Preface

Objective

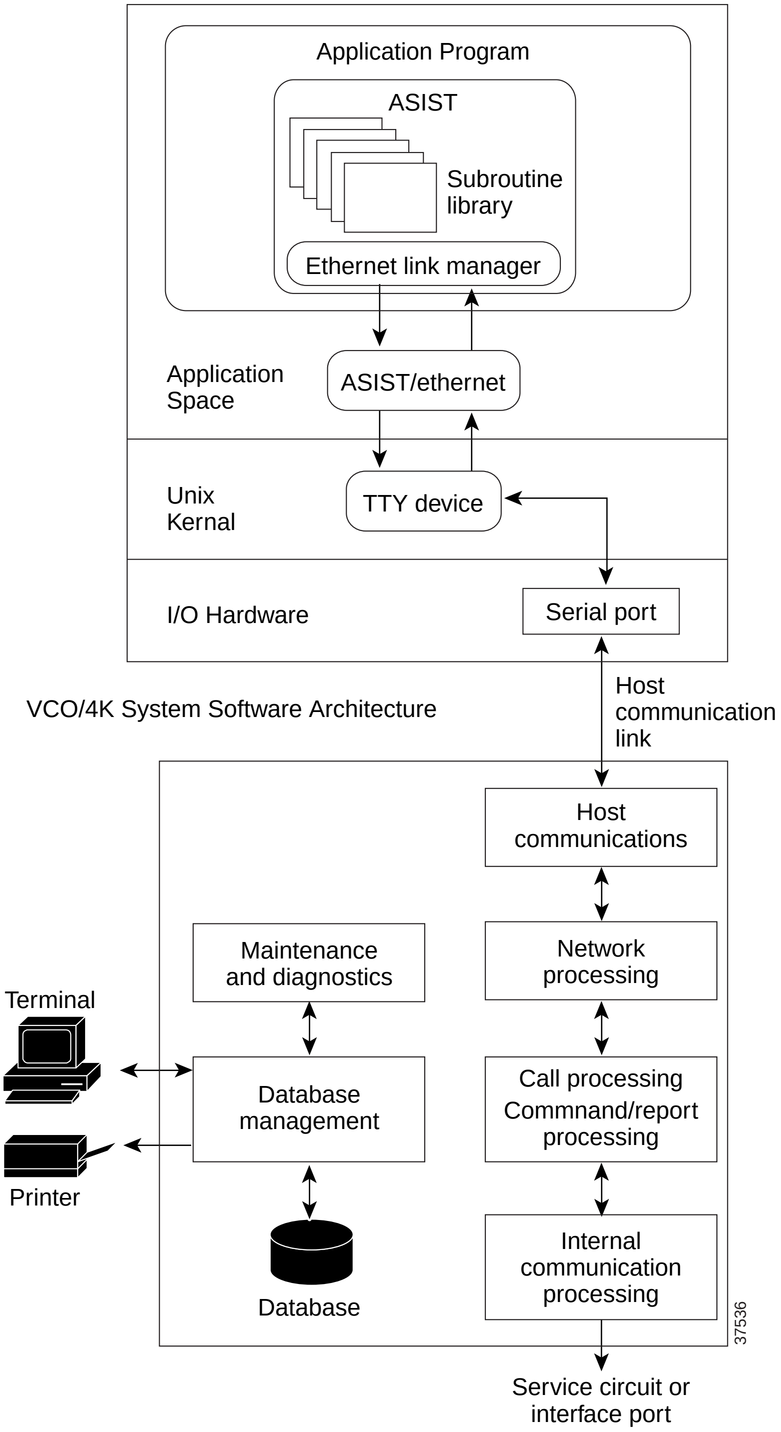
The *Cisco VCO/4K ASIST Programming Reference* describes the Application Software Integration Support Tools (ASIST) software product. The ASIST software product is a set of application development tools to help Cisco VCO/4K customers develop host-controlled applications.

ASIST is a C language representation of the command and report host interface protocols described in the *Cisco VCO/4K Standard Programming Reference* and the *VCO/4K Extended Programming Reference*. The command and report protocols allow a host-based application to control system resources, including the following:

- Network interface circuits
- Service circuits
- Voice paths
- System controller

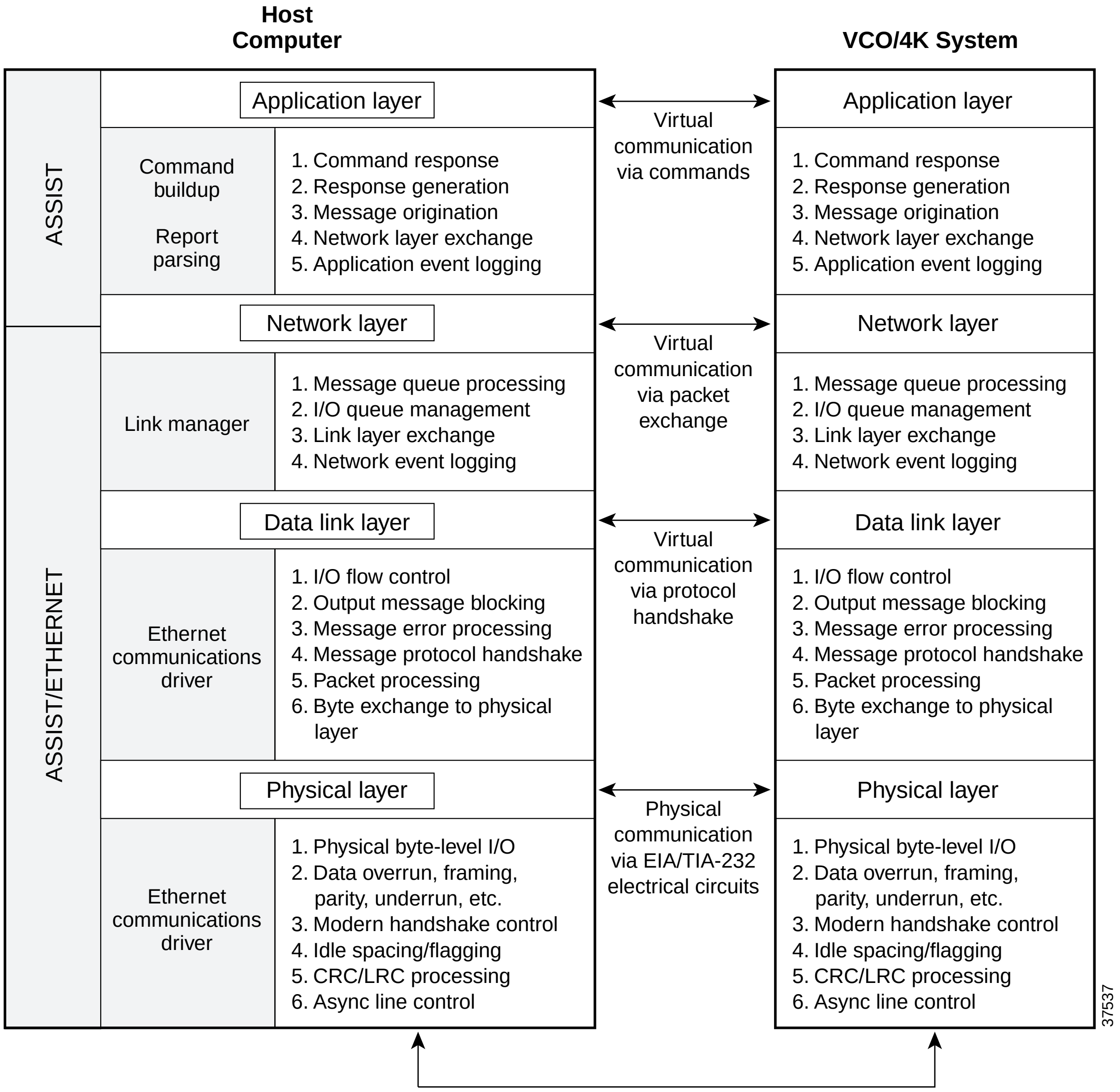
Other commands and reports are dedicated to system status and statistics. The ASIST product is organized around the system architecture of the VCO/4K system software (see Figure 1).

Figure 1 Software Architecture with ASIST Integration



The C language ASIST product is independent of any specific host operating system (see Figure 2). The ASIST product requires a separate communications driver, such as the ASIST/Ethernet component described in Chapter 3, “Ethernet Communications,” to transmit commands and receive reports.

Figure 2 Host-to-Switch Model with ASIST


Host Computer
VCO/4K System
ASSIST
ASSIST/ETHERNET
Application layer
Command buildup
Report parsing
1. Command response
2. Response generation
3. Message origination
4. Network layer exchange
5. Application event logging
Virtual communication via commands
Application layer
1. Command response
2. Response generation
3. Message origination
4. Network layer exchange
5. Application event logging
Network layer
Link manager
1. Message queue processing
2. I/O queue management
3. Link layer exchange
4. Network event logging
Virtual communication via packet exchange
Network layer
1. Message queue processing
2. I/O queue management
3. Link layer exchange
4. Network event logging
Data link layer
Ethernet communications driver
1. I/O flow control
2. Output message blocking
3. Message error processing
4. Message protocol handshake
5. Packet processing
6. Byte exchange to physical layer
Virtual communication via protocol handshake
Data link layer
1. I/O flow control
2. Output message blocking
3. Message error processing
4. Message protocol handshake
5. Packet processing
6. Byte exchange to physical layer
Physical layer
Ethernet communications driver
1. Physical byte-level I/O
2. Data overrun, framing, parity, underrun, etc.
3. Modern handshake control
4. Idle spacing/flagging
5. CRC/LRC processing
6. Async line control
Physical communication via EIA/TIA-232 electrical circuits
Physical layer
1. Physical byte-level I/O
2. Data overrun, framing, parity, underrun, etc.
3. Modern handshake control
4. Idle spacing/flagging
5. CRC/LRC processing
6. Async line control
37537

Audience

This guide is intended for all personnel designing applications for the VCO/4K switch. You should be familiar with the components of the switch as well as the system administrator master console. The master console is your access to the system administration functions. This guide offers programmers a means of easily implementing the call processing aspects of a telecommunications application.

Document Organization

Chapter 1, “ASIST Installation,” describes how to install the ASIST software product on your system.

Chapter 2, “Detailed Description,” describes the C language functions and structures of ASIST that allow you to create applications used with the VCO/4K system.

Chapter 3, “Ethernet Communications,” describes the ASIST/Ethernet software component—a set of application development tools designed to assist in the development of host-controlled applications used with the VCO/4K.

Document Conventions

This guide uses the following conventions:

Note Means *reader take note*. Notes contain helpful suggestions.

Tips Means *the following are useful tips*.

Caution Means *reader be careful*. In this situation, you might do something that could result in loss of data.

Warning **Means danger. You are in a situation that could cause bodily injury. Before you work on any equipment, you must be aware of the hazards involved with electrical circuitry and be familiar with standard practices for preventing accidents.**

Related Documentation

You may want to refer to the following documents that apply to your Cisco VCO/4K configuration:

- *Cisco VCO/4K System Software Version 5.n(n) Release Notes*
- *Cisco VCO/4K System Administrator’s Guide*
- *Cisco VCO/4K System Messages*
- *Cisco VCO/4K Software Installation Guide*
- *Cisco VCO/4K Hardware Installation Guide*
- *Cisco VCO/4K Card Technical Descriptions*
- Product supplements for optional software, including:
  - *Cisco VCO/4K Management Information Base (MIB) Reference*
  - *Cisco VCO/4K Extended Programming Reference*
  - *Cisco VCO/4K Standard Programming Reference*
  - *Cisco VCO/4K TeleRouter Reference Guide*
  - *Cisco VCO/4K ISDN Supplement*
  - *Cisco VCO/4K Ethernet Supplement*

- – *Cisco VCO/4K IPRC Supplement*
- – Applicable tone plan supplements

Obtaining Documentation

The following sections provide sources for obtaining documentation from Cisco Systems.

World Wide Web

You can access the most current Cisco documentation on the World Wide Web at the following sites:

- http://www.cisco.com
- http://www-china.cisco.com
- http://www-europe.cisco.com

Documentation CD-ROM

Cisco documentation and additional literature are available in a CD-ROM package, which ships with your product. The Documentation CD-ROM is updated monthly and may be more current than printed documentation. The CD-ROM package is available as a single unit or as an annual subscription.

Ordering Documentation

Cisco documentation is available in the following ways:

- Registered Cisco Direct Customers can order Cisco Product documentation from the Networking Products MarketPlace:

  http://www.cisco.com/cgi-bin/order/order_root.pl
- Registered Cisco.com users can order the Documentation CD-ROM through the online Subscription Store:

  http://www.cisco.com/go/subscription
- Nonregistered Cisco.com users can order documentation through a local account representative by calling Cisco corporate headquarters (California, USA) at 408 526-7208 or, in North America, by calling 800 553-NETS(6387).

Documentation Feedback

If you are reading Cisco product documentation on the World Wide Web, you can submit technical comments electronically. Click **Feedback** in the toolbar and select **Documentation**. After you complete the form, click **Submit** to send it to Cisco.

You can e-mail your comments to bug-doc@cisco.com.

To submit your comments by mail, for your convenience many documents contain a response card behind the front cover. Otherwise, you can mail your comments to the following address:

Cisco Systems, Inc.
Document Resource Connection
170 West Tasman Drive
San Jose, CA 95134-9883

We appreciate your comments.

Obtaining Technical Assistance

Cisco provides Cisco.com as a starting point for all technical assistance. Customers and partners can obtain documentation, troubleshooting tips, and sample configurations from online tools. For Cisco.com registered users, additional troubleshooting tools are available from the TAC website.

Cisco.com

Cisco.com is the foundation of a suite of interactive, networked services that provides immediate, open access to Cisco information and resources at anytime, from anywhere in the world. This highly integrated Internet application is a powerful, easy-to-use tool for doing business with Cisco.

Cisco.com provides a broad range of features and services to help customers and partners streamline business processes and improve productivity. Through Cisco.com, you can find information about Cisco and our networking solutions, services, and programs. In addition, you can resolve technical issues with online technical support, download and test software packages, and order Cisco learning materials and merchandise. Valuable online skill assessment, training, and certification programs are also available.

Customers and partners can self-register on Cisco.com to obtain additional personalized information and services. Registered users can order products, check on the status of an order, access technical support, and view benefits specific to their relationships with Cisco.

To access Cisco.com, go to the following website:

http://www.cisco.com

Technical Assistance Center

The Cisco TAC website is available to all customers who need technical assistance with a Cisco product or technology that is under warranty or covered by a maintenance contract.

Contacting TAC by Using the Cisco TAC Website

If you have a priority level 3 (P3) or priority level 4 (P4) problem, contact TAC by going to the TAC website:

http://www.cisco.com/tac

P3 and P4 level problems are defined as follows:

- P3—Your network performance is degraded. Network functionality is noticeably impaired, but most business operations continue.
- P4—You need information or assistance on Cisco product capabilities, product installation, or basic product configuration.

In each of the above cases, use the Cisco TAC website to quickly find answers to your questions.

To register for Cisco.com, go to the following website:

http://www.cisco.com/register/

If you cannot resolve your technical issue by using the TAC online resources, Cisco.com registered users can open a case online by using the TAC Case Open tool at the following website:

http://www.cisco.com/tac/caseopen

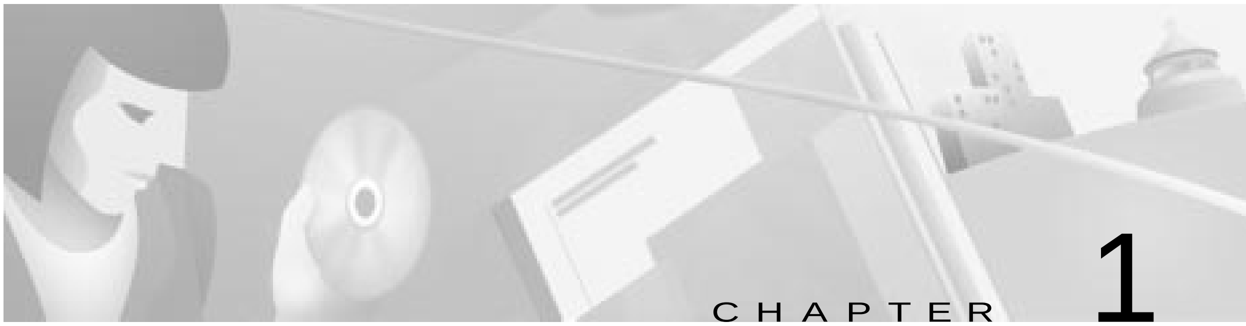
Contacting TAC by Telephone

If you have a priority level 1(P1) or priority level 2 (P2) problem, contact TAC by telephone and immediately open a case. To obtain a directory of toll-free numbers for your country, go to the following website:

http://www.cisco.com/warp/public/687/Directory/DirTAC.shtml

P1 and P2 level problems are defined as follows:

- P1—Your production network is down, causing a critical impact to business operations if service is not restored quickly. No workaround is available.
- P2—Your production network is severely degraded, affecting significant aspects of your business operations. No workaround is available.



ASIST Installation

The ASIST products were developed on a Sun Microsystems SPARCstation running SunOS 4.1. The ASIST product is independent of any particular operating system. The media contents and installation/compilation procedures for ASIST are described in this chapter.

ASIST Media

All ASIST files reside on a single 3.5-inch diskette (1.44-MB double sided, high-density), shown in the form of a tar file. When expanded, the files listed in Table 1-1are included:

Table 1-1 ASIST Files

File Name	File Name	File Name
./asist/api/Makefile	api_isdn.c	api_sub_switch.c
api_ctrl.c	api_isdn.h	api_sub_switch.h
api_ctrl.h	api_mf.c	api_tone.h
api_dcc.c	api_mf.h	asist.h
api_dcc.h	api_msg.h	asist_test.c
api_digit.c	api_net.c	asist_test.h
api_dtmf.c	api_net.h	command.c
api_dtmf.h	api_nsbmsg.h	dvc_prompts.h
api_dvc.c	api_path.c	isdn_ie.c
api_dvc.h	api_path.h	isdn_ie.h
api_dvcmsg.h	api_src.c	nsb_errmsg.h
api_ex1.c	api_src.h	report.c
api_ex2.c	api_stat.c	sds_cardtype.h
api_hook.c	api_stat.h	types.h
api_hook.h	—	—

Installing and Compiling

To copy the ASIST files from the supplied media, enter the appropriate Unix **tar** command as follows:

```
tar xvf /dev/rfd0 ./asist/api (SunOS)
tar xvf /dev/f0t ./asist/api (System V)
```

To guide you when compiling the source code, this ASIST product includes a makefile for all the source modules. To compile the ASIST product, use the **make file**.

Note The ASIST example program modules, **api_ex1.c** and **api_ex2.c**, must be compiled separately. The ASIST makefile does not compile the example modules.

A C language preprocessor flag, "-DBSD", is used in each makefile. When present, this flag indicates that the target operating system is SunOS; its absence indicates a System V environment.

make all

Build all modules that are out of date and create libasist.a, the make utility. Compile each file, and the header files it uses, into the object (.o) file. A new object file is created when the source file or any of the header files have changed, and when the object file doesn’t exist.

Object files are created, then combined into a library using the ar utility. **ranlib** randomizes the library when the object files are combined. This makes the library link much faster.

make clean

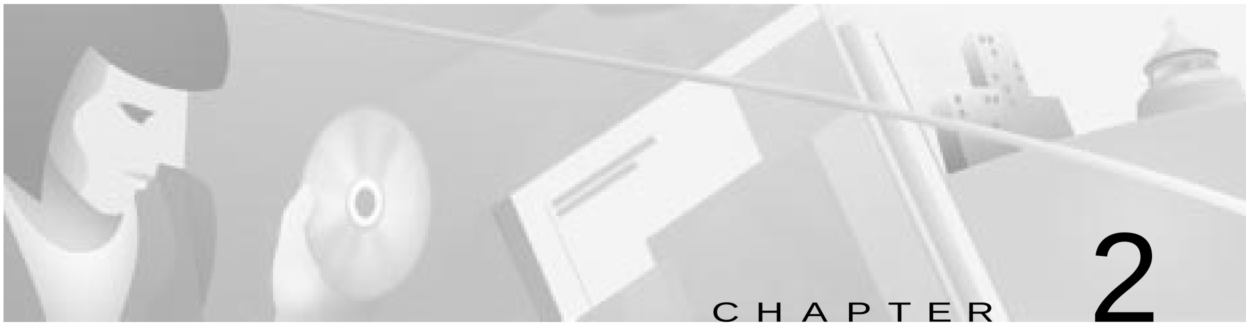
With the **clean** target, the **make** removes the object files (.o) and the library.

make install

With the **install** target, the **make** builds the library if it needs to be built, then copies it to the release areas as determined by INCDEST and LIBDEST.

