Table of Contents

Preface

This document, *Linux QETH Implementation*, is entirely based upon information made public by IBM via web sites, IBM manuals, public presentations, for example, Share, and Linux source code. At no time was access to any IBM proprietary or trade secret information used in the creation of this document.

Introduction

This overview addresses two primary areas:

Open Systems Adapter Hardware

Linux QETH Components

Appendix A is an overview of how Linux QDIO Layer-2 operations are established. The remaining appendices should be viewed a research notes created during this document's preparation and are not necessarily complete. Their purposes were to assist the author in understanding various aspects of the Linux implementation.

For a discussion of QDIO data transfers between the program and the adapter, refer the document *Linux QDIO*.

A future version of this document will detail Layer-3 IP Assist command structures. Only minimal documentation is provided in this version.

Open Systems Adapter (OSA)

Four versions of the OSA exist. Each supports one or more of four channel types:

OSD – Queued Direct Input/Output (QDIO). Also referred to as Internal QDIO (IQDIO) when this channel type is emulated by a HiperSocket.

OSE – Non-queued Direct Input/Output (non-QDIO), LAN Channel Station emulation,

OSC – OSA-Express Integrated Console Controller and

OSN – Network Control Program (NCP) under Communication Controller for Linux (CCL)

The following table identifies the channel types supported by the various network devices of this class.

Network Device	OSD Channels	OSE Channels	OSC Channels	OSN Channels
OSA-2	no	yes	no	no
OSA Express	yes	yes	no	no
OSA Express2	yes	yes	yes	yes
OSA Express3	yes	yes	yes	yes
HiperSocket	yes	no	no	no

Frame sizes associated with the different physical interface types:

Physical Interface	802.3 Frames	DIX Frames	Jumbo
Gigabit Ethernet	1492	1492	8992
10 Gigabit Ethernet	1492	1492	8992
Fast Ethernet	1492	1492	Not supported
1000Base-T Ethernet	1492	1492	8992

QDIO Configuration

The following options are configured as part of the hardware definition of the OSA Express OSE (Ethernet) or Hipersocket interface:

Device numbers: three consecutive devices for read, write and data.

Port Number

Port Name

Maximum number of queues

Physical interface

Linux Support of QDIO (OSD Channel) Features – Real Interfaces

Reference: SC33-8411-03 Device Drivers, Features, and Commands, Development stream (Kernel 2.6.31)

QDIO operates at either Layer-3 or Layer-2. The primary difference is how data is forwarded. In Layer-3 mode, packets are forwarded based upon destination IP address. In Layer-2 mode, frames are forwarded based upon MAC address. The following table identifies which features are available with which mode.

The following color coding is used in the diagram:

red – feature not applicable to Hercules

blue – potential feature for Hercules

yellow – feature differs between real device and z/VM emulation

green – features to be implemented by Hercules

QDIO Feature	OSA Layer 2	OSA Layer 3	HiperSocket Layer 2	HiperSocket Layer 3
Layer Operation	OSD default	dynamic	dynamic	HiperSocket default
DMA with store-and-forward	Express2	Express2	yes	yes
DMA without store-and-forward	Express3	Express3	no	no
Dynamic LAN Idle	yes	yes	no	no
Enhanced IP Network Availability (IPA)				
ARP Takeover (IPv4 only)	implicit	yes	implicit	no
IP broadcast support	implicit	yes	implicit	yes
IP multicast support	implicit	yes	implicit	yes
Internet Protocol version 4 (IPv4)	implicit	yes	implicit	yes
Routing Driver Setup	no	required	no	yes
VLAN Support	sw	hw	sw	hw
Layer-3 Virtual MAC	Not applicable	yes	no	yes

QDIO Feature	OSA Layer 2	OSA Layer 3	HiperSocket Layer 2	HiperSocket Layer 3
ARP	implicit	No (hw)	implicit	no
ARP Cache management	no	yes	no	no
Checksum offload - receive	no	yes	no	no
Checksum offload - transmit	no	no	no	no
Internet Protocol version 6 (IPv6)	implicit	yes	implicit	yes
Routing Driver Setup	no	required	no	yes
VLAN Support	sw	sw	sw	no
Layer-3 Virtual MAC	Not applicable	yes	no	yes
Neighbor Discovery	implicit	yes	implicit	no
Jumbo Frame (8992 byte frame size)	yes	yes	yes	yes
Large Send for TCP/IP	no	yes	no	no
Link aggregation (LACP, z/VM only)	yes	no	yes	no
LPAR-to-LPAR communications	yes	yes	yes	yes
Port Sharing	Yes?	yes		
Data Connection (port) Isolation		yes	yes	no
IP Takeover	no	required	no	yes
Layer-2 to Layer-3 support	Yes by hw	yes	no	yes
Primary/Secondary Router	no	yes	no	no
OSA for NCP	Note 2	Note 2	Note 2	Note 2
MAC Address (unique)	yes (random)	no	yes	yes
MAC Address (change)	yes	no	yes	no
MAC Headers (send)	yes	faked	yes	faked
MAC Headers (receive)	yes	faked	yes	faked
Multi-port Support	yes	yes	no	no
Network Traffic Analyzer (z/OS only)	no	no	no	no
Non-IP Traffic	implicit	yes	implicit	no
Priority Queuing	no	yes	no	yes
Promiscuous Mode	no	no	no	no
Proxy ARP Driver Setup	no	required	no	yes
QDIO Diag. Synch. (z/OS only)	no	no	no	no
Receive buffer count	yes	yes	yes	yes
SNMP Support (Note 3)				
OSA/SF (z/VM, z/OS)	yes	yes	no	no

QDIO Feature	OSA Layer 2	OSA Layer 3	HiperSocket Layer 2	HiperSocket Layer 3
Direct SNMP (Linux)	yes	yes	no	no
TCP Segmentation offload	no	TSO > large_send	no	no
Virtual IP Address (VIPA) Driver Setup	no	required	no	yes
VLAN Support for GVRP	yes	yes	yes	no

Note 1: Layer-2 operation is protocol independent and therefore supports this feature.

Note 2: OSA for NCP mode is a third mode of operation for OSD subchannels, distinct from either Layer-2 or Layer-3 modes.

Note 3: SNMP support utilizes the OSA/SF device on z/OS, defined as device type OSAD rather than device type OSA. This protocol used on this device is proprietary. Linux uses an open source SNMP subagent, osasnmpd provided by the s390-tools package. This subagent needs to be reviewed to determine feasibility of implementation on Hercules.

Priority queuing sorts outgoing IP message traffic according to the priority assigned in the IP header (using the IP Type of Service field). This is available only with z/OS environments.

With enhanced IP network availability, all home IP addresses are stored in the OSA. The OSA-Express feature port then responds to ARP requests for own IP address, as well as for other IP addresses active in the TCP/IP stack, in particular for virtual IP addresses.

ARP Takeover allows one OSA port to take over responsibility for an IP address by providing a gratuitous ARP to update network forwarding tables of the new location of the MAC address. TCP/IP drives this operation.

Linux Support of QDIO (OSD Channel) Features – Emulated Interfaces

Reference: SC33-8411-03 Device Drivers, Features, and Commands, Development stream (Kernel 2.6.31)

Emulated interfaces are provided by z/VM. QDIO operates at either Layer-3 or Layer-2. The primary difference is how data is forwarded. In Layer-3 mode, packets are forwarded based upon destination IP address. In Layer-2 mode, frames are forwarded based upon MAC address. The following table identifies which features are available with which mode.

QDIO Feature	Emulated OSA Layer 2	Emulated OSA Layer 3	HiperSocket Layer 2 (Guest LAN)	HiperSocket Layer 3 (Guest LAN)
Layer operation	Static (default)	static	static	Static (default)

QDIO Feature	Emulated OSA Layer 2	Emulated OSA Layer 3	HiperSocket Layer 2 (Guest LAN)	HiperSocket Layer 3 (Guest LAN)
ARP	yes (note 1)	No (hw)	no	no
ARP Cache management	no	yes	no	no
ARP Takeover	no	yes	yes (note 1)	no
Checksum offload (IPv4 only)		yes	no	yes
Receive	no	no	no	no
Transmit	no	no	no	no
DMA with store-and-forward	Express2	Express2	yes	yes
DMA without store-and-forward	Express3	Express3	no	no
Dynamic LAN Idle (z/OS only)	no	no	no	no
Enhanced IP Network Availability (IPA)				
ARP Takeover	no	yes	yes (note 1)	no
IP broadcast support	yes (note 1)	yes	yes (note 1)	yes
IP multicast support	yes (note 1)	yes	yes (note 1)	yes
Internet Protocol version 4 (IPv4)	yes (note 1)	yes	yes (note 1)	yes
Broadcast	yes	yes	yes	yes
Multicast	yes	yes	yes	yes
Routing Driver Setup	no	required	no	required
VLAN Support	sw	hw	hw	hw
Layer-3 Virtual MAC	no	yes	no	yes
Internet Protocol version 6 (IPv6)	yes (note 1)	yes	no	no
Broadcast	yes	yes	no	no
Multicast	yes	yes	no	no
Routing Driver Setup	no	required	no	required
VLAN Support	sw	sw	no	no
Layer-3 Virtual MAC	no	yes	no	no
Jumbo Frame (8992 byte frame size)	yes	yes	yes	yes
Large Send for TCP/IP	no	no	no	no
Link aggregation	yes	no	no	no
LPAR-to-LPAR communications	no	no	no	no
Port Sharing	no	no	no	no
Data Connection (port) Isolation	no	no	no	no
IP Takeover	no	required	no	yes

QDIO Feature	Emulated OSA Layer 2	Emulated OSA Layer 3	HiperSocket Layer 2 (Guest LAN)	HiperSocket Layer 3 (Guest LAN)
Layer-2 to Layer-3 support	no	no	no	no
Primary/Secondary Router	no	no	no	no
OSA for NCP		yes	Note 2	Note 2
MAC Address (unique)	yes (random)	no	yes	yes
MAC Address (change)	yes	no	no	no
MAC Headers (send)	yes	faked	faked	faked
MAC Headers (receive)	yes	faked	faked	faked
Multi-port Support	no	no	no	no
Network Traffic Analyzer (z/OS only)	no	no	no	no
Neighbor Solicitation	yes (note 1)	yes	no	no
Non-IP Traffic	yes (note 1)	no	no	no
VLAN Support	sw	no	no	no
Priority Queuing	yes	yes	yes	yes
Promiscuous Mode	yes	yes	no	no
Proxy ARP Driver Setup	no	required	no	required
QDIO Diag. Synch. (z/OS only)	no	no	no	no
Receive buffer count	yes	yes	yes	yes
Segmentation offload	no	no	no	no
SNMP Support (Note 3)	no	no	no	no
Virtual IP Address (VIPA) Driver Setup	no	required	no	required
VLAN Support for GVRP	yes	yes	yes	no

Note 1: Layer-2 operation is protocol independent and therefore supports this feature.

Note 2: OSA for NCP mode is a third mode of operation for OSD subchannels, distinct from either Layer-2 or Layer-3 modes.

Note 3: SNMP support utilizes the OSA/SF device, defined as device type OSAD rather than device type OSA. The protocol used on this device is proprietary.

Priority queuing sorts outgoing IP message traffic according to the priority assigned in the IP header (using the IP Type of Service field). This is available only with z/OS environments.

With enhanced IP network availability, all home IP addresses are stored in the OSA. The OSA-Express feature port then responds to ARP requests for own IP address, as well as for other IP addresses active in the TCP/IP stack, in particular for virtual IP addresses.

ARP Takeover allows one OSA port to take over responsibility for an IP address by providing

a gratuitous ARP to update network forwarding tables of the new location of the MAC address. TCP/IP drives this operation.

OSN Channels

OSN channels utilize the local routing between interfaces on the OSA Express2 or later hardware to allow a SNA PU.T5 using channel data link control on an NCP channel interface to communicate with a Linux resident NCP over another OSN configured interface. This provides the appearance of a channel attached Communications Controller to the SNA PU.T5.

The qeth_l2 driver provides a command channel interface for Communications Controller for Linux. This is likely implemented via a CC4L specific driver which calls the OSN related functions within qeth_l2 exposed by Linux EXPORT_SYMBOL statements. This interface allows CC4L to send its own OSA adapter commands and handle responses outside of the qeth_l2 driver itself. A Linux network device is also created for CC4L usage for SNA PDU transfers.

A Communications Controller ID (CCID) is used exclusively by OSN devices. The CCID is placed in data transfer buffers. The CCID is likely used by the OSN channels for directing data between the correct OSN group device in Linux and PU.T4 channel data link control interface. Three IP Assist Layer-2 commands are associated with CCID management.

No further analysis of the OSN interface is currently provided.

Device Information

Control Unit Type: 0x1731

Control Unit Models:

- 0x01 – OSA Enhanced
- 0x05 – HiperSocket
- 0x06 – OSN

Device Type: 0x1732

Device Models:

- 0x01 – OSA Enhanced
- 0x05 – HiperSocket
- 0x06 – OSN

Model List Data

The known_device array in net/qeth_core_main lists the recognized QETH devices. The

recognition is based upon Sense ID data returned by the read device of the device group. There are three entries each containing 10 integers. The values are set from the QETH_MODELLIST_ARRAY definition in net/qeth_core.h. The following table documents the values provided.

Entry	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Column Desc.	CU Type	CU Model	Device Type	Device Model	Card Type				Max Queues	
[0]	0x1731	0x01	0x1732	0x01	10	1	Note 2	Note 3	4	0
[1]	0x1731	0x05	0x1732	0x05	1234	0	Note 6	Note 7	4	0x103
[2]	0x1731	0x06	0x1732	0x06	11	0	Note 2	Note 3	4	0

Card type identification is based upon the device type, column 2, and device model, column 3. The card type, column 4, is moved into the card.info.type field. The maximum number of queues is derived from column 8. The card.info.is_multicast_different field is set from column 9. Columns 6 and 7 provide information used during IDX_ACTIVATE processing.

Note 1 – QETH card types:

10 = OSA Express (QETH_CARD_TYPE_OSAE)

11 = NCP Channel (QETH_CARD_TYPE_OSN)

1234 = HiperSocket (QETH_CARD_TYPE_IQD)

Note 2: QETH_IDX_FUNC_LEVEL_OSAE_ENA_IPAT (0x0101)

Note 3: QETH_IDX_FUNC_LEVEL_OSAE_DIS_IPAT (0x0101)

Note 4: Maximum number of queues

Note 6: QETH_IDX_FUNC_LEVEL_IQD_ENA_IPAT (0x4108)

Note 7: QETH_IDX_FUNC_LEVEL_IQD_DIS_IPAT (0x5108)

OSA Express Interface

The OSA Express interface consists of three separate devices, each of which operates on its own I/O subsystem subchannel and have assigned to it its own device number. Each of the three devices has its own role in the overall OSA Express Interface:

Read device – Supports the submission of commands to the OSA Express

Write device – Provides responses from previously submitted commands

Data device – Transports network data between main storage and the network adapter using QDIO queues.

The read and write devices must be configured with sequential device numbers, the write device being the next in sequence beyond the read device's device number. Once the read and write devices have successfully performed an IDENTIFICATION EXCHANGE with the program, the two devices will act as a single command channel for the OSA Express adapter.

Validation that the correct data device is in use occurs during the OSA command channel activation sequence when IDENTIFICATION EXCHANGE data is shared.

Refer to the Linux QDIO document for details on QDIO data device operation. The remainder of this document describes the operation of the read and write devices.

The read and write devices respond to the following channel command words:

0x01 – WRITE

0x02 – READ

0x04 – SENSE – See below, “Sense Data” section

0xE4 - SENSE ID – See above “Device Information” section.

Device Status

Only unit exception and unit check are examined as device status error conditions related to OSA QDIO devices. Sense Data, see below, should be examined for more information about the condition when unit check is presented.

Unit exception is suspected of being used for signaling that a pending read on the read device is not present when a write to the write device occurs. It might also be used to indicate more generally any invalid channel state. Such states might be receiving an transport PDU before the IDX_ACTIVATE has established the transport channel, or a WRITE or READ is issued to a read device or write device, respectively, after the transport channel has been established.

Sense Data

Linux uses concurrent sense. How many of the bytes of the 32 available for concurrent sense

is unclear. Only bytes 0-3 are examined by Linux.

In a post on the Marist Linux for System z email list, 16 Dec 2010, a poster sent the output from a OSA activation error with an invalid port name. Device dependent bit 6 of sense byte 0 was set to one in this case, which would have been in response to a ULP_ENABLE request write CCW to the write device. This bit or any additional bits set in these four bytes, other than the values identified below, are treated as device configuration errors. Configuration errors are suspected of being reported during IDX_ACTIVATE handling. If this is the case, then no IDX_ACTIVATE response would be provided.

Byte	Name	
0	Standard meaning	Bit 0 ==1 -> A command reject has occurred
1	Resetting Event Byte	Bit 0 ==1 -> A resetting event has occurred
2,3	AFFE	Checked for 0xAFFE, meaning undocumented

Identification Exchange – Establishing Transport Operations

The purpose of the Identification Exchange operations with the read and write devices is to

validate the set of devices the program is using are part of the same OSA configuration group and

establish the full duplex transport channel for a correct set of devices.

An IDX_ACTIVATE_WRITE command is sent to the write device using a WRITE CCW to activate the command channel write role of the write device.

An IDX_ACTIVATE_READ command is sent to the read device using a WRITE CCW to the activate the command channel read role of the read device.

The structure of the 34-byte command channel transport operation activate command is as follows. “SbP” indicates fields set by the program.

Disp. (hex)	Size	IDX_ACTIVATE_READ Request Template	IDX_ACTIVATE_WRITE Request Template	Description
+00	2	0x0000	0x0000	
+02	2	0x8000	0x8000	Device directed command
+04	4	0x00000000 (SbP)	0x00000000 (SbP)	Transport Header Sequence Number (Note 5)
+08	2	0x1901	0x1501	IDX_ACTIVATE type (READ or WRITE)
+0A	1	0x01	0x01	
+0B	1	0x8x	0x8x	OSA Express Port number (with bit 0 set to 1)
+0C	4	0x00000000 (SbP)	0x00000000 (SbP)	Issuer rm_w token (0x00010103UL)
+10	2	0x0000	0xFFFF	Function level (Note 1)
+12	4	0x00000000	0x00000000	Microcode level in response

Disp. (hex)	Size	IDX_ACTIVATE_READ Request Template	IDX_ACTIVATE_WRITE Request Template	Description
+16	8	EBCDIC 'HALL0LE '	EBCDIC 'HALL0LE '	EBCDIC Dataset name (Note 4)
+1E	2	0x0000 (SbP)	0x0000 (SbP)	Data device device number (Note 6)
+20	1	0x00 (SbP)	0x00 (SbP)	Data device Control Unit Address (Note 2)
+21	1	0x00 (SbP)	0x00 (SbP)	Data device Unit Address (Note 3)

Note 1: Function level is derived from columns 6 and 7 in the model list data table.

Note 2: Data device control unit address is provided by the read device RCD CCW response data. The control unit address is found in byte 31 of the I/O Device Node Element Descriptor.

Note 3: Data device unit address is provided by the read device RCD CCW response data. The unit address is provided in byte 31 of the Emulation Node Element Descriptor.

Note 4: This name is a program supplied string used to identify the OSA adapter. Linux uses the name 'HALL0LE “, or, in ECBDIC, X'C8C1D3D3D6D3C540”. Operating systems that support VTAM style configurations may use a name supplied by the configuration node for the adapter. This is not the port name, which is specified in an ULP_ENABLE request.

Note 5: The transport sequence number in the IDX_ACTIVATE_COMMAND has the effect of establishing the sequence number while identifying its own sequence number. This number will be incremented for the first and subsequent command channel requests.

Note 6: This is the data device number that the host expects to used with this OSA.

The responses is retrieved from the read or write device using a CCW count of 4096 with a READ CCW command. The length actually read is not possible to determine. However, the response data beyond byte 22, if any, is ignored by Linux.

Disp. (hex)	Size	IDX_ACTIVATE_READ Response	IDX_ACTIVATE_WRITE Response	Description or Comment
+00	2	ignored	ignored	
+02	1	Bits 0,1: IDX response	ignored	See “IDX Response” below.
+03	1	ignored	ignored	
+04	1	0x22 => invalid portname	ignored	IDX TERMINATE cause code
+05	3	ignored	ignored	
+08	1	Bits 6,7 == 10	Bits 6,7 == 10	Positive response
+09	1	0x19 ==> exclusively owned by another host	0x19 ==> exclusively owned by another host	Non-positive response cause code. Only 0x19 explicitly recognized by Linux.
+0A	1	ignored	ignored	
+0B	1	Bit 0 == 1 implies **no** port name required	ignored	

Disp. (hex)	Size	IDX_ACTIVATE_READ Response	IDX_ACTIVATE_WRITE Response	Description or Comment
+0C	4	issuer_rm_r token response	ignored	Value used in other commands
+10	2	Function level returned	Function level returned, bit seven may be set to 1	See “Function Level Processing” below.
+12	4	Microcode Level returned	ignored	Used for reporting purposes only

IDX Response: These two bits set to 1 indicates an IDX_TERMINATE has been received. Normal response for these two bits is unclear. Suspect that the first bit means a response, but this can not be validated.

Following a successful exchange of IDX_ACTIVATE_READ messages with the read device and IDX_ACTIVATE_WRITE messages with the write device, the read and write devices enter command channel operational mode.

Function Level Processing

It is impossible to determine through reverse engineering the meaning of these responses. The code indicates these responses will be accepted. The request values suggest that at the level of the IDX_ACTIVATE command, OSA Express devices always use a function level that indicates IP take over assist is available. Whether IP take over assist is enabled or not is determined in Linux by the sysfs attribute “ipato”.

Card Type	IP TO Assists	ACTIVATE_READ Request	ACTIVATE_READ Response	ACTIVATE_WRITE Request	ACTIVATE_WRITE Response
OSA Express	disabled	0x0101	0x0201	0x0101	0x0201
OSA Express	enabled	0x0101	0x0201	0x0101	0x0201
Hipersocket	disabled	0x5108	0x0408	0x5108	0x0408
Hipersocket	enabled	0x4108	0x0408	0x4108	0x0408

Sequence Numbers

Multiple series of sequence numbers are maintained between the program and the OSA.

Transport Sequence Numbers – are used for all command channel requests and corresponding responses.

PDU Sequence Numbers – are used with all Protocol Data Unit messages.

PDU ACK Sequence Numbers – are used to specify the expected response PDU sequence number. Although, Linux does not manage PDU ACK sequence numbers or increment them.

IP Assist Sequence Numbers – are used with all IP assists request and responses.

Linux performs no monitoring of sequence numbers in responses. In the case of Transport

Sequence, PDU Sequence and IP Assist Sequence numbers, Linux immediately increments the sequence number following the placement of the current maintained sequence number into the message. Hence, Linux is maintaining the “next” sequence number to be used.

Tokens

Tokens are four-byte unsigned values. The host provides tokens in some commands and the adapter provides tokens used in other commands. The following table identifies the tokens used.

Token	Source	Usage by Program
issrmr	IDX_ACTIVATE_READ response	CM_ENABLE request. CM_SETUP request
issrmw	Program, 0x00010103	IDX_ACTIVATE_READ command
cmfilw	Program, 0x00010108	CM_ENABLE request
cmfilr	CM_ENABLE response	Not used by program
cmconr	CM_SETUP response	ULP_ENABLE request, ULP_SETUP request, DM_ACT request
ulpfilw	Program, 0x0001010B	ULP_ENABLE request
ulpfilr	ULP_ENABLE response	ULP_SETUP request
ulpconw	Program, 0x0001010D	ULP_SETUP request
ulpconr	ULP_SETUP response	IPA command request, DM_ACT request
	Program, 0x00010111	CM_SETUP request template

The actual role of the tokens is unclear. They may operate as simply data meaningful to either the OSA implementation or the program implementation, similar in concept to the interrupt parameter used in subchannel I/O operations. Three of the tokens provided by the OSA adapter operate in the role of “destination tokens”. These three are identified above in **bold** font.

Command Channel Operations

In command channel mode, the read and write devices restrict themselves to specific roles.

> Write device – accepts command channel requests for processing when a WRITE CCW command is issued to the device for the length of the command.
>
> Read device – provides a command channel response when the next READ CCW is presented to the device. Normally the program will initiate a READ CCW to the device which will remain active while waiting for the next response. A HALT I/O or CLEAR I/O is required to terminate an active read waiting for data.

The first action taken upon entering command channel mode by the program is to issue a READ CCW to the read device. Linux processing issues a READ CCW, with suppress-length-indication set and a data length of 4096 bytes.

Command Channel Protocol Data Units

Command channel request has the following standard 64-byte Protocol Data Unit (PDU) header. Inspection of the usage of the fields in channel command messages suggests that there are in fact two areas of what Linux calls the IPA_PDU_HEADER: a transport header and a true Protocol Data Unit Header. This is suggested by two factors:

> In the “Transport Header” the length is the total message length including all headers and a separate transport sequence number is maintained vs.
>
> In the “Protocol Data Unit Header” the length of the PDU content is used, excluding any headers, and separate sets of sequence numbers are maintained for the PDU and the PDU acknowledgment.

Why the PDU content length occurs three time in the “Protocol Data Unit Header” is unclear. The format could suggest that actually multiple PDU's in a single transport message are possible.

Command Channel Requests and Responses

Of the three OSD subchannels required to support a OSA Express QDIO interface, the first two are used to send commands to the adapter and receive responses to the commands. The commands and command structures are defined in **drivers/s390/net/qeth_core_mpc.h**. Linux manages the options using a bit map that identifies each command.

Six forms of requests and responses are used: CM_ENABLE, CM_SETUP, ULP_ENABLE, ULP_SETUP, DM_ACT and IPA, Other than the IPA requests and responses, which have their own structure, the requests and responses follow a similar pattern. Each contains multiple “items”. Each item starts with a length field that includes itself. In the table below, the items are highlighted with different colors. There appears to also be in use some mechanism for describing the content of individual fields in each item. The scheme is not obvious from the data, so fields suspected of providing this role are simply marked with “desc”.

Locating the PDU response content is achieved by the following approach:

1. Locate the Response Header from the Transport Header length.
2. Add the Response Header length to the Transport Header length to locate the start of the Protocol Data Unit header.
3. From the Protocol Data Unit header determine the total length of the headers preceding the Protocol Data Unit content. This value can then be used to locate the start of the Protocol Data Unit content from the start of the response.

This calculation is performed in Linux strictly for locating data within a response. Using these values largely protects the program from changes that might occur in the header sizes in the future or actual variances in the message headers of adapter responses.

The following table compares the template content for each form of request. All field contents are in hex. “SbP” indicates the value is supplied by the program. “Resp” indicates a response

field that should be zeros in the template.

Disp. (hex)	Len.	CM_ENABLE	CM_SETUP	ULP_ENABLE	ULP_SETUP	IPA	DM_ACT
Transport Header							
+00	1	00	00	00	00	00	00
+01	1	E0	E0	E0	E0	E0	E0
+02	2	0000	0000	0000	0000	0000	0000
+04	4	SbP (transeq)	SbP (transeq)	SbP (transeq)	SbP (transeq)	SbP (transeq)	SbP (transeq)
+08	2	0000	0000	0000	0000	0000	0000
+0A	2	0014 (Note 5)	0014 (Note 5)	0014 (Note 5)	0014 (Note 5)	0014 (Note 5)	0014 (Note 5)
+0C	2	0000	0000	0000	0000	0000	0000
+0E	2	0063 (Note 10)	0064 (Note 10)	006B (Note 10)	006C (Note 10)	SbP (Note 10)	0055 (Note 10)
+10	4	10000001	10000001	10000001	10000001	10000001	10000001
Request/Response Header							
+14	4	00000000	00000000	00000000	00000000	00000000	00000000
+18	1	81 (CM)	81 (CM)	41 (ULP)	41 (ULP)	C1 (IPA))	41 (ULP)
+19	1	7E	7E	7E	7E	SbP (Note 1)	7E
+1A	2	0001	0001	0001	0001	0001	0001
+1C	4	SbP (pduseq)	SbP (pduseq)	SbP (pduseq)	SbP (pduseq)	SbP (pduseq)	SbP (pduseq)
+20	4	SbP (ackseq)	SbP (ackseq)	SbP (ackseq)	SbP (ackseq)	SbP (ackseq)	SbP (pduseq)
+24	2	0024 (Note 6)	0024 (Note 6)	0024 (Note 6)	0024 (Note 6)	0024 (Note 6)	0024 (Note 6)
+26	2	0023 (Note 8)	0024 (Note 8)	002B (Note 8)	002C (Note 8)	SbP (Note 8)	0015 (Note 8)
+28	1	00	00	00	00	00	00
+29	2	0023 (Note 8)	0024 (Note 8)	002B (Note 8)	002C (Note 8)	SbP (Note 8)	002C (Note 8)
+2B	1	05	05	05	05	05	05
+2C	4	SbP (issrmr)	SbP (issrmr)	SbP (cmconr)	SbP (cmconr)	SbP (ulpconr)	SbP (cmconr)
+30	4	00000000	00000000	00000000	00000000	00000000	00000000
+34	4	00000000	00000000	00000000	00000000	00000000	00000000
Protocol Data Unit Header							
+38	2	0100	0100	0100	0100	0100	0100
+3A	2	0023 (Note 8)	0024 (Note 8)	002B (Note 8)	002C (Note 8)	SbP (Note 8)	0015 (Note 8)
+3C	2	0000	0000	0000	0000	0000	0000
+3E	2	0040 (Note 7)	0040 (Note 7)	0040 (Note 7)	0040 (Note 7)	0040 (Note 7)	0040 (Note 7)
Protocol Data Unit							
+40	2	000C (itemlen)	000C (itemlen)	000C (itemlen)	000C (itemlen)	Note 9	000C (itemlen)
+42	2	41 (OSA cmd)	41 (OSA cmd)	41 (OSA cmd)	41 (OSA cmd)		43 (QDIO cmd)
+43	1	02 (ENABLE)	04 (SETUP)	02 (ENABLE)	04 (SETUP)		60 (ACTIVATE)
+44	2	0017 (Note 12)	0018 (Note 12)	001F (Note 12)	0020 (Note 12)		0009 (Note 12)
+46	2	0000	0000	0000	0000		0000

Disp. (hex)	Len.	CM_ENABLE	CM_SETUP	ULP_ENABLE	ULP_SETUP	IPA	DM_ACT
+48	4	00000000	00000000	00000000	00000000		00000000
+4C	2	000B (itemlen)	0009 (itemlen)	000B (itemlen)	0009 (itemlen)		0009 (itemlen)
+4E	1	04 (desc)	04 (desc)	04 (desc)	04 (desc)		04 (desc)
+4F	1	01 (desc)	04 (desc)	01 (desc)	04 (desc)		04 (desc)
+50	1	7E	05 (desc)	SbP (Note 1)	05 (desc)		05 (desc)
+51	1	04 (desc)	00 (undoc tok)	04 (desc)	SbP (ulpconw)		SbP (ulpconr)
+52	1	05 (desc)	01 (undoc tok)	05 (desc)	SbP (ulpconw)		SbP (ulpconr)
+53	2	SbP (cmfilw) / Resp (cmfilr)	0111 (undoc tok)	SbP (ulpfilw)/ Resp (ulpfilr)	SbP (ulpconw)		SbP (ulpconr)
+55	2	SbP (cmfilw) / Resp (cmfilr)	0009 (itemlen)	SbP (ulpfilw) / Resp (ulpfilr)	0009 (itemlen)		
+57	2	000C (itemlen)	0405 (desc)	0014 (Note 2)	0405 (desc)		
+59	1	04 (desc)	05 (desc)	04 (desc)	05 (desc)		
+5A	1	02 (desc)	Resp (cmconr)	0A (desc)	SbP (ulpfilr) / Resp (ulpconr)		
+5B	1	SbP (user)	Resp (cmconr)	SbP (portno)	SbP (ulpfilr) / Resp (ulpconr)		
+5C	2	SbP (user)	Resp (cmconr)	2000	SbP (ulpfilr) / Resp (ulpconr)		
+5E	1	SbP (user)	00 (itemlen)	00	00 (itemlen)		
+5F	1	SbP (user)	06 (itemlen)	Resp (Note 3)	06 (itemlen)		
+60	1	SbP (user)	04 (desc)	Resp (Note 3)	04 (desc)		
+61	1	SbP (user)	06 (desc)	00	06 (desc)		
+62	1	SbP (user)	C8	SbP (portname ll)	40		
+63	1		00	SbP (portname)	00		
+64	2			SbP (portname)	0008 (item len)		
+66	2			SbP (portname)	040B (desc)		
+68	2			SbP (portname)	SbP (devno)		
+6A	1			SbP (portname)	SbP (rcua)		
+6B	1			Resp (Note 4)	SbP (rcua)		

Notes:

1. Protocol Type: 0x08 = Layer2, 0x03 = TCP/IP, 0x0A = NCP
2. If this item length field in the ULP_ENABLE response is greater than 24, then the link type response is present in byte 107.
3. For Hipersocket only, Maximum MTU values: 0x4000=8192, 0x6000=16384, 0xA000=32768, 0xFFFF=57344

4. This field, if present in the response, identifies the link type.

   0x01= Fast (100 Mbps) Ethernet,

   0x02 = High Speed Token Ring,

   0x03 = Gigabit Ethernet,

   0x04 = NCP,

   0x10 = 10 gigabit Ethernet,

   0x81 = LANE 100Mbps Ethernet,

   0x82 = LANE Token Ring,

   0x88 = LANE

   0x90 = Native ATM

5. The length of the transport header. Linux only uses the low order byte of the halfword,
6. Request header length. Linux only uses the low-order byte of the halfword.
7. Total length of message headers. Hence displacement to the PDU content of the message, that is, the actual request or response from the targeted destination (the destination token). Linux only uses the low-order bytes of this length.
8. PDU content length. Length of the actual request or response data following the PDU header.
9. IP Assist commands use a different format than the other PDU's. See the section “IP Assist Commands (IPA)” for details.
10. Total length of the transport message.
11. Command data length. Length of the data following the initial command descriptor item.
12. Combined length of all data items of a command. This value does not include the length of the initial command descriptor item in which the value resides.

The handling of the information from one set of request and response to the next dictates that the requests and responses must flow in a specific sequence once the IDX_ACTIVATE exchanges have succeeded:

1. CM_ENABLE
2. CM_SETUP
3. ULP_ENABLE
4. ULP_SETUP

Various IPA commands may then follow. Activation is then finalized by the DM_ACT request/response sequence. Additional IPA commands may then be initiated during the process of creating the operational environment.

In addition to exchanging tokens, the primary role of these four exchanges by Linux is to validate the following configuration information.

Protocol discipline (Layer-2 vs. Layer-3) – ULP_ENABLE

Port number – ULP_ENABLE

Port name, if required – ULP_ENABLE

QDIO queues operational – DM_ACT

Maximum Transmission Unit (MTU) Size

The adapter type is initially identified based upon the SENSE ID CCW response. This will initially determine the maximum and initial default MTU for the adapter.

Card Type	Device Type	Device Model	Maximum MTU	Default MTU
OSA Express	0x1731	0x01	61440	1492
NCP	0x1731	0x05	61440	1500
Hipersocket	0x1731	0x06	57344	57344

For Hipersocket devices, the MTU values may be influenced by the ULP_ENABLE response. For this reason, Linux does not establish the MTU until the ULP_ENABLE response has been received.

While the MTU for the group device may be manipulated by an attribute, any settings are not communicated to the adapter itself in Layer-2 mode. Only in Layer-3 mode is the MTU communicated to the adapter to allow the adapter to properly fragment TCP segments.

IP Assist Commands (IPA)

The previous commands are exclusive to OSA Express operations. OSA Express predecessor, OSA or LAN Channel Station, used a set of IP Assist commands. A portion of the LCS header data is retained in OSA Express operation and a small subset of commands are shared with LCS.

The structure qeth_ipa_info (in qeth_core.h) uses two bit maps: one for the supported commands and one for the enabled commands. The structure qeth_card contains three instances of this structure (itself containing the two bit maps):

ipa4 – IPv4 assists (from QIPASSIST command with IPv4 protocol version)

ipa6 – IPv6 assists (from QIPASSIST command with IPv6 protocol version)

adp – adapter commands (from SETADPPARMS Query-Commands-Supported sub-command)

IP Assist commands contain a standard IP Assist Header followed by the specific command data for the respective command.

Each IP Assist command or response is part of a channel message and resides in the PDU portion of the channel command message.

IP Assist Header

Fields shared with LCS are identified in *italics*.

Disp. (hex)	Length	Description	Program Supplied Value
+00	1	*Command identifier*	See IPA Commands table below
+01	1	*Initiator identifier*	0x00 == Host initiated command (Note 1)
+02	2	*IP Assist sequence number*	Next number, then increment (Note 2)
+04	2	*Return code*	0x0000 == positive response
+06	1	*Adapter type*	0x01
+07	1	*Relative adapter number*	OSA port number
+08	1	Primary version number	0x02 for Layer 2 support, 0x01 otherwise
+09	1	Parameter count	always 0x01 for Linux
+0A	2	Protocol version	0x0004 = IPv4, 0x0006 = IPv6, or 0x0000
+0C	4	IP Assist supported	Set to all zeros
+10	4	IP Assist enabled	Set to all zeros

Note 1: This field identifies an IP assist reply when the message received contains either 0x00, for a host initiated request, or 0x81, for OSA initiated error reply. Normal OSA replies will likely contain 0x01. The assumption is that the high-order bit being set from the OSA adapter is a generic error response indicator. Linux only recognizes 0x81 and 0x00 as initiator values that may trigger directly an error message. Some other value is required to process the reply call back function, the assumption being that value is 0x01.

Note 2: Response sequence numbers are checked by Linux. The response sequence number must match sequence number of the original IP Assist command request.

IP Assist Commands

Commands shared with LCS are identified in italics. The two columns for Layer-2 and Layer-3 discipline identify the IP protocol version used in the IP Assist header with the command. "---" indicates the command is not used in this discipline.

Name	Identifier	Layer-2	Layer-3	Description
UNKNOWN	0x00	---	---	Not surprisingly, not used by Linux
STARTLAN	0x01	0x0000	0x0000	Start LAN operations
STOPLAN	0x02	---	0x0000	Stop LAN operations
SETVMAC	0x21	IPv4	---	Set Layer-2 MAC address

Name	Identifier	Layer-2	Layer-3	Description
DELVMAC	0x22	IPv4	---	Delete Layer-2 MAC address (use physical NIC address)
SETGMAC	0x23	IPv4	---	Set Layer-2 Group Multicast address
DELGMAC	0x24	IPv4	---	Delete Layer-2 Group Multicast address
SETVLAN	0x25	IPv4	---	Set Layer-2 VLAN
DELVLAN	0x26	IPv4	---	Delete Layer-2 VLAN
SETCCID	0x41	---	---	Used with OSN (NCP) subchannel
DELCCID	0x42	---	---	Used with OSN (NCP) subchannel
MODCCID	0x43	---	---	Used with OSN (NCP) subchannel
SETIP	0xB1	---	IPv4/6	Set Layer-3 IP unicast address
QIPASSIST	0xB2	---	IPv4/6	Query Layer-3 IP assist capability
SETASSPARMS	0xB3	---	IPv4/6	Set Layer-3 IP assist parameters
SETIPM	0xB4	---	IPv4/6	Set Layer-3 IP multicast address
DELIPM	0xB5	---	IPv4/6	Delete Layer-3 IP multicast address
SETRTG	0xB6	---	IPv4/6	Set Layer-3 routing information
DELIP	0xB7	---	IPv4/6	Delete Layer-3 IP unicast address
SETADPPARMS	0xB8	IPv4	IPv4	Various adapter directed sub-commands
SETDIAGASS	0xB9	---	0x0000	Set Layer-3 diagnostic assists
CREATEADDR	0xC3	---	IPv6	Create Layer-3 IPv6 address from Layer-2 MAC
DESTROYADDR	0xC4	---	IPv6	Destroy Layer-3 IPv6 address from Layer-2 MAC
REGLCLADDR	0xD1	---	---	Not used by Linux
UNREGLCLADDR	0xD2	---	---	Not used by Linux

As can be seen from the above header's fields shared with LCS and those that are not, OSA IP Assist command header explicitly identifies more information or changes the header field sizes. It is likely that the first thing that is done with the a command is to identify the logic that processes the command. If the command processing logic requires any of the additional header data or expects a specific parameter structure that differs from LCS, a different command is required. OSA non-QDIO commands follow LCS rules with LCS IP assist commands.

IP Assist Return Codes

IP Assist return codes are provided in the IP Assist header of the response to the program's command. The following table describes the return code and the commands expected to provide it.

Code	Description	Command
0x0000	Success	All may provide this return code

Code	Description	Command
0x0001	Command not supported for Layer-2 or Layer-3	
0x0002	Add address IP table full - IPv6	
0x0003	IP Assist command failed – unknown reason	Any
0x0004	Command not supported	
0x0005	Hipersocket trace already active	
0x0006	Invalid format	
0x0008	IPv6 address already registered remote	
0x0010	IPv6 address already registered locally	
0x0011	IP Address not registerd	
0x0012	No CCID's available	
0x0013	CCID not found	
0x0020	IP version incorrect	Note 1
0x0040	LAN and frame mismatch	
0x2003	Unsupported Layer-2 command	
0x2005	Duplicate MAC Address	SETVMAC, SETGMAC
0x2006	Layer-2 address table full	SETVMAC, SETGMAC, SETVLAN
0x200A	Duplicate with Layer 3 MAC	SETVMAC
0x200C	Layer-2 MAC not authorized by hypervisor	SETVMAC
0x200D	Layer-2 MAC not authorized by adapter	SETVMAC
0x2015	Layer-2 invalid VLAN id	DELVLAN
0x2016	Layer-2 duplicate VLAN id	SETVLAN
0x2017	Layer-2 VLAN id not found	DELVLAN
0xE001-0xE00D	Various Layer-3 related return codes	
0xE00E	Unsupported assist sub-command	SETADPPARMS
0xE00F	Multicast address already defined	
0xE080	Layer-3 LAN offline	Any
0xF001	Invalid IP assist header protocol version	Any (Note 1)
0xFFFF	Unknown error	Any

Note 1: In which scenario return code 0x0020 vs 0xF001 is unclear. Future analysis of IP Assist Layer-3 messaging may make this clear.

STARTLAN (0x01)

Start LAN operations. Requires IP Assist Header protocol version to be set to 0x0000. No command data is associated with STARTLAN. Successful completion will result in the driver

reporting to Linux that the network device carrier is “on.”

STOPLAN (0x02)

Stops LAN operations. Requires IP Assist Header protocol version to be to 0x0000. No command data is associated with STOPLAN. Successful completion will result in the driver reporting to Linux that the network device carrier is “off.” STOPLAN is only used with Layer-3 discipline.

SETVMAC (0x21), DELVMAC (0x22), SETGMAC (0x23) and DELGMAC (0x24)

Set or delete a MAC address.

Disp. (hex)	Length	Description
+14	4	Length of MAC Address (6)
+18	6	MAC Address

SETVMAC Return Codes: 0x2005, 0x2006, 0x200A, 0x200C, 0x200D

DELVMAC Return Codes: None

SETGMAC Return Codes: 0x2005, 0x2006, 0x200A, 0x200C, 0x200D

DELGMAC Return Codes: None

SETVLAN (0x25) and DELVLAN (0x26)

Add or delete a VLAN from the adapter interface.

Disp. (hex)	Length	Description
+14	2	VLAN Id

The IPA IP protocol version is set to IPV4 for this command.

SETVLAN Return Codes: 0x2006, 0x2016

DELVLAN Return Codes: 0x2015, 0x2017

SETIP (0xB1) and DELIP(0xB7)

Add or delete an IPv4 or IPv6 address. Which structure is in use is determined by the IP Assist Header Protocol Version field.

For IPv4:

Disp. (hex)	Length	Description
+16	4	IPv4 address
+18	4	IPv4 subnet mask

Disp. (hex)	Length	Description
+1C	4	Flags

For IPv6:

Disp. (hex)	Length	Description
+14	16	IPv6 address
+24	16	IPv6 mask
+34	4	Flags

Only the byte of the 4-byte flags field is used by Linux:

0x00000000 – normal default setting for DELIP and SETIP

0x00000001 – SETIP VIPA flag (no gratuitous ARP)

0x00000002 – SETIP VIPA takeover flag (no failure on gratuitous ARP)

0x00000020 – DELIP address to be taken over

0x00000040 – DELIP VIPA flag

0x00000080 – DELIP address needs SETIP

SETADPPARMS (0xB8)

Performs a set of subcommands. A standard header is used for all subcommands as described below. Support by the OSA is assumed for Layer-2 operations. This suggests this adapter command was made available with version 2 OSA, a prerequisite for Layer-2 operations. In Layer-3 operations, the QIPASSIST command will indicate whether this command is available or not.

Disp. (hex)	Length	Description
+14	4	Supported hardware commands
+18	4	reserved
+1C	2	SETADPPARMS Command length (this header plus sub-command data)
+1E	4	Sub-command code
+22	2	Sub-command return code (Note 1)
+24	1	Used total (always set to 0x01 by Linux)
+25	1	Sequence number (always set to 0x01 by Linux)
+26	4	reserved

Note 1: Linux treats the sub-command return code as if it were the IPA return code. So from

the list of recognized return codes it is not possible to identify the return codes truly returned by a SETADPPARMS IPA command sub-command. Although, one return code stands out: 0xE00E – unsupported assist sub command.

SETADPPARMS Return Codes: 0xE00E

Query-Commands-Supported Sub-Command (0x00000001)

Response indicates the LAN Type and supported SETADPPARMS sub-commands.

Disp. (hex)	Length	Description
+2A	4	Number of LAN types supported
+2E	1	LAN Type response (see Note 1)
+2F	1	reserved
+30	4	Supported commands bit map response (see Note 2)
+34	8	reserved

Note 1: If not zeros, this field contains the LAN Type response. It uses the same values as Link Type in ULP_ENABLE extended response, Note 4 above.

Note 2: The sub commands are identical to the bit-map of supported commands. This bit map is used to determine which commands may be issued. The following bit settings are associated with the following commands:

0x00000001 – Query-Commands-Supported sub-command

0x00000002 – Alter MAC Address (Layer-3 only)

0x00000004 – Add/Delete Group Address – not used by Linux

0x00000008 – Add/Delete Functional Address – not used by Linux

0x00000010 – Set Addressing Mode – not used by Linux

0x00000020 – Set Config Parms – not used by Linux

0x00000040 – Set Config Parms Extended – not used by Linux

0x00000080 – Set Broadcast Mode (Layer-3 only)

0x00000100 – Send OSA Message – not used by Linux

0x00000200 – Set SNMP Control (Layer-3 only)

0x00000400 – Query Card Information – not used by Linux

0x00000800 – Set Promiscuous Mode

0x00002000 – Set Diagnostic Assists (Layer-3 only)

0x00010000 – Set Access Control

Set Promiscuous Mode Sub-Command (0x00000800)

Turn promiscuous mode on or off. A non-zero sub-command return code indicates that promiscuous mode was unable to be initialized.

Disp. (hex)	Length	Description
+2A	4	Promiscuous mode: 0x00000000 == off, 0x00000001 == on

Promiscuous mode is used with multicast.

Data Device Buffer Formats

Four different buffer headers are used in QDIO data device buffers, identified by the first byte of the buffer:

0x01 – Layer-3, see “Layer-3 Buffers” below,

0x02 – Layer-2, see “Layer-2 Buffers” below,

0x03 – Layer-3 with TCP Segmentation Offload, or

0x04 – OSN, not documented

Regardless of the card discipline, the Hipersocket trace will cause Layer-2 frames to be used as well as Layer-3.

Layer-2 Buffers

Layer-2 buffers transfered over the QDIO interface contain a Layer-2 OSA header followed by the Layer-2 frame content. The following is the structure of the Layer-2 OSA header.

Disp. (hex)	Length	Name	Description
+00	1	id	0x02 for Layer-2
+01	3	flags	See “Layer-2 Buffer Flags” below
+04	1	portno	Adapter port number
+05	1	hdr_length	0x20 (for 32-byte Layer-2 header)
+06	2	pkt_length	Length of MAC frame in buffer (not including hdr_length)
+08	2	seq_no	Not used by Linux
+0A	2	vlan_id	If byte 3 bit 3 is one, the VLAN id of the MAC frame, zeros otherwise
+0C	20		Reserved, must be zeros

z/VM VSWITCH relies upon the *vlan_id* information in the Layer-2 buffer. Native OSA adapter relies upon VLAN information in the Ethernet frame when the protocol is 0x8100 in the MAC header.

Layer-2 Buffer Flags

Byte	Bits	Description
0	0-7	Not used, must be zeros
1	0-7	Not used, must be zeros
2	0-2	Not used, must be zeros
2	3	802.1Q VLAN information present in MAC frame

Byte	Bits	Description
2	4	Not used, must be zero
2	5	MAC frame is unicast
2	6	MAC frame is broadcast
2	7	MAC frame is multicast

Layer-3 Buffers

Layer-3 buffers transfered over the QDIO interface contain a Layer-3 OSA header followed by the Layer-3 IP packet content. The following is the structure of the Layer-3 OSA header when TCP Segmentation Offload is not in use.

Disp. (hex)	Length	Name	
+00	1	id	0x01 for Layer-3
+01	1	flags	See “Layer-3 Buffer Flags” below
+02	2	in_cksum	Inbound checksum
+04	4	token	Not used by Linux
+08	2	length	Packet size (not including this Layer-3 header)
+0A	1	vlan_prio	Not used by Linux
+0B	1	ext_flags	See “Layer-3 Extended Buffer Flags” below
+0C	2	vlan_id	If ext_flags bit 5 ==1, contains VLAN id, otherwise zeros
+0E	2	frame_offset	Note 1
+10	16	dest_addr	Destination Layer-3 address (IPv4 – bytes 2-5, IPv6 – bytes 2-9) Note 2

Note 1: The frame-offset likely differs between Layer-3 buffers with and without the TCP Segmentation Offload feature. Because Linux does not use this value, it is not possible to determine from where the offset is relative. It is likely from the start of the buffer.

Note 2: Because only data starting at byte 2 of the dest_addr field is used, it is possible that a length field is present in the first two bytes indicating the size of the data present. If such a field is actually present, Linux does not use it.

Layer-3 Buffer Flags

Byte	Bits	Description
0	0	`0` -> IPv4, `1` -> IPv6
0	1, 2	Not used, must be zeros
0	3	Pass through packet

Byte	Bits	Description
0	4	Not used, must be zero
0	5-7	Cast Description: `000` -> no cast `100` -> multicast `101` -> broadcast `110` -> unicast `111` -> anycast

Layer-3 Extended Buffer Flags

Byte	Bits	Description
0	0	Not used, must be zero
0	1	`0` -> TCP with TCP Segmentation Offload, `1` -> UDP
0	2	Transport checksum : 0 -> not checked by hardware, 1-> checked by hardware
0	3	Packet header checksum: 0 -> not checked by hardware, 1-> checked by hardware
0	4	External Source MAC address present in dest_addr bytes 2-15
0	5	VLAN Tag included in header (VLAN id in vlan_id field)
0	6	Token ID
0	7	VLAN ID in Frame (VLAN id in bytes 12 and 13 of dest_addr field)

Layer-3 Buffers with TCP Segmentation Offload

An additional header is inserted between the IP Layer-3 packet data and the Layer-3 Header. To be documented with Layer-3 processing.

OSA Logical Structure

The following diagram gives a conceptual overview of the OSA QDIO functionality as exposed to the program by the various messages and structures that participate in its operation. Three interfaces are utilized:

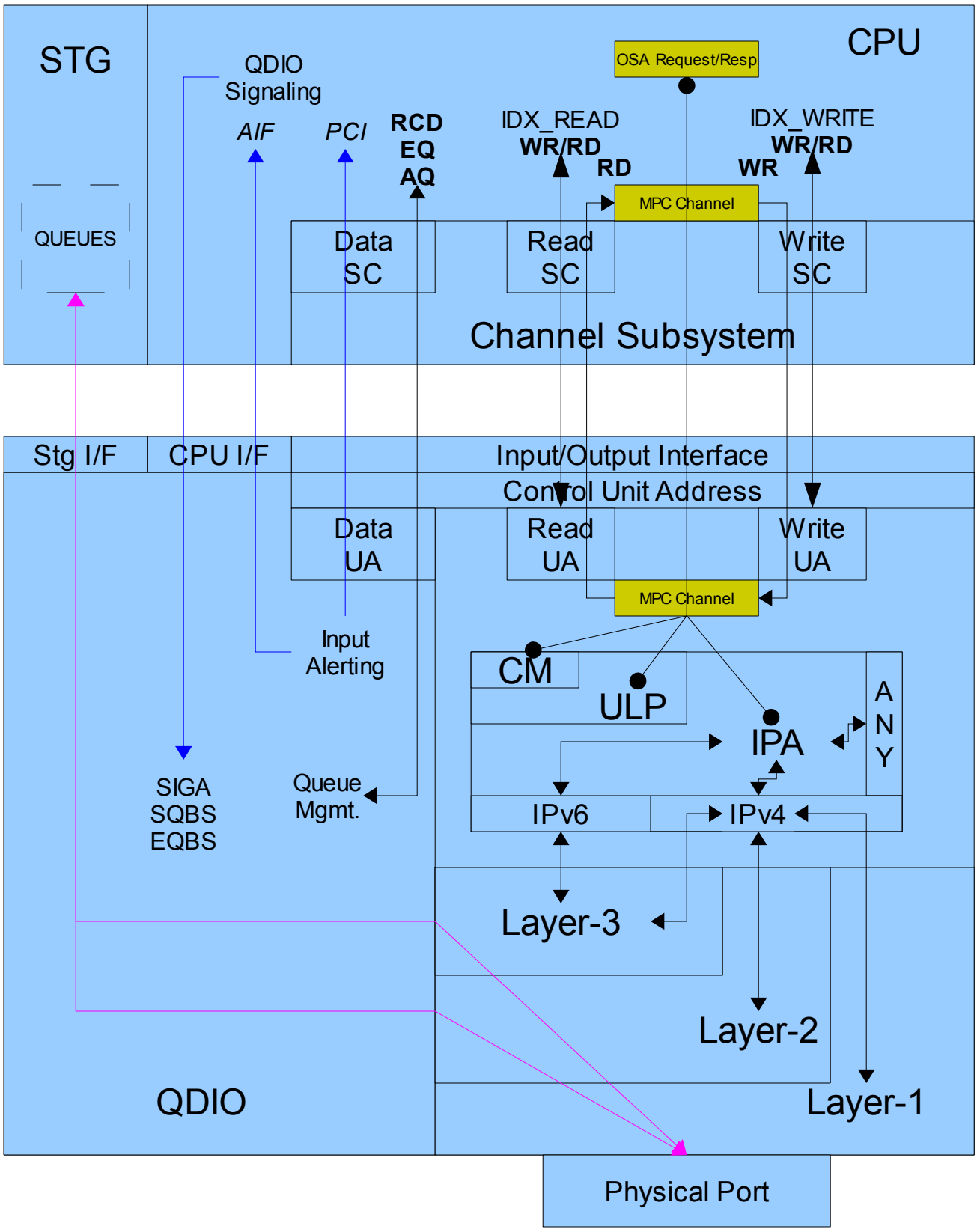
A main storage interface for queue and queue related data transfers (in magenta),

A CPU interface that provides communication between the CPU (in blue), and

an Input/Output interface that uses CCW. The CCW's are in **bold font**.

Note that the details of the QDIO operations are documented separately in *Linux QDIO*.

The upper portion of the diagram represents the CEC and the lower portion represents the OSA adapter that physically resides in an I/O cage of the mainframe system.


STG
CPU
QDIO
Signaling
AIF
PCI
RCD
EQ
AQ
OSA Request/Resp
IDX_READ
WR/RD
RD
IDX_WRITE
WR/RD
WR
MPC Channel
QUEUES
Data
SC
Read
SC
Write
SC
Channel Subsystem
Stg I/F
CPU I/F
Input/Output Interface
Control Unit Address
Data
UA
Read
UA
Write
UA
MPC Channel
Input
Alerting
CM
ULP
IPA
ANY
SIGA
SQBS
EQBS
Queue
Mgmt.
IPv6
IPv4
Layer-3
Layer-2
Layer-1
QDIO
Physical Port

Appendix A - Linux QETH Components

The Queued Input/Output Ethernet facility operates with an Open System Adapter for network access. Input and Output may be use either Layer 3 IP packets or Layer 2 Ethernet frames. Packets or frames transmitted to or from an adapter connection use buffers. Each buffer is referenced by a number between 0 and 255, inclusive.

The primary Linux network device structure, in `linux/include/linux/netdevice.h`, is net_device. This structure contains a pointer to the mid-layer private data. For QETH, the mid-layer private data is contained in the structure `qeth_card` defined in `drivers/s390/net/qeth_core.h`.

The Linux QETH support is provided by four source modules:

drivers/s390/net/qeth_core_main.c

drivers/s390/net/qeth_core_mpc.c

drivers/s390/net/qeth_core_sys.c – OSA sysfs attribute support

drivers/s390/net/qeth_l2_main.c

drivers/s390/net/qeth_l3_main.c

drivers/s390/net/qeth_l3_sys.c – OSA sysfs Layer-3 attribute support

Drivers

Three device drivers are used by the Linux QETH implementation:

ccwgroup

qeth_l2 and

qeth_l3.

A set of driver functions are provided. The following table illustrates the relationships between the drivers.

Function	ccwgroup	qeth_l2	qeth_l3
probe	qeth_core_probe_device	qeth_l2_probe_device	qeth_l3_probe_device
remove	qeth_core_remove_device	qeth_l2_remove_device	qeth_l3_remove_device
set_online	qeth_core_set_online	qeth_l2_set_online	qeth_l3_set_online
set_offline	(wrapper for qeth_lx)	qeth_l2_set_offline	qeth_l3_set_offline
shutdown	(wrapper for qeth_lx)	qeth_l2_shutdown	qeth_l3_shutdown
prepare	(wrapper for qeth_lx)	Not implemented	Not implemented
complete	(wrapper for qeth_lx)	Not implemented	Not implemented
freeze	(wrapper for qeth_lx)	qeth_l2_pm_suspend	qeth_l3_pm_suspend

Function	ccwgroup	qeth_l2	qeth_l3
thaw	(wrapper for qeth_lx)	qeth_l2_pm_resume	qeth_l3_pm_resume
restore	(wrapper for qeth_lx)	qeth_l2_pm_resume	qeth_l3_pm_resume

Device Attributes

Each group device has a set of attributes. The sysfs attribute management is provided by two modules: qeth_core_sys.c (layer 2 attributes and attributes common to both layer 2 and 3) and qeth_l3_sys.c (layer 3 specific attributes).

The following table describes the device attributes supported by qeth_core_sys.c.

Attribute	Read	Write
state	card.state card->lan_online	
chpid	card->info.chpid	
if_name	QETH_CARD_IFNAME(card)	
card_type	qeth_get_cardname_short	
inbuf_size	card->qdio.in_buf_size	
portno	card->info.portno	card->info.portno
portname	card->info.portname_required card->info.portname	card->info.portname
priority_queueing	card->qdio.do_priority_queueing card->qdio.default_out_queue	card->qdio.no_output_queues card->qdio.do_priority_queueing card->qdio.default_out_queue
buffer_count	card->qdio.in_buf_pool.buf_count	card->qdio.in_buf_pool.buf_count qeth_realloc_buffer_pool(card,cnt)
recover		qeth_schedule_recovery(card)
performance_stats	card->options.performace_stats	card->options.performace_stats
layer2	card->option.layer2	card->discipline.ccwgdriver->remove(card->gdev) qeth_core_free_discipline(card) qeth_core_load_discipline(card) card->discipline.ccwgdriver->probe(card->gdev)
isolation	card->options.isolation	card->options.isolation
blkt/total	card->info.blkt.time_total	card->info.blkt.time_total
blkt/inter	card->info.blkt.inter_packet	card->info.blkt.inter_packet
blkt/inter_jumbo	card->info.blkt.inter_packet_jumbo	card->info.blkt.inter_packet_jumbo

The following table describes the attributes supported by qeth_l3_sys.c.

Attribute	Read	Write
route4	card->options.route4	card->options.route4 qeth_l3_setrouting_v4
route6	card->options.route6	card->options.route6 qeth_l3_setrouting_v6
fake_broadcast	card->options.fake_broadcast	card->options.fake_broadcast
broadcast_mode (Token-Ring)	card->options.broadcast_mode	card->options.broadcast_mode
canonical_macaddr (Token-Ring)	card->options.macaddr_mode	card->options.macaddr_mode
checksumming	card->options.checksum_type	qeth_l3_set_rx_csum(card,csum_type)
large_send	card->options.large_send	qeth_l3_set_large_send(card, type)
ipa_takeover/enable	card->ipato.enabled	card->ipato.enabled
ipa_takeover/invert4	card->ipato.invert4	card->ipato.invert4
ipa_takeover/add4	card->ipato.entries	qeth_l3_add_ipato_entry(card,ipatoe)
ipa_takeover/del4		qeth_l3_del_ipato_entry(card,proto,addr,mask_bits)
ipa_takeover/invert6	card->ipato.invert6	card->ipato.invert6
ipa_takeover/add6	card->ipato.entries	qeth_l3_add_ipato_entry(card,ipatoe)
ipa_takeover/del6		qeth_l3_del_ipato_entry(card,proto,addr,mask_bits)
vipa/add4	card->ip_list	qeth_l3_add_vipa(card,proto,addr)
viap/del4		qeth_l3_del_vipa(card,proto,addr)
vipa/add6	card->ip_list	qeth_l3_add_vipa(card,proto,addr)
vipa/del6		qeth_l3_del_vipa(card,proto,addr)
rxip/add4	card->ip_list	qeth_l3_add_rxip(card,proto,addr)
rxip/del4		qeth_l3_del_rxip(card,proto,addr)
rxip/add6	card->ip_list	qeth_l3_add_rxip(card,proto,addr)
rxip/del6		qeth_l3_del_rxip(card,proto,addr)

Linux OSA Layer-2 Ethernet Adapter Management

Step 1 – Kernel Initialization

During kernel initialization the various drivers and modules are registered with the Linux kernel.

Step 2 – I/O Probes

The I/O probe process identifies the type of device by means of the SENSE ID. The SENSE

ID information ties the subchannel to the appropriate ccw device driver. At this stage the OSA devices will be associated with the *qeth_ccw_driver*. The devices are not yet part of a group.

Step 3 – User Configuration

Configuration of the adapter is performed by various values being written to the sysfs file system thereby storing in the adapter structure the information for later reference. The following steps are initiated by setting the group device online.

Refer to the appropriate *Device Drivers, Features and Commands* manual for details on user configuration.

Bash scripts are provided to assist with this configuration. Such scripts convert the options supplied by the user into sysfs attributes that are read or written.

Creating the Group Device

The first step is to create the root device from the read device as the groups root device. This is accomplished by writing a string to the read devices "group" attribute the list of bus id's of the individual device numbers of the participating subchannels. The group device is created in *ccwgroup.c/ccwgroup_create_from_string()*. The group device is assigned to the *qeth_core_ccwgroup_driver*. When the group device is added it will automatically be probed by the Linux driver core using the function *qeth_core_probe_device*. This processing is common to both Layer-2 and Layer-3 operation.

ccwgroup.c/ccwgroup_create_from_string

- qeth_core_probe_device()
  - qeth_determine_card_type() - Read device previously queried SENSE ID data inspected
  - qeth_core_create_device_attributes() - Create group sysfs attributes (qeth_core_sys.c)
  - qeth_determine_capabilities()
    - ccw_device_set_online() - **Enable subchannel in SCHIB of the data device (preferably with concurrent sense)**
    - qeth_read_conf_data() - **Issue RCD to data device**
    - qdio_get_ssqd_desc() - **Issue CHSC to acquire queue descriptor**

Configuring the Group Device

Additional attributes can now be written to the sysfs group device entry. One of the most important is the one specifying the discipline to be used with the OSA: Layer-2 vs. Layer-3. Layer-3 is the default discipline for the OSA. Setting the discipline must occur before the group device is set online. Setting the discipline has the effect of identifying the group device driver, "qeth_l2" or "qeth_l3", for which the ccwgroup driver acts as a wrapper.

Setting the Group Device Online

Writing "1" to the group device "online" attribute has the effect of calling the *set_online*

function of either the qeth_l2 or qeth_l3 driver. This function triggers the next step in the initialization process.

Step 4 – Bringing the Layer-2 Group Device Online

The following functions are driven by setting the group device on line. At the completion of this step the network device has been created in Linux, eth0, for example. Further initialization occurs through the Linux network device, for example, eth0, in step 5.

ccwgroup set_online()

qeth_core_set_online()

- qeth_l2_probe_device()
- qeth_l2_set_online()
  - __qeth_l2_set_online()
  - **/* Initialize Transport Operations */**
    - qeth_core_hardsetup_card()
      - ccw_device_set_offline() - data device set offline
      - ccw_device_set_offline() - write device set offline
      - ccw_device_set_offline() - read device set offline
      - ccw_device_set_online() - read device set online
      - ccw_device_set_online() - write device set online
      - ccw_device_set_online() - data device set online
      - qeth_qdio_clear_card()
        - qdio_cleanup() - **CLEAR I/O or HALT I/O issued to data device**
      - qeth_idx_activate_channel() - **IDX_ACTIVATE for read device**
      - qeth_idx_activate_channel() - **IDX_ACTIVATE for write device**
  - **/* Transport Operation Initialized */**
  - **/* Setup Adapter Hardware */**
    - qeth_mpc_initialize()
      - qeth_issue_next_read() - prime the channel read device
      - qeth_cm_enable() - **Send CM_ENABLE to channel**
      - qeth_cm_setup() - **Send CM_SETUP to channel**
      - qeth_ulp_enable() - **Send ULP_ENABLE to channel**
      - qeth_ulp_setup() - **Send ULP_SETUP to channel**
      - qeth_alloc_qdio_buffers() - Allocate queues in storage
      - qeth_qdio_establish()
        - qdio_initialize()

qdio_establish() - **EQ to data device**

qeth_qdio_activate()

qdio_activate() - **AQ to data device**

qeth_dm_act() - **Send DM_ACT to channel**

/* Adapter Hardware Setup Complete */

/* Setup Adapter Software */

qeth_l2_setup_netdev – establish the Linux Ethernet device (ethn)

qeth_l2_request_initial_mac

qeth_query_setadapterparms – **Send to channel IPA_SETADP_QUERY_COMMANDS_SUPPORTED**

FOR OSA: generate a random MAC address

FOR HYPERSOCKET OR GUEST LAN: query MAC with adapter command

qeth_l2_send_setmac – **Send to channel IPA_CMD_SETVMAC**

qeth_send_startlan – **Send to channel IPA_CMD_STARTLAN**

qeth_set_access_ctrl_online – If supported, Send to channel IPA_SETADP_SET_ACCESS_CONTROL

For each VLAN in a list either send IPA_CMD_DELVLAN or IPA_CMD_SETVLAN

Initialize the QDIO processing – **SIGA** usage starts

/* Adapter Software Setup Complete */

At this point the physical interface is operational, although the logical interface is not. Packets will not be sent from the IP stack because the interface is not UP and any packets received from the network interface will be discarded, also because the interface is not UP.

Step 5 – Bringing the Layer-2 Ethernet Interface Up

Various attributes can be applied to the Ethernet interface, some of which are backed by the qeth_l2 or qeth_l3 driver. A set of network device operations are supported by the qeth_l2 group driver. Various tools are used in user space to configure the interface: ifconfig, ethtool, miitool, and multicast supporting applications.

Operation	Tool	Function	Description
open	ifconfig	qeth_l2_open	Does netif_start_queue (Linux processes frames)
stop	ifconfig	qeth_l2_stop	Does netif_tx_disable (Linux sends no frames)
get_stats		geth_get_stats	
hard_start_xmit		qeth_l2_hard_start_xmit	Sends a frame via data device QDIO.
validate_addr		eth_validate_addr	
set_multicast_list		qeth_l2_set_multicast_list	
do_ioctl	ethtool	qeth_l2_do_ioctl	
set_mac_address	ifconfig	qeth_l2_set_mac_address	Uses DELVMAC and SETVMAC

Operation	Tool	Function	Description
change_mtu	ifconfig	qeth_change_mtu	
vlan_rx_add_vid	vconfig	qeth_l2_vlan_rx_add_vid	Uses SETVLAN
vlan_rx_kill_vid	vconfig	qeth_l2_vlan_rx_kill_vid	Uses DELVLAN
tx_timeout		qeth_tx_timeout	Called by watchdog timer

Promiscuous mode is a network device flag set or reset by ifconfig. The SETADPPARMS Set Promiscuous Mode Sub-command is used to change the adapter interface status.

Step 6 – Bringing the Layer-2 Ethernet Interface Down

When the Ethernet interface is taken down, Linux will call the network device stop function. This has the effect of calling netif_tx_disable, causing Linux to cease sending frames to the OSA group device. Linux will mark the network device as down. At this point the OSA group device is still online. Frames from the network will continue to be received by the qeth_l2 driver. When the driver sees the network device as down, these inbound packets or frames will be discarded.

Step 7 – Taking the Layer-2 Group Device Offline

When a value is written to the group device attribute online, managed by the ccwgroup driver in cio/ccwgroup.c, the group driver set_online or set_offline function is called, depending upon the value. By writing a “0” to online, the ccwgroup driver's set_offline function is called, qeth_l2_offline.

qeth_l2_set_offline()

__qeth_l2_set_offline()

netif_carrier_off() informs the network device that the phsyical interface is down

qeth_l2_stop card()

dev_close() closes the network device

qeth_l2_send_delmac – **Use DELVMAC to channel**

qeth_l2_process_vlans – with clear flag for all active VLANS

qeth_l2_send_setdelvlan – **Use DELVLAN to channel**

qeth_qdio_clear_card()

qdio_cleanup() - cleanup data device using CLEAR I/O

qdio_shutdown() - **results in CLEAR I/O to data device**

Storage buffers freed

ccw_device_set_offline() - Data device set offline – **SCHIB set to disabled**

ccw_device_set_offline() - Write device set offline – **SCHIB set to disabled**

ccw_device_set_offline() - Read device set offline – **SCHIB set to disabled**

Inspection of the above indicates that, in addition to tearing down the configuration for the OSA adapter, the state is appropriately reflected to the Ethernet device as well.

Read/Write Device Transport Channel Handling

Following the successful IDX_ACTIVATE sequence with both the read and write devices, the two devices enter Transport Channel mode of operation. In transport mode, an adapter request is sent to the adapter by issuing a WRITE CCW to the write device subchannel, followed by the adapter completing the transfer of the response to storage and terminating either a previously initiated READ CCW or the next READ CCW issued to the read device subchannel.

The transmission of the adapter request is performed when the request is issued on the write device. Normal operation has a READ CCW issued to the read device subchannel waiting for completion most of the time. The time when this is not the case is immediately following the completion of the READ CCW. During the handling of the interrupt, a new READ CCW will be immediately issued to the read device. The driver, though must wait for the response from the adapter. A common routine is used to send requests to the adapter. This routine will either wait on a wait queue for the response to be presented or spin waiting for a time period to expire by inspecting the current jiffy count.

Each command has a set of command specific functions that initialize the request and that analyzes the response.

Appendix B - Module Relationships

The following table describes the relationship between the modules by identifying the functions exposed as exported symbols. Exposed symbols are in bold font. Functions that reference the exposed symbol are in regular font.

Line	core_main.c	l2_main.c	l3_main.c
50	**qeth_core_card_list**		
		qeth_l2_verify_device	qeth_l3_verify_device
		qeth_l2_netdev_by_devno	
	qeth_core_probe_device		
	qeth_core_remove_device		
	qeth_core_init		
52	**qeth_core_header_cache**		
		qeth_l2_hard_start_xmit	qeth_l3_hard_start_xmit
	__qeth_clear_output_buffer		
	qeth_core_init		
	qeth_core_exit		
181	**qeth_set_allowed_threads**		
		qeth_l2_stop_card	qeth_l3_stop_card
		qeth_l2_remove_device	qeth_l3_remove_device
		__qeth_l2_set_online	__qeth_l3_set_online
			qeth_l3_pm_suspend
			qeth_l3_pm_resume
195	**qeth_threads_running**		
	qeth_wait_for_threads	qeth_l2_set_multicast_list	qeth_l3_set_multicast_list
		qeth_l2_remove_device	qeth_l3_remove_device
		qeth_l2_pm_suspend	qeth_l3_pm_suspend
207	**qeth_wait_for_threads**		
		qeth_l2_vlan_rx_add_vid	
		qeth_l2_vlan_rx_kill_vid	qeth_l3_vlan_rx_kill_vid
		qeth_l2_set_mac_address	
214	**qeth_clear_working_pool_list**		
	qeth_realloc_buffer_pool	qeth_l2_stop_card	qeth_l3_stop_card
398	**qeth_clear_ipacmd_list**		
	qeth_send_control_data_cb	qeth_l2_stop_card	qeth_l3_stop_card

Line	core_main.c	l2_main.c	l3_main.c
	qeth_irq		
473	**qeth_release_buffer**		
	qeth_clear_cmd_buffers	qeth_osn_send_control_data	
	qeth_send_control_data_cb		
	qeth_idx_write_cb		
	qeth_idx_read_cb		
	qeth_prepare_control_data		
	qeth_send_control_data		
506	**qeth_wait_for_buffer**		
	qeth_cm_enable	qeth_osn_assist	
	qeth_cm_setup		
	qeth_ulp_enable		
	qeth_ulp_setup		
	qeth_dm_act		
	qeth_get_ipacmd_buffer		
508	**qeth_clear_cmd_buffers**		
	qeth_idx_activate_get_answer	qeth_l2_stop_card	qeth_l3_stop_card
	qeth_idx_activate_channel		
643	**qeth_clear_thread_start_bit**		
		qeth_l2_recover	qeth_l3_recover
654	**qeth_clear_thread_running_bit**		
		qeth_l2_recover	qeth_l3_recover
684	**qeth_do__run_thread**		
		qeth_l2_recover	qeth_l3_recover
694	**qeth_schedule_recovery**		
	qeth_issue_next_read	qeth_l2_qdio_input_handler	qeth_l3_input_handler
	qeth_check_ipa_data		
	qeth_flush_buffers		
	qeth_qdio_output_handler		
	qeth_tx_timeout		
	core_sys.c/qeth_dev_recover_store		
933	**qeth_clear_qdio_buffers**		
		qeth_l2_stop_card	qeth_l3_stop_card
		qeth_l2_shutdown	qeth_l3_shutdown
1279	**qeth_qdio_clear_card**		

Line	core_main.c	l2_main.c	l3_main.c
	qeth_mpc_initialize	qeth_l2_stop_card	qeth_l3_stop_card
	qeth_core_hardsetup_card	qeth_l2_shutdown	qeth_l3_shutdown
1630	**qeth_prepare_control_data**		
	qeth_send_control_data	qeth_osn_send_control_data	
1648	**qeth_send_control_data**		
	qeth_cm_enable		qeth_l3_send_ipa_arp_cmd
	qeth_cm_setup		
	qeth_ulp_enable		
	qeth_ulp_setup		
	qeth_dm_act		
	qeth_send_ipa_cmd		
	qeth_send_snmp_cmd		
2227	**qeth_print_status_message**		
		__qeth_l2_set_online	__qeth_l3_set_online
2326	qeth_init_input_buffer		
	qeth_init_qdio_queues		
2355	**qeth_init_qdio_queues**		
		__qeth_l2_set_online	__qeth_l3_set_online
2427	**qeth_get_ipacmd_buffer**		
	qeth_send_statstoplan	qeth_l2_send_setdelvlan	
	qeth_get_adapter_cmd	qeth_l2_send_setdelmac	
			qeth_l3_send_setdelmc
			qeth_l3_send_setdelip
			qeth_l3_send_setrouting
			qeth_l3_get_setassparms_cmd
			qeth_l3_query_ipassists
			qeth_l3_put_unique_id
			qeth_l3_iqd_read_initial_mac
			qeth_l3_get_unique_id
2441	**qeth_prepare_ipa_cmd**		
	qeth_send_ipa_cmd	qeth_osn_send_ipa_cmd	
2451	**qeth_send_ipa_cmd**		
	qeth_send_statstoplan		
	qeth_query_setadapterparms		

Line	core_main.c	l2_main.c	l3_main.c
	qeth_setad_promisc_mode		
	qeth_setadpparms_change_macaddr		
	qeth_setadpparms_set_access_ctrl		
		qeth_l2_send_setdelvlan	
		qeth_l2_send_setdelmac	
			qeth_l3_send_setdelmc
			qeth_l3_send_setdelip
			qeth_l3_send_setrouting
			qeth_l3_send_setadp_mode
			qeth_l3_send_setassparms
			qeth_l3_query_ipassists
			qeth_l3_put_unique_id
			qeth_l3_iqd_read_initial_mac
			qeth_l3_get_unique_id
2487	**qeth_send_startlan**		
		__qeth_l2_set_online	__qeth_l2_set_online
2498	**qeth_send_stoplan**		
			qeth_l3_stop_card
2514	**qeth_default_setadapterparms_cb**		
	qeth_query_setadapterparms_cb		qeth_l3_send_setadp_mode
	qeth_setadp_promisc_mode_cb		
	qeth_setadpparms_change_macaddr		
	qeth_setadpparms_set_access_ctrl_cb		
2545	**qeth_get_adapter_cmd**		
	qeth_query_setadpapterparms		qeth_l3_send_setadp_mode
	qeth_setadp_promisc_mode		
	qeth_setadpparms_change_macaddr		
	qeth_setadpparms_set_access_ctrl		
	qeth_snmp_command		
2563	**qeth_query_setadpterparms**		
		qeth_l2_request_initial_mac	qeth_l3_setadpter_parms
2576	**qeth_check_qdio_errors**		
	qeth_handle_send_error	qeth_l2_qdio_input_handler	qeth_l3_qdio_input_handler
2593	**qeth_queue_input_buffer**		
		qeth_l2_qdio_input_handler	qeth_l3_qdio_input_handler

Line	core_main.c	l2_main.c	l3_main.c
2881	**qeth_qdio_output_handler**		
		qeth_l2_probe_device	qeth_l3_probe_device
2922	**qeth_get_priority_queue**		
		qeth_l2_hard_start_xmit	qeth_l3_hard_start_xmit
2960	**qeth_get_elemets_no**		
		qeth_l2_hard_start_xmit	qeth_l3_hard_start_xmit
3096	**qeth_do_send_packet_fast**		
		qeth_l2_hard_start_xmit	qeth_l3_hard_start_xmit
3150	**qeth_do_send_packet**		
		qeth_l2_hard_start_xmit	qeth_l3_hard_start_xmit
3256	**qeth_setadp_promisc_mode**		
		qeth_l2_multicast_list	qeth_l3_multicast_list
3283	**qeth_change_mtu**		
		qeth_l2_netdev_ops	qeth_l3_netdev_ops
			qeth_l3_osa_netdev_ops
3306	**qeth_get_stats**		
		qeth_l2_netdev_ops	qeth_l3_netdev_ops
			qeth_l3_osa_netdev_ops
3337	**qeth_setadpparms_change_macaddr**		
		qeth_l2_request_initial_mac	qeth_l3_setadapter_parms
3481	**qeth_set_access_ctrl_online**		
	core_sys.c/qeth_dev_isolation_store	__qeth_l2_set_online	qeth_l3_start_ipassists
3508	**qeth_tx_timeout**		
		qeth_l2_netdev_ops	qeth_l3_netdev_ops
			qeth_l3_osa_netdev_ops
3518	**qeth_mdio_read**		
		qeth_l2_do_ioctl	qeth_l3_do_ioctl
3671	**qeth_snmp_command**		
		qeth_l2_do_ioctl	qeth_l3_do_ioctl
3848	**qeth_core_hardsetup_card**		
		__qeth_l2_set_online	__qeth_l3_set_online
3985	**qeth_core_get_next_skb**		
		qeth_l2_process_inbound_buffer	qeth_l3_process_inbound_buffer
4103	**qeth_sbf_longtext**		

Line	core_main.c	l2_main.c	l3_main.c
4431	**qeth_core_get_sset_count**		
		qeth_l2_ethtool_ops	qeth_l3_ethtool_ops
		qeth_l2_osn_ops	
4442	**qeth_core_get_ethtool_stats**		
		qeth_l2_ethtool_ops	qeth_l3_ethtool_ops
		qeth_l2_osn_ops	
4490	**qeth_core_get_strings**		
		qeth_l2_ethtool_ops	qeth_l3_ethtool_ops
		qeth_l2_osn_ops	
4504	**qeth_core_get_drvinfo**		
		qeth_l2_ethtool_ops	qeth_l3_ethtool_ops
		qeth_l2_osn_ops	
4522	**qeth_core_ethtool_get_settings**		
		qeth_l2_ethtool_ops	qeth_l3_ethtool_ops
		qeth_l2_osn_ops	
47	**qeth_dbf**		
1162		**qeth_l2_ccwgroup_driver**	
	qeth_core_load_discipline		
44			qeth_l3_set_large_send
			qeth_l3_sys.c/qeth_l3_set_large_send
581			qeth_l3_setrouting_v4
			qeth_l3_sys.c/qeth_l3_dev_route_store
701			qeth_l3_setrouting_v6
			qeth_l3_sys.c/qeth_l3_dev_route_store
742			qeth_l3_add_ipato_entry
			qeth_l3_sys.c/qeth_l3_dev_ipato_add_store
768			qeth_l3_ipato_entry
			qeth_l3_sys.c/qeth_l3_del_ipato_entry
792			qeth_l3_add_vipa
			qeth_l3_sys.c/qeth_l3_dev_vip

Line	core_main.c	l2_main.c	l3_main.c
			a_add_store
829			qeth_l3_del_vipa
			qeth_l3_sys.c/qeth_l3_dev_vipa_del_store
856			qeth_l3_add_rxip
			qeth_l3_sys.c/qeth_l3_dev_rxip_add_store
893			qeth_l3_del_rxip
			qeth_l3_sys.c/qeth_l3_dev_rxip_del_store
1468			qeth_l3_set_rx_csum
			qeth_l3_sys.c/qeth_l3_dev_checksum_store
3431			**qeth_l3_ccwgroup_driver**
	qeth_core_load_discipline		

The following diagram illustrates the relationship between key QETH structures. The table provides a high level reference of the structures.

Structure	Location	Description
net_device	`include/linux/netdevice.h`	Generic network interface
device	`include/linux/device`	Generic device
qeth_card	`drivers/s390/net/qeth_core.h`	QDIO card resources
qeth_channel	`drivers/s390/net/qeth_core.h`	Read/Write CCW support
ccw_device	`arch/s390/include/asm/ccwdev.h`	Generic CCW device
ccwgroup_device	`arch/s390/include/asm/ccwgroup.h`	Group device

Each of these structures is related to supporting Linux generic constructs, related driver constructs or the generic s390 Common Device Support utilized by the driver.

Linux	s390 Driver	Common Device Support
net_device	qeth_card	ccwgroup_device
device	qeth_channel	ccw_device

Structure Usage by qeth_core_main.c

Line	Function	net _device	qeth _card	ccwgroup _device	qeth _channel	ccw _device
109	qeth_get_cardname (doc)		R			
136	qeth_get_cardname_short		R			
226	qeth_alloc_buffer_pool (doc)		?			
257	qeth_realloc_buffer_pool (doc)		?			
273	qeth_issue_next_read (doc)		R		?	
303	qeth_alloc_reply (doc)		W			
329	qeth_issue_ipa_msg (doc)		R			
344	qeth_ipa_cmd (doc)		R			
398	qeth_clear_ipacmd_list		W			
438	qeth_setup_ccw		R		W	
453	__qeth_get_buffer				W	
473	**qeth_release_buffer**				W	
488	qeth_get_buffer				W	
499	**qeth_wait_for_buffer**				W	
508	**qeth_clear_cmd_buffers**				W	
519	qeth_send_control_data_cb (doc)		R		W	
598	qeth_setup_channel				W	
702	qeth_get_problem					R
749	__qeth_check_irb_error (doc)					R
785	qeth_irq		R		W	R
933	**qeth_clear_qdio_buffers** (doc)		R			
948	qeth_free_buffer_pool		W			
962	qeth_free_qdio_buffers		W			
996	qeth_is_1920_device		W			data
1024	qeth_init_qdio_info (doc)		W			
1036	qeth_set_initial_options (doc)		W			
1079	qeth_setup_card (doc)		W			
1125	qeth_core_sl_print (doc)		R			
1134	qeth_alloc_card (doc)		W			

Line	Function	net _device	qeth _card	ccwgroup _device	qeth _channel	ccw _device
1158	qeth_determine_card_type (doc)		W			read W
1188	qeth_clear_channel		W		R	W
1212	qeth_halt_channel		W		R	W
1235	qeth_halt_channels		R		all R	
1265	qeth_clear_halt_card		R			
1279	**qeth_qdio_clear_card** (doc)		W			data R
1310	qeth_read_conf_data (doc)		W			data W
1358	qeth_get_unitaddr (doc)		W			data
1380	qeth_init_tokens (doc)		W			
1389	qeth_init_func_level (doc)		W			
1410	qeth_idx_activitate_get_answer (doc)		W		W	W
1455	qeth_idx_activate_channel (doc)		W		W	data, write W
1538	qeth_idx_write_cb (doc)		R		W	write R
1576	qeth_idx_read_cb (doc)		R		W	read R
1630	**qeth_prepare_control_data** (doc)		W			
1648	**qeth_send_control_data** (doc)		W		W	write W
1740	qeth_cm_enable_cb (doc)		W			
1755	qeth_cm_enable (doc)		R			
1774	qeth_cm_setup_cb (doc)		W			
1790	qeth_cm_setup (doc)		R			
1811	qeth_get_initial_mtu_for_card (doc)		R			
1846	qeth_get_mtu_out_of_mpc (doc)		R			
1856	qeth_get_mtu_outof_framesize (doc)					
1872	qeth_mtu_is_valid		R			
1887	qeth_ulp_enable_cb (doc)		W			read
1930	qeth_ulp_enable (doc)		R			write
1965	qeth_ulp_setup_cb (doc)		W			read

Line	Function	net _device	qeth _card	ccwgroup _device	qeth _channel	ccw _device
1980	qeth_ulp_setup		R			write
2008	qeth_alloc_qdio_buffers (doc)		W			
2076	qeth_create_qib_param_field (doc)		R			
2089	qeth_create_qib_param_field_blkt (doc)		R			
2102	qeth_qdio_activate (doc)		R			data
2108	qeth_dm_act (doc)		R			write, read
2126	qeth_mpc_initialize (doc)		X			
2185	qeth_print_status_with_portname (doc)		R			
2206	qeth_print_status_no_portname (doc)		R			
2227	**qeth_print_status_message** (doc)		R			
2268	qeth_initialize_working_pool_list (doc)		R			
2280	qeth_buffer_pool_entry (doc)		X			
2326	qeth_init_input_buffer (doc)		X			
2355	**qeth_init_qdio_queues** (doc)		R			data
2407	qeth_fill_ipacmd_header (doc)		R			
2427	**qeth_get_ipacmd_buffer** (doc)		R		write	
2441	**qeth_prepare_ipa_cmd** (doc)		R			
2451	**qeth_send_ipa_cmd** (doc)					write, read
2475	qeth_send_startstoplan (doc)		R			
2487	**qeth_send_startlan** (doc)		R			
2498	qeth_send_stoplan (doc)		R			
2514	**qeth_default_setadapterparms_cb** (doc)					
2529	qeth_query_setadapterparms_cb (doc)					

Line	Function	net _device	qeth _card	ccwgroup _device	qeth _channel	ccw _device
2545	**qeth_get_adapter_cmd** (doc)		R			
2563	**qeth_query_setadapterparms** (doc)		R			write, read
2576	**qeth_check_qdio_errors** (doc)					
2593	**qeth_queue_input_buffer** (doc)					data
2657	qeth_handle_send_errors (doc)		R			
2881	**qeth_qdio_output_handler** (doc)		W			
2922	**qeth_get_priority_queue** (doc)		R			
2960	qeth_get_elements_no (doc)		R			
3096	**qeth_do_send_packets_fast** (doc)		R			
3150	**qeth_do_send_packet** (doc)		R			
3236	qeth_setadp_promisc_mode_cb (doc)		R			
3256	**qeth_setadp_promisc_mode** (doc)					Write, read
3283	**qeth_change_mtu** (doc)	W	R			
3306	**qeth_get_stats** (doc)	R	R			
3318	qeth_setadpparms_change_macaddr_cb (doc)		W			
3337	**qeth_setadpparms_change_macaddr** (doc)		R			Write, read
3358	qeth_setadpparms_set_access_ctrol_cb (doc)		W			
3455	qeth_setadpparms_set_access_ctrl (document)		R			
3481	**qeth_set_access_ctrl_online** (doc)		R			
3508	**qeth_tx_timeout**	R				
3518	**qeth_mdio_read**	R				
3583	qeth_send_ipa_snmp_cmd (doc)		R			Write, read

Line	Function	net _device	qeth _card	ccwgroup _device	qeth _channel	ccw _device
3607	qeth_snmp_command_cb		R			
3671	**qeth_snmp_command**					Write, read
3730	qeth_get_qdio_q_format (doc)		R			
3740	*qeth_qdio_establish (doc)*		R			
3811	qeth_core_free_card		W			
3848	***qeth_core_hardsetup_card*** (doc)		W			
3985	**qeth_core_get_next_skb** (doc)					
4147	qeth_core_load_discipline (doc)		W			
4171	qeth_core_free_discipline (doc)		W			
4180	*qeth_core_probe_device* (doc)		W	R		
4254	qeth_core_remove_device (doc)			W		
4279	*qeth_core_set_online* (doc)		W	l2/l3		
4302	qeth_core_set_offline (doc)		R	l2/l3		
4308	qeth_core_shutdown (doc)		R	l2/l3		
4316	qeth_core_prepare (doc)		R	l2/l3		
4325	qeth_core_complete (doc)		R	l2/l3		
4333	qeth_core_freeze (doc)		R	l2/l3		
4342	qeth_core_thaw (doc)		R	l2/l3		
4351	qeth_core_restore (doc)		R	l2/l3		
4504	qeth_core_get_drvinfo (doc)	R	R			

Linux QETH Implementation

Initialization of the QETH driver is performed via net/qeth_l2_main.c/qeth_l2_set_online. This function is identified in the CCW group driver structure.

Linux QETH support has two major components:

the OSA card that uses standard subchannels and

the Queued input/output mechanisms.

Appendix C - Linux Structures

Structure Values

QDIO_MAX_QUEUES_PER_IRQ – 32

QDIO_MAX_BUFFERS_PER_Q – 128

QDIO_MAX_ELEMENTS_PER_BUFFER – 16

SDIO_SBAL_SIZE – 256

drivers/s390/net/qeth_core.h

qeth_buffer_pool_entry

Structure	Key References
qeth_buffer_pool_entry	
list_head list	
list_head init_list	
elements *	[QDIO_MAX_ELEMENTS_PER_BUFFER] Pointers to the elements composing this buffer.

qeth_card

This structure has a one-to-one correspondence with the Linux IP layer interface structure.

The following summarizes the high-level components of this structure relating to QDIO.

Structure	Key References
qeth_card	
ccwgroup	
qeth_channel read	
qeth_channel write	
qeth_channel data	
qeth_card_info info	qeth_get_unitaddr, qeth_ulp_enable_cb, qeth_l2_setup_netdev,

Structure	Key References
	qeth_l3_setup_netdev
qeth_card_options option	qeth_l3_ipassists qeth_l3_probe_device qeth_l3_setrouting_v4 qeth_l3_setrouting_v6 qeth_l3_setadapter_hstr qeth_l3_default_setassparms_cb qeth_l3_query_ipassists_cb qeth_l3_start_ipa_checksum qeth_l3_start_ipa_tso
qeth_qdio_info qdio	
qeth_discipline discipline	

Appendix D - Linux Modules and Drivers

The following Linux modules are related to QETH support in Linux (in drivers/s390):

cio/ccwgroup.c – Generic CCW group device bus driver: ccwgroup

cio/device.c – Generic individual CCW device bus driver: ccw

cio/qdio_main.c – Generic QDIO group driver

cio/qdio_setup.c – Establishes queues.

cio/qdio_thinint.c – Adapter Interrupt Facility handler (Linux thin interrupts)

net/qeth_core_main.c – QETH module

net/qeth_l2_main.c – Layer 2 CCW group driver

net/qeth_l3_main.c – Layer 3 CCW group driver

Appendix E - Linux Functions

The following source modules support QDIO processing in Linux:

qeth_core_main.c – qeth generic module

qeth_core_mpc.c

qeth_core_sys.c

qeth_l2_main.c – qeth_l2 module, the CCW group device driver for Layer 2.

qeth_l3_main.c – qeth_l3 module, the CCW group device driver for Layer 3.

qeth_l3_sys.c

net/qeth_core_main.c

```
static struct ccwgroup_driver qeth_core_ccwgroup_driver = {
     .owner = THIS_MODULE,
     .name = "qeth",
     .driver_id = 0xD8C5E3C8,
     .probe = qeth_core_probe_device,
     .remove = qeth_core_remove_device,
     .set_online = qeth_core_set_online,
     .set_offline = qeth_core_set_offline,
     .shutdown = qeth_core_shutdown,
     .prepare = qeth_core_prepare,
     .complete = qeth_core_complete,
     .freeze = qeth_core_freeze,
     .thaw = qeth_core_thaw,
     .restore = qeth_core_restore,
};
```

qeth_core_init()

cio/drivers.c/ccw_driver_register – registers the 'ccw ' bus driver

cio/ccwgroup.c/ccwgroup_driver_register – registers the 'ccwgroup' bus driver

driver_create_file – create file 'group' in sysfs 'ccwgroup' directory

root_device_register – create root device in sysfs of 'qeth'

qeth_core_hardsetup_card()

qeth_mpc_initialize

qeth_qdio_establish

cio/qdio_main.c/**qdio_get_ssqd_desc** – issue CHSC to card data device

qeth_core_clear_card()

qeth_core_probe_device()

qeth_determine_card_type

qeth_is_1920_device

cio/device_ops.c/ccw_device_get_chp_desc

cio/chp.c/chp_get_chp_desc

net/qeth_l2_main.c

struct ccwgroup_driver qeth_l2_ccwgroup_driver = {

```
	.probe = qeth_l2_probe_device,
	.remove = qeth_l2_remove_device,
	.set_online = qeth_l2_set_online,
	.set_offline = qeth_l2_set_offline,
	.shutdown = qeth_l2_shutdown,
	.freeze = qeth_l2_pm_suspend,
	.thaw = qeth_l2_pm_resume,
	.restore = qeth_l2_pm_resume,
};
```

Function Call Tree

qeth_l2_set_online

	__qeth_l2_set_online

Linux Source Inventory

Various Linux components are provided in linux/drivers/s390. The following table identifies the Linux components implemented by each source file.

Source File	Module (module_init)	Subsystem (subsys_initcall)	Device Driver	Thin Interurpt Handler	Init (__initcall)
cio/blacklist.c					cio_ignore boot parm
cio/ccwgroup.c	Bus ccwgroup				
cio/chp.c		cio_chp workq			
cio/chsc.c					
cio/chsc_sch.c	chsc device				
cio/cio.c					
cio/cmf.c	s390cmf boot parm				
cio/crw.c			device_initcall		
cio/css.c cio/idset.c		Bus css			
cio/device.c cio/device_fsm.c cio/device_id.c cio/device_ops.c cio/device_pgid.c cio/device_status.c cio/fcx.c cio/ioasm.h cio/itcw.c cio/scsw.c		Bus ccw			
cio/isc.c					
cio/qdio_thinint.c cio/airq.c (AIF support)				QDIO thin	
net/qeth_core_main.c net/qeth_core_sys.c	“qeth” ccw driver reg.				

Source File	Module (module_init)	Subsystem (subsys_initcall)	Device Driver	Thin Interurpt Handler	Init (__initcall)
	“qeth” ccwgroup reg.				
net/qeth_l2_main.c	stub		qeth L2 ccwgroup		
net/qeth_l3_main.c net/qeth_l3_sys.c net/qeth_core_mpc.c	stub		qeth L3 ccwgroup		