
- BPF_STMT (BPF_LD + BPF_B + BPF_ABS, 23),
+ BPF_STMT (BPF_LD + BPF_B + BPF_ABS, 9),
+ BPF_JUMP (BPF_JMP + BPF_JEQ + BPF_K, IPPROTO_UDP, 0, 6),
+
+ /* Make sure this isn't a fragment... */
- BPF_STMT (BPF_LD + BPF_H + BPF_ABS, 20),
+ BPF_STMT (BPF_LD + BPF_H + BPF_ABS, 6),
+ BPF_JUMP (BPF_JMP + BPF_JSET + BPF_K, 0x1fff, 4, 0),
+
+ /* Get the IP header length... */
- BPF_STMT (BPF_LDX + BPF_B + BPF_MSH, 14),
+ BPF_STMT (BPF_LDX + BPF_B + BPF_MSH, 0),
+
+ /* Make sure it's to the right port... */
- BPF_STMT (BPF_LD + BPF_H + BPF_IND, 16),
+ BPF_STMT (BPF_LD + BPF_H + BPF_IND, 2),
+ BPF_JUMP (BPF_JMP + BPF_JEQ + BPF_K, 67, 0, 1), /* patch */
+
+ /* If we passed all the tests, ask for the whole packet. */
diff -ru dhcp-3.0rc12/common/lpf.c dhcp-3.0rc12-qeth/common/lpf.c
--- dhcp-3.0rc12/common/lpf.c Tue Apr 24 09:36:00 2001
+++ dhcp-3.0rc12-qeth/common/lpf.c Wed Feb 26 23:19:26 2003
@@ -84,7 +84,7 @@
    /* Make an LPF socket. */
    if ((sock = socket(PF_PACKET, SOCK_PACKET,
- htons((short)ETH_P_ALL))) < 0) {
+ htons((short)ETH_P_ALL))) < 0) {
+ htons((short)ETH_P_ALL))) < 0) {
        if (errno == ENOPROTOOPT || errno == EPROTONOSUPPORT ||
            errno == ESOCKTNOSUPPORT || errno == EPFNOSUPPORT ||
            errno == EAFNOSUPPORT || errno == EINVAL) {
@@ -234,7 +234,7 @@
        /* Patch the server port into the LPF program...
        XXX changes to filter program may require changes
        to the insn number(s) used below! XXX */
- dhcp_bpf_filter [8].k = ntohs ((short)local_port);
+ dhcp_bpf_filter [6].k = ntohs ((short)local_port);

        if (setsockopt (info -> rfdesc, SOL_SOCKET, SO_ATTACH_FILTER, &p,
            sizeof p) < 0) {
@@ -349,7 +349,7 @@
        unsigned char ibuf [1536];
        unsigned bufix = 0;

- length = read (interface -> rfdesc, ibuf, sizeof ibuf);
+ length = read (interface -> rfdesc, ibuf+ETHER_HEADER_SIZE, sizeof ibuf);
        if (length <= 0)
            return length;

@@ -365,7 +365,6 @@
    }

    bufix += offset;
- length -= offset;

    /* Decode the IP and UDP headers... */
    offset = decode_udp_ip_header (interface, ibuf, bufix, from,
```

The following patch shows an example of how to make the DHCP client *dhcpcd* (version 1.3.22-pl1) work properly on Linux for zSeries.

The sample patch is for illustration purposes only and not intended for you to use directly. You need to write your own patch that is suited to your environment.

DHCP sample client patch for dhcpcd 1.3.22-pl1

```
--- client.c Wed Apr 23 21:44:47 2003
+++ client-new.c Wed Apr 23 21:54:24 2003
@@ -505,6 +505,17 @@
     gettimeofday(&current, NULL);
     timeval_subtract(&diff, &current, &begin);
     timeout = j - diff.tv_sec*1000000 - diff.tv_usec + random()%200000;
+/* start of changes here ----- */
+    if ( (len<=sizeof(udpipMessage)+sizeof(struct packed_ether_header)) &&
+        ( *(char*)&UdpIpMsgRecv.ethhdr == 0x45 ) ) {
+        memmove(((char*)&UdpIpMsgRecv)+
+        sizeof(struct packed_ether_header),
+        &UdpIpMsgRecv,len);
+        len+=sizeof(struct packed_ether_header);
+        UdpIpMsgRecv.ethhdr.ether_type = htons(ETHERTYPE_IP);
+    }
+/* end of changes here ----- */
+    if ( UdpIpMsgRecv.ethhdr.ether_type != htons(ETHERTYPE_IP) )
+        continue;
+    /* Use local copy because ipRecv is not aligned. */
```

Required options for using DHCP on Linux for zSeries and S/390

The following applies if you are using *dhcpcd* on Linux for zSeries and S/390 without the *layer2* option:

- You need to run *dhcpcd* with option **-B**.

This option instructs the DHCP server to broadcast its response to the DHCP client. Because the OSA-Express adapter card in QDIO mode forwards packets to Linux based on IP addresses, a DHCP client that requests an IP address could not receive the response from the DHCP server without this option.

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

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

There are no special required options for using *dhcp* on Linux for zSeries and S/390.

Setting up for tcpdump

tcpdump uses the packet capture library *libpcap*.

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

libpcap requires an Ethernet LLC header for the packets it captures. To make it work properly in a Linux for zSeries and S/390 environment you can do either:

- Enable the layer2 option (see “The layer2 option” on page 278).
- Make some changes in the libpcap library.

The following is a sample patch for the libpcap version 0.6.2. The patch is for illustration purposes only and not intended for you to use directly. You need to write your own patch that is suited to your environment.

The sample patche assumes, that all incoming packets are missing layer 2 Ethernet header information. The sample patch might change the application behavior when handling packets that contain layer 2 Ethernet header information.

libpcap 0.6.2. sample patch

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

The first part of the patch adds a fake LLC header to all network packets that do not have one. The second part prevents filtering of network packets in the kernel

so that packets are filtered in user mode after a fake LLC header has been added. There is no impact on other network device types, like LCS devices.

The layer2 option

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

Be aware that in conjunction with the *layer2* option, the following cannot be configured as described in the respective sections:

- Router definitions (see “Examples: OSA-Express CHPID in QDIO mode” on page 148)
- HiperSockets network concentrator (see Chapter 23, “HiperSockets Network Concentrator,” on page 265)
- IP address takeover (see “IP address takeover” on page 152)
- Proxy ARP (see “Proxy ARP” on page 154)
- VIPA (see “OSA-Express CHPID in QDIO mode – Virtual IP address (VIPA)” on page 155)
- Stateless autoconfiguration in IPv6 (see “Stateless autoconfiguration in IPv6” on page 156)

Accordingly, you cannot use the following commands if you are using the *layer2* option:

- `qetharp` (see “qetharp - Query and purge OSA and HiperSockets ARP data” on page 197)
- `qethconf` (see “qethconf - configure qeth devices” on page 199)

See also “*layer2* | *no_layer2*” on page 146.

If your environment does not fulfill the requirements, or you want to use a feature that you cannot use in conjunction with the *layer2* option, consider using the *fake_ll* option instead (see “The *fake_ll* option” on page 279).

The *layer2* option stops the OSA-Express adapter from stripping the MAC addresses from incoming packets. Incoming and outgoing packets are complete with an LLC header at all stages between the Linux network stack and the LAN as shown in Figure 24 on page 271. This layer 2 based forwarding requires unique MAC addresses for all concerned Linux instances.

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

For Linux instances that are directly attached to an OSA-Express adapter card in QDIO mode, you need to assign the MAC addresses yourself. Consult your distribution documentation on how to assign a MAC address. Be sure not to assign the MAC address of the OSA-Express adapter card to your Linux instance. See Chapter 4 “Planning for Guest LANs and Virtual Switches” in *z/VM: Connectivity*, SC24-6080, for further information on configuring MAC addresses.

Using layer2

Use the channel device layer **add_parms** command to enable *layer2*. For example, include a line like this in `/etc/chandev.conf` to include fake LLC headers in all packets received by `eth0`:

```
qeth0,0x500,0x501,0x502;add_parms,0x10,0x500,0x502,layer2
```


The fake_ll option

If your environment does not support the *layer2* option (see “The layer2 option” on page 278), use the *fake_ll* option to be able to use network programs that require incoming packets with LLC headers.

What fake_ll does

The *fake_ll* option instructs qeth to insert a fake LLC header in all incoming packets. The packets are then passed to the Linux network stack and finally to the recipient programs or applications.

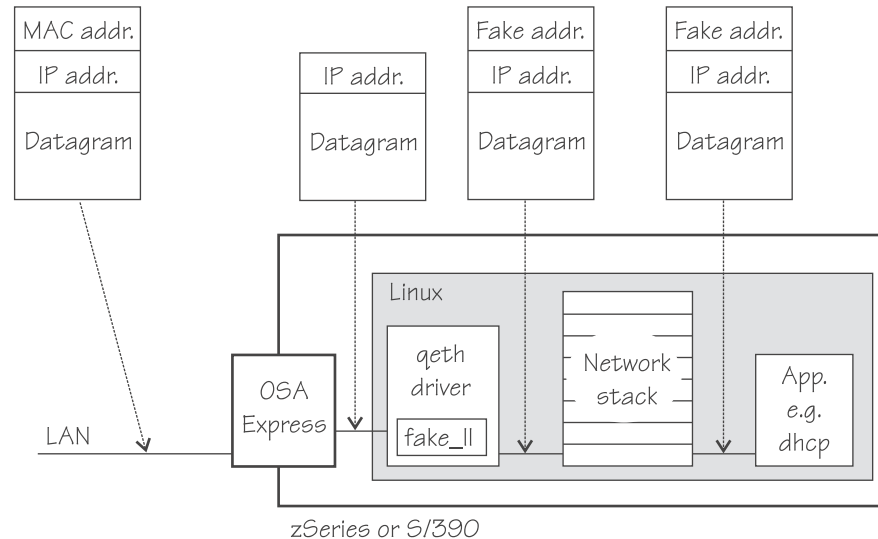


Figure 26. qeth with *fake_ll* option for incoming packets

Network programs that expect incoming packets to have a LLC header can then be used as usual, without any patches. Examples for programs that expect an LLC header are:

- The DHCP server program *dhcp*
- The DHCP client program *dhcpcd*
- libpcap based applications like *tcpdump*

Shortcomings of fake_ll

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

Because *fake_ll* is a qeth option (see Figure 26), it also cannot supply fake LLC headers for programs that intercept outgoing packets before they have reached the qeth driver.

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

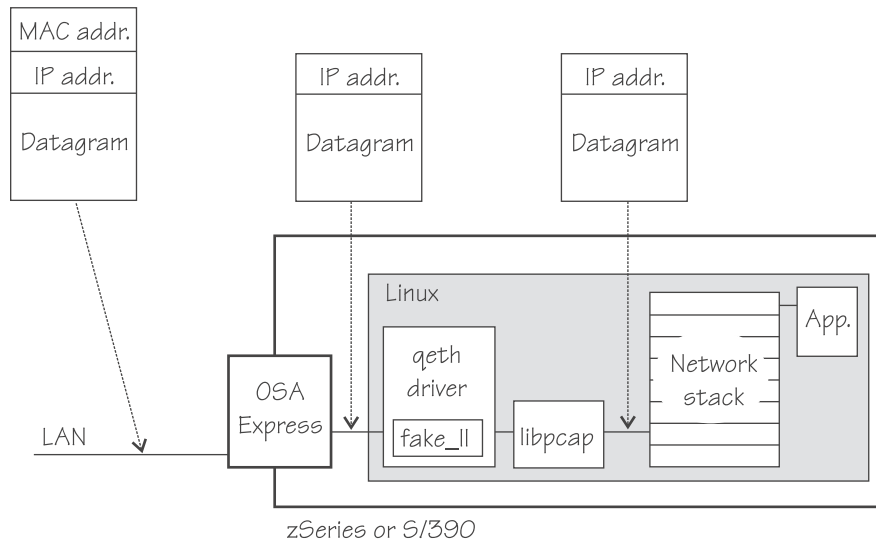


Figure 27. qeth with fake_ll option for outgoing packets

An example of a program that expects LLC headers in outgoing packets is:

- *libpcap* a program that captures outgoing packets for *tcpdump*

See “The layer2 option” on page 278 or “Setting up for tcpdump” on page 276 for information on how to make tcpdump work correctly for IPv4.

Using fake_ll

Use the channel device layer **add_parms** command to enable *fake_ll*. For example, include a line like this in `/etc/chandev.conf` to include fake LLC headers in all packets received by `eth0`:

```
qeth0,0x500,0x501,0x502;add_parms,0x10,0x500,0x502,fake_ll
```

DHCP and tcpdump for HiperSockets devices

The HiperSockets network connectivity was designed to be very similar to connectivity by the OSA-Express adapter card in QDIO mode. Therefore, most of the information of Chapter 24, “Enabling OSA-Express QDIO devices in Linux for DHCP and tcpdump,” on page 271 also applies to HiperSockets.

For connections through an OSA-Express card in QDIO mode, the OSA-Express card off-loads ARP, LLC header, and MAC address handling. Because a HiperSockets connection does not go out on a physical network there are no ARP, LLC headers, and MAC addresses for packets in a HiperSockets LAN.

The resulting problems for DHCP and tcpdump are the same in both cases. The *layer2* option is not applicable to HiperSockets. The other fixes for connections through the OSA-Express card also be used for HiperSockets.

Chapter 25. Selected kernel parameters

There are two different ways of passing parameters to Linux:

- Passing parameters to your kernel at startup time. (The parameter line)
- Configuring your boot loader to always pass those parameters.

The kernel can only handle a parameter line file that is no larger than 896 bytes.

The parameters which affect Linux for zSeries and S/390 in particular are:

- `cio_ignore`
- `cio_msg`
- `cio_notoper_msg`
- `ipldelay`
- `maxcpus`
- `mem`
- `noinitrd`
- `ramdisk_size`
- `ro`
- `root`
- `use_pfix`
- `vmhalt`
- `vmpoff`

cio_ignore

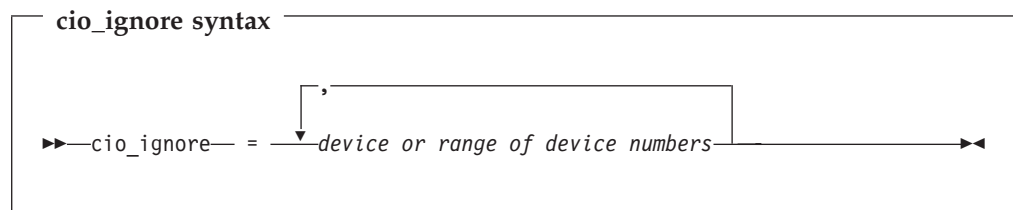
Usage

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 in hexadecimal.

Devices that are not known already can later be ignored, or the "ignored" state for known devices reset, by piping commands to the `/proc/cio_ignore` interface. Refer to examples section for further information.

Format



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
```

Listed here are examples of piping commands to `/proc/cio_ignore`.

```
cat /proc/cio_ignore
```

will list the ranges of device numbers which are ignored by common I/O.

You can also use this method to un-ignore certain or all devices.

```
free all
```

will un-ignore all ignored devices.

```
free <devnorange>,<devnorange>, ...
```

will un-ignore the specified devices.

If devices 23 to 42 and 4711 are to be ignored,

```
echo free 0x30-0x32 > /proc/cio_ignore
```

will un-ignore devices 30 to 32 and will leave devices 23 to 2F, 33 to 42 and 4711 ignored.

```
echo free 0x41 > /proc/cio_ignore
```

will furthermore un-ignore device 41.

```
echo free all > /proc/cio_ignore
```

will un-ignore all remaining ignored devices.

When a device is un-ignored, device recognition and sensing is performed and the device driver will be notified if possible, so the device will become available to the system. You can also add ranges of devices to be ignored by piping to `/proc/cio_ignore`.

```
add <devnorange>, <devnorange>, ...
```

will ignore the specified devices.

Note: Devices that are known already cannot be ignored; this also applies to devices which are gone after a machine check.

For example, if device `abcd` is known already and all other devices `a000-afff` are not known,

```
echo add 0xa000-0xaccc, 0xaf00-0xffff > /proc/cio_ignore
```

will add `af00-afff` to the list of ignored devices and skip `a000-accc`.

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
```

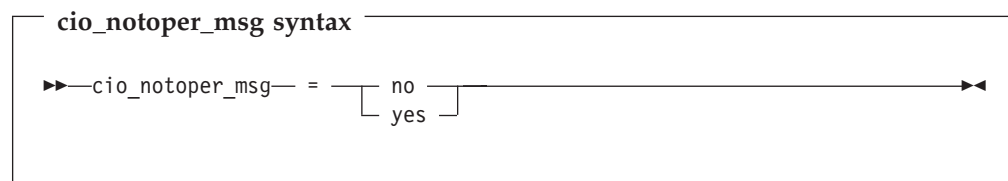
cio_notoper_msg

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



Examples

This example suppresses 'not operational' messages on startup:

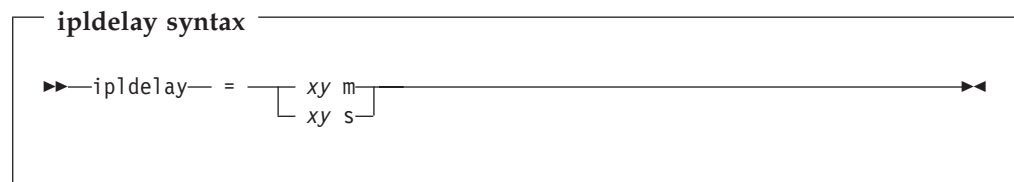
```
cio_notoper_msg=no
```

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



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

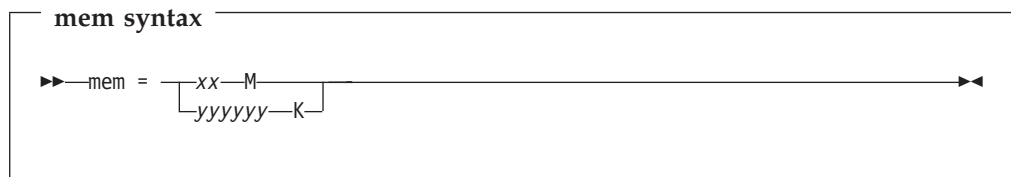
maxcpus=2

mem

Usage

Restricts memory usage to the size specified. This must be used to overcome initialization problems on a P/390.

Format



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
```

Note: A RAM disk cannot be specified using the devfs syntax (/dev/rd/0) but rather only as /dev/ram0.

use_pfix

Usage

Attention

Security restriction: granting the DIAG98 option to a guest can compromise privacy and integrity of the entire VM system. Anyone who gains root access to such a guest could read or alter arbitrary pages in the entire LPAR. Do not use the use_pfix parameter, unless you have understood this security exposure and feel confident about it.

This parameter is applicable only if Linux runs as a VM guest under z/VM in 31-bit mode. The use_pfix parameter does not apply to z/VM in 64-bit mode.

The option fixes all Linux pages in the VM memory. As a result, VM does not have to do any address translation and QDIO transactions can be handled with significantly less overhead. With the option enabled, all qeth-queues use real storage channel programs instead of translated channel programs. use_pfix does not affect zfcps queues.

Before you start: In the VM directory, DIAG98 must be specified for the guest.

Recommendation: Fixing a large number of pages can have an adverse effect on overall VM performance. Only use fix pages for Linux router guests that have a small amount of memory (not more than 64 MB).

Format

use_pfix syntax

►►—use_pfix—◄◄

vmhalt

Usage

Specifies a command to be issued to CP after a system halt. This command is only applicable if the system runs as a VM guest.

Format

vmhalt syntax

►—vmhalt— = —*COMMAND*—◄◄

Examples

This example specifies that an initial program load of CMS should follow the Linux "halt" command:

```
vmhalt="I CMS"
```

Note: The command must be entered in uppercase.

vmpoff

Usage

Specifies a command to be issued to CP after a system power off. This command is only applicable if the system runs as a VM guest.

Format

vmpoff syntax

► `vmpoff` = `—COMMAND—` ◄

Examples

This example specifies that CP should clear the guest machine after the Linux "power off" or "halt -p" command:

```
vmpoff="SYSTEM CLEAR"
```

Note: The command must be entered in uppercase.

Selected kernel parameters

Chapter 26. 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.

Comments

The parameters allowed in the parameter line are described in Chapter 25, “Selected kernel parameters,” on page 281 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 <code>zip1</code> (option <code>-p</code>)
Tape	Second file on tape
VM reader	Second file in reader
CD-ROM	Third entry in the <code>.ins</code> 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. Building the kernel

Introduction	299	Compiling the kernel	301
Preliminary steps	300	Installing the modules	302
Configuring the parameters	300	Finishing up.	302
Checking the configuration.	301	Kernel configuration options	303
Checking the dependencies.	301		

Introduction

The following instructions apply to the "June 2003 stream" on developerWorks.

Your build system must have the following software installed (as a minimum):

- kernel source 2.4.21 (depending on your base distribution) with the Linux for zSeries and S/390 patch
- gcc version 3.2 or later supporting Linux for zSeries and S/390
- binutils version 2.12.90.0.15 or later supporting Linux for zSeries and S/390
- glibc 2.2.5 or later supporting Linux for zSeries and S/390
- 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 and S/390 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.

1. "Preliminary steps" on page 300
2. "Configuring the parameters" on page 300
3. "Checking the configuration" on page 301
4. "Checking the dependencies" on page 301
5. "Compiling the kernel" on page 301
6. "Installing the modules" on page 302
7. "Finishing up" on page 302

Note: The object-code-only (OCO) device driver, `tape_3590.o`, is compiled without version information. You may encounter problems, if you switch on *Loadable module support* -> *Set version information on all module symbols*.

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.

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 and S/390 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 the bash shell. 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. `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` — X Window based configuration tool.

Notes on `make config`:

- Keeping unnecessary drivers in the Linux for zSeries and S/390 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 and S/390 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 `Documentation/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 and S/390 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 and S/390 kernel image. This compiles the source code and leaves the kernel image in the current directory

Building the kernel

(usually `/usr/src/linux/arch/s390/boot` for 31-bit or `/usr/src/linux/arch/s390x/boot/image` for 64-bit).

Note that `make zImage` and `make bzImage` are not supported by the Linux for zSeries and S/390 kernel.

Compiling for IPL from tape

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 and S/390 kernel are met.

Installing the modules

If you configured any of the parts of the Linux for zSeries and S/390 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.x (indicating 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 and S/390 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` for 31-bit or `/usr/src/linux/arch/s390x/boot/image` for 64-bit after compilation) to the place where your regular bootable kernel is located and then run `ZIPL`.

Kernel configuration options

The following table summarizes the configuration options pertaining specifically to the Linux for zSeries and S/390 environment or representing special prerequisites for these options.

Table 14. Kernel configuration options specific to the Linux for zSeries and S/390 environment

"menuconfig" menu item	Configuration option	Module name if "M" selected	Description
Submenu: Processor type and features			
Kernel support for 31-bit emulation	CONFIG_S390_SUPPORT		Select this option if you want to enable your system kernel to handle system-calls from ELF binaries for 31 bit ESA. This option (and some other stuff like libraries and such) is needed for executing 31 bit applications. It is safe to say "Y". Note: This option is available only when building a 64-bit kernel.
Kernel support for 31-bit ELF binaries	CONFIG_BINFMT_ELF32		This allows you to run 32-bit Linux/ELF binaries on your machine. "Y" is recommended. Note: This option is available only when building a 64-bit kernel.
IEEE FPU emulation	CONFIG_MTHEMU		This option is required for IEEE compliant floating point arithmetic on the Alpha. The only time you would ever not say "Y" is to say "M" in order to debug the code. Say "Y" unless you know what you are doing.

Table 14. Kernel configuration options specific to the Linux for zSeries and S/390 environment (continued)

"menuconfig" menu item	Configuration option	Module name if "M" selected	Description
Submenu: General setup			
Fast IRQ handling	CONFIG_FAST_IRQ		Select this option in order to have interrupts processed faster on your S/390 or zSeries machine. If selected, after an interrupt is processed, the channel subsystem will be asked for other pending interrupts, which will also be processed before leaving the interrupt context. This speeds up I/O considerably. "Y" is recommended.
Process warning machine checks	CONFIG_MACHCHK_WARNING		Select this option if you want the machine check handler on IBM S390 or zSeries to process warning machine checks (such as on power failures). If unsure, say "Y".
Use chscs for Common I/O	CONFIG_CHSC		Select this option if you want the common I/O layer to use information obtained by channel subsystem calls. This will enable Linux to process link failures and resource accessibility events. Moreover, if you have procs enabled, you will be able to toggle CHPIDs logically offline and online. If unsure, say "Y".
QDIO support	CONFIG_QDIO	qdio.o	This driver provides Queued Direct I/O base support for the IBM S/390 (G5 and G6) and eServer™ zSeries (z800, z900, and z990). For details, refer to the documentation provided by IBM at ibm.com/developerworks/linux/linux390 This driver is also available as a module (code which can be inserted in and removed from the running kernel whenever you want). If you want to compile it as a module, say "M" here and read file <code>Documentation/modules.txt</code> . If unsure, say "Y".
Performance statistics in /proc	CONFIG_QDIO_PERF_STATS		Say "Y" to obtain performance statistics in <code>/proc/qdio_perf</code> . If unsure, say "N".
Built-in IPL record support	CONFIG_IPL		If you want to use the produced kernel to IPL directly from a device, you have to merge a bootsector specific to the device into the first bytes of the kernel. You will have to select the IPL device on another question, that pops up, when you select <code>CONFIG_IPL</code> .

Table 14. Kernel configuration options specific to the Linux for zSeries and S/390 environment (continued)

"menuconfig" menu item	Configuration option	Module name if "M" selected	Description
IPL method generated into head.S	CONFIG_IPL_VM		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.
Pseudo page fault support	CONFIG_PFAULT		Select this option if you want to use PFAULT pseudo page fault handling under VM. If running native or in an LPAR, this option has no effect. If your VM does not support PFAULT, PAGEEX pseudo page fault handling will be used. VM 4.2 supports PFAULT but has a bug in its implementation that causes some problems. This option should be selected by anyone wanting to run Linux under VM other than VM4.2.
VM shared kernel support	CONFIG_SHARED_KERNEL		Select this option if you want to share the text segment of the Linux kernel among different VM guests. This reduces memory usage with a large number of guests, but it greatly increases kernel size. You should only select this option if you are confident in your ability and want to exploit this feature.
Virtual CPU timer support	CONFIG_VIRT_TIMER		This provides a kernel interface for virtual CPU timers.
Linux - VM Monitor Stream, base infrastructure	CONFIG_APPLDATA_BASE		This option provides a kernel interface for creating and updating z/VM APPLDATA monitor records. The monitor records are updated at given time intervals, once the timer is started.
Monitor memory management statistics	CONFIG_APPLDATA_MEM	appldata_mem.o	See "zfcpx HBA API (FC-HBA) support" on page 83 for details. This option is subordinate to CONFIG_APPLDATA_BASE. It provides memory management related data to the Linux - VM Monitor Stream, for example, the paging/swapping rate and the utilisation.
Monitor OS statistics	CONFIG_APPLDATA_OS	appldata_os.o	This option is subordinate to CONFIG_APPLDATA_BASE. It provides operating system related data to the Linux - VM Monitor Stream, for example, the CPU utilisation.
Monitor overall network statistics	CONFIG_APPLDATA_NET_SUM	appldata_net_sum.o	This option is subordinate to CONFIG_APPLDATA_BASE. It provides network related data to the Linux - VM Monitor Stream. The data gives a total sum of network I/O statistics, no per-interface data.

Table 14. Kernel configuration options specific to the Linux for zSeries and S/390 environment (continued)

"menuconfig" menu item	Configuration option	Module name if "M" selected	Description
Submenu: SCSI support / SCSI low-level drivers			
FCP host bus adapter driver for IBM z800, z900, z990 (GA2)	CONFIG_ZFCP	zfcp.o	This driver supports the IBM eServer zSeries 800/900 FCP adapter. If you want to access SCSI devices attached to your zSeries by means of Fibre Channel interfaces, enter "Y". For details, refer to the documentation provided by IBM at ibm.com/developerworks/linux/1inux390/index.shtml. A prerequisite for this driver is QDIO (see CONFIG_QDIO) and SCSI support. This driver is also available as a module (code which can be inserted in and removed from the running kernel whenever you want). If you want to compile it as a module, say "M" here and read file <code>Documentation/modules.txt</code>. The kernel level must be at least 2.4.17.
HBA API support for FCP host bus adapter driver for IBM z990 (GA2)	CONFIG_ZFCP_HBAAPI	zfcp_hbaapi.o	This option is subordinate to "FCP host bus adapter driver for IBM z800, z900, z990 (GA2)". Say Y here to include HBA API (FC-HBA) support for z990 (GA2). This support is also available as a separate module. If you want to compile it as a module, say M here and read file: <code>Documentation/modules.txt</code>. If unsure, say N.
Submenu: Block device drivers			
XPRAM disk support	CONFIG_BLK_DEV_XPRAM	xpram.o	Select this option if you want to use your expanded storage on S/390 or zSeries as a disk. This is useful as a <i>fast swap device</i> if you want to access more than 2G of memory when running in 31 bit mode. This option is also available as a module which will be called <code>xpram.o</code>. If unsure, say "N". See "XPRAM kernel parameter syntax" on page 30 for more information.

Table 14. Kernel configuration options specific to the Linux for zSeries and S/390 environment (continued)

"menuconfig" menu item	Configuration option	Module name if "M" selected	Description
z/VM discontinuous saved segments (DCSS) block device driver	CONFIG_DCSSBLK	dcssblk.o	A block device driver for DCSS segments. It can be used to create a file system in such a segment. See Chapter 4, "z/VM discontinuous saved segments device driver," on page 33 for more information.
Support for DASD devices	CONFIG_DASD	dasd_mod.o	Enable this option if you want to access DASDs directly utilizing S/390's or zSeries' channel subsystem commands. This is necessary for running natively on a single image or an LPAR.
Support for ECKD disks	CONFIG_DASD_ECKD	dasd_eckd_mod.o	ECKD (Extended Count Key Data) devices are the most commonly used devices on zSeries and S/390. You should enable this option unless you are certain that you have no ECKD devices.
Automatic activation of ECKD module	CONFIG_DASD_AUTO_ECKD		This option enables demand loading of the ECKD module. You should say "Y" if you selected CONFIG_DASD_ECKD as a module.
Support for FBA disks	CONFIG_DASD_FBA	dasd_fba_mod.o	Select this option if you want to use FBA (fixed block) devices. If you are not sure, say "Y".
Automatic activation of FBA module	CONFIG_DASD_AUTO_FBA		This option enables demand loading of the FBA module. You should say "Y" if you selected CONFIG_DASD_FBA as a module.
Support for Channel Measurement on DASD devices	CONFIG_S390_CMF	cmf.o dasd_cmf.o	This option is subordinate to "Support for DASD devices". Select this option if you want to run applications that read statistical data about DASD I/O from the Channel Measurement Facility. If you say "M" here, two modules, "dasd_cmb.o" and "cmf.o" will be created. If unsure, say "N". See "DASD Channel Measurement Facility" on page 25 for more details.
Support for DIAG access to CMS reserved disks	CONFIG_DASD_DIAG	dasd_diag_mod.o	Select this option if you want to use CMS reserved disks under VM with the Diagnose250 command. If you are not running under VM, or are uncertain, say "N". Note: This option is available only when building a 31-bit kernel.

Table 14. Kernel configuration options specific to the Linux for zSeries and S/390 environment (continued)

"menuconfig" menu item	Configuration option	Module name if "M" selected	Description
Automatic activation of DIAG module	CONFIG_DASD_AUTO_DIAG		This option enables demand loading of the DIAG module. If you selected CONFIG_DASD_DIAG as a module, you should say "Y". Note: This option is available only when building a 31-bit kernel.
Submenu: Character device drivers			
Support for locally attached 3270 tubes	CONFIG_TN3270		Include support for IBM 3270 line-mode terminals. For more information, see Chapter 5, "Console device drivers," on page 39.
Support for console on 3270 line mode terminal	CONFIG_TN3270_CONSOLE		This option is subordinate to CONFIG_TN3215. It includes support for using an IBM 3270 line-mode terminal as a Linux system console.
Support for 3215 line mode terminal	CONFIG_TN3215		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 and S/390 environment, so the 3215 driver can only be used under VM. On a native zSeries and S/390 system, the initialization function of the 3215 driver returns without registering the driver to the system. Entering ("Y") for this option switches on the compilation of parts 1 and 2 of the 3215 terminal driver. The option makes it possible to use the option CONFIG_TN3215_CONSOLE. For more information, see Chapter 5, "Console device drivers," on page 39.
Support for console on 3215 line mode terminal	CONFIG_TN3215_CONSOLE		This option is subordinate to CONFIG_TN3215. It 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.

Table 14. Kernel configuration options specific to the Linux for zSeries and S/390 environment (continued)

"menuconfig" menu item	Configuration option	Module name if "M" selected	Description
Support for SCLP	CONFIG_SCLP		Include support for the IBM SCLP interface to the service element. The hardware console (SCLP) is an alternative terminal, usually required for a native Linux for zSeries or Linux for S/390 installation although it is also run under VM. You would normally enter ("Y") for this option in a native or LPAR installation if your hardware configuration includes a hardware console. In a VM installation, without a hardware console, you would normally enter ("N"). Note: This option is required for the "signal quiesce" capability, which enables Linux for zSeries and S/390 to shut down on request from the zSeries machine or z/VM. See Chapter 5, "Console device drivers," on page 39 for more information.
Support for SCLP line mode terminal	CONFIG_SCLP_TTY		This option is subordinate to CONFIG_SCLP and required if you want to use an SCLP line-mode terminal as the Linux system console.
Console on SCLP line mode terminal	CONFIG_SCLP_CONSOLE		This option is subordinate to CONFIG_SCLP_TTY. It prints kernel errors and kernel warnings to the hardware console in addition to the normal output on the TTY device.
Support for SCLP VT220-compatible terminal	CONFIG_SCLP_VT220_TTY		This option is subordinate to CONFIG_SCLP and required if you want to use an SCLP VT220-compatible terminal in full-screen mode as the Linux system console.
Support for console on SCLP VT220-compatible terminal	CONFIG_SCLP_VT220_CONSOLE		This option is subordinate to CONFIG_SCLP_VT220_TTY. It prints kernel errors and kernel warnings to the full-screen hardware console in addition to the normal output on the TTY device.
Control Program Identification	CONFIG_SCLP_CPI	sc1p_cpi.o	Allows for Control Program Identification via the SCLP interface, i.e., provides a means to pass an OS instance name (system name) to the machine. This option should only be selected as a module, since the system name must be passed as a module parameter.

Table 14. Kernel configuration options specific to the Linux for zSeries and S/390 environment (continued)

"menuconfig" menu item	Configuration option	Module name if "M" selected	Description
S/390 tape device support	CONFIG_S390_TAPE	tape390.o	Select this option if you want to access channel-attached tape devices on IBM S/390 or zSeries. If you select this option you will also need to select at least one tape discipline (hardware) option in order to access a tape device. This option is also available as a module, which will include all selected interfaces. The tape disciplines will be separate modules in this case. For more information, see Chapter 6, "Channel-attached tape device driver," on page 49. If you are not certain, say "Y".
Support for tape block devices	CONFIG_S390_TAPE_BLOCK	tape390.o	Select this option if you want to access your channel-attached tape devices using the block device interface. This interface is similar to CD-ROM devices on other platforms. The tapes can only be accessed read-only when using this interface. Refer to file <code>Documentation/s390/TAPE</code> for further information about creating volumes for and using this interface. It is safe to say "Y" here. If <code>CONFIG_S390_TAPE</code> is selected as a module, the block-device front-end will be built into "tape390.o". For more information, see Chapter 6, "Channel-attached tape device driver," on page 49.
Support for 3480/3490 tape hardware	CONFIG_S390_TAPE_34XX	tape_34xx.o	Select this option if you want to access IBM 3480 or 3490 magnetic tape subsystems and 100% compatibles. This option is also available as a module. If <code>CONFIG_S390_TAPE</code> is selected as a module, this discipline will also be a module. It is safe to say "Y" here. For more information, see Chapter 6, "Channel-attached tape device driver," on page 49.

Table 14. Kernel configuration options specific to the Linux for zSeries and S/390 environment (continued)

"menuconfig" menu item	Configuration option	Module name if "M" selected	Description
Support for the z/VM recording system services	CONFIG_VMLOGRDR	vmlogrdr.o	Select this option if you want to be able to receive records collected by the z/VM recording system services, eg. from *LOGREC. This option should be build as a module since the actual service to connect to has to be specified at module load time. This driver depends on the IUCV support driver. See Chapter 9, "z/VM recording device driver," on page 89 for details.
Submenu: Network device drivers			
Dummy net driver support	CONFIG_DUMMY		This option is required to support the VIPA functionality (see Chapter 18, "VIPA – minimize outage due to adapter failure," on page 237).
Ethernet (10 or 100Mbit)	CONFIG_NET_ETHERNET		If you want to use OSA-Express or HiperSockets connections under Linux, enter ("Y") here. See Chapter 15, "QETH device driver for OSA-Express (QDIO) and HiperSockets," on page 139 for more information.
Token Ring driver support	CONFIG_TR		To participate on a Token Ring network, you need a special Token Ring network card. If you are connected to such a Token Ring network and want to use your Token Ring card under Linux, say "Y" here and to the driver for your particular card below and read the Token-Ring mini-HOWTO, available from http://www.tldp.org/docs/html/howto .
Channel Device Configuration	CONFIG_CHANDEV		This option enables the Channel Device Layer. The currently supported devices are: 1. LCS 2. CTC/ESCON 3. CTCMPC 4. QETH 5. OSAD 6. CLAW See Chapter 10, "Channel device layer," on page 93 for more information.

Table 14. Kernel configuration options specific to the Linux for zSeries and S/390 environment (continued)

"menuconfig" menu item	Configuration option	Module name if "M" selected	Description
LAN Channel Station interface	CONFIG_LCS	lcs.o	Select this option if you want to use LCS networking on IBM S/390 or zSeries. This device driver supports Token Ring (IEEE 802.5), FDDI (IEEE 802.7), and Ethernet. It will use the channel device configuration if this is available. If you are not familiar with this option, it is safe to say "Y". For more information, see Chapter 14, "LCS device driver," on page 137.
Support for Gigabit Ethernet	CONFIG_QETH	qeth.o	This driver supports the IBM S/390 and zSeries OSA-Express adapters in QDIO mode (all media), HiperSockets interfaces and VM Guest LAN interfaces in QDIO and HIPER mode. CONFIG_NET_ETHERNET, CONFIG_TR, CONFIG_IPV6, CONFIG_SHARED_IPV6_CARDS and CONFIG_VLAN_8021Q should all be selected when compiling this. For details refer to: ibm.com/developerworks/linux/linux390/index.shtml If you want to compile this driver as a module, say 'M' here and read file Documentation/modules.txt.
IPv6 support for qeth	CONFIG_QETH_IPV6		If CONFIG_QETH is selected, this option includes IPv6 support in the qeth device driver.
IEEE 802.1q VLAN support for qeth	CONFIG_QETH_VLAN		If CONFIG_QETH is selected, this option includes IEEE 802.1q VLAN support in the qeth device driver.
Performance statistics in /proc	CONFIG_QETH_PERF_STATS		If CONFIG_QETH is selected, this option adds a file /proc/qeth_perf_stats in the proc-fs containing performance statistics. It may slightly impact performance, so this is only recommended for internal tuning of the device driver.
CTC device support	CONFIG_CTC	ctc.o	Select this option if you want to use channel-to-channel networking on IBM S/390 or zSeries. This device driver supports real CTC coupling using ESCON. It also supports virtual CTCs when running under VM. It will use the channel device configuration if this is available. If in doubt, it is safe to say "Y". See Chapter 11, "CTC/ESCON device driver," on page 105 for more information.

Table 14. Kernel configuration options specific to the Linux for zSeries and S/390 environment (continued)

"menuconfig" menu item	Configuration option	Module name if "M" selected	Description
CTCMPC device support	CONFIG_MPC	ctcmpc.o	Select this option if you want to use Communications Server Linux over Multi-Path Channel MPC connections. These MPC connections can be to VM/VTAM, VSE/VTAM, MVS/VTAM or CS/390 and the ctcmpc driver can use virtual CTCs under VM or real channels between processors. See Chapter 12, "CTCMPC device driver," on page 113 for more information.
IUCV device support	CONFIG_IUCV	iucv.o	This is a VM-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 and S/390 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. For more information, see "IUCV Application Programming Interface (API)" on page 119.
IUCV network device support	CONFIG_NETIUCV	netiucv.o	This option is subordinate to CONFIG_IUCV. This is a VM-only device driver based on the Inter-User Communications Vehicle of VM. Enter "Y" to enable a fast communication link between VM guests. Using ifconfig a point-to-point connection can be established to the Linux for zSeries and S/390 system running on the other VM guest. For more information, see Chapter 13, "IUCV device driver," on page 115.
IUCV special message support	CONFIG_MSGIUCV	msgiucv.o	This option is subordinate to CONFIG_IUCV. This is a VM-only special message support based on IUCV. Select this option if you want to be able to receive MSG messages from other VM guest systems.
Submenu: Networking options			

Table 14. Kernel configuration options specific to the Linux for zSeries and S/390 environment (continued)

"menuconfig" menu item	Configuration option	Module name if "M" selected	Description
Prepare net_device struct for shared IPv6 cards	CONFIG_SHARED_IPV6_CARDS		This option is required to facilitate IPv6 stateless address generation on the OSA-Express feature using QDIO. If it is selected, IPv6 stateless autoconfigured addresses are generated on the same OSA-Express feature by different guest images. For more information, see "Stateless autoconfiguration in IPv6" on page 156. Note: This option is available only when building a 64-bit kernel.
IBM disk label and partition support	CONFIG_IBM_PARTITION		You must say "Y" here if you want to be able to read volume labels of IBM DASD disks. These can be ECKD DASD disks with compatible disk layout (cdl) or standard Linux disk layout (ldl), FBA DASD disks, and CMS reserved minidisks. If you say "N", you will not be able to access these disks. For more information, see "DASD partitioning" on page 14. Note: This option is presented only if you select "Advanced partition selection".
Submenu: Kernel hacking			
Magic System Request Key support	CONFIG_MAGIC_SYSRQ		If you say Y here, you will have some control over the system even if the system crashes for example during kernel debugging. For example, you will be able to flush the buffer cache to disk, reboot the system immediately or dump some status information. See also "Magic sysrequest function" on page 44.

Building the kernel

Appendix B. Hotplug support

Linux on S/390 and zSeries supports hotplug for:

- DASD devices
- Channel attached tape devices
- Cryptographic devices
- z/VM recording

The purpose of this support is the dynamic creation of device nodes on non-devfs systems.

To use the kernel hotplug support you need to install the hotplug package from <http://linux-hotplug.sourceforge.net> and the corresponding hotplug agents need to be present. These agents are not provided by IBM.

DASD hotplug events

For every hotplug event generated by the DASD device driver, the following command line parameters are provided to `/sbin/hotplug`:

Only one command line parameter is provided. It is always “dasd”.

The following environment variables are set:

DEVNO=<DEVNO>

<DEVNO> is the 4 digit hexadecimal device number of the device.
Lowercase characters are used to represent values a-f.

MAJOR=<MAJOR>

<MAJOR> is the decimal major number of the device.

MINOR=<MINOR>

<MINOR> is the decimal minor number of the device. This always refers to the first minor number which represents the whole device and not to a minor number of a partition.

DASDNAME=<DASDNAME>

<DASDNAME> is the name of the device in the classic naming scheme (dasda ... dasdz, dasdaa ... dasdzz, dasdaaa ... dasdzzz).

ACTION=<ACTION>

<ACTION> depending on the type of event that has occurred, contains a keyword that indicates the desired action to be performed by the hotplug agent.

ACTION can be:

add The device has appeared. The hotplug agent is requested to create a device node for the device. The device can now be formatted.

remove

The device has disappeared. The hotplug agent is requested to remove all device nodes for the device. The device can no longer be accessed.

partchk

A partition check has been done for the device. The hotplug agent is requested to check for available partitions (e.g. by parsing

/proc/partitions), create the device nodes of partitions that have appeared and remove device nodes of partitions the have disappeared.

partremove

All partitions for the device have disappeared because the device was unregistered at genhd. The hotplug agent is requested to remove all device nodes that correspond with a partition but leave the device node for the whole device in place (for formatting etc.).

Tape hotplug events

For every hotplug event generated by the channel attached tape device driver, the following command line parameters are provided to /sbin/hotplug:

Only one command line parameter is provided. It is always "tape". The following environment variables are set:

DEVNO=<DEVNO>

<DEVNO> is the 4 digit hexadecimal device number of the device.
Lowercase characters are used to represent values a-f.

MAJOR=<MAJOR>

<MAJOR> is the decimal major number of the device.

MINOR=<MINOR>

<MINOR> is the decimal minor number of the device. This always refers to the first minor number (non-rewinding character device or block device).

INTERFACE=<INTERFACE>

<INTERFACE> depending on the type of event that has occurred contains either "block" or "char" to indicate on which device nodes <ACTION> is to act.

ACTION=<ACTION>

<ACTION> depending on the type of event that has occurred, contains a keyword that indicates the desired action to be performed by the hotplug agent.

The following actions are defined:

add The device has appeared. The hotplug agent is requested to create all required device nodes for the device that used the selected interface.

remove

The device has disappeared. The hotplug agent is requested to remove all device nodes for the device that used the selected interface.

Crypto hotplug events

As of version 1.2.1, z90crypt generates hotplug events when the crypto module is loaded (for example, with **insmod** or **modprobe**) or unloaded (for example, with **rmmod** or **modprobe -r**). You can use these hotplug events to automatically generate or remove the device node for accessing z90crypt. Usually, this node is /dev/z90crypt.

The hotplug event causes a hotplug agent to be called. The path for the agent is defined through `/proc/sys/kernel/hotplug`. You can use the default, `/sbin/hotplug`, or specify your own path.

For every hotplug event generated by the cryptographic device driver, the following command line parameters are provided to `/sbin/hotplug`:

Only one command line parameter is provided. It is always “z90crypt”. The following environment variables are set:

MAJOR=<MAJOR>

<MAJOR> is set to the major number of the device node used for accessing z90crypt. The major number is created dynamically at module load time.

MINOR=<MINOR>

<MINOR> is set to the minor number of the device node used for accessing z90crypt, usually 0.

HOME=<HOME>

<HOME> is set to “/”.

PATH=<PATH>

<PATH> is set to “/sbin:/bin:/usr/sbin:/usr/bin”.

ACTION=<ACTION>

<ACTION> depending on the type of event that has occurred, contains a keyword that indicates the desired action to be performed by the hotplug agent.

The following actions are defined:

add The device driver has been loaded. The hotplug agent is requested to create the device node for accessing z90crypt.

remove The device driver has been unloaded. The hotplug agent is requested to remove the device node for accessing z90crypt.

z/VM recording device driver hotplug events

For every hotplug event generated by the z/VM recording device driver, one command line parameter is provided to `/sbin/hotplug`. It is always “vmlogdr”. The following environment variables are set:

ACTION=<ACTION>

<ACTION> depending on the type of event that has occurred, contains a keyword that indicates the desired action to be performed by the hotplug agent. The following actions are defined:

add The device has appeared. The hotplug agent is requested to create the required device node for the device.

remove The device has disappeared. The hotplug agent is requested to remove the device node for the device.

MAJOR=<MAJOR>

<MAJOR> is the device’s major number in decimal notation.

MINOR=<MINOR>

<MINOR> is the device’s minor number in decimal notation.

SERVICE=<SERVICE>

<SERVICE> is the name of the z/VM system service that can be accessed through the device, for example LOGREC.

Glossary

This glossary includes IBM product terminology as well as selected other terms and definitions.

Additional information can be obtained in:

- The American National Standard Dictionary for Information Systems , ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036.
- The ANSI/EIA Standard-440-A, Fiber Optic Terminology. Copies may be purchased from the Electronic Industries Association, 2001 Pennsylvania Avenue, N.W., Washington, DC 20006.
- The Information Technology Vocabulary developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1).
- The IBM Dictionary of Computing , New York: McGraw-Hill, 1994.
- Internet Request for Comments: 1208, Glossary of Networking Terms
- Internet Request for Comments: 1392, Internet Users' Glossary
- The Object-Oriented Interface Design: IBM Common User Access[®] Guidelines , Carmel, Indiana: Que, 1992.

Numerics

10 Gigabit Ethernet. An Ethernet network with a bandwidth of 10000-Mbps.

3215. IBM console printer-keyboard.

3270. IBM information display system.

3370, 3380 or 3390. IBM direct access storage device (disk).

3480, 3490, 3590. IBM magnetic tape subsystem.

9336 or 9345. IBM direct access storage device (disk).

A

asynchronous transfer mode (ATM). A transfer mode in which the information is organized into cells; it is asynchronous in the sense that the recurrence of cells containing information from an individual user is not necessarily periodic. ATM is specified in international standards such as ATM Forum UNI 3.1.

auto-detection. Listing the addresses of devices attached to a card by issuing a query command to the card.

C

cdl. compatible disk layout. A disk structure for Linux for zSeries and S/390 which allows access from other zSeries and S/390 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 and S/390 via the channel subsystem.

channel subsystem. The programmable input/output processors of the zSeries and S/390, 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.

Glossary

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 and S/390 devices using device numbers.

D

DASD. direct access storage device. A mass storage medium on which a computer stores data.

device driver. (1) A file that contains the code needed to use an attached device. (2) A program that enables a computer to communicate with a specific peripheral device; for example, a printer, a videodisc player, or a CD-ROM drive. (3) A collection of subroutines that control the interface between I/O device adapters and the processor.

E

ECKD. extended count-key-data device. A disk storage device that has a data transfer rate faster than some processors can utilize and that is connected to the processor through use of a speed matching buffer. A specialized channel program is needed to communicate with such a device.

ESCON. enterprise systems connection. A set of IBM products and services that provide a dynamically connected environment within an enterprise.

Ethernet. A 10-Mbps baseband local area network that allows multiple stations to access the transmission medium at will without prior coordination, avoids contention by using carrier sense and deference, and resolves contention by using collision detection and delayed retransmission. Ethernet uses CSMA/CD.

F

Fast Ethernet (FENET). Ethernet network with a bandwidth of 100 Mbps

FBA. fixed block architecture. A type of DASD on Multiprise 3000 or P/390 or emulated by VM.

FDDI. fiber distributed data interface. An American National Standards Institute (ANSI) standard for a 100-Mbps LAN using optical fiber cables.

FTP. file transfer protocol. In the Internet suite of protocols, an application layer protocol that uses TCP and Telnet services to transfer bulk-data files between machines or hosts.

G

Gigabit Ethernet (GbE). An Ethernet network with a bandwidth of 1000-Mbps

G3, G4, G5 and G6. The generation names of the S/390 CMOS based product family.

H

hardware console. A service-call logical processor that is the communication feature between the main processor and the service processor.

Host Bus Adapter (HBA). An I/O controller that connects an external bus, such as a Fibre Channel, to the internal bus (channel subsystem).

HMC. hardware management console. A console used to monitor and control hardware such as the zSeries and S/390 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 and S/390. Now replaced by cdl.

LDP. Linux Documentation Project. An attempt to provide a centralized location containing the source material for all open source Linux documentation. Includes user and reference guides, HOW TOs, and FAQs. The homepage of the Linux Documentation Project is <http://www.linuxdoc.org>

Linux. a variant of UNIX which runs on a wide range of machines from wristwatches through personal and small business machines to enterprise systems.

Linux for zSeries and S/390. the port of Linux to the IBM zSeries and S/390 architecture.

LPAR. logical partition of a zSeries or S/390.

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 or S/390 and the network.

O

OCO. Object-code only. A loadable module supplied by IBM without the associated source code.

OS. operating system. (1) Software that controls the execution of programs. An operating system may provide services such as resource allocation, scheduling, input/output control, and data management. (2) A set of programs that control how the system works. (3) The software that deals with the most basic operations that a computer performs.

OSA-2. Open Systems Adapter-2. A common zSeries and S/390 network interface feature

OSA-Express. Abbreviation for S/390 and zSeries Open Systems Adapter-Express networking features. These include 10 Gigabit Ethernet, Gigabit Ethernet, Fast Ethernet, Token Ring, and ATM.

OSPF. open shortest path first. A function used in route optimization in networks.

P

POR. power-on reset

POSIX. Portable Operating System Interface for Computer Environments. An IEEE operating system standard closely related to the UNIX system.

Glossary

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 or more LANs that are configured (using management software) so that they can communicate as if they were attached to the same wire, when in fact they are located on a number of different LAN segments. Because VLANs are based on logical rather than physical connections, they are extremely flexible.

volume. A data carrier that is usually mounted and demounted as a unit, for example a tape cartridge or a disk pack. If a storage unit has no demountable packs the volume is the portion available to a single read/write mechanism.

Z

zSeries and S/390. The family of IBM enterprise servers that demonstrate outstanding reliability, availability, scalability, security, and capacity in today's network computing environments.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information about the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

- AIX
- Common User Access
- developerWorks
- ECKD
- Enterprise Storage Server
- ESCON
- @server
- eServer
- FICON
- HiperSockets
- IBM
- Multiprise
- OS/390
- RAMAC
- RISC System/6000
- RS/6000
- S/390
- TotalStorage
- VSE/ESA
- VTAM
- z/Architecture
- z/OS
- z/VM
- zSeries

Intel is a trademark of Intel Corporation in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

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

Special characters

/proc
 /appldata 257
/proc/tapedevices
 interfaces 55
*LOGREC 89

Numerics

10 Gigabit Ethernet 139, 189
1000Base-T, Ethernet 139, 189
3088, control unit 113
31-bit
 values for monitor records 259
 z90crypt 67
3215 line- mode terminal 39
3270 line- mode terminal 39
3380 17
3390 17
3480 tape drive 49
3490 tape drive 49
3590 tape drive 49
9345 17

A

Address Resolution Protocol (ARP) 274
AgentX protocol 189
API
 FC-HBA 83
 API, zSeries and S/390 207
 APPLDATA 257
ARP
 Proxy ARP 154
 query/purge OSA-Express ARP
 cache 197
ARP (Address Resolution Protocol) 274
ATM feature 143
auto-detection 93, 95, 98, 99, 100, 101,
 137, 141
autoconfiguration, IPv6 157

B

block device
 tape 51
block device drivers 7
boot utility 209
broadcast 141

C

CEX2C (Crypto Express2) 63
chandev
 See channel device layer
channel device layer 93
 CTC 96
 CTCMPC 96
 ESCON 96

channel device layer (*continued*)
 LCS 96
 qeth 141
Channel Measurement Facility 25
character device
 tape 50
character device drivers 7
checksum 137, 146
Chinese-Remainder Theorem 63
CHPID
 toggling on- and offline 247
cio_ignore 284
cio_msg 286
cio_notoper_msg 287
CLAW
 device driver 159
codepage
 for x3270 45
commands, Linux
 dasdfmt 168
 dasdview 172
 depmod 231
 fdasd 182
 ifconfig 221
 insmod 225
 lsmod 230
 mke2fs 233
 modprobe 227
 osasnmppd 189
 qetharp 157, 197
 qethconf 199
 snIPL 201
 tape390_display 58
 vconfig 234
 ziPL 209
common device support 3
compatibility mode, z90crypt 67
condev, kernel parameter 43
CONFIG_DCSSBLK 33
CONFIG_Z90CRYPT 65
conmode, kernel parameter 42
connections
 CTC 108
 ESCON 108
console
 mainframe versus Linux 39
console device driver 39
 defining to P/390 43
 device node names 41
 features 39
 overriding default driver 42
 specifying preferred console 42
console, kernel parameter 42
control characters 44
CP Error Logging System Service 89
CRC 137
CRT 63
crypto
 hotplug events 320
Crypto Express2 63

cryptographic device driver
 See z90crypt

CTC 137
 chandev example 105
 channel device layer 96
 connection 108
 device driver 105
 features 105
 kernel example 107
 kernel parameter 106
 module example 108
 module options 107
 recovery 111
 syntax 105, 106, 107
CTCMPC
 channel device layer 96
 device driver 113
 subchannels 113
cua 137

D

DASD
 access by VOLSER 22
 API 23
 boxed 21
 Channel Measurement Facility 25
 device driver 9
 dynamic attach 20
 enabling and disabling 20
 features 17
 hotplug events 319
 partitioning 14
 preparing for use 16
 reserve and release 21
dasdfmt, Linux command 168
dasdview, Linux command 172
DCSS device driver 33
decryption 63, 69
depmod, Linux command 231
developerWorks xiii
device driver 139
 block and character 7
 console 39
 crypto 63
 CTC 105
 CTCMPC 113
 DASD 9
 DCSS 33
 ESCON 105
 HyperSockets 139
 LCS 137
 module overview 3
 network 91
 OSA-Express (non-QDIO) 137
 OSA-Express (QDIO) 139
 SCSI via Fibre Channel 73
 tape 49
 XPRAM 29
 z/VM recording 89
 z90crypt 63

- device driver (*continued*)
 - zfc 73
- device major number 49
- device nodes
 - z90crypt 67
- devices, ignoring 284
- DHCP
 - for HiperSockets 282
 - required options for zSeries and S/390 278
- DHCP (Dynamic Host Configuration Protocol) 276
- Direct SNMP 189
- discontiguous saved segments device driver 33
- domain=
 - kernel parameter 65
 - module parameter 66
- dump tools xiii
- Dynamic Host Configuration Protocol (DHCP) 276
- dynamic routing, and VIPA 237

E

- ECKD 17
- edit characters
 - VM console 45
- encryption 63
- Enterprise Storage Server 17
- Error Logging System Service 89
- ESCON
 - chandev example 105
 - channel device layer 96
 - connection 108
 - device driver 105
 - features 105
 - kernel example 107
 - module example 108
 - recovery 111
 - syntax 105, 106, 107
- Ethernet 137, 139, 189
- examples
 - CTC chandev 105
 - CTC kernel 107
 - CTC module 108
 - ESCON chandev 105
 - ESCON kernel 107
 - ESCON module 108
 - Source VIPA 242
 - tape driver 59
 - toggleing CHPIDs online/offline 248
 - VARY ON/OFF notification 23
 - VIPA (Virtual IP address) use 238
- expanded memory 29

F

- fake_ll 281
- Fast Ethernet 137
- FBA 17
- FC-HBA 83
- FCP 73
- fdasd, Linux command 182
- features
 - console 39

- features (*continued*)
 - CTC 105
 - ESCON 105
- Fibre Channel 73
- file system
 - ISO9660 52
 - proc 5
 - tape 52
- full-screen mode terminal 40

G

- Gigabit Ethernet 139, 189

H

- hardware console 39
- Hardware Management Console
 - See* HMC
- hardware status, z90crypt 67
- HBA API 83
- high availability project 207
- High Speed Token Ring 137
- HiperSockets 139
- HiperSockets Network Concentrator 267
- HMC 39
- HMC (Hardware Management Console)
 - usage 45
- hotplug 319

I

- I/O message suppression 286, 287
- IBM TotalStorage Enterprise Storage Server 17
- ifconfig, Linux command 221
- insmod, Linux command 107, 225
- interfaces
 - /proc/tapedevices 55
 - FC-HBA 83
 - MTIO 54
 - tape390_display tool 58
- ioctl
 - return codes 71
- IP address
 - confirming 158
 - virtual 155
- ipldelay 288
- IPv4
 - OSA-Express 274
- IPv6
 - OSA-Express 274
 - stateless autoconfiguration 157
 - support for 157
- IRQ 4
- ISO9660 file system 52
- IUCV
 - device driver 115

K

- kernel
 - building 301
 - source tree xiii

- kernel configuration menu options
 - z90crypt 65
- kernel module
 - z90crypt 65
- kernel parameter
 - console 41
 - CTC 106
 - tape device driver 52
 - XPRAM 30
- kernel parameter line file 299
- kernel parameters 283
 - domain= 65
 - z90crypt 65

L

- layer2 280
- LCS
 - channel device layer 96
 - device driver 137
- libica library 66
- libpcap 278
- license xiv
- line edit characters
 - VM console 45
- line-mode terminal 39
- line-mode terminals
 - special characters 44
- LLC header 275
- log-in at terminals 40
- Logical Link Control (LLC) header 275
- Logical Volume Manager 249
- LOGREC 89
- lsmod, Linux command 230
- LVM 249

M

- MAC addresses 273
- management information base 189
- maxcpus 289
- maximum devices
 - tape 49
- Media Access Control (MAC)
 - addresses 273
- mem 290
- memory
 - expanded 29
- MIB (management information base) 189
- mke2fs, Linux command 233
- modprobe, Linux command 107, 227
- module options
 - CTC 107
- module parameter
 - qeth 147
 - tape device driver 53
 - XPRAM 31
- module parameters
 - domain= 66
 - z90crypt 66
- modulus-exponent 63
- monitor stream 257
- MTIO interface 54
- MTU
 - qeth 158

multicast 138
multipathing 249
Multiprise 17

N

net-snmp 189
Network Concentrator 267
network device drivers 91
noinitrd 291
non-rewinding tape devices 50
notices 327

O

object ID 189
OID (object ID) 189
openCryptoki 66
options 107
OSA-2 137
OSA-Express 137
 ATM feature 143
 device driver 139
 IPv4 handling 274
 IPv6 handling 274
osasnmpd, OSA-Express SNMP
 subagent 189

P

P/390 39, 290
padding, z90crypt 64
parameter file 107
parameter line 299
partitioning DASD 14
patches
 dhcp 276
 dhcpcd 278
 tcpdump 279
path
 weight value 253
PCI Cryptographic Accelerator 63
PCI Cryptographic Coprocessor 63
PCI-X Cryptographic Coprocessor 63
PKCS #11 API 63, 66
port name 145
preferred console 42
prerequisites 7, 91, 166
proc file system 5
Proxy ARP 154
Purge ARP 157
pvpath 251
pvpathrestore 254
pvpathsave 253

Q

qdio 139
qeth
 channel device layer 141
 fake_ll option 281
 layer2 option 280
 module parameter 147
 MTU 158
qeth_sparebuffs 147

qetharp, Linux command 157, 197
qethconf, Linux command 199
Query ARP 157
queueing 141

R

RAMAC 17
recovery
 CTC 111
 ESCON 111
restrictions 7, 91, 166
return codes, ioctl and read 71
rewinding tape devices 50
ro 293
root 294
routing 141, 145
RSA exponentiation 63
RVA 17

S

S/390 Application Programming
 Interfaces 207
SCLP (service-call logical processor)
 interface 39
SCSI
 driver (via Fibre Channel) 73
SE (Service Element) 39
Seascape 17
service-call logical processor interface 39
Shoot The Other Node In The Head 207
SILO 209
simple network IPL 207
Simple Network Management
 Protocol 189
snIPL 207
snIPL, Linux command 201
 description 201
 syntax 203
SNMP 189, 207
Source VIPA 240
 example 242
special characters
 line-mode terminals 44
 VM console 45
stateless autoconfiguration, IPv6 157
static routing, and VIPA 237
STONITH 207
subchannels
 CTCMPC 113
syntax diagrams xiv
Systems Management Application
 Programming, z/VM 208

T

tape 49
 block device 51
 character device 50
 control operations 51
 device driver 49, 53
 device major number 49
 discipline modules 50
 display support, tape 58
 driver examples 59

tape (*continued*)
 file system 52
 hotplug events 320
 kernel parameter 52
 maximum devices 49
 module parameter 53
 reserving devices 56
 tape390_display tool 58
tape390_display, Linux command 58
TCP/IP 105, 115
tcpdump 278
TERM, environment variable 40
terminal
 enabling user log-ins 40
 mainframe versus Linux 39
Token Ring 137, 139, 189
trademarks 328
TTY routines 43

U

ucd-snmp 189
use_pfix 295

V

VARY ON/OFF, notification of 247
vconfig, Linux command 234
VInput 46
VIPA (virtual IP address)
 description 155, 237
 example 238
 Source VIPA 240
 static routing 237
 usage 237
virtual
 IP address 155
VLAN (virtual LAN) 263
VM console
 line edit characters 45
vmhalt 296
vmlogrdr.o 89
vmpoff 297
VOLSER, DASD device access by 22
VTOC 12

W

weight value 253

X

x3270 codepage 45
XPRAM
 device driver 29
 features 29
 kernel parameter 30
 loading 31
 module parameter 31

Z

z/VM
 monitor stream 257

- z/VM (*continued*)
 - Systems Management Application
 - Programming 208
- z/VM discontinuous saved segments
 - device driver 33
- z/VM recording device driver 89
 - hotplug events 321
- z90crypt
 - deactivating 68
 - decryption 69
 - device driver 63
 - device nodes 67
 - hardware status 67
 - kernel configuration menu option 65
 - kernel module 65
 - kernel parameter 65
 - module parameter 66
- zfcp driver 73
- zfcp HBA API 83
- zIPL, Linux command 209
- zSeries Application Programming Interfaces 207

Readers' Comments — We'd Like to Hear from You

Linux on zSeries
Device Drivers and Installation Commands
March 23, 2005
Linux Kernel 2.4 - June 2003 stream

Publication No. SC33-8282-00

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.



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



SC33-8282-00