INCDEST declares the path name of the directory where the include files are stored. The ASIST library is shipped with the INCDEST set to **/usr/local/include/asist**.

LIBDEST declares the path name of the directory where the library file is stored. The ASIST library is shipped with the LIBDEST set to **/usr/local/lib**.



CHAPTER 2

Detailed Description

ASIST provides application developers with C-language functions to build commands and parse reports. Structures are provided for each message in individual include files. Host applications use these structures to set up the data as required and ASIST creates the encoded binary command message to send to the VCO/4K. In reverse, the report messages from the host are unpacked by the ASIST parsing functions.

The V5.x system software includes support for two host API modes: standard and extended. The extended API mode supports the expanding capabilities offered in VCO/4K and V5.x system software, including extended port addresses and the addition of tone plans. The standard host API mode provides backward compatibility to previous system software releases.

ASIST supports both standard and extended API modes. An ASIST function allows you to specify which form of the messages should be used.

Note Cisco Systems recommends using the extended mode host API for all new application development.

API Message Format

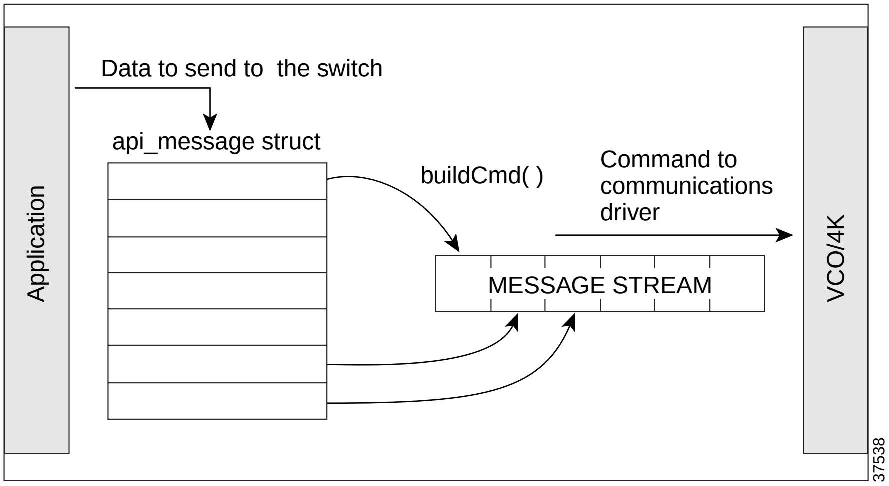
A data structure called `api_message` (defined in the `api_msg.h` file) serves as the interface to all of the command and report functions in ASIST. The `api_message` structure follows the guidelines in both the *Cisco VCO/4K Standard Programming Reference* and the *Cisco VCO/4K Extended API Programming Reference*. The api_message structure implements the command/report data field via a union of all the primary command and report data structures defined in the include files for each message type. This union, called `api_cmd_rep`, with the header data in `api_message`, is defined in `api_msg.h`.

Each command buildup function does the following:

- Refers to the appropriate structure pointed to by `msg`.
- Extracts the various data elements.
- Copies them into the correct `buf` location.
- Returns the length of the `buf` to the caller.

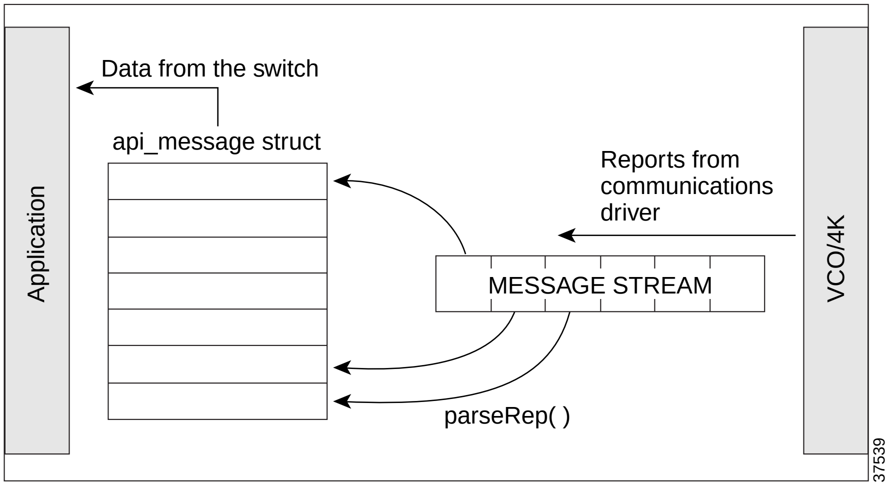
At this point, the `buf` array is ready to be sent to a communication driver and transmitted to the system. Figure 2-1 shows the command build up capability.

Figure 2-1 Command Buildup Flow



The report parsing functions basically perform the inverse of the command buildup functions. They take a byte array filled with a report just received by the communications driver, parse the array byte-by-byte, and initialize the data elements in the appropriate report structure. Figure 2-2 shows the report parsing capability.

Figure 2-2 Report Parsing Flow



Standard versus Extended Operational Mode Host API

The V5.x system software supports 4096 ports and multiple tone plans. An extended mode host application programming interface (API) accommodates these capabilities. The extended mode host API is a superset of the API provided in systems prior to V5.x—the standard mode host API. The same fields, in the same order, are provided. Differences between the two modes include the following:

- Larger field sizes for specific data types.
- Bit flags realigned.

- Additional fields to support new capabilities (i.e., tone plans).

ASIST supports several variations of systems, which include:

- V3.x systems (refer to the *Cisco VCO/4K Standard Programming Reference*)
- V4.x systems (refer to the *Cisco VCO/4K Standard Programming Reference*)
- V5.x (and beyond) systems operating with the standard mode host API
- V5.x (and beyond) systems operating with the extended mode host API

The operational mode which dictates the API, either standard, or extended, is set in the VCO/4K switch at install time for V5.x systems. The API cannot be dynamically changed during operation. Systems using software versions prior to V5.0 support the standard mode API only.

Note The host API mode used by the application and ASIST **must** match the mode set in the switch.

You set ASIST at run time to support either API by invoking the SetMessageMode() function described in the "Set Message Mode" section on page 2-5. ASIST builds or parses messages according to the mode you specify. This allows the same host application to support both forms of the host API. All function and data definition names are the same.

The data structures within previous releases of ASIST must be expanded to accommodate the extended mode host API. This includes additional fields and larger field sizes. ASIST provides both versions of the structures in order to remain backward compatible. The smaller structures (V4.x and earlier) are referred to as V4.x structures and the updated expanded structures (V5.x and beyond) are referred to as V5.x structures.

V5.x structures must be used when the extended mode API is used. You can use either structure set with the standard mode API. If the V5.x structures are used with the standard mode API, unused fields, such as tone plan, are ignored, and the larger fields sizes are masked in the message being built. Initialize any unused fields to 0.

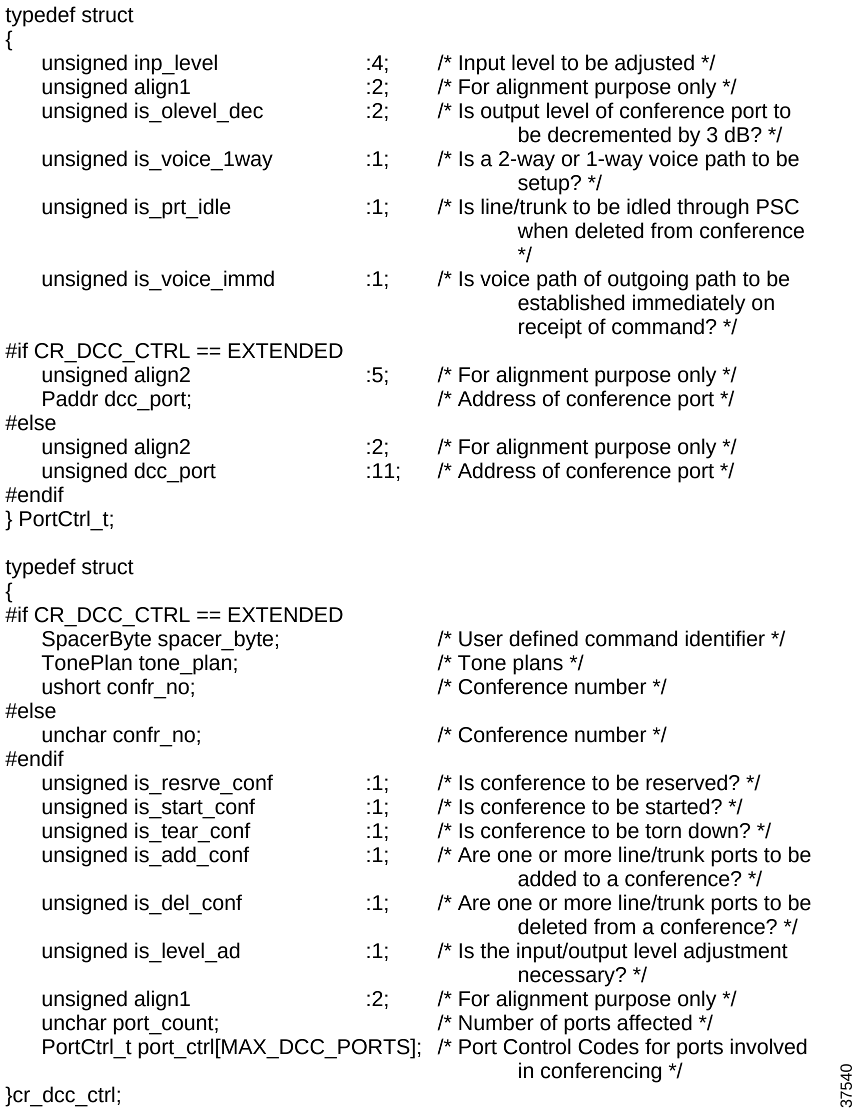
Instead of providing two physically separate sets of files to handle the two structure sets, the files are combined. Conditional compiling specifies which structure set to use. Set constants to STANDARD to specify the V4.x structures. Set constants to EXTENDED to specify the V5.x structures. These constants are set up on a per message basis. This allows you to migrate from the V4.x structures to the V5.x structures one message at a time. The constants may also be set as a group by setting DEFAULT_API_MODE to the desired mode (see Table 2-1). The constants are provided in asist.h.

Table 2-1 ASIST Structures According to Standard or Extended API

Structures in ASIST as Set by DEFAULT_API_MODE	VCO/4K Set to Standard API at Install Set Message Mode (STANDARD)	VCO/4K Set to Extended API at Install Set Message Mode (EXTENDED)
Standard (V4.x structure).	OK. Provided for backward compatability.	Not allowed. Extended (V5.x) structures must be used for extended API.
Extended (V5.x structure).	OK. Provided for backward compatability.	Recommended setting.

Figure 2-3 shows an example of a structure definition set up with the conditional compile information. The figure shows a portion of api_dcc.h set to use the V5.x structure set.

Figure 2-3 Example of an ASIST Structure



```
typedef struct
{
     unsigned inp_level                :4;     /* Input level to be adjusted */
     unsigned align1                   :2;     /* For alignment purpose only */
     unsigned is_olevel_dec            :2;     /* Is output level of conference port to
                                                          be decremented by 3 dB? */
     unsigned is_voice_1way            :1;     /* Is a 2-way or 1-way voice path to be
                                                          setup? */
     unsigned is_prt_idle              :1;     /* Is line/trunk to be idled through PSC
                                                          when deleted from conference
                                                          */
     unsigned is_voice_immd            :1;     /* Is voice path of outgoing path to be
                                                          established immediately on
                                                          receipt of command? */
#if CR_DCC_CTRL == EXTENDED
     unsigned align2                   :5;     /* For alignment purpose only */
     Paddr dcc_port;                           /* Address of conference port */
#else
     unsigned align2                   :2;     /* For alignment purpose only */
     unsigned dcc_port                 :11;    /* Address of conference port */
#endif
} PortCtrl_t;

typedef struct
{
#if CR_DCC_CTRL == EXTENDED
     SpacerByte spacer_byte;                   /* User defined command identifier */
     TonePlan tone_plan;                       /* Tone plans */
     ushort confr_no;                          /* Conference number */
#else
     unchar confr_no;                          /* Conference number */
#endif
     unsigned is_resrve_conf           :1;     /* Is conference to be reserved? */
     unsigned is_start_conf            :1;     /* Is conference to be started? */
     unsigned is_tear_conf             :1;     /* Is conference to be torn down? */
     unsigned is_add_conf              :1;     /* Are one or more line/trunk ports to be
                                                          added to a conference? */
     unsigned is_del_conf              :1;     /* Are one or more line/trunk ports to be
                                                          deleted from a conference? */
     unsigned is_level_ad              :1;     /* Is the input/output level adjustment
                                                          necessary? */
     unsigned align1                   :2;     /* For alignment purpose only */
     unchar port_count;                        /* Number of ports affected */
     PortCtrl_t port_ctrl[MAX_DCC_PORTS];      /* Port Control Codes for ports involved
                                                          in conferencing */
}cr_dcc_ctrl;
```

Unique Field Data Types in V4.x and V5.x

To provide flexibility, consistency, and future growth, unique data types are available for fields which differ between the V4.x and V5.x structure sets. They are specified in types.h and are as follows:

- Paddr—All extended mode port addresses are declared Paddr data types. Paddr is an unsigned long integer. Do not change the definition of the Paddr type or the port address fields.
- SpacerByte—All spacer_byte fields are declared SpacerByte data types. SpacerByte is an unsigned long integer. Do not change the definition of the SpacerByte type or the spacer_byte field.

- TonePlan—All tone_plan fields are declared TonePlan data types. TonePlan is an unsigned long integer. Do not change the definition of the TonePlan type or the tone_plan fields.
- Rule—All extended mode inpulse/outpulse rule numbers are declared Rule data types. Rule is an unsigned short integer. Do not change the definition of the Rule type or the inpulse/outpulse rule number fields.
- Group—Some extended mode resource group number fields are declared Group data types—in reports that have a resource group field, such as the $DA and $DB reports. Group is an unsigned short integer. Do not change the definition of the Group type or the resource group number fields.

External Functions

This section specifies the functions called by the application. The functions in turn access the internal message processing described in the "Message Processing" section on page 2-11.

For the following externals functions, the include files are:

- #include <sys/types.h>
- #include "types.h"
- #include "api_msg.h"

Set Message Mode

Function

int SetMessageMode (int mode).

The message mode determines whether the library will build and parse standard messages or extended messages. The setMessageMode function may be called to adjust the message mode. The parameter may be STANDARD in all systems, or EXTENDED in VCO V5.0 systems only.

The message mode defaults to STANDARD. The setMessageMode function returns the selected message mode when it is successful, and a – 1 when it is unsuccessful.

Parse Report

Function

int parseRep(unchar *buf, int rep_len, api_message *msg)

This function is a single function call interface to all the report parsing functions. parseRep() uses msg->func_id to select the appropriate report parsing function. It then calls the report parsing function and returns the report in the api_message structure pointed to by msg. If msg->func_id is invalid, the function returns a value greater than zero; if invalid, a – 1.

Retrieve Incoming Port

Function

- ushort getIport(api_message *msg)
- Paddr getExtIport(api_message *msg)

This function retrieves the ctrl_port variable of the report structure identified by msg->func_id and returns it to the caller; if the func_id is not valid, it returns a – 1. This function only accesses the following report structures:

- rr_mf_digits
- rs_res_limit
- rr_ipulse_rule
- rr_dtmf_digits
- rs_port_stats
- rc_dvc_status
- rr_psc
- rr_oport_cos
- rr_isdn_pcos
- rs_port_status
- rr_iport_cos
- rr_isdn_irule
- rr_spoken_dig

Retrieve Outgoing Port

Function

- ushort getOport(api_message *msg)
- Paddr getExtOport(api_message *msg)

This function retrieves the resource or outgoing port of the report structure identified by msg->func_id and returns it to the caller; if the func_id is not valid, it returns a – 1. This function only accesses the following report structures:

- rr_mf_digits
- rc_dvc_status
- rr_dtmf_digits
- rr_isdn_pcos.
- rr_spoken_dig
- rr_oport_cos

Build Command

Function

int buildCmd(api_message *msg, unchar *buf)

This function is a single function call interface to all the command buildup functions. buildCmd() uses msg->func_id to select the appropriate command buildup function. It then calls the command buildup function and returns the command in the buf array and the length, in bytes, of the array. If msg->func_id is invalid, the function returns a value greater than zero; if invalid, a – 1.

Substitute Incoming Port

Function

- int subsIport(api_message *msg, ushort iport)
- int subsExtIport(api_message *msg, Paddr iport)

This function initializes the ctrl_port or iport variable of the command identified by msg->func_id, with the value of the iport parameter. The function returns TRUE if the func_id is valid, otherwise FALSE. This function only affects the following command structures:

- cd_path_ctrl
- cr_iport_ctrl
- cd_psupv_ctrl
- cr_isdn_ctrl
- cr_dtmf_ctrl
- cr_ch_iport
- cr_mf_ctrl
- cr_dvc_ctrl
- cr_oport_ctrl.
- cr_src_ctrl

Substitute Outgoing Port

Function

- int subsOport(api_message *msg, ushort oport)
- int subsExtOport(api_message *msg, Paddr oport)

This function initializes the outgoing port variable of the command identified by msg->func_id, with the value of the oport parameter. The function returns TRUE if the func_id is valid, otherwise FALSE. This function only affects the following command structures:

- cd_path_ctrl
- cr_oport_ctrl
- cr_ch_iport, cr_isdn_ctrl

Parse Command

Function

int parseCmd (unchar *buf, int buflen, api_message *msg)

This function processes a command (pointed to by buf and returned by the system) and populates the appropriate command data structure in msg. It returns a value greater than zero if successful; if an invalid command is passed to it, it returns –1.

Processing ISDN IEs: Calling/Called Party Number

The functions in this section can be called directly to parse or build up ISDN message information elements (IEs). Refer to the “Controlling ISDN Primary Rate Interfaces” section on page 2-52 for ISDN message processing.

- #include "isdn_ie.h"

- #include <stdio.h>

Find Information Element

This function performs the following actions:

- Searches for the IE identifier, **ieid**, in the received IE message segment pointed to by **iebuf**.
- Returns the index into **iebuf** where **ieid** is found; else it returns –1 if it fails to find **ieid**.

Function

findIE(unchar ieid, unchar *iebuf, int seg_count)

Mode Constant

None

Parameters

The parameters for this message are in Table 2-2.

Table 2-2 Find Information Element Parameters

Parameter	Type	Description
ieid	unsigned char	IE identifier to search for in iebuf
iebuf	pointer, unsigned char	Contains the received IE message from the system
seg_count	integer	The number of IE segments contained in iebuf

Calling/Called Party Number IE Buildup

This function performs the following actions:

- Translates an API IE message (IE_MSG) into a Calling Party Number or Called Party Number information element, depending upon the **ieid**.
- Allows the user to specify the number type and plan to be used with the address digits.
- If the Calling Party Number IE is specified, the user has access to the number presentation and screening parameters. Set the **ps_ind_flag** to TRUE in this case.
- Allows the user to specify the number of digits and the address digits.
- Returns the length, in bytes, of the IE message contained in **iebuf**.

Function

IE_buildCallNum(IE_MSG *iemsgp, unchar *iebuf)

Mode Constant

None

Parameters

The parameters for this message are in Table 2-3.

Table 2-3 Calling/Called Party Number IE Buildup Parameters

Parameter	Type	Description
iemsgp->ieid	unsigned char	IE identifier.
type	unsigned char	Number type.
plan	unsigned char	Number plan.
ps_ind_flag	unsigned char	Indicates if presentation and screening are required.
presentation_ind	unsigned char	Is calling party number presented to called user?
screening_ind	unsigned char	Is the calling party number screened?
digit_count	unsigned char	Number of digits.
digits	character	IA5 (ASCII) formatted digits.
iebuf	pointer, unsigned char	Contains the Calling/Called Party IE message.

Calling/Called Party Number IE Parsing

This function performs the following actions:

- Translates a Calling Party Number or Called Party Number information element into API IE message (IE_MSG).
- Provides the user with the number type and plan of the address digits.
- If the Calling Party Number IE is received, the user may be provided the number presentation and screening parameters. The **ps_ind_flag** is set to TRUE in this case.
- Provides the user with the number of digits and the address digits.

Function

IE_parseCallNum(unchar *iebuf, int ielen, IE_MSG *iemsgp)

Mode Constant

None

Parameters

The parameters for this message are in Table 2-4.

Table 2-4 Calling/Called Party Number IE Parameters

Parameters	Type	Description
iebuf	pointer, unsigned char	Contains the Calling/Called Party IE.
ielen	integer	Length, in bytes, of message.
iemsgp->ieid	unsigned char	IE identifier.
type	unsigned char	Number type.

Table 2-4 Calling/Called Party Number IE Parameters (continued)

Parameters	Type	Description
plan	unsigned char	Number plan.
ps_ind_flag	unsigned char	Indicates presentation and screening are set.
presentation_ind	unsigned char	Is calling party number presented to called user?
screening_ind	unsigned char	Is the calling party number screened?
digit_count	unsigned char	Number of digits.
digits	character	IA5 (ASCII) formatted digits.

Network Facility Information Element

This function translates Network Facility Information Elements into API IE message (IE_MSG). No parsing function is available at this time.

Function

IE_buildNetFacil(IE_MSG *iemsgp, unchar *iebuf)

Mode Constant

None

Parameters

The parameters for this message are in Table 2-5.

Table 2-5 Network Facility Information Element Parameters

Parameter	Type	Description
ie_id	unchar	The ID number of the IE, network-specific facility messages use a value of 0x20.
ie_len	unchar	Length of the information element beginning with octet number 3. The maximum length is 25 octets, per TR 41449.
netfacil.length	unchar	The length of network identification in octets, which include octet 3.1 and optional octet(s) 3.2.
netfacil.ext	unchar	Extention.
netfacil.type	unchar	Type of network identification.
netfacil.plan	unchar	Network identification plan.
netfacil.ident[]	unchar	Network identification.
netfacil.par_bin	unchar	Parameters/binary.
netfacil.expan	unchar	Expansion.
netfacil.ftr_svc	unchar	Requested facility is feature/service.
netfacil.codeval	unchar	Facility coding value.
netfacil.param[]	unchar	Parameters.

Message Processing

This section defines the supported command and report messages. The messages are grouped by functional areas.

For each functional area, the include files which describe the associated structures are specified for reference. These include files are included in api_msg.h and as a result, they do not need to be called out individually.

For each message, the following data is included:

- Description
- Prototype for the function supporting this message
- The mode constant used in the conditional compile of the structure
- Parameter definitions

For parameters that are different between standard and extended structure definitions, the type specified in the table is as described in the “Standard versus Extended Operational Mode Host API” section on page 2-2. The actual size varies.

For example, the parameter definitions in all cases of port addresses are specified as type Paddr. If you compile the structure in standard mode, the type is actually either an unsigned short or an 11-bit field.

For all of the following externals functions, the include files are:

- #include <sys/types.h>
- #include “types.h”
- #include “api_msg.h”

Internal Functions

Individual internal functions perform the actual parsing and building of messages. These functions are not called directly.

The naming conventions adhered to by ASIST functions are as follows:

fXY_funcname:

If X is:

r—Decoding a report from the system.

c—Sending a command to the system.

And Y is:

r—Resource control command/report.

c—Configuration control command/report.

s—System status command/report.

d—System diagnostics command.

m—System maintenance command.

Similarly, the naming conventions for data structures are as follows:

*XY*_structname—where *X* and *Y* have the same meaning as defined above.

Structname and funcname are identical for the data structure and function that they identify. Table 2-6 maps the commands and reports to their corresponding abbreviations.

Table 2-6 Commands/Reports and Function Names

Hex	Command/Report	Function/Structure Name for Command Buildup	Function/Structure Name for Report Parsing
$65	Subrate Path Control	(f)cr_subrate_ctrl	(f)rr_subrate_ctrl
$67	DTMF Collection Control	(f)cr_dtmf_ctrl	(f)rr_dtmf_ctrl
$D1	DTMF Digit Collection	—	(f)rr_dtmf_digits
$68	MF Collection Control	(f)cr_mf_ctrl	(f)rr_mf_ctrl
$DO	MF Digit Collection	—	(f)rr_mf_digits
$6C	DVC Port Control	(f)cr_dvc_ctrl	(f)rr_dvc_ctrl
$DE	DVC Port Status	—	(f)rc_dvc_status
$91	Voice Prompt Maintenance	(f)cr_vpm_ctrl	(f)rr_vpm_ctrl
$6D	Conference Control	(f)cr_dcc_ctrl	(f)rr_dcc_ctrl
$70	Port Hook State Control	(f)cd_hook_ctrl	(f)rd_hook_ctrl
$69	Outgoing Port Control	(f)cr_oport_ctrl	(f)rr_oport_ctrl
$DA	Outgoing Port Change of State	—	(f)rr_oport_cos
$6A	Incoming Port Control (Macro)	(f)cr_iport_ctrl	(f)_iport_ctrl
$DB	Incoming Port Change of State	—	(f)rr_iport_cos
$DD	Inpulse Rule Complete (Macro)	—	(f)rr_ipulse_rule
$72	Port Supervision Control	(f)cr_psupv_ctrl	(f)rr_psupv_ctrl
$66	Voice Path Control	(f)cd_path_ctrl	(f)rd_path_ctrl
$6B	Change Incoming Port	(f)cr_ch_iport	(f)rr_ch_iport
$D5	Route Action	—	(f)rr_route_action
$C0 00	Configure VCA/Set System Clock	(f)cc_confg_vca	—
$C0 01	Change Active Controllers	(f)cc_ch_sysctrl	—
$C0 02	T1 Synchronization Control	(f)cc_confg_t1	(f)rc_confg_t1
$C0 03	Set/Reset Host Alarms	(f)cc_set_alarms	—
$C0 04	Host Load Control	(f)cc_load_ctrl	—
$C0 05	Host Assume/Relinquish Port Control	(f)cc_port_ctrl	(f)rc_port_ctrl
$DC	Active/Standby Mode	—	(f)rc_act_sby
$80	Resource Allocation	(f)cs_res_alloc	(f)rs_res_alloc
$81	Hardware Allocation	(f)cs_hw_alloc	(f)rs_hw_alloc
$82	Card Status	(f)cs_card_statreq	(f)rs_card_statreq
$83	Change Port Status	(f)cs_port_statreq	(f)rs_port_statreq
$D2	Permanent Signal Condition	—	(f)rr_psc
$90	Change Port Status	(f)cm_ch_pstatus	(f)rm_ch_pstatus

Table 2-6 Commands/Reports and Function Names (continued)

Hex	Command/Report	Function/Structure Name for Command Buildup	Function/Structure Name for Report Parsing
$D3	Port Status	—	(f)rs_port_status
$D6	Resource Limitation	—	(f)rs_res_limit
$D9	Card Status	—	(f)rs_card_status
$F0	Alarm Condition	—	(f)rs_alarm_cond
$49	ISDN Port Control	(f)cr_isdn_ctrl	(f)rr_isdn_ctrl
$EA	ISDN Port Change of State	—	(f)rr_isdn_pcos
$ED	ISDN Inpulse Rule Complete	—	(f)rr_isdn_irule

Subrate Switching

#include "api_sub_switch.h"

Subrate Path Control ($65) Command and Report

Subrate switching gives the system the ability to connect portions of DSO links to other DSO links, called paths. The width of a subrate path can be from 8 kilobits to 64 kilobits, in 8-kilobit increments. The subrate switch command is structured with a single source end point and one or more destination end points. An endpoint is the combination of a port address and a bit offset.

Note When ASIST is operating in the extended mode, and more than 98 destination endpoints are specified, the built command is too long. Limit the number of destination endpoints to 98 or fewer.

Function

int fcr_subrate_ctrl(api_message *msg, char *buf)

int frr_subrate_ctrl(unchar *buf, int len, api_message *msg)

Mode Constant

CR_SUBRATE_SWITCH

Parameters

The parameters for this message are in Table 2-7 and Table 2-8.

Table 2-7 Subrate Switch ($65) Parameters

Parameter	Type	Description
spacer_byte	SpacerByte	User-definable spacer bytes—compiled in extended mode only.
tone_plan	TonePlan	Reserved for tone plans—compiled in extended mode only.
detach_bearer	unsigned:1	Detach bearer.
idle_channel	unsigned:1	Idle channel.
is_multi_dest_mode	unsigned:1	Multiple destination mode?
is_bulk_mode	unsigned:1	Bulk mode?
is_teardown	unsigned:1	Tear path down?
path_control	unsigned:2	Path control.
parameter	unsigned short	Parameter.
width	unchar	Subrate path width.
source	endpoint_struct	Source end port address and offset.
dest []	endpoint_struct	Destination port addresses and offsets, up to 166.

Table 2-8 Subrate Switch ($65) Endpoint and Structure

Parameter	Type	Description
port_address	Paddr	Port address of the end point
offset	unchar	Bit offset at which the path begins

DTMF Digit Collection

#include api_dtmf.h

#include api_tone.h

DTMF Collection Control ($67) (Standard and Enhanced) Command and Report

Use these functions to perform the following:

- Generate the DTMF Collection Control ($67) (Standard and Enhanced) command to be sent to the system, and decode the report.
- Instruct the system to collect DTMF digits sent over any line or trunk circuit without an inpulse rule
- Collect dial pulse (DP) digits on a SLIC or DID circuit.
- Attach/detach DTMF receivers to/from a trunk (SLIC, DID, and UTC cards have an onboard DTMF receiver per port).
- Instruct the IPRC to play a voice prompt/announcement via an attached Enhanced Voice Port Control ($6C) command.

Function

fcr_dtmf_ctrl(api_message *msg, char *buf)

frr_dtmf_ctrl(unchar *buf, int len, api_message *msg)

Mode Constant

CR_DTMF_CTRL

Parameters

For parameters that are different between standard and extended structure definitions, the type specified in the table is as described in the "Standard versus Extended Operational Mode Host API" section on page 2-2. The actual size varies.

If msg->cmd_rep.dtmf_ctrl.type = DTMF_STD, then parameters are as in Table 2-9.

If msg->cmd_rep.dtmf_ctrl.type = DTMF_ENH, then parameters are as inTable 2-10.

Table 2-9 DTMF Collection Control (Standard) ($67) Parameters

Parameter	Type	Description
spacer_byte	SpacerByte	User-definable spacer bytes—compiled in extended mode only.
tone_plan	TonePlan	Reserved for tone plans—compiled in extended mode only.
ctrl_port	Paddr	Controlling port address.
is_switch_reqd	unsigned:1	Is switching action required?
is_port_attach	unsigned:1	Is port to be attached or detached?
is_dtmf_rgrp	unsigned:1	Is a specific DTMF receiver to be used or one from DTMF receiver resource group?
is_dtmf_retain	unsigned:1	Is DTMF receiver to be retained after report?
dtmf_port	Paddr	DTMF receiver port address. If controlling port address resides on DID, UTC or SLIC, then $00.
is_dtmf_enable	unsigned:1	Is the DTMF receiver to be enabled?
max_digits	unsigned:6	Maximum number of digits to be collected.
reenter_digits	unchar	Reenter digits.
end_digits	unchar	End of string digits.
col_timeout	unchar	Number of seconds allowed for the user to enter max_digits.
is_reenter_beep	unsigned:1	Is a beep tone connected when the user enters reenter code?
is_strend_beep	unsigned:1	Is a beep tone connected when the end of string code is detected?
is_enable_beep	unsigned:1	Is a beep tone connected when DTMF receiver is enabled?

Table 2-10 DTMF Collection Control (Enhanced) ($67) Parameters

Parameter	Type	Description
spacer_byte	SpacerByte	User definable spacer bytes—compiled in extended mode only.
tone_plan	TonePlan	Reserved for tone plans—compiled in extended mode only.
ctrl_port	Paddr	Controlling port address.
is_switch_reqd	unsigned:1	Is switching action required?
is_port_attach	unsigned:1	Is port to be attached or detached?
is_dtmf_rgrp	unsigned:1	Is a specific DTMF receiver to be used or one from DTMF receiver resource group?
is_dtmf_retain	unsigned:1	Is DTMF receiver to be retained after report?
dtmf_port	Paddr	DTMF receiver port address. If controlling port address resides on DID, UTC or SLIC, then $00.
is_dtmf_enable	unsigned:1	Is the DTMF receiver to be enabled?
is_enhanced	unsigned:1	Should always be 1.
is_4th_col_enable	nsigned:1	Is fourth column DTMF enabled?
is_tmr_seg	unsigned:1	Is Collection Timers segment attached?
is_eos_seg	unsigned:1	Is Reenter/End of String segment attached?
is_fdig_seg	unsigned:1	Is First Digit Processing String segment attached?
is_enopt_seg	unsigned:1	Is Enabling Options Segment attached?
max_digits	unsigned:6	Maximum number of digits to be collected
is_store_dig	unsigned:1	Are digits to be stored in ports digit field?
is_app_dig	unsigned:1	Are digits to be appended in ports digit field?
dig_field	unsigned:3	Fields in which digits stored/appended.
If is_tmr_seg = TRUE		
fdig_tout	unchar	First Digit Timeout.
idig_tout	unchar	Inter Digit Timeout.
fldig_tout	unchar	Field Timeout.
If is_eos_seg = TRUE		
is_1dig_reenter	unsigned:1	Is a single-digit reenter code used?
is_2dig_reenter	unsigned:1	Is a two-digit reenter code used?
is_1dig_eos	unsigned:1	Is a single-digit end of string code used?
is_2dig_eos	unsigned:1	Is a two-digit end of string code used?
is_rent_notone	unsigned:1	Is no tone on reenter code detection to be connected?
is_eos_notone	unsigned:1	Is no tone on end of string code detection to be connected?
rent_dig_code	unsigned:1	One- or two-digit DTMF digit reenter code.
eos_code	unsigned:1	One- or two-digit DTMF digit end of string code.
rent_tone_code	unsigned:1	Reenter tone on detection of reenter code.
eos_tone_code	unsigned:1	Reenter tone on detection of end of string code.

Table 2-10 DTMF Collection Control (Enhanced) ($67) Parameters (continued)

Parameter	Type	Description
field_tout	unchar	Field Timeout.
If is_fdig_seg = TRUE		
is_rep_fdig	unsigned:1	Is DTMF digit report sent to host on detection of first digit?
is_fdig_tone	unsigned:1	Is a tone to be presented on detection of first digit?
is_fdig_wink	unsigned:1	Is a wink to be presented on detection of first digit?
is_fdig_abort	unsigned:1	Is DVC voice prompt being presented to be aborted on detection of first digit?
is_detach_og	unsigned:1	Is line/trunk attached to port to be detached on detection of first digit?
fdig_tone	unchar	Tone to be presented on detection of first digit.
If is_enopt_seg = TRUE		
is_enab_rec	unsigned:1	Is receiver to be enabled immediately or after condition satisfied?
is_enh_dvc	unsigned:1	Is an enhanced $6C segment attached?
is_tone_renb	unsigned:1	Is a tone to be presented when receiver enabled?
is_wink_renb	unsigned:1	Is a wink to be presented when receiver enabled?
is_tmr_pause	unsigned:1	Pause before starting first digit timer?
is_tmr_sup	unsigned:1	Wait for supervision event before starting digit timer?
is_tmr_dvc	unsigned:1	Present (up to 14) voice prompts before starting digit timer?
If is_enh_dvc = TRUE		
dvc_ctrl	cr_dvc_ctrl	$6C segment (refer to the “DVC Port Control ($6C) Command and Report” section on page 2-20).

DTMF Digit Collection ($D1) (Standard and Enhanced) Report Parsing

This function does the following:

- Analyzes the DTMF Digit Collection ($D1) report sent from the system.
- Transfers DTMF/DP digit collection information to the host application.
- Generates a report indicating whether digit report is valid, and the line/trunk for which digits were collected. If a timeout occurs, any digits collected up to that point are returned.
- Collects a maximum of 40 digits.

The report produced by this function also indicates:

- Report generated for first digit receipt.
- DVC prompt being presented was aborted.
- Timeout occurred while waiting for supervision.
- Digit field overflow occurred.
- Receiver port not available at first request (hunt only).

Function

frr_dtmf_digits(unchar *buf, int len, api_message *msg)

Mode Constant

RR_DTMF_DIGITS

Parameters

The parameters for this message are in Table 2-11.

Table 2-11 DTMF Digit Collection Report ($D1) Parsing Parameters

Parameter	Type	Description
ctrl_port	Paddr	Controlling port address.
is_prmt_abort	unsigned:1	Was DVC prompt aborted after the user entered the first digit?
was_out_detach	unsigned:1	Was the outgoing port detached on first digit detection?
did_sup_fire	unsigned:1	Was digit collection aborted and the receiver removed because the supervision timer fired?
did_dig_flow	unsigned:1	Did digit field overflow?
why_sup_fired	unchar	Indicates why supervision timer fired.
dtmf_port	Paddr	DTMF receiver port address. Identical to controlling port address for SLICs, DIDs and UTCs.
is_enh_dtmf	unsigned: 1	0 (zero) if report following is old style. 1 (one) if report following is enhanced report format.
did_idig_fire	unsigned:1	Did the interdigit timer fire?
is_fdig_rep	unsigned:1	Is it the first digit report?
was_rec_avail	unsigned:1	Was DTMF receiver available on initial request?
did_ddig_fire	unsigned:1	Did digit collection timer fire?
did_fdig_fire	unsigned:1	Did first digit timer fire?
is_rep_valid	unsigned:1	Is DTMF digit report valid?
field_id	unchar	Field in which the system stores reported digits when Enable Digit Field Reporting feature is enabled.
digits	array, unchar	Pointer to array containing digits collected.
digit_count	unchar	Number of digits collected.

MF Digit Collection

#include “api_mf.h”

MF Collection Control ($68) Command and Report

These functions do the following:

- Generate the MF Collection Control ($68) command that is sent to the system, and parse the reports.

- Enable the host to collect MF digits sent over a trunk.
- Attach/detach MF receivers to/from a trunk. Up to 40 MF digits can be collected in a 30-second period.
- Allow call to remain in active state if garbled digits are received or no KP/ST is detected. The default is to tear down the call.

Function

fcr_mf_ctrl(api_message *msg, char *buf)

frr_mf_ctrl(unchar *buf, int len, api_message *msg)

Mode Constant

CR_MF_CTRL

Parameters

The parameters for this message are in Table 2-12.

Table 2-12 MF Collection Control ($68) Parameters

Parameter	Type	Description
spacer_byte	SpacerByte	User-definable spacer bytes—compiled in extended mode only.
tone_plan	TonePlan	Reserved for tone plans—compiled in extended mode only.
ctrl_port	Paddr	Controlling port address.
is_switch_reqd	unsigned:1	Is switching action required?
is_port_attach	unsigned:1	Is port to be attached or detached?
is_mf_rgrp	unsigned:1	Is a specific MF receiver to be used or one from MF receiver resource group?
is_mf_retain	unsigned:1	Is MF receiver to be retained after report?
is_tearcall	unsigned:1	Is the call to be torn down upon digit collection failure?
mf_port	unsigned:11	MF receiver port address.
spacer_byte	unchar	Spacer byte.
mf_enable	unchar	MF receiver can either be enabled or not (MF_ENABLE or MF_DISABLE).

MF Digit Collection ($D0) Report Parsing

This function does the following:

- Analyzes the MF Digit Collection ($D0) report sent from the system.
- Transfers MF digit collection information from the system to the host application.
- Generates a report indicating whether the digit report is valid, and the incoming port from which digits were collected. The report also indicates the present state of controlling port (CP_SETUP or forced to idle) and whether it detected garbled MF digits.
- Collects a maximum of 40 digits.

Function

frr_mf_digits(unchar *buf, int len, api_message *msg)

Mode Constant

RR_MF_DIGITS

Parameters

The parameters for this function are in Table 2-13.

Table 2-13 MF Digit Collection ($D0) Report Parsing Parameters

Parameter	Type	Description
ctrl_port	Paddr	Incoming port address.
spacer_bytes	ushort	Spacer bytes.
mf_port	Paddr	MF receiver port address.
is_rep_garbled	unsigned:1	Is MF report garbled?
is_port_idle	unsigned:1	If MF report is garbled, has the controlling port been forced to idle state or placed in setup state?
is_mf_avail	unsigned:1	Is MF receiver available when initially requested?
did_mf_fire	unsigned:1	Did the MF digit collection timer fire?
is_rep_valid	unsigned:1	Is MF digit report valid?
field_id	unchar	Field in which the system stores reported digits when Enable Digit Field Reporting feature is enabled.
digits	array, unchar	Array containing digits collected.
digit_count	unchar	Number of digits collected.

Playing Digitized Voice Prompts

#include “api_dvc.h”

DVC Port Control ($6C) Command and Report

Use these functions do the following:

- Generate the DVC Port Control ($6C) command that is sent to the system, and parse the report returned from the system.
- Instruct the system to play up to 14 prompts (or 20, with Enhanced $6C command) to a line or trunk port. All prompts are downloaded to the Digital Voice Card (DVC) at system boot.
- Link or remove a DVC port to or from a call's resource chain.
- Serve as a command segment in an Incoming Port Control ($6A) command.
- Play or record voice prompts.

Function

fcr_dvc_ctrl(api_message *msg, char *buf)

frr_dvc_ctrl(unchar *buf, int len, api_message *msg)

Mode Constant

CR_DVC_CTRL

Parameters

The parameters for this message are in Table 2-14.

Table 2-14 DVC Port Control ($6C) Parameters

Parameter	Type	Description
spacer_byte	SpacerByte	User-definable spacer bytes—compiled in extended mode only.
tone_plan	TonePlan	Reserved for tone plans—compiled in extended mode only.
ctrl_port	Paddr	Controlling port address.
is_switch_reqd	unsigned:1	Is switching action required?
is_port_attach	unsigned:1	Is a DVC port to be attached or detached?
is_dvc_rgrp	unsigned:1	Is a specific DVC port to be used or one from a resource group?
is_dvc_rel	unsigned:1	Is DVC port to be released after prompts have been played?
dvc_port	Paddr	DVC port address group to search for.
is_play_prompt	unsigned:1	Is a prompt to be played on a line/trunk?
is_genrep_prmt	unsigned:1	Is a $DE report to be generated when all the prompts have been played?
enh_dvc	unsigned:1	Is this an enhanced DVC command?
seg_attach	unsigned:1	Is play (0) or record (1) segment attached (enhanced only)?
no_prompts	unsigned:4	Number of prompts to be played (maximum 14).
phrases	char pointer	Pointer to a char array of phrase numbers to be played.
enh_record_seg	rec_seg	Record segment for (enhanced only).
enh_play_seg	play_seg	Play segment for (enhanced only).

DVC Port Status ($DE) Report Parsing

This function analyzes a DVC Port Status ($DE) report sent from the system and indicates when all voice prompts specified in a DVC Control ($6C) command have completed.

Function

frc_dvc_status(unchar *buf, int len, api_message *msg)

Mode Constant

RC_DVC_STATUS

Parameters

The parameters for this message are in Table 2-15.

Table 2-15 DVC Port Status ($DE) Parsing Parameters

Parameter	Type	Description
ctrl_port	Paddr	Address of incoming port to which voice prompts were played.
spacer_bytes	ushort	Spacer bytes.
dvc_port	Paddr	Address of DVC port used to present prompts.
status	unchar	Indicated status of the digit report.

Voice Prompt Maintenance ($91) Command and Report

These functions generate the Voice Prompt Maintenance Control ($91) command that is sent to the system, and parse the report returned from the system. The $91 command provides a mechanism for the host to:

- Upload voice prompt information from one or more IPRCs
- Download prompt information to one or more IPRCs.

Function

fcr_vpm_ctrl(api_message *msg, char *buf)

frr_vpm_ctrl(unchar *buf, int len, api_message *msg)

Mode Constant

CR_VPM_CTRL

Parameters

The parameters for this message are in Table 2-16.

Table 2-16 Voice Prompt Maintenance ($91) Parameters

Parameters	Type	Description
spacer_byte	SpacerByte	User-definable spacer byte—compiled in extended mode only.
tone_plan	TonePlan	Reserved for tone plans—compiled in extended mode only.
control_code	unchar	0 = download prompt information; 1 = upload.
access_ code	unchar	0 = Access card containing port specified by port address code; 1 = access card specified by RLS code; 2 = access all cards supporting specified prompt library (download only).
rls_code	unchar	See Hardware Allocation ($81) report for RLS code specification; set to $00 if accessing by port address.
ctrl_port	Paddr	Card that contains port address will be accessed. Set to $0000 if accessing by RLS code.
source_library	unchar	Hexadecimal representation of library ID ($00 to $0F).

Table 2-16 Voice Prompt Maintenance ($91) Parameters (continued)

Parameters	Type	Description
temp_prompt	unsigned: 1	Is this a temporary prompt?
source_prompt_id	ushort	Source prompt ID.
dest_lib	unchar	Destination library (upload only) ($00 to $0F).
dest_prompt_id	ushort	Destination prompt ID (upload only) ($0001 to $00FF).

Controlling Multi-Party Conferences

#include “api_dcc.h”

Conference Control ($6D) Command and Report

These functions generate the Conference Control ($6D) command that is sent to the system, and the report returned by the system. It controls conferencing features. Up to eight conference ports can be used for a conference. The system supports up to 128 simultaneous conferences.

Function

fcr_dcc_ctrl(api_message *msg, char *buf)

frr_dcc_ctrl(unchar *buf, int len, api_message *msg)

Mode Constant

CR_DCC_CTRL

Parameters

The parameters for this message are in Table 2-17.

Table 2-17 Conference Control ($6D) Parameters

Parameter	Type	Description
spacer_byte	SpacerByte	User-definable spacer byte—compiled in extended mode only.
tone_plan	TonePlan	Reserved for tone plans—compiled in extended mode only.
confr_no	unsigned short	Conference number.
is_resrve_conf	unsigned:1	Is conference to be reserved?
is_start_conf	unsigned:1	Is conference to be started?
is_tear_conf	unsigned:1	Is conference to be torn down?
is_add_conf	unsigned:1	Are one or more line/trunk ports to be added to conference?
is_del_conf	unsigned:1	Are one or more line/trunk ports to be deleted from a conference?
is_level_adj	unsigned:1	Is the input/output level adjustment necessary?
port_count	unchar	Number of ports affected.

Table 2-17 Conference Control ($6D) Parameters (continued)

Parameter	Type	Description
For Each Port (port_count):		
inp_level_adj	unchar	Input level adjustment (00 to 15).
is_olevel_dec	unsigned:1	Is output level of conference port associated with the line/trunk port to be decremented by 3 dB?
is_voice_2way	unsigned:1	Is a two-way or one-way voice path to be set up?
dcc_port	Paddr	Address of port involved in conference.

Port Hook Control

#include “api_hook.h”

Port Hook State Control ($70) Command and Report

The Port Hook State Control ($70) command provides the host with the ability to cause onhook and offhook processing on a line or trunk port. The host may also start inpulse or outpulse rule processing with this command. Hook state control is useful when hook state events are received external to the switch, such as SS7.

Function

fcd_hook_ctrl(api_message *msg, char *buf)

frd_hook_ctrl(unchar *buf, int len, api_message *msg)

Mode Constant

CD_HOOK_CTRL

Parameters

The parameters for this message are in Table 2-18.

Table 2-18 Port Hook State Control ($70) Parameters

Parameter	Type	Description
spacer_byte	SpacerByte	User-definable spacer bytes—compiled in extended mode only.
tone_plan	TonePlan	Reserved for tone plans—compiled in extended mode only.
ctrl_port	Paddr	Address of the port being affected.
supp_dadb	unsigned:1	Suppress DA and DB reports.
hook_state	unsigned:1	Hook State: • 0 = On hook. • 1 = Off hook.

Table 2-18 Port Hook State Control ($70) Parameters (continued)

Parameter	Type	Description
class	unchar	Class of Service: • 0 = Use port’s configured COS. • 1 = Force port to incoming. • 2 = Force port to outgoing.
do_irule	unsigned:1	Inpulse rule control.
do_orule	unsigned:1	Outpulse rule control.
rule_id	Rule	Rule number.

Controlling Line/Trunk Network Interfaces

#include “api_dtmf.h”

#include “api_mf.h”

#include “api_src.h”

#include “api_dvc.h”

#include “api_net.h”

Outgoing Port Control ($69) Command and Report

Use these functions to do the following:

- Generate the Outgoing Port Control ($69) command that is sent to the system, and parse the report from the system.
- Link or remove outgoing circuits to or from a call's resource chain.
- Begin outpulse/inpulse rule processing for an outgoing port.
- Overwrite digit strings contained in call record fields and supply new digits.
- Disconnect call at teardown.

Function

fcr_oport_ctrl(api_message *msg, unchar *buf)

frr_oport_ctrl(unchar *buf, int len, api_message *msg)

Mode Constant

CR_OPORT_CTRL

Parameters

The parameters for this message are in Table 2-19.

Table 2-19 Outgoing Port Control ($69) Parameters

Parameter	Type	Description
spacer_byte	SpacerByte	User-definable spacer bytes—compiled in extended mode only.
tone_plan	TonePlan	Reserved for tone plans—compiled in extended mode only.
ctrl_port	Paddr	Incoming port address.
is_switch_reqd	unsigned:1	Is switching action required?
is_port_attach	unsigned:1	Is port to be attached or detached?
is_oport_rgrp	unsigned:1	Is a specific outgoing port to be used or one from a resource group?
twoway_path	unsigned:2	Defer 2-way path until end of outpulse rule (DEFER_OPUL), defer until outgoing answers (DEFER_OANS), or cut 2-way speech instantly (CUT_SPEECH).
oport	Paddr	Outgoing port address/resource group to search for.
is_opul_exec	unsigned:1	Is an outpulse rule to be executed?
is_discon_byte	unsigned:1	Is Disconnect Control byte included in the command?
is_ipul_exec	unsigned:1	Is an inpulse rule to be executed?
rule_number	Rule	Inpulse/outpulse number.
is_irep_sup	unsigned:1	Is onhook report for incoming port to be suppressed if outgoing port goes onhook first?
is_ic_setup	unsigned:1	Is incoming port to return to CP_SETUP if outgoing port goes onhook first?
is_orep_sup	unsigned:1	Is onhook report for outgoing port to be suppressed?
field_no	unsigned:3	Call record field to receive digit string.
no_digits	unsigned:5	Number of digits in string between 0 to 40 (0 to 12 for ANI).
dig_string	array, unchar	Digit string.

Outgoing Port Change of State ($DA) Report Parsing

This function does the following:

- Analyzes Outgoing Port Change of State ($DA) report sent from the system.
- Informs host of a change in hardware state of an outgoing the system port.
- Indicates whether an outpulse rule has been successfully completed.

Note In order for the system to generate a report indicating outpulse rule completion, a REP END token must be contained in the outpulse rule.

- Indicates supervision errors in change byte **oport_change**.
- Indicates when a rehunt of an outgoing port is performed.

Function

frr_oport_cos(unchar *buf, int len, api_message *msg)

Mode Constant

RR_OPORT_COS

Parameters

The parameters for this message are in Table 2-20.

Table 2-20 Outgoing Port Change of State Report ($DA) Parameters

Parameter	Type	Description
res_group	Group	Resource group number.
oport_change	unchar	Change occurring on outgoing port. Can be ACT_OPORT, INACT_OPORT, SUPRERR_OPORT, SUPDET_OPORT, SUPOPUL_OPORT, OUTPUL_OPORT or HUNT_OPORT.
oport	Paddr	Outgoing port address.
iport	Paddr	Incoming port address.
supv_code	ushort	Answer Supervision Code.
is_og_ans	unsigned:1	Is outgoing port considered answered?
supv_tmplate	unsigned:6	Answer supervision template used.
new_oport	Paddr	Address of new outgoing port selected by the system as a result of rehunt.

Incoming Port Control (Macro) ($6A) Command and Report

These functions do the following:

- Generate the Incoming Port Control ($6A) command that is sent to the system, and parse the report from the system.
- Instruct the system to force call origination or disconnect, begin an inpulse or outpulse rule, or execute one of the following:
  - Port Supervision Control ($72) command
  - DTMF Collection Control (Standard or Enhanced) ($67) command
  - MF Collection Control ($68) command
  - Outgoing Port Control ($69) command
  - DVC Port Control ($6C) command
  - Outpulse Control segment

Note Specify only one inpulse rule, outpulse rule or command segment in a single command. You can include up to five outpulse control segments in a single command when you specify an outpulse or inpulse rule for an incoming port.

Function

fcr_iport_ctrl(api_message *msg, char *buf)

frr_iport_ctrl(unchar *buf, int len, api_message *msg)

Mode Constant

CR_IPORT_CTRL

Parameters

The parameters for this message are in Table 2-21.

Table 2-21 Incoming Port Control (Macro) ($6A) Command and Report Parameters

Parameter	Type	Description
spacer_byte	SpacerByte	User-definable spacer bytes—compiled in extended mode only.
tone_plan	TonePlan	Reserved for tone plans—compiled in extended mode only.
iport	Paddr	Incoming port address.
oport	Paddr	Outgoing port address/resource group to search for.
is_switch_reqd	unsigned:1	Is switching action required?
is_port_attach	unsigned:1	Is a call to be originated or disconnected?
is_iport_rgrp	unsigned:1	Is a specific incoming port to be used or one from a resource group?
is_ipdcon_sup	unsigned:1	Should incoming go to CP_SETUP state on forced disconnect?
is_opdcon_sup	unsigned:1	Should outgoing go to CP_SETUP state on forced disconnect?
is_ipon_rep	unsigned:1	Are on hooks for incoming ports to be reported?
is_opon_rep	unsigned:1	Are on hooks for outgoing ports to be reported?
is_opul_exec	unsigned:1	Is an outpulse rule to be executed?
is_ipul_exec	unsigned:1	Is an inpulse rule to be executed?
rule_number	Rule	Inpulse/outpulse number.
is_psup_seg	unsigned:1	Is Port Supervision Control Command segment attached?
is_dtmf_seg	unsigned:1	Is DTMF Collection Control Command segment attached?
is_mf_seg	unsigned:1	Is MF Collection Control Command segment attached?
is_dvc_seg	unsigned:1	Is DVC Port Control Command segment attached?
is_ogcon_seg	unsigned:1	Is Outgoing Port Control Command segment attached?
is_iccon_seg	unsigned:1	Is Incoming Port Control Command segment attached?
is_outpul_seg	unsigned:1	Is Outpulse Rule Control Command segment attached?
If is is_psup_seg=TRUE		
is_psup_exe	unsigned:1	Is supervision action to be executed or cancelled?
psup_action	unsigned:3	Port supervision action.
If is_mf_seg = TRUE		
is_switch_reqd	unsigned:1	Is switching action required?

Table 2-21 Incoming Port Control (Macro) ($6A) Command and Report Parameters (continued)

Parameter	Type	Description
is_port_attach	unsigned:1	Is port to be attached or detached?
is_mf_rgrp	unsigned:1	Is a specific MF receiver to be used or one from MF receiver resource group?
is_mf_retain	unsigned:1	Is MF receiver to be retained after report?
is_tearcall	unsigned:1	Is the call to be torn down upon digit collection failure?
mf_port	Paddr	MF receiver port address.
mf_enable	unchar	MF receiver enabled?
If is_dtmf_seg = TRUE:		
If the Standard DTMF Collection segment is used:		
is_switch_reqd	unsigned:1	Is switching action required?
is_port_attach	unsigned:1	Is port to be attached or detached?
is_dtmf_rgrp	unsigned:1	Is a specific DTMF receiver to be used or one from DTMF receiver resource group?
is_dtmf_retain	unsigned:1	Is DTMF receiver to be retained after report?
dtmf_port	Paddr	DTMF receiver port address. If controlling port address resides on DID, UTC or SLIC, then $00.
is_dtmf_enable	unsigned:1	Is the DTMF receiver to be enabled?
max_digits	unsigned:6	Maximum number of digits to be collected.
reenter_digits	unchar	Reenter digits.
end_digits	unchar	End of string digits.
col_timeout	unchar	Number of seconds allowed for the user to enter max_digits.
is_reenter_beep	unsigned:1	Is a beep tone connected when user enters reenter code?
is_strend_beep	unsigned:1	Is a beep tone connected when end of string code detected?
is_enable_beep	unsigned:1	Is a beep tone connected when DTMF receiver is enabled?
If Enhanced DTMF Collection Segment Used:		
is_switch_reqd	unsigned:1	Is switching action required?
is_port_attach	unsigned:1	Is port to be attached or detached?
is_dtmf_rgrp	unsigned:1	Is a specific DTMF receiver to be used or one from DTMF receiver resource group?
is_dtmf_retain	unsigned:1	Is DTMF receiver to be retained after report?
dtmf_port	Paddr	DTMF receiver port address. If controlling port address resides on DID, UTC or SLIC, then $00.
is_dtmf_enable	unsigned:1	Is the DTMF receiver to be enabled?
is_enhanced	unsigned:1	Should always be 1.
is_tmr_seg	unsigned:1	Is Collection Timers segment attached?
is_eos_seg	unsigned:1	Is Reenter/End of String segment attached?
is_fdig_seg	unsigned:1	Is First Digit Processing String segment attached?

Table 2-21 Incoming Port Control (Macro) ($6A) Command and Report Parameters (continued)

Parameter	Type	Description
is_enopt_seg	unsigned:1	Is the Enabling Options segment attached?
max_digits	unsigned:6	Maximum number of digits to be collected.
is_store_dig	unsigned:1	Are digits to be stored in ports digit field?
is_app_dig	unsigned:1	Are digits to be appended in ports digit field?
dig_field	unsigned:3	Fields in which digits stored/appended.
If is_tmr_seg = TRUE:		
fdig_tout	unchar	First Digit Timeout.
idig_tout	unchar	Inter Digit Timeout.
fldig_tout	unchar	Field Timeout.
If is_eos_seg = TRUE:		
is_1dig_reenter	unsigned:1	Is a single-digit reenter code used?
is_2dig_reenter	unsigned:1	Is a two-digit reenter code used?
is_1dig_eos	unsigned:1	Is a single-digit end of string code used?
is_2dig_eos	unsigned:1	Is a two-digit end of string code used?
is_rent_notone	unsigned:1	Is no tone on reenter code detection to be connected?
is_eos_notone	unsigned:1	Is no tone on end of string code detection to be connected?
rent_dig_code	unsigned:1	One- or two-digit DTMF digit reenter code.
eos_code	unsigned:1	One- or two-digit DTMF digit end of string code.
rent_tone_code	unsigned:1	Reenter tone on detection of reenter code.
eos_tone_code	unsigned:1	Reenter tone on detection of end of string code.
field_tout	unchar	Field Timeout.
If is_fdig_seg = TRUE:		
is_rep_fdig	unsigned:1	Is DTMF digit report sent to host on detection of first digit?
is_fdig_tone	unsigned:1	Is a tone to be presented on detection of first digit?
is_fdig_wink	unsigned:1	Is a wink to be presented on detection of first digit?
is_fdig_abort	unsigned:1	Is a DVC voice prompt being presented to be aborted on detection of first digit?
is_detach_og	unsigned:1	Is line/trunk attached to port to be detached on detection of first digit?
fdig_tone	unchar	Tone to be presented on detection of first digit.
If is_enopt_seg = TRUE		
is_enab_rec	unsigned:1	Is receiver to be enabled immediately or after condition satisfied?
is_tone_renb	unsigned:1	Is a tone to be presented when the receiver is enabled?
is_wink_renb	unsigned:1	Is a wink to be presented when the receiver is enabled?
is_tmr_pause	unsigned:1	Pause before starting first digit timer?
is_tmr_sup	unsigned:1	Wait for supervision event before starting digit timer?

Table 2-21 Incoming Port Control (Macro) ($6A) Command and Report Parameters (continued)

Parameter	Type	Description
is_tmr_dvc	unsigned:1	Present (up to 14) voice prompts before starting digit timer?
Incoming Port Control (Macro) Command ($6A) Buildup		
If is_dvc_seg = TRUE:		
is_switch_reqd	unsigned:1	Is switching action required?
is_port_attach	unsigned:1	Is a DVC port to be attached or detached?
is_dvc_rgrp	unsigned:1	Is a specific DVC port to be used or one from a resource group?
is_dvc_retain	unsigned:1	Is DVC port to be retained after prompts have been played?
dvc_port	Paddr	DVC port address group to search for.
is_play_promp	unsigned:1	Is a prompt to be played on a line/trunk?
is_genrep_prmt	unsigned:1	Is a $DE report to be generated when all the prompts have been played?
no_prompts	unsigned:4	Number of prompts to be played (maximum 14).
phrases	char pointer	Pointer to a char array of phrase numbers to be played.
If is_ogcon_seg = TRUE:		
is_switch_reqd	unsigned:1	Is switching action required?
is_port_attach	unsigned:1	Is port to be attached or detached?
is_oport_rgrp	unsigned:1	Is a specific outgoing port to be used or one from a resource group?
twoway_path	unsigned:2	Defer 2-way path until end of outpulse rule (DEFER_OPUL), defer until outgoing answers (DEFER_OANS) or cut 2-way speech instantly (CUT_SPEECH).
oport	Paddr	Outgoing port address/resource group to search for.
is_opul_exec	unsigned:1	Is an outpulse rule to be executed?
is_discon_byte	unsigned:1	Is Disconnect Control byte included in the command?
is_ipul_exec	unsigned:1	Is an inpulse rule to be executed?
rule_number	Rule	Inpulse/outpulse rule number.
field_no	unsigned:3	Call record field to receive digit string.
no_digits	unsigned:5	Number of digits in string between 0 to 40 (0 to 12 for ANI)
dig_string	array, unchar	Digit string.
is_irep_sup	input	Is onhook report for incoming port to be suppressed if outgoing port goes onhook first?
is_ic_setup	unsigned:1	Is incoming port to return to CP_SETUP if outgoing port goes onhook first?
is_orep_sup	unsigned:1	Is onhook report for outgoing port to be suppressed?
If is_outpul_seg = TRUE:		

Table 2-21 Incoming Port Control (Macro) ($6A) Command and Report Parameters (continued)

Parameter	Type	Description
no_dopul_segs	integer	The number of outpulse control segments contained within the array iport_segs.dopul_seg[].
dopul_seg[]	array of typedig_opul_ seg	Contains one to five outpulse control segments.

Incoming Port Change of State ($DB) Report Parsing

This function does the following:

- Analyzes Incoming Port Change of State ($DB) report sent from the system.
- Informs host of a change in hardware state of an incoming system port.
- Indicates whether an inpulse rule has been successfully completed.

Function

frr_iport_cos(unchar *buf, int len, api_message *msg)

Mode Constant

RR_IPORT_COS

Parameters

The parameters for this message are in Table 2-22.

Table 2-22 Incoming Port Change of State ($DB) Report Parameters

Parameter	Type	Description
res_group	Group	Resource group number.
iport_change	unchar	Change occurring on incoming port. Can be ACT_IPORT, INACT_IPORT, SUPRERR_IPORT or OUTPUL_IPORT.
iport	Paddr	Incoming port address.
supv_code	ushort	Answer supervision code.
supv_tmplate	unsigned:6	Answer supervision template used.

Inpulse Rule Complete (Macro) ($DD) Report Parsing

This function does the following:

- Analyzes Inpulse Rule Complete ($DD) report sent from the system.
- Informs the host that an inpulse rule has been processed. The content of the report is controlled by the type of reporting specified in the inpulse rule. If REP EACH is specified, the report will indicate only that inpulse rule processing has ended. If REP END is specified, the report is a macro containing resource control reports (segments) to represent all actions taken during inpulse rule execution.

Function

frr_ipulse_rule(unchar *buf, int len, api_message *msg)

Mode Constant

RR_IPULSE_RULE

Parameters

The parameters for this message are in Table 2-23.

Table 2-23 Inpulse Rule Complete (Macro) ($DD) Report Parameters

Parameter	Type	Description
ctrl_port	Paddr	Address of controlling port for which inpulse rule executed.
is_port_inc	unsigned:1	Is port incoming?
is_loop_abort	unsigned:1	Has rule been aborted because of looping?
route_action	unsigned:1	Has a ROUTE token been executed?
seg_count	unsigned:3	Number of segments in inpulse report
is_dvc_avail	unsigned:1	Was DVC port available on initial request?
is_inpul_abort	unsigned:1	Was inpulse processing aborted?
is_no_outch	unsigned:1	Was rule aborted because of exhaustion of outpulse channel?
rule_id	Rule	Inpulse rule executed.
seg_type	unchar	Type of segment—ic port change of state, DTMF collection or MF collection.
If seg_type = FR_IPORT_COS:		
iport_change	unchar	Change occurring on incoming port.
If seg_type = FR_MF_DIGIT:		
mf_port	Paddr	MF receiver port address.
is_rep_garbled	unsigned:1	Is MF receiver garbled?
is_port_idle	unsigned:1	If MF report is garbled, is port in idle state or setup?
is_mf_avail	unsigned:1	Is MF receiver available when initially requested?
did_mf_fire	unsigned:1	Did MF digit collection timer fire?
is_rep_valid	unsigned:1	Is MF digit collection valid?
digits	array, unchar	Array containing MF digits collected.
digit_count	unchar	Number of digits collected.
field_id	unchar	Field number in which digits are stored.
If seg_type = FR_DTMF_DIGIT:		
dtmf_port	Paddr	DTMF receiver port address.
did_idig_fire	unsigned:1	Did interdigit timer fire?
is_fdig_rep	unsigned:1	Is it first digit report?
was_rec_avail	unsigned:1	Is DTMF receiver available when initially requested?

Table 2-23 Inpulse Rule Complete (Macro) ($DD) Report Parameters (continued)

Parameter	Type	Description
did_ddig_fire	unsigned:1	Did digit collection timer fire?
did_fdig_fire	unsigned:1	Did first digit timer fire?
is_rep_valid	unsigned:1	Is MF digit collection valid?
digits	array, unchar	Array containing DTMF digits collected.
digit_count	unchar	Number of digits collected.
field_id	unchar	Field number in which digits stored.

Port Supervision Control ($72) Command and Report

These functions generate the Port Supervision Control ($72) command that is sent to the system, and parse the report from the system. It is used for manual host control of outward handshake and supervision signals on both incoming and outgoing circuits.

Function

fcr_psupv_ctrl(api_message *msg, char *buf)

frr_psupv_ctrl(unchar *buf, int len, api_message *msg)

Mode Constant

CD_PSUPV_CTRL

Parameters

The parameters for this message are in Table 2-24.

.

Table 2-24 Port Supervision Control ($72) Command and Report Parameters

Parameter	Type	Description
spacer_byte	SpacerByte	User-definable spacer bytes—compiled in extended mode only.
tone_plan	TonePlan	Reserved for tone plans—compiled in extended mode only.
ctrl_port	Paddr	Controlling port address.
is_psup_exe	unsigned:1	Is supervision action to be executed (seize/wink) EXEC_PSUP or cancelled (seize) CANCEL_PSUP?
psup_action	unsigned:3	Port supervision action (can be SEIZE_SUP or WINK_PSUP).

Establishing Voice Paths Between Ports

#include “api_path.h”

#include “api_tone.h”

Voice Path Control ($66) Command and Report

These functions generate the Voice Path Control ($66) command that is sent to the system, and parse the report from the system. It is used for immediate setup of voice paths between receivers and senders. A receiver can be an incoming circuit, outgoing circuit, MF receiver, DTMF receiver, or SRC port. A sender can be an incoming circuit, outgoing circuit, system tones, DVC port, or DCC port. The voice path remains established until it is torn down by:

- Release of one of the circuits involved
- A call processing action
- An inpulse or outpulse rule
- Another voice path control command
- Resource Control command

It is possible to tear down a two-way path in only one direction, converting it to a one-way path. It can also be used to set a conference party to listen to a tone. A second command will send it back to conference.

Function

fcd_path_ctrl(api_message *msg, char *buf)

frd_path_ctrl(unchar *buf, int len, api_message *msg)

Mode Constant

CD_PATH_CTRL

Parameters

The parameters for this message are in Table 2-25.

.

Table 2-25 Voice Path Control ($66) Parameters

Parameter	Type	Description
spacer_byte	SpacerByte	User-definable spacer bytes—compiled in extended mode only.
path_type	unchar	Type of voice path to be constructed. Can be either BREAK_PATH, ONEWAY_PATH or TWOWAY_PATH (for non-ISDN channels) or BREAK_ISDNHC_PATH, ONEWAY_ISDN_PATH or TWOWAY_ISDNHC_PATH (for ISDN channels).
recv_port	Paddr	Port address of receiver.
send_port	Paddr	Port address of sender.
num_b_chans	unchar	Number of adjacent ISDN B-channels.

Change Incoming Port ($6B) Command and Report

These functions generate the Change Incoming Port ($6B) command that is sent to the system, and parse the report returned from the system. It switches all resources for an active call from one incoming port to another. The original port is forced to an idle state.

Function

fcr_ch_iport(api_message *msg, char *buf)

frr_ch_iport(unchar *buf, int len, api_message *msg)

Mode Constant

CR_CH_IPORT

Parameters

The parameters for this message are in Table 2-26.

Table 2-26 Change Incoming Port ($6B) Command and Report Parameters

Parameter	Type	Description
spacer_byte	SpacerByte	User definable spacer bytes—compiled in extended mode only.
tone_plan	TonePlan	Reserved for tone plans—compiled in extended mode only.
old_iport	Paddr	Old incoming port address.
is_ipdcon_clr	unsigned:1	Are disconnect control bits on old incoming port to be cleared or carried?
is_opdcon_clr	unsigned:1	Are disconnect control bits on outgoing port to be cleared or retained?
is_iport_rgrp	unsigned:1	Is a specific incoming port to be used or one from a resource group?
new_iport	Paddr	New incoming port address group to search for.

Route Action ($D5) Report Parsing

This function parses a Routing Action ($D5) report. The $D5 report provides the host with information about the outcome of a ROUTE inpulse rule operation. This information includes the action (inpulse or outpulse rule) performed, the status of that action, and the connected incoming and outgoing ports.

Function

frr_route_action(unchar *buf, int len, api_message *msg)

Mode Constant

RR_ROUTE_ACTION

Parameters

The parameters for this message are in Table 2-27.

Table 2-27 Route Action ($D5) Report Parameters

Parameter	Type	Description
iport	Paddr	Incoming controlling port.
action	unchar	Routing action performed; either inpulse or outpulse rule executed.

Table 2-27 Route Action ($D5) Report Parameters (continued)

Parameter	Type	Description
status	unchar	Status of the routing action; subset of network status byte values.
oport	Paddr	Outgoing port involved in the routing action.

Modify System Controller Operation

#include api_ctrl.h

Configure VCA/Set System Clock ($C0 00) Command Buildup

This function does the following:

- Generates the Configure VCA/Set System Clock ($C0 00) command that is used to configure any specific Virtual Communication Addresses (VCA) needed besides global communication address $DF. Specific VCAs are especially important when a single host controls multiple systems.
- Allows setting of the system real time clock.

Function

fcc_confg_vca(api_message *msg, char *buf)

Mode Constant

None

Parameters

The parameters for this message are in Table 2-28.

Table 2-28 Configure VCA/Set System Clock ($C0 00) Command Parameters

Parameter	Type	Description
funcid2	unchar	Second function ID byte.
vca	unchar	Virtual communications address; specifying $FF leaves it unchanged address to $DF.
hour_clock	unchar	System clock hour value to set.
min_clock	unchar	System clock minute value to set.
sec_clock	unchar	System clock second value to set.

Change Active Controllers ($C0 01) Command Buildup

This function generates the Change Active Controllers ($C0 01) command that is sent to the system. In redundant systems, it transfers system control from the active side to the standby side. Command can be sent to either the active or standby system controller. Optional reset of previously active side is also available.

Function

fcc_ch_sysctrl(api_message *msg, char *buf)

Mode Constant

None

Parameters

The parameters for this message are in Table 2-29.

Table 2-29 Change Active Controllers ($C0 01) Command Parameters

Parameter	Type	Description
funcid2	unchar	Second function ID byte.
prev_act_cond	unchar	Condition of transfer for previously active side. Can be NORESET_PREVACT or RESET_PREVACT.

Synchronization Control ($C0 02) Command and Report

These functions generate the T1 Synchronization Control ($C0 02) command that is sent to the system, and parses the report returned from the system. It alters Administration Console Master Timing Link parameters as follows:

- Switch to internal synchronization.
- Switch to external synchronization.
- Switch to incoming synchronization and specify/change Master Timing Link.
- Primary only.
- Secondary only.
- Both primary and secondary.

Note T1, E1 and ISDN PRI cards can be selected as the source for incoming timing.

Function

fcc_confg_t1(api_message *msg, char *buf)

frc_confg_t1(unchar *buf, int len, api_message *msg)

Mode Constant

CC_CONFG_T1

Parameters

The parameters for this message are in Table 2-30.

Table 2-30 T1 Synchronization Control ($C0 02) Command Parameters

Parameter	Type	Description
funcid2	unchar	Second function ID byte.
t1_sync_mode	unchar	Specifies Master Timing Link parameters to be altered. Can be INT_SYNC, EXT_SYNC, PR_MASTER PR_SEC_MASTER or SEC_MASTER.
prim_link_val	Paddr	Port address if t1_sync_mode = PR_SEC_MASTER or PR_MASTER.
sec_link_val	Paddr	Port address if t1_sync_mode = PR_SEC_MASTER or SEC_MASTER.

Set/Reset Host Alarms ($C0 03) Command Buildup

This function generates the Set/Reset Host Alarms ($C0 03) command that is sent to the system. Use it to set/clear major, minor and auxiliary alarms controlled by the host.

Function

fcc_set_alarms(api_message *msg, char *buf)

Mode Constant

None

Parameters

The parameters for this message are in Table 2-31.

.

Table 2-31 Set/Reset Host Alarms (C0 03) Command Parameters

Parameter	Type	Description
funcid2	unchar	Second function ID byte.
alarm_clear	unsigned:4	Determines if alarm is to be set or cleared. Can be ALARM_SET or ALARM_CLEAR.
alarm_type	unsigned:4	Type of alarm to be set/reset. Can be MAJOR_ALARM, MINOR_ALARM, AUX1_ALARM or AUX2_ALARM.

Host Call Load Control ($C0 04) Command Buildup

This function generates the Host Call Load Control ($C0 04) command that is sent to the system. Use it in conjunction with Enable Host Control of Call Load feature. When enabled, this command allows the host to start or stop sending Inpulse Rule Complete ($DD) reports and Incoming Port Change of State ($DB) reports. This effectively stops call processing, since calls cannot be completed through the system without host intervention.

Function

fcc_load_ctrl(api_message *msg, char *buf)

Mode Constant

None

Parameters

The parameters for this message are in Table 2-32.

Table 2-32 Host Call Load Control ($C0 04) Command Parameters

Parameter	Type	Description
funcid2	unchar	Second function ID byte.
is_rep_suspend	unsigned:1	Determines if system should process incoming calls by reporting them to host.

Host Assume/Relinquish Port Control ($C0 05) Command

These functions extend the system controller functionality to allow a host process to relinquish control of a call assigned to itself, or assume control of a call that has been assigned to a different host.

Function

fcc_port_ctrl (api_message *msg, char *buf)

frc_port_ctrl(unchar *buf, int len, api_message *msg)

Mode Constant

CC_PORT_CTRL

Parameters

The parameters for this message are in Table 2-33.

Table 2-33 Host Assume/Relinquish Port Control ($C0 05) Command Parameters

Parameter	Type	Description
funcid2	unchar	Always 0x05.
port_ctrl	unchar	Port control modifier.
port	Paddr	Extended message mode.

Active/Standby Mode ($DC) Report Parsing

This function does the following:

- Analyzes Active/Standby mode ($DC) report sent from the system.
- Informs host of a system boot or transfer in control between the active and standby sides for a redundant system.
- Reports when a link between the host and the system becomes established. In a redundant system, each link sends the report.

Function

frc_act_sby(unchar *buf, int len, api_message *msg)

Mode Constant

None

Parameters

The parameters for this message are in Table 2-34.

Table 2-34 Active/Standby Mode ($DC) Report Parameters

Parameter	Type	Description
funcid2	unchar	Second function ID byte.
is_sysboot	unsigned:1	Has system booted/data link established or is it a run time transfer?
is_sds_b	unsigned:1	Has report originated from the A-side of the system or the B-side of the system?
is_sds_act	unsigned:1	Is the system originating the report currently active or standby?

System Status

#include api_stat.h

Request Resource Allocation ($80) Command Buildup

This function generates the Request Resource Allocation ($80) command that is sent to the system. The system returns a bit map of the on-line and off-line status of every port within a given resource group and port address range. The add_range can be RANGE_00, RANGE_512, RANGE_1024, RANGE_1536, RANGE_2048, RANGE_2560, RANGE_3072, or RANGE_3584.

Function

fcs_res_alloc(api_message *msg, char *buf)

Mode Constant

CS_RES_ALLOC

Parameters

The parameters for this message are in Table 2-35.

Table 2-35 Request Resource Allocation ($80) Command Parameters

Parameter	Type	Description
add_range	ulong	Port address range for which resource allocation report desired. Can be RANGE_00, RANGE_512, RANGE_1024, RANGE_1536, RANGE_2048, RANGE_2560, RANGE_3072, or RANGE_3584.
res_grp_no	Group	Resource group number.

Resource Allocation ($80) Report Parsing

This function analyzes the Resource Allocation ($80) report sent from the system. A bit map is returned containing the status of each port within a given range and specified resource group. The report shows whether the port is of the specified resource group and whether it is on-line or off-line.

Function

frs_res_alloc(unchar *buf, int len, api_message *msg)

Mode Constant

RS_RES_ALLOC

Parameters

The parameters for this message are in Table 2-36.

Table 2-36 Resource Allocation ($80) Report Parameters

Parameter	Type	Description
low_port	Paddr	Lower address range.
high_port	Paddr	Higher address range.
res_grp	Group	Resource group number.
status	char array	128-byte array containing the status byte.

Request Hardware Allocation ($81) Command Buildup

This function generates the Request Hardware Allocation ($81) command that is sent to the system. The resulting bit map allows the host to match logical port addresses used in system commands to their corresponding physical rack, level, slot (R-L-S) hardware addresses of all ports within a given port address range. The add_range can be HW_RANGE_00, HW_RANGE_512, HW_RANGE_1024, HW_RANGE_1536, HW_RANGE_2048, HW_RANGE_2560, HW_RANGE_3072, or HW_RANGE_3584.

Function

fcs_hw_alloc(api_message *msg, char *buf)

Mode Constant

CS_HW_ALLOC

Parameter

The parameter for this message is in Table 2-37.

Table 2-37 Request Hardware Allocation ($81) Command Parameters

Parameter	Type	Description
add_range	unchar	Port address range for which resource allocation report desired.

Hardware Allocation ($81) Report Parsing

This function analyzes the Hardware Allocation ($81) report sent from the system and returns a bit map of ports within a given range. Each byte in the 64-byte array represents eight port addresses.

Function

frs_hw_alloc(unchar *buf, int len, api_message *msg)

Mode Constant

RS_HW_ALLOC

Parameters

The parameters for this message are in Table 2-38.

Table 2-38 Hardware Allocation ($81) Report Parameters

Parameter	Type	Description
low_port	Paddr	Lower address range.
high_port	Paddr	Higher address range.
rls	char array	64-byte array containing R-L-S information.

Card Status ($82) Command Buildup

Use this function to obtain the Card Status ($82) report. You can specify a single card or a range of cards. Specify the card by its rack, level, and slot (R-L-S) position. Specify a range of cards by encoding the starting R-L-S and ending R-L-S in the command. One Card Status ($82) report is generated for each card specified in the range. For single cards, the starting R-L-S and the ending R-L-S are the same.

Function

fcs_card_statreq(api_message *msg, char *buf)

Mode Constant

CS_CARD_STATREQ

Parameters

The parameters for this message are in Table 2-39.

Table 2-39 Card Status ($82) Command Parameters

Parameter	Type	Description
first_rack	unchar	First rack number.
first_level	unchar	First level number.
first_slot	unchar	First slot number.
first_span	unchar	First span number.
last_rack	unchar	Last rack number.
last_level	unchar	Last level number.
last_slot	unchar	Last slot number.
last_span	unchar	Last span number.

Card Status ($82) Report Parsing

This function informs the host of the status of a card. The card location is represented by both the port address and the physical rack, level, and slot (RLS) address. The report includes the status of the card and the type of the card. One Card Status ($82) report is generated for each of the cards specified in the Card Status ($82) command. In the case of a multispan card, an $82 report is generated for each span in the slot. The Card Status ($82) report takes the form of a command returned with a network status byte set to $01.

Function

frs_card_statreq(api_message *msg, char *buf)

Mode Constant

RS_CARD_STATREQ

Parameters

The parameters for this message are in Table 2-40.

Table 2-40 Card Status Report ($82) Parameters

Parameter	Type	Description
low_port	Paddr	Lower address range.
high_port	Paddr	Higher address range.
hour	unchar	Hour.
min	unchar	Minute.
sec	unchar	Second.
rack	unchar	Rack number.
level	unchar	Level number.

Table 2-40 Card Status Report ($82) Parameters (continued)

Parameter	Type	Description
slot	unchar	Slot number.
span	unchar	Span number.
card_type	unchar	Card type.
card_state	unchar	Card state.

Port Status ($83) Command Buildup

This function is used to obtain the Port Status ($83) report. The command queries the status of a range of ports specified by either a port address or the rack, level, and slot (RLS) encoding of a card, with the specified span or resource group. In the case of a query for ports in a multispan card (through RLS specification), you must also specify the span (interface).

Function

fcs_port_statreq(api_message *msg, char *buf)

Mode Constant

CS_PORT_STATREQ

Parameters

The parameters for this message are in Table 2-41.

Table 2-41 Port Status ($83) Command Parameters

Parameter	Type	Description
more_frags	unsigned	More fragments to follow?
align1	usigned:3	Unused—for alignment purposes.
pa_range	unsigned:1	Is port address range used?
rg	unsigned:1	Is a resource group used?
rls	unsigned:1	Is rack, level, slot, span used?
align2	unsigned:1	Unused—for alignment purposes.
If pa_range = TRUE:		
start_pa	Paddr	Starting port address.
end_pa	Paddr	Ending port address.
If rg = TRUE:		
align:1	ushort	Unused—for alignment purposes.
rg_num	ushort	Resource group number
align:2	Paddr	Unused—for alignment purposes.
If rls = TRUE:		
rack	unchar	Rack number.

Table 2-41 Port Status ($83) Command Parameters (continued)

Parameter	Type	Description
level	unchar	Level number.
slot	unchar	Slot number.
span	unchar	Span number.
align1	Paddr	Unused—for alignment purposes.

Port Status ($83) Report Parsing

This function informs the host of a range of ports. The command, for which the report is generated, forms the leading portion of the report. This leading portion is followed by a series of port status report elements, each of which is three bytes long. The first two bytes specify the port address; the third byte specifies the call processing status of the port. If the port range for which the status report is requested is such that one $83 report cannot accommodate all the port status report elements, the report is split into as many $83 reports as necessary. Such fragments are distinguished from each other through a continuity bit. Up to 82 port status elements can be in one $83 report, considering that the maximum length of the report is 256 bytes.

Function

(f)rs_port_statreq(api_message *msg, char *buf)

Mode Constant

RS_PORT_STATREQ

Parameters

The parameters for this message are in Table 2-42 and Table 2-43.

Table 2-42 Port Status ($83) Report Parameters

Parameter	Type	Description
more_frags	unsigned:1	More fragments to follow?
align1	unsigned:3	Unused—for alignment purposes.
pa_range	unsigned:1	Is port address range used?
rg	unsigned:1	Is a resource group used?
rls	unsigned:1	Is rack, level, slot, span used?
align2	unsigned:1	Unused—for alignment purposes.
if pa_range = TRUE		
start_pa	Paddr	Starting port address.
end_pa	Paddr	Ending port address.
if rg = TRUE		
align1	ushort	Unused—for alignment purposes.
rg_num	ushort	Resource group number.

Table 2-42 Port Status ($83) Report Parameters (continued)

Parameter	Type	Description
align2	Paddr	Unused—for alignment purposes.
if rls = TRUE		
rack	unchar	Rack number.
level	unchar	Level number.
slot	unchar	Slot number.
span	unchar	Span number.
align1	Paddr	Unused—for alignment purposes.
report_element[]	rep_ele_struct	Report Elements (MAX_REPORT_ELEMENTS). See Table 2-43.

Table 2-43 Port Status ($83) Report Elements

Parameter	Type	Description
port_address	Paddr	Port address.
cp_state	unchar	Call processing state.
supp_state	unchar	Supplementary state.
isdn_state	unchar	ISDN state.
isdn_substate	unchar	ISDN sub state.

Permanent Signal Condition ($D2) Report Parsing

This function analyzes the Permanent Signal Condition ($D2) report sent from the system. It informs a host that a line/trunk port has not released within 30 seconds of a release by the system. The report is also sent when a line/trunk goes back on hook. This function analyzes the $D2 report sent from the system.

Function

frr_psc(unchar *buf, int len, api_message *msg)

Mode Constant

RR_PSC

Parameters

The parameters for this message are in Table 2-44.

Table 2-44 Permanent Signal Condition ($D2) Report Parameters

Parameter	Type	Description
port	Paddr	Port address.
is_on_hook	unsigned:1	Is PSC due to on hook on one end of stable call?

Table 2-44 Permanent Signal Condition ($D2) Report Parameters (continued)

Parameter	Type	Description
is_error	unsigned:1	Is PSC due to error condition?
did_host_tout	unsigned:1	Was PSC caused when host timed out because of no response to initial call report?
did_host_dcon	unsigned:1	Was PSC caused when the host forcibly disconnected the port?
no_mf_rcvrs	unsigned:1	Is PSC caused because of MF receiver resource limitation?
is_sds_prob	unsigned:1	Is PSC due to an internal system problem?
is_mf_garbled	unsigned:1	Is PSC due to garbled MF digits?
is_psc	unsigned:1	Does PSC exist?
res_grp	Group	Resource group number.
hour	unchar	Hour of report generation.
min	unchar	Minute of report generation.
sec	unchar	Second of report generation.

Change Port Status ($90) Command and Report

These functions generate and parse the Change Port Status ($90) command. They enable the host to activate and deactivate individual ports on an internal service circuit, or network interface card. They also perform the same action as taking ports out-of-service using the system administration Card Maintenance utility.

Function

fcm_ch_pstatus(api_message *msg, char *buf)

frm_ch_pstatus(unchar *buf, int len, opi_message *msg)

Mode Constant

CM_CH_PSTATUS

Parameters

The parameters for this message are in Table 2-45.

Table 2-45 Change Port Status ($90) Command and Report Parameters

Parameter	Type	Description
spacer_byte	SpacerByte	User-definable spacer bytes—compiled in extended mode only.
port_deact	unsigned:2	Conditions to activate/deactivate a port.
is_port_seize	unsigned:1	Seize out on a port.
is_upd_disk	unsigned:1	Is port status change to be updated on a disk?
list_ports	unsigned:2	Change status of single port, cluster of ports or ports listed in the port address map.

Table 2-45 Change Port Status ($90) Command and Report Parameters

Parameter	Type	Description
first_port	Paddr	Port address of the first port affected by change of state.
padd_map	array, unchar	Port address map.

Port Status ($D3) Report Parsing

This function analyzes the Port Status ($D3) report sent from the system. It informs the host of a change in status of an system resource report. Change can be as a result of the following:

- Activation/deactivation of a port through system administration.
- Setting a voice path between ports through system administrative menu.
- Path/Port Reset screen.
- Inward seize detected for a port with COS=O or COS=2 and internal COS=U; port busied out by connected equipment.
- Auto Makebusy feature; port busied out after specified number of supervision errors have been detected.

Function

frs_port_status(unchar *buf, int len, api_message *msg)

Mode Constant

RS_PORT_STATUS

Parameters

The parameters for this message are in Table 2-46.

Table 2-46 Port Status ($D3) Report Parameters

Parameter	Type	Description
port	Paddr	Address of port.
old_status	unsigned:4	Old status of port before change occurred. Can be either RES_UNAVAIL or RES_AVAIL.
new_status	unsigned:4	New status of port before change occurred. Can be either RES_UNAVAIL or RES_AVAIL.
originator	unchar	Specifies whether the system or host originated the change in status and the reason. Can be HOST_UNK, SDS_UNK, SDS_SYSADM, SDS_SPATH, SDS_DEND, SDS_ABUSY or SDS_CARD.
res_grp	Group	Resource group number.
hour	unchar	Hour of report generation.
min	unchar	Minute of report generation.
sec	unchar	Second of report generation.

Resource Limitation ($D6) Report Parsing

This function analyzes the Resource Limitation ($D6) report sent from the system. It informs the host when a resource limitation condition is detected or cleared. It is generated only for the first occurrence in a specific group for a limitation condition (in response to a resource control command, inpulse rule, outpulse rule) until the condition clears.

Function

frs_res_limit(unchar *buf, int len, api_message *msg)

Mode Constant

RS_RES_LIMIT

Parameters

The parameters for this message are in Table 2-47.

Table 2-47 Resource Limitation ($D6) Report Parameters

Parameter	Type	Description
is_limitation	unsigned:1	Is resource limitation present?
res_grp	Group	Resource group number.
port	Paddr	Requesting port address.
hour	unchar	Hour of report generation.
min	unchar	Minute of report generation.
sec	unchar	Second of report generation.

Card Status ($D9) Report Parsing

This function analyzes the Card Status ($D9) report sent from the system. It informs the host of a change in the status of an system resource card. The card location is represented both by the port address and the physical rack-level-slot address. It is generated when status changes are caused by system administration, host command, or physical removal/replacement of a card.

Function

frs_card_status(unchar *buf, int len, api_message *msg)

Mode Constant

RS_CARD_STATUS

Parameters

The parameters for this message are in Table 2-48.

Table 2-48 Card Status ($D9) Report Parameters

Parameter	Type	Description
low_port	Paddr	Address of lowest port affected.
high_port	Paddr	Address of highest port affected.
is_card_off	unsigned:1	Is the card offline?
is_card_del	unsigned:1	Has the card been deleted?
is_card_add	unsigned:1	Has the card been added?
is_sds_resp	unsigned:1	Is system responsible?
is_host_resp	unsigned:1	Is the host responsible?
hour	unchar	Hour of report generation.
min	unchar	Minute of report generation.
sec	unchar	Second of report generation.
card_state	unchar	State of the card. Can be either UNK_STATE,ACT_STATE,MAINT_STATE, DIAG_STATE,OOS_STATE,SBY_STATE (only BRC and DTG) or CON_STATE.
rack	unchar	Rack number.
level	unchar	Level number.
slot	unchar	Slot number.

Alarm Condition ($F0) Report Parsing

This function performs the following actions:

- Translates the system $F0 report, pointed to by **buf**, into the Alarm Condition API message, pointed to by **msgp**.
- Provides the user with type of alarm that occurred, the level of severity of that alarm, and the number of alarms that exist.

Function

frs_alarm_cond(unchar *buf, int len, api_message *msgp)

Mode Constant

None

Parameters

The parameters for this message are in Table 2-49.

Table 2-49 Alarm Condition ($F0) Report Parameters

Parameter	Type	Description
alarm_type	unsigned integer	Identifies the type of alarm that occurred.
alarm_level	unsigned integer	Indicates the level of severity.
alarm_count	integer	Indicates the number of alarms that exist.
count_level	unsigned integer	Indicates the level of severity for the number of alarms.
almdata_ind	integer	Indicates if alarm data is present: TRUE or FALSE.
alarm_data	unsigned integer	Additional data for some alarm types.

Controlling ISDN Primary Rate Interfaces

#include “api_dtmf.h”

#include “api_dvc.h”

#include “api_mf.h”

#include “api_src.h”

#include “api_net.h”

#include “api_isdn.h”

ISDN Port Control ($49) Command and Report

These functions perform the following actions:

- Translates the ISDN Port Control API message, pointed to by **msgp**, into the corresponding system $49 message, pointed to by **buf**, or parse the message returned from the system.
- Allows the user to specify the controlling and associated ports involved with the call.
- Processes up to five ISDN Outpulse Control Segments, any one of which may contain a Binary Code Digit string, Information Element (IE) data, IE header, or a complete IE message.
- Allows the user to specify whether or not to execute an inpulse or outpulse rule.
- Copies IE segments, if any, into the correct location of the system message. The IE segments are created with the **IE_buildCallNum()** function and then copied into the **ie_segs[]** array.
- Returns the length, in bytes, of the $49 system message pointed to by **buf**.

Function

fcr_isdn_ctrl(api_message *msgp, unchar *buf)

frr_isdn_ctrl(unchar *buf, int len, api_message *msg)

Mode Constant

CR_ISDN_CTRL

Parameters

The parameters for this message are in Table 2-50.

Table 2-50 ISDN Port Control ($49) Command and Report Parameters

Parameter	Type	Description
spacer_byte	SpacerByte	User definable spacer bytes—compiled in extended mode only.
tone_plan	TonePlan	Reserved for tone plans—compiled in extended mode only.
ctrl_port	Paddr	Controlling port address.
ctrl_callid	unsigned integer	Controlling call identifier.
ctrl_addrid	unsigned integer	Controlling address identifier.
assc_port	unsigned integer	Associated port address.
assc_callid	unsigned integer	Associated call identifier.
assc_addrid	unsigned integer	Associated address identifier.
conn_switch_reqd	unsigned integer	Switching action.
conn_port_attach	unsigned integer	Attach or remove resource.
conn_port_rgrp	unsigned integer	Port or resource group.
conn_path	unsigned integer	Speech path control options.
disconn_d_bit	unsigned integer	Receive report on disconnect.
disconn_r_bit	unsigned integer	Receive report on port release.
disconn_t_bit	unsigned integer	Receive report if release was received.
disconn_i_bit	unsigned integer	Return iport to CP_SETUP when oport releases.
disconn_c_bit	unsigned integer	Send EA report if ISDN oport is released.
disconn_u_bit	unsigned integer	Force oport to IDLE or SETUP.
rule_orule_exec	unsigned integer	Is outpulse rule executed?
rule_irule_exec	unsigned integer	Is inpulse rule executed?
rule_number	Rule	Inpulse or outpulse rule number.
no_opulse_segs	unsigned integer	Outpulse segment count.
no_ie_segs	unsigned integer	Information element count.
ie_seg_len	unsigned integer	Length in bytes of the IE segment.
opulse_segs	ISDN_OPULSE_SEGS array	Outpulse segments.
ie_segs	array, unsigned char	IE segments.

ISDN Port Change of State ($EA) Report Parsing

This function performs the following actions:

- Translates the system $EA report, pointed to by **buf**, into the ISDN Port Change of State API message, pointed to by **msgp**.

- Informs the user of a network event or change of state on an ISDN PRI port.
- Provides the affected controlling and associated port.
- Provides the ISDN D-channel message type.
- Provides error status.
- Provides the outpulse rule number that was executed.
- Provides the ISDN Supervision Template number.
- Provides optional IE message segments that may have been collected by the outpulse rule. The IE message segments must be processed by the **IE_parseCallNum()** function.

Function

frr_isdn_pcos(unchar *buf, int len, api_message *msgp)

Mode Constant

RR_ISDN_PCOS

Parameters

The parameters for this message are in Table 2-51.

Table 2-51 ISDN Port Change of State ($EA) Report Parameters

Parameter	Type	Description
ctrl_port	Paddr	Controlling port address.
ctrl_callid	unsigned integer	Controlling call identifier.
ctrl_addrid	unsigned integer	Controlling address identifier.
assc_port	Paddr	Associated port address.
assc_callid	unsigned integer	Associated call identifier.
assc_addrid	unsigned integer	Associated address identifier.
change	unsigned integer	Type of change on the port.
d_channel_msg	unsigned integer	Codeset 0, Q.931 D-channel msgs.
supv_answer	unsigned integer	Is outgoing port answered or not?
supv_template	unsigned integer	ISDN supervision template used.
status	unsigned integer	Error status.
rule_number	Rule	Outpulse rule processed.
no_ie_segs	integer	IE count.
ie_seg_len	integer	Length in bytes of the ie segment.
ie_segs	array, unsigned char	IE segments.
data[5]	array, unsigned char	Optional data field.

ISDN Inpulse Rule Complete ($ED) Report Parsing

This function performs the following actions:

- Translates the system $ED report, pointed to by **buf**, into the ISDN inpulse rule complete API message, pointed to by **msgp**.
- Informs the user of the completion of an inpulse rule on a ISDN PRI port.
- Provides the affected controlling port.
- Provides the ISDN D-channel message type.
- Provides error status.
- Provides the inpulse rule number that was executed.
- Provides optional DTMF and MF digit reports in the same format as the regular Inpulse Rule Complete ($DD) report.
- Provides optional IE message segments that may have been collected by the outpulse rule. The IE message segments must be processed by the **IE_parseCallNum()** function.

Function

frr_isdn_irule(unchar *buf, int len, api_message *msgp)

Mode Constant

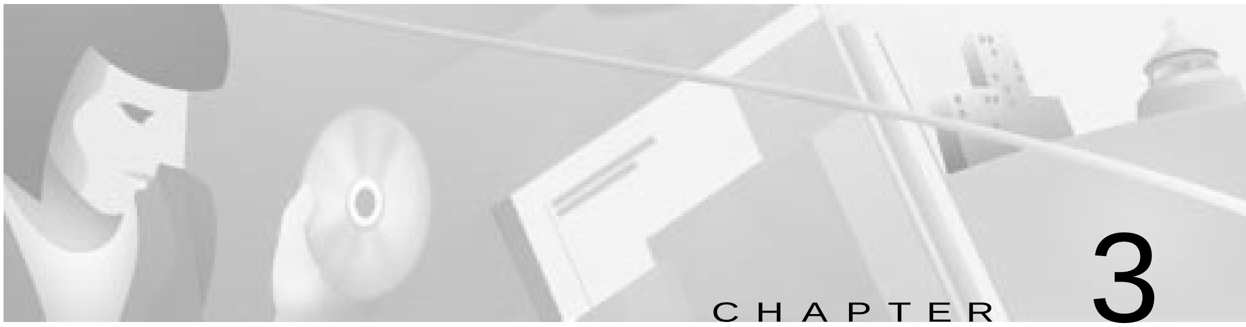
RR_ISDN_IRULE

Parameters

The parameters for this message are in Table 2-52.

Table 2-52 ISDN Inpulse Rule Complete ($ED) Report Parameters

Parameter	Type	Description
ctrl_port	Paddr	Controlling port address.
ctrl_callid	unsigned integer	Controlling call identifier.
ctrl_addrid	unsigned integer	Controlling address identifier.
rule_number	Rule	Inpulse rule executed.
d_channel_msg	unsigned integer	Received D-channel message type.
no_status_bytes	unsigned integer	Number of inpulse rule status bytes.
no_digit_segs	integer	Number of digit segment reports.
no_ie_segs	integer	Number of ie messages that follow.
ie_seg_len	integer	Length in bytes of the IE segment.
irule_status	array, unsigned char	Completion status.
digit_segs	array, ipulse_seg_t	Digit segment storage.
ie_segs	array, unsigned char	Information element segments.



Ethernet Communications

The ASIST/Ethernet software component is a set of application development tools that assists VCO/4K customers in developing host-controlled applications. This tool kit offers programmers a means to provide data communications services to a call processing application.

The ASIST/Ethernet component is application-layer software that allows a UNIX-based host to communicate with a VCO/4K. This product implements the Ethernet protocol, and allows the programmer to concentrate on the application rather than on low-level, host-to-switch communication issues. ASIST/Ethernet is ideal for the application developer who needs to prototype applications quickly in a laboratory environment.

ASIST/Ethernet is written specifically for the Sun Microsystems SunOS operating system.

ASIST/Ethernet Features

The ASIST/Ethernet component was designed with the following requirements:

- Communication with a VCO/4K over Ethernet via a single application-layer process
- Interprocess Communication Support
- Multiblocked Message Support
- Error Reporting
- Link Control
- Link Configuration
- Link Statistics
- Application access to the Ethernet process via a function call interface using a first-in, first-out (FIFO) queue mechanism.

ASIST/Ethernet is written in the C programming language.

ASIST/Ethernet Interface Format

The Ethernet Link Manager and the Ethernet Communication Driver communicate information to each other via a standard message format. This message format consists of Commands, Acknowledgments, Ethernet Packet Data, Error Reports and Link Statistics.

Message Format—Commands

The application can send the following commands to the driver:

- Initialization command—The application issues this command to initialize a host port which it intends to use for communicating with the VCO/4K. This command must supply the UNIX socket via the configuration file. The Initialization Command initializes the port for raw input/output, but does not start polling.
- Activate⁄Deactivate command—These commands start or stop polling on the link. Only a previously initialized link can be activated.
- Deinitialize command—This command turns off a given link and restores the previous state of the socket.
- Send Statistics command—This command requests link statistics from the Ethernet Communications Driver. The application receives a Statistics message in response to this command.
- Set FIFO Debug command—This command turns on verbose debugging to the standard output for all FIFO actions performed by the Ethernet Communications driver.
- Send Link Debug command—This command turns on verbose debugging to the standard output for all link actions performed by the Ethernet Communications driver.

Message Format—Acknowledgments

These message types are sent by the Ethernet Driver to the application process in response to a successfully executed command.

Message Format—Data

Data packets contain Ethernet message data. Data packets are sent in both directions—Application to Ethernet Communications Driver and Ethernet Communications Driver to Application.

Message Format—Errors

Error packets are sent by the Ethernet Communictions driver to the application to report faulty conditions on a link or the state of the read and write queues. Error packets are also sent if a command could not be executed successfully.

Message Format—Statistics

The Ethernet Driver responds to a Send Statistics command by sending a Statistics packet. This packet contains various counter values for a given link.

Link Manager—Communications Driver Interface

The following structure is the C language representation of the Link Manager-Communication Driver interface format. This data structure is located in the enet_if.h file.

```
typedef struct
}
    ushort bytecount;
    byte  pkt_type;
    union {
        DATA_PKT          qdata_pkt;
        LINK_STATS        qstat_pkt;
        ERROR_PKT         qerr_pkt;
        ACK_PKT           qack_pkt;
        CMD_PKT           qcmd_pkt;
        PARAMS            qpara_pkt;
        q_data;
}
Q_PKT;
```

ASIST/Ethernet Configuration File

The following is a sample configuration file for the Ethernet Communications Driver:

```
/*Sample configuration file for VCO/4K system #1*/
log file: test.log
log level: 1

network a name: j_jetson
network a local: 2121
network a remote: 2122

network b name: g_jetson
betwork b local: 2021
network b remote: 2022
```

The definitions of the configuration file parameters are as follows:

log file—The name of the log file you want to create (or append to) for each log message function call. The log file name is a standard UNIX file name and may be up to 256 characters in length. If a log file name is not specified, the system uses the default, which is ./enetdrvr.log.

log level—The level of logging you want to output to the file. The level is from 0 to 4, where level 0 logs the least information and level 4 logs the most information.

network a name—The Ethernet node name assigned to the ACTIVE side of the VCO/4K system to which you are communicating.

network a local—The local port number for the Ethernet host configuration on the ACTIVE side of the VCO/4K system. The value for this parameter is specified in the Local Port field on the VCO/4K Host Configuration screen.

network a remote—The remote number associated with “network a local” for the ACTIVE side of the VCO/4K system. The value for this parameter is specified in the Rem Port field on the VCO/4K Host Configuration screen.

network b name—The Ethernet node name assigned to the standby side of the VCO/4K system to which you are communicating.

network b local—The local port number for the Ethernet host configuration on the STANDBY side of the VCO/4K system. The value for this parameter is specified in the Local Port field on the VCO/4K Host Configuration screen.

network b remote—The remote number associated with “network b local” for the ACTIVE side of the VCO/4K system. The value for this parameter is specified in the Rem Port field on the VCO/4K Host Configuration screen.

ASIST/Ethernet Installation

The ASIST products were developed on a Sun Microsystems SPARCstation. While the ASIST product is independent of any particular operating system, the ASIST/Ethernet product requires a UNIX operating system environment.

This section contains a list of the media contents of the ASIST/Ethernet disk and a description of the installation and compilation procedure.

ASIST/Ethernet Media

All ASIST/Ethernet files reside on a 3.5-inch diskette (1.44 Mb). The directory of this diskette is as follows:

```
/asist/enet/control.c
    datapath.c
    enet.h
    enetdrvr.c
    enetdrvr.make
    enet_if.h
    enet_mgr.c
    enet_mgr.make
    enet_util.c
    host.c
    network.c
    types.h
```

Installing and Compiling

To copy the ASIST files from the supplied media, enter the appropriate UNIX `tar` command as shown below.

For systems running SunOS:

```
tar xvf /dev/rfd0 ./asist/enet (SunOS)
```

For systems running System V:

```
tar xvf /dev/f0t ./asist/enet (system V)
```

Each ASIST product includes a makefile for all the source modules. To compile the ASIST/Ethernet product, use the enet_mgr.make file.

The C language flag -DBSD is used in each makefile. When present, this flag indicates that the target operating system is SunOS; its absence indicates a System V environment.

Function Description

The ASIST/Ethernet product contains the following functional areas:

- ENET Link Manager—This module facilitates the interface of the customer application code to the Ethernet Communications Driver.
- ENET Communication Driver—This module executes commands received from the ENET Link Manager, transmits and receives ENET packets, and manages the link between the host and the VCO/4K.
- ENET Utilities—This module contains functions that are used by the ENET Link Manager and ENET Communications Driver modules.

Note The ENET Utilities module uses the same data structures and constants as the ENET Link Manager.

The C language source files and functions provided by each of the functional areas are as follows.

ENET Link Manager Source Files and Functions

Files:	enet_if.h/enet_mgr.c
Functions:	Prepare Data Packet Prepare Queue Packet Check Receive Queue Send Queue Packet to Driver Spawn Another Driver Process Initialize ENET Driver Stop ENET driver Initialize Link Manager Process ENET Driver Commands Send ENET Driver Commands Create ENET Driver Process Display ENET Attributes Read Data From ENET Link Write Data To Enet Link Create FIFO Name FIFO Terminate ENET Communications Driver

ENET Communication Driver Source Files and Functions

Files: enet.h/control.c/datapath.c/enetdrvr.c/host.c/network.c

Functions:
Create FIFO Name
Read Configuration File
Get Token
Print Driver Structures
Process Report
Control Data Path
Stop Data
Main
Handle Application Output Queue
Create Q_PKT
Send ACK_PKT
Send ERROR_PKT
Send Status
Open Read/Write FIFOs
Read Packet From Host Application Queue
Close Driver Connection
Display Network Statistics
Log Socket Information
Open Driver Socket
Open TCP Link
Reopen TCP Link
Close Network Side
Close Network Link
Read TCP Link
Write TCP Link
Send Report To Buffer
Send Message To VCO
Check For Report

ENET Utilities Source Files and Functions

File: enet_if.h/enet_util.c

Functions:
Open And Write To File
Convert Hex to ASCII
Convert ASCII to Hex
Convert ASCII String Into Hexadecimal Byte Stream
Convert Hexadecimal Byte Stream Into ASCII String
Display Q_PKT Contents
Output ASCII Representation of Hexadecimal Byte Stream

ENET Link Manager Data Structures and Constants

File Name: enet_if.h

```
*/


#define MAX_CFG_NAME           30     /* Max characters in config file name */
#define MAX_BLKS               10     /* Max blocking factor */
#define DONE                   1
#define FAILED                 0
#define MAX_MSG_LENGTH         256    /* max length of link message */

/*
```

Data packets contain single link message. Data packets are sent by the applicaton to the driver,to be transmitted . Similarly driver after deblocking the received link packet, sends it to the application.

```
*/

typedef struct

                  {

                         long       unused;
                         byte       link_no          /* for/from which link */

                         unsigned short bytecount;   /* no of data characters */
                         unsigned short  bytecount;
                         unsigned short  msg_no;     /* for future use */

                         unchar  data[MAX_MSG_LENGTH];

                  }

DATA_PKT;

#define DATAPKTSIZE                    sizeof (DATA_PKT)

typedef struct

                  {
                  long   unused      /* type of command */
                  byte   cmd;        /* for this link */
                  byte   link_no;
                  char   cfg_name    [MAX_CFG_NAME];
                  }

CMD_PKT;

#define CMDPKTSIZE                     sizeof (CMD_PKT)

/*
```

```
     command packets are sent only by the application to the driver.
     The following commands are supported.

*/



#define PORT_DEINIT             0      /* turn off the initialized port */
#define PORT_INIT               1      /* initialize only, no polling */
#define PORT_ACTIVE             2      /* start polling */
#define PORT_DEACTIVE           3      /* stop polling */
#define SEND_STATS              4      /* get statistics */
#define SET_FIFO_DEBUG          5      /* Set SHOW_DRVR_FIFO debug option */
#define SET_LINK_DEBUG          6      /* Set SHOW_DATA_LINK debug option */



typedef struct

       {

           long   unused;                 /* info pertains to this link */
           byte   link_no                 /* equals cmd that is being acknowledged
           char   cfg_name[MAX_CFG_NAME];  */

       }



#define ACKPKTSIZE

sizeof (ACK_PKT)

typedef
struct

           {

                  long   unused;
                  byte   error                       /* type of error */
                  byte   link_no;                    /* info pertains to this link
                  char   cfg_name{[MAX_CFG_NAME];     */
                                                     /* only for init fail*/

           }

ERROR_PKT;

#define ERRPKTSIZE       sizeof (ERROR_PKT)

/*
error packets are sent by the driver to the application in case of faulty
conditions on any of the link or the states of the FIFO.
*/
```

```
#define INIT_FAIL           0       /* Port initialization failed */
#define PORT_INACTIVE       1       /* port is not in active state */
#define POLLING_FAIL        2       /* No response in max attempts */
#define RNR                 3       /* Secondary not ready */
#define TX_FAIL             4       /* No ack till max xmt attempts */
#define NACKS_2MANY         5       /* primary sent max nacks in a row */
#defineMSG_2LONG            6       /* Number of characters > MAX_MSG_LENGTH */
```

```
typedef struct

     {

          ulong        unused;
          ulong        Npolled;                    /* Total No of times tty polled */
          ulong        Nno_eots;                   /* Total no of times no EOTS */
          ulong        Ntxed_msgs;                 /* No. of msgs txed successfully */
          ulong        Ntx_attempts;               /* No. of transmissioins */
          ulong        Nnacks;                     /* No of NACKS sent */
          ulong        Ntx_blk_msgs[MAX_BLKS];     /* No. of multiblocked msgs txed */
          ulong        Nrxed_msgs;                 /* No of msgs rxed */
          ulong        Nlrc_errs;                  /* No of LRC errors */
          ulong        Nrx_blk_msgs[MAX_BLKS];     /* No. of multiblocked msgs rxed */
          ulong        Nno_response;               /* No of times secondary failed to
                                                   */

     }

LINK_STATS;

#define STATPKTSIZE     sizeof (LINK_STATS)

/*
structure containing the configurable parameter values. On getting this structure the
driver updates its own copy. For the changes concerning baud rate, parity and stopbits
to become effective, PORT_INIT command needs to be issued.
*/
```

```
#define MAXPARACHARS8               /* Maximum chars for any parameter name
                                    */

       typedef struct

       {

       long        unused;

       char        cfg_name[MAX_CFG_  /* tty device name */
                   NAME];

       }
```

```
PARAMS
#define PARAPKTSIZEsizeof (PARAMS)
typedef struct

       {

       unsigned short  bytecount;

       byte       pkt_type;          /*type of this queue packet */

       union

          {

          DATA_PKT qdata_pkt;
          LINK_STATS qstat_pkt;
          ERROR_PKT qerr_pkt;
          ACK_PKT qack_pkt;
          CMD_PKT qcmd_pkt;
          PARAMS  qpara_pkt;


       q_data;
       }

Q_PKT;




/*
Type of packets to be exchanged between the driver and the application of the FIFO.
/*

       #define DATA_TYPE                  1
       #define STATS_TYPE                 2
       #define EROR_TYPE                  3
       #define CMD_TYPE                   4
       #define ACK_TYPE                   5
       #define PARAM_TYPE                 6

/*     used as an index into the enet_mngr [] table */

       #define LOGICAL_LINK_0             0
       #define LOGICAL_LINK_1             1
       #define LOGICAL_LINK_2             2
       #define LOGICAL_LINK_3             3
       #define LOGICAL_LINK_4             4
       #define LOGICAL_LINK_5             5
       #defineLOGICAL_LINK_6              6
       #define LOGICAL_LINK_7             7


       #define MAX_LINKS             8      /*Maximum possible communication links */
       #define MAX_FIFO_NAME         20    /* Maximum characters for FIFO name */
```

```
/* Commands supported by the driver interface */


      #defineCREATE_DRV                  1
      #define TERM_DRVR                  2
      #define LINK_INIT                  3
      #define LINK_DEINIT                4
      #define LINK_ACTIVATE              5
      #define LINK_DEACTIVATE            6
      #define SEND_PARAMS                8
      #define GET_STATS                  9
      #define SHOW_LINKS                 10




typedef struct

     {

        int          drvr_pid;
        char         destn_fifo [MAX_FIFO_NAME];
        char         source_fifo [MAX_FIFO_NAME];
        int          destn_fd;
        int          source_fd;
        char         cfg_name [MAX_CFG_NAME];
        byte         link_state;

     }

APP_ENET;
```

ENET Link Manager Functions

Names

make_datapkt, get_qpkt, check_qstate, send_toq, create_comm,init_enet, stop_enet, init_linkmngr, process_cmd, send_drvr_cmd, create_drvr, show_process, enet_read, enet_write, makefifo, fifoname, terminate_drvr

Synopsis

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <signal.h>
```

```
#if defined(BSD)
#include <sys/wait.h>
#endif
#include "types.h"
#include "enet_if.h"
DATA_PKT *make_datapkt();
int get_qpkt();
uchar check_qstate();
int send_toq();
int create_comm();
int init_enet();
int stop_enet();
void init_linkmngr();
void process_cmd();
void send_drvr_cmd();
void create_drvr();
void show_process();
int enet_read();

int enet_write();
int makefifo();
void terminate_drvr();
```

Global Data

```
APP_ENET enet_mngr[MAX_LINKS];
Q_PKT ipdata, *ipptr;
DATA_PKT thisdata, *dataptr;
CMD_PKT thiscmd, *cmdptr;
ACK_PKT thisack, *ackptr;
ERROR_PKT thiserror, *errorptr;
LINK_STATS thisstatus, *statusptr;
```

Prepare Data Packet

Name

DATA_PKT *make_datapkt(uchar *buf, ushort bytecount)

Data Structure(s)

DATA_PKT thisdata

Parameter	Type	Use	Description
buf	uchar*	input	Data from application
bytecount	ushort	input	Buffer size

Description

This function copies “bytecount” number of bytes from “buf” into the DATA_PKT “thisdata.” It returns a pointer to “thisdata.”

Prepare Queue Packet

Name

int get_qpkt(ushort type)

Data Structure(s)

Q_PKT *ipptr;
DATA_PKT *dataptr;
CMD_PKT *cmdptr;

Parameter	Type	Use	Description
type	ushort	input	Q_PKT identifier

Description

This function prepares the Q_PKT structure, pointed to by “ipptr,” to be sent to the ENET Driver process. Valid “type” identifiers for this function are as follows:

```
DATA_TYPE
CMD_TYPE
PARAM_TYPE
```

This function copies the appropriate information into the correct location of the Q_PKT. For PARAM_TYPE it calls the “make_parapkt()” function. If successful, it returns a value greater than zero; otherwise it returns FAILED.

Check Receive Queue

Name

uchar check_qstate(ushort linkid)

Data Structure(s)

APP_ENET enet_mngr[];
uchar process_stat[];
Q_PKT *ipptr;

Parameter	Type	Use	Description
linkid	ushort	input	Link identifier

Description

This function checks if the ENET Driver process has sent any Q_PKT structures to the ENET Link Manager. The queue is identified in “enet_mngr[linkid].source_fd.” It copies available data into the appropriate location in the Q_PKT structure, pointed to by “ipptr” and calls the “read()” system call. It returns the type of Q_PKT received; otherwise NULL.

Send Queue Packet To Driver

Name

int send_toq(ushort linkid)

Data Structure(s)

APP_ENET enet_mngr[];
uchar process_stat[];
Q_PKT *ipptr;

Parameter	Type	Use	Description
linkid	ushort	input	Link identifier

Description

This function writes the Q_PKT structure to the ENET Driver process. The queue is identified in “enet_mngr[linkid].destn_fd”. It calls the “write()” system call. This function returns one of the following values:

- The number of the byte written to the queue if the function completes successfully.
- 0 if the driver/link identified by “linkid” is not active
- –1 if there is an error during the “write()” system call. The global variable “errno” will contain more information about the error.

Spawn Another Driver Process

Name

int create_comm(char path[], char name[], ushort linkid)

Parameter	Type	Use	Description
path	char	input	“./enetdrvr” for ENET (or file name and directory path)
name	char	input	“enetdrvr” for ENET
linkid	ushort	input	Link identifier

Description

This function spawns another ENET Communications Driver Process. It sets and opens the FIFO files and returns one of the following values:

- 1 (DONE) if the function completes successfully
- 0 (FAILED) if the function fails

Initialize ENET Driver

Name

int init_enet(ushort linkid, char *config, char *path, char *program)

Data Structure(s)

Q_PKT ipdata *ipptr;
DATA_PKT thisdata *dataptr;
CMD_PKT thiscmd *cmdptr;
ACK_PKT thisack, *ackptr;
ERROR_PKT thiserror, *errorptr;
LINK_STATUS thisstatus, *statusptr;

Parameter	Type	Use	Description
linkid	ushort	input	Link ID
*config	char pointer	input	ENET configuration file name
*path	char pointer	input	ENET driver pathname
*program	char pointer	input	ENET driver program name

Description

This function performs the following tasks:

1. Initializes the global pointers “ipptr”, “dataptr”, “cmdptr”, “ackptr”, “errorptr”, and “statusptr.”
2. Creates the ENET driver process.
3. Activates the ENET link.

Stop ENET Driver

Name

int stop_enet(ushort linkid)

Parameter	Type	Use	Description
linkid	ushort	input	Link ID

Description

This function sends a deactivate, deinitialize, and terminate message to the ENET Driver process, which is indicated by linkid (the logical link number).

Initialize Link Manager

Name

void init_linkmngr(void)

Data Structure(s)

APP_ENET enet_mngr[], *mngrprt

Description

This function initializes the link manager structures. The process IDs are initialized to 0 indicating that the process has not been created. The link state is set to deinitialized (LINK_DEINIT).

Process ENET Driver Commands

Name

void process_cmd(int command, ushort linkid)

Parameter	Type	Use	Description
command	int	input	A valid ENET Driver command
linkid	ushort	input	Link ID

Description

This function invokes a ENET Communications Driver “command.” The commands used by this function are as follows:

```
CREATE_DRVR
TERM_DRVR
LINK_INIT
LINK_DEINIT
LINK_ACTIVATE
LINK_DEACTIVATE
GET_STATUS
SEND_PARAMS
SHOW_LINKS
```

This function calls “create_drvr(),” “terminate_drvr(),” “send_drvr_cmd(),” and “show_process().” The Link ID should reflect the logical link you want the command to operate on.

Send ENET Driver Commands

Name

void send_drvr_cmd (byte command, ushort linkid)

Parameter	Type	Use	Description
command	byte	input	A valid command packet identifier
linkid	ushort	input	Link ID of driver process

Description

This function sends “commands” to the ENET Communications Driver process. The commands sent by this function are as follows:

```
LINK_INIT
LINK_DEINIT
LINK_ACTIVATE
LINK_DEACTIVATE
GET_STATUS
SEND_PARAMS
```

This function calls “get_qpkt()” to prepare the command Q_PKT, and then send Q_PKT by calling “send_toq().”

Create ENET Driver Process

Name

void create_drvr(ushort linkid, char *path, char *name)

Parameter	Type	Use	Description
linkid	ushort	input	Link ID of driver process
*path	charpointer	input	"./enetdrvr" for ENET (or file name and directory path)
*name	charpointer	input	"enetdrvr" for ENET

Description

This function creates the ENET Communications Driver process, sets up the communication queue, and updates the link manager data structure.

Display ENET Attributes

Name

void show_process(ushort linkid)

Parameter	Type	Use	Description
linkid	ushort	input	Link ID of driver process

Description

Displays the attributes of the running process associated with the logical link number. The process ID of the ENET Communications Driver process, source and destination communication queue names, respective file descriptors, and the state of the link are shown.

Read Data from ENET Link

Name

int enet_read(int linkid, unchar *buf, int *len)

Data Structure(s)

Q_PKT *ipptr

Parameter	Type	Use	Description
linkid	int	input	ENET Driver/link identifier
buf	pointer, unsigned char	input	Receive buffer

Parameter	Type	Use	Description
len	pointer, int	input	Maximum length (in bytes) of “buf”
len	pointer, int	output	Actual length of received data

Description

This function performs the following functions:

- Reads the receive queue for messages from the ENET/link identified by “linkid.” If a DATA message is read, it copies the received message into “buf” and updates “len” with actual message length. If another message type is read, no copying takes place.
- Returns the packet type: NULL, DAT_TYPE, STATS_TYPE, ERROR_TYPE, or ACK_TYPE.
- Returns –1 if the length of the received message is longer than the buffer “len” (DATA_TYPE only).

Write Data to ENET Link

Name

int enet_write(int linkid, unchar *buf, int len)

Data Structure(s)

Q_PKT *ipptr

Parameter	Type	Use	Description
linkid	int	input	ENET Driver/link identifier
buf	pointer, unsigned char	input	Transmit buffer
len	int	input	Length (in bytes) of “buf”

Description

This function performs the following functions:

- Copies “len” bytes from “buf” into the correct Q_PKT structure and writes it to the ENET driver/link identified by “linkid.”
- Returns the number of bytes written to the driver upon successful completion.
- Returns 0 if the link identified by “linkid” is not active.
- Returns –1 if there is a write error.
- Returns –2 if the length of “buf” is greate than MAX_MSG_LENGTH.

Create FIFO

Name

int makefifo(char *path)

Parameter	Type	Use	Description
*path	char	input	FIFO pathname

Description

Communication is facilitated between the host application and the ENET Communications Driver via a data packet passing scheme involving two FIFOs: a read FIFO and a write FIFO. This function creates the FIFO indicated by "path" and returns the value from the mknod (make node) call.

Create FIFO Name

Name

char *fifoname(char *prefix, long key)

Parameter	Type	Use	Description
*prefix	char	input	Always "src_" for host's read FIFO and "dst_" for its write FIFO.
key	long	input	The process ID of the associated ENET driver.

Description

This function creates a file name for a FIFO using prefix and key arguments. These arguments are preceded by "/tmp/" in the FIFO name. This function returns the file name of the desired FIFO.

Terminate ENET Communications Driver Process

Name

void terminate_drvr(ushort linkid)

Parameter	Type	Use	Description
linkid	ushort	input	Link ID of driver process

Description

Used by the host application to terminate the communication driver process associated with the link ID. The driver process closes the socket and FIFO communications and then ends.

ENET Communications Driver Data Structures and Constants

Note The ENET Communications Driver is accessed via the ENET Link Manager. The descriptions that follow are provided for informational purposes. This module also uses all the data structures found in "enet_if.h."

File Name: enet.h

```
/* structure for host side global daa */

typedef struct {

              char           read_fifo_name [256];
              char           write_fifo_name [256];
              int            read_fifo_fd;
              int            write_fifo_fd;
              int            link_no;
              int            connected

}HOST;

/* structure for Ethernet side global data */

typedef struct {

              char           name [256];
              char           official_name [256];
              int            local _port;
              int            remote_port;
              int            status;

} LINK;

/* structure for this application global data */

typedef struct {

              int            pid;
              char           LogFile [256];
              int            LogLevel;
              char           cfg_name [256];

} DRVR;

typedef char string 10 [11];
```

```
/*

     host message structures

This is the set of possible host application data structures which may be used to
transfer data to and from the interface manager.

/*


enum host_message_types {

     SDS_RECORD,     /* the host application uses SDS message record */

};


/*

     link_type

This is the set of possible SDA interfaces, or links, which may be used to connect
to the switch.

*/



enum link_type {

                CLOSED,                 /* the link is not open */

                TCP_LINK                /* TCP_IP socket */

};


#define BROADCAST_BCA 0xDF
#define MAX_MESSAGE_LEN 512
#defineMIN_ADLC_PACKET 6
#define MIN_MESSAGE_LEN 6


#define NET_SIDE_A                      0
#define NET_SIDE_B                      1
#define HOST_CMD_A2                     2
#define HOST_RPT_A                      3

/* received message states */
```

```
#define MSG_NOTOK0                          0
#define MSG_OK                              1
#define ACCEPT_MSG                          2
#define NO_SPACE                            3


/* the data structure of the transfer buffer */

typedef struct

                unsigned short length;   /* message length */
                unsigned char data       /* message date*/
                [MAX_MESSAGE_LEN];

}
tcp_xfr_type;



typedef unsigned char byte_array [MAX_MESSAGE_LEN];          /* the raw data */



typedef union xbfr {

         tcp_xfr_type tcp;            /* TCP message */

         byte _array raw;

{ msg_buffer_type;

#define A_SIDE_ACTIVE 0x01
#define B_SIDE_ACTIVE 0x03
#define A_SIDE_STANDBY 0x00
#define B_SIDE_STANDBY 0x02


/* network status values */
#define NOTOK                             0
#define OK                                1
#define LINK_IS_BROKEN                    -1
#define LRC_CHECK_ERROR                   -2
#define LINK_LOGIC_ERROR                  -3
#define INVALID_LINK_TYPE                 -4
#define TOO_MANY_RETRIES                  -5
#define MESSAGE_CHECK_ERROR               -6
#define NO_HOST_APPLICATION               -7

#define MAX_MSG_SIZE                      256

extern int Log_Level;                     /* The amount of logging performed */

extern int errno;
```

ENET Communications Driver Functions

The ENET Communications Driver Process, enetdrvr, consists of 28 functions in 5 source code modules: control.c, datapath.c, enetdrvr.c, host.c, and network.c.

The following are the function names, synopsis and global data in the control.c file.

control.c Names

char *fifoname, int read_configuration, int get_token, print_structs

control.c Synopsis

```
#include <stdio.h>
#include <fcntl.h>
#include <poll.h>
#include <malloc.h>
#include "types.h"
#include "enet.h"
int read_configuration();
int get_token();
void print_structs();
char *fifoname();
```

control.c Global Data

```
extern HOST             host;
extern LINK             link [ ];
extern DRVR             drvr;
extern struct poll fd   connection [ ];
```

The following are the function names, synopsis and global data in the datapath.c file.

datapath.c Names:

process_report, data_path_control, stop_data_path_control

datapath.c Synopsis:

```
#include <stdio.h>
#include <errno.h>
#include <poll.h>
#include <sys/time.h>
#include "enet.h"
#include "types.h"
#include "enet_if.h"

#define POLL_TIMEOUT   1000
    void               process_report ( );
    int                data_path_control ( );
    void               stop_data_path_control ( );
    int                check (report);
```

datapath.c Global Data

```
extern HOST  host;
extern LINK  link[];
extern DRVR  drvr;
extern char  LogStr[];
extern int  shut_down;
extern int  active_network;
extern struct pollfd  connection[4];
extern Q_PKT  ipdata, *ipptr;
```

The following are the function names, synopsis and global data in the enetdrvr.c file.

enetdrvr.c Names

```
main
```

enetdrvr.c Synopsis

```
#include <stdio.h>
#include <signal.h>
#include <poll.h>
#include <fcntl.h>
#include "enet.h"
#include "types.h"
#include "enet_if.h"


/* function prototypes */

     int main( );

/* external function prototypes */

     int               parse_arguments ( );
     int               data_path_control ( );
     void              stop_data_path_control ( );
```

enetdrvr.c Global Data

```
HOST  host;
LINK  link[2];
DRVR  drvr;
Q_PKT  ipdata,  *ipptr;
DATA_PKT  thisdata, *dataptr;
CMD_PKT  thiscmd, *cmdptr;
ACK_PKT  thisack, *ackptr;
ERROR_PKT  thiserror, *errptr;
LINK_STATS  thisstatus, *statptr;
PARAMS  paras, *paraptr;
```

The following are the function names, synopsis and global data in the host.c file.

host.c Names

handle_que, send_toque, ack_toque, error_toq, stat_toq, setup_que, read_que, close_host

host.c Synopsis

```
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <stropts.h>
#include <poll.h>
#include <sys/types.h>
#include "enet.h"
#include "types.h"
#include "enet_if.h"

#define MAX_QTRIES   10       /* max attempts to open fifos */



/* function prototypes */

       int        handle_que( );
       int        send_toque( );
       void       ack_toque( );
       void       error_toq( );
       void       stat_toq( );
       int        setup_que( );
       int        read_que( );
       int        close_host( );

/* external function prototypes */

       extern char  *fifoname();
```

host.c Global Data

```
extern HOST           host;
extern LINK           link[];
extern DRVR           drvr;
extern Q_PKT          ipdata,*ipptr;
extern DATA_PKT       thisdata,*dataptr;
extern CMD_PKT        thiscmd,*cmdptr;
extern ACK_PKT        thisack,*ackptr;
extern ERROR_PKT      thiserror,*errptr;
extern LINK_STATS     thisstatus,*statptr;
extern PARAMS         paras, *paraptr;
```

The following are the function names, synopsis and global data in the network.c file.

network.c Names

print_netstat, print_sockaddr, open_network, open_network_link,reopen_network_link, close_network, close_network_link,read_network_link, write_network_link, network_receive, pass_to_network, check_report

network.c Synopsis

```
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <termio.h>
#include <poll.h>
#include <signal.h>
#include <time.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include "types.h"
#include "enet_if.h"
#include "enet.h"

typedef unsigned char link_packet[258];

typedef struct sockaddr_in SOCK_INET;    /*just a typedef of sockaddr_in */



/* function prototypes */

        void  print_netstat( );
        void  print_sockaddr( );
        int  open_network_link( );
        void  reopen_network_link( );
        int  close_network_link( );
        int  read_network_link( );
        int  write_network_link( );
        int  open_network( );
        int  close_network( );
        int  network_receive( );
        int  pass_to_network( );
        int  check_report( );

/* external function prototypes */

        int  get_token ();
```

network.c Global Data

```
extern HOST  host;
extern LINK  link[2];
extern DRVR  drvr;
extern char  LogStr[];
extern int  shut_down;
extern int  link_debug;
extern int  active_network;
```

Create FIFO Name

Module: control.c

Name

char *fifoname(char *prefix, longkey)

Parameter	Type	Use	Description
*prefix	char	input	A character string to prefix the file name
key	long	input	A unique ID for this file name

Description

This function returns a file name for the FIFO using the prefix and key arguments. The prefix argument is always either “src_” for the host applications read FIFO, or “dst_” for the host applications write FIFO. The key argument is the process ID number for the associated communications driver. These arguments are preceded by “/tmp” in the FIFO name. This function returns a pointer to the character string containing the file name.

Read Configuration File

Module: control.c

Name

int read_configuration(char *name)

Parameter	Type	Use	Description
*name	char	input	A valid UNIX file name

Description

This function reads the configuration file which is pointed to by “name.” The specified items in the configuration file are read in to the appropriate global variables contained in the driver program.

Returns

This function returns 0 (TRUE) if the file exists and contains valid data, –1 (FALSE) if the file does not exist or the data contained in the file is invalid.

Get Token

Module: control.c

Name

int get_token(FILE *fil, char *tok, int siz)

Parameter	Type	Use	Description
*fil	char pointer	input	An opened file pointer
*tok	char pointer	input	Return value of the next token read
siz	int	input	Size of the token

Description

This function is used by “read_configuration” to read individual portions of the configuration file.

Returns

This function returns the following values:

- 1 if tok contains a token
- 0 if the end of the file is reached
- –1 for any other error

Print Driver Structures

Module: control.c

Name

void print_structs(void)

Description

This function prints the contents of the three major structures (extern HOST, extern LINK, and extern DRVR) of the ENET Communications Driver process to standard output.

Process Report

Module: datapath.c

Name

void process_report(int net, unsigned char *msg, int len)

Parameter	Type	Use	Description
net	int	input	The network from which the report came
*msg	uchar	input	Pointer to the report data
len	int	input	Length of report data

Description

This function takes an incoming report from the VCO/4K and sends it to the application by placing it in the FIFO. This function also checks for redundant switchover and adjusts its global variables so future commands are sent to the correct TCP link.

Control Data Path

Module: datapath.c

Name

int data_path_control(void)

Description

This function polls the TCP and FIFO links for data awaiting processing. If there is data on the link, this function calls the appropriate functions to parse and send the data to its destination. Polling is continuous as long as the global "shut_down" variable is set to –1 (FALSE).

Returns

This function always returns 0 (TRUE).

Stop Data

Module: datapath.c

Name

void stop_data_path_control(int sig, int code, struct sigcontext *scp, char *addr)

Parameter	Type	Use	Description
sig	int	input	Information sent by the signal handler
code	int	input	Information sent by the signal handler

Parameter	Type	Use	Description
*scp	struct sigcontext	input	Information sent by the signal handler
*addr	char	input	Information sent by the signal handler

Description

This function is used for the three UNIX signals that are caught and processed by the driver process (SIGTERM, SIGQUIT, and SIGINT). It sets the global "shut_down" variable to 0 (TRUE), which sets up a graceful shutdown of the driver.

Main

Module: enetdrvr.c

Name

int main(int argc, char**argv)

Parameter	Type	Use	Description
argc	int	input	Standard C command line argument
**argv	char	input	Standard C command line argument

Description

This function is the main function of the enetdrvr process. It is responsible for initializing global variables, setting up the signal handlers and starting the polling loop. When the polling is complete, it shuts down the network and host connections and returns to the operating system.

Returns

This function returns the status of "data_path_control," which is always 0 (TRUE).

Handle Application Output Queue

Module: host.c

Name

int handle_que(void)

Description

This function is called by the polling rouine when data is available on the host applications output queue. This packet is read and the proper actions are taken depending on the type of packet sent. In general, a data packet contains a command to the VCO/4K, a command packet indicates the host wants the driver to perform a specific action.

Returns

This function returns either a valid pkt_type, or a NULL if nothing is read.

Create Q_PKT

Module: host.c

Name

int send_toque(void)

Description

This function writes the packet in the global Q_PKT ippr structure to the host applications FIFO queue.

Returns

This function returns OK if the data is written. Otherwise, NOTOK.

Send ACK_PKT

Module: host.c

Name

void ack_toque(byte ack)

Parameter	Type	Use	Description
ack	byte	input	One of the valid commands (PORT_INT, PORT_DEINT, etc.)

Description

This function is used by the ENET Communications Driver software to send an ACK packet to the host application. ACK packets are sent to acknowledge to the host application that its command was carried out.

Send ERROR_PKT

Module: host.c

Name

void error_toque(byte ack)

Parameter	Type	Use	Description
error	ushort	input	A valid error code

Description

This function is used by the ENET Communications Driver software to send an error packet to the host application. An error packet is sent to notify the applications that a command it sent has failed. Valid error packet types can be found in enet_if.h.

Send Status

Module: host.c

Name

void stat_toq(void)

Description

This function is used by the ENET COmmunications Driver to send statistics about its links to the host applications process in response to a SEND_STATS command. Statistics are reset to zero (and begin accumulating again) after they are sent to the host applications.

Open Read/Write FIFOs

Module: host.c

Name

int setup_que(void)

Description

This function instructs the ENET Communications Driver to open the host applications FIFOs for reading and writing. The queues must have been created by the host application prior to calling this function.

Returns

This function returns OK if the queue is opened successfully. Otherwise, NOTOK.

Read Packet from Host Application Queue

Module: host.c

Name

int read_que(void)

Description

This function is used by the ENET Communications Driver to read one packet from the host application's output queue. It is used internally by "handle_que."

Returns

This function returns DONE if data has been read. Otherwise, NULL.

Close Driver Connection

Module: host.c

Name

int close_host(void)

Description

This function is used to close the enetdrvr process connection to the host applications FIFOs. A message is sent to the log file to indicate this function has been called.

Returns

This function always returns 0.

Display Network Statistics

Module: network.c

Name

void print_netstat(int network)

Parameter	Type	Use	Description
network	int	input	The network side for which the statistics are being displayed.

Description

This function is used to print the log file various statistics about the network TCP connection to standard output. The statistics include connection type, packets received, number of faults, reports received, average size, and number of link responses. "network" should indicate either NET_SIDE_A or NET_SIDE_B.

Log Socket Information

Module: network.c

Name

void print_sockaddr(SOCK_NET *sock, char *lab)

Parameter	Type	Use	Description
*sock	SOCK_NET	input	The socket for which the data is being printed
*lab	char	input	Text label

Description

This function prints information about the specific socket into a log file. Information includes the family, port, and address of the socket. "lab" is a user-definable string, which adds information to the file output.

Open Driver Socket

Module: network.c

Name

int open_network(int side)

Parameter	Type	Use	Description
side	int	input	The network side to open

Description

This function is used to set up the TCP sockets for the ENET Communications Driver to talk to the VCO/4K system. Two sides can be opened for redundant systems. (this feature is set in the configuration file.) “side” should indicate either NET_SIDE_A or NET_SIDE_B.

Returns

If this function is successful, a 0 is returned. Otherwise, a –1 is returned.

Open TCP Links

Module: network.c

Name

int open_network_link(int side, struct pollfd *pfd)

Parameter	Type	Use	Description
side	int	input	The network side to open
*pfd	struct pollfd	input	The poll structure of the UNIX system

Description

This function is used internally by the open_network() to open the TCP links to the VCO/4K system. “*pfd” structure should be set up as indicated in he UNIX poll function intsructions. “side” should indicate either NET_SIDE_A or NET_SIDE_B.

Returns

If the function completes successfully, it returns the file ID. Otherwise, it returns a –1.

Reopen TCP Link

Module: network.c

Name

void repoen_network_link(int sig, int code, struct sigcontext *scp, char *addr)

Parameter	Type	Use	Description
*sig	int	input	Information sent by signal handler
code	int	input	Information sent by signal handler

Parameter	Type	Use	Description
*scp	struct	input	Information sent by signal handler
*addr	char	input	Information sent by signal handler

Description

This function is called when the VCO/4K system sends a signal indicating that the TCP link is down. A 90-second signal alarm is sent to allow the Ethernet link on the VCO/4K system to reset. When the alarm is raised, it calls this function to attempt to reopen the link. If the link cannot be reopened, the link variables are cleared and a POLLING_FAIL error is sent to the host application.

Close Network Side

Module: network.c

Name

int close_network(int side)

Parameter	Type	Use	Description
side	int	input	The network side to close

Description

This function is used to stop all communications from the ENET Communications Driver to the VCO/4K system. “side” should be either NET_SIDE_A or NET_SIDE_B.

Returns

If the network side is closed successfully, this function returns a 0. Otherwise, it returns a –1.

Close Network Link

Module: network.c

Name

int close_network_link(struct pollfd *pfd)

Parameter	Type	Use	Description
*pfd	struct	input	The poll structure for the link

Description

This function is used internally by “close_network” to handle the mechanics of closing the network link.

Returns

If the link is closed successfully, this function returns a 0. Otherwise, it returns a –1.

Read TCP Link

Module: network.c

Name

int read_network_link(int fd, unsigned char *msg, int size)

Parameter	Type	Use	Description
fd	int	input	The file ID of the link
*msg	uchar pointer	input	Pointer to the buffer which will receive the data
size	int	input	The size of the buffer message

Description

This function is used internally by “network_receive.” It handles the mechanics of reading data from the TCP link and storing it in a buffer for processing.

Returns

If the function completes successfully, it returns the number of bytes stored. Otherwise, it returns a –1.

Write TCP link

Module: network.c

Name

int write_network_link(int fd, unsigned char *msg, int size)

Parameter	Type	Use	Description
fd	int	input	The file ID of the link

Parameter	Type	Use	Description
*msg	uchar pointer	input	Pointer to the buffer from which to write
size	int	input	The number of bytes from the buffer to write

Description

This function writes “size” number of bytes from the msg buffer to the VCO/4K system which is pointed to by the fd descriptor.

Returns

If the function completes successfully, it returns 0. Otherwise, it returns a –1.

Send VCO Report

Module: network.c

Name

int network_receive(int network, unsigned char *message, int size)

Parameter	Type	Use	Description
network	int	input	The network side from which to read
*message	unsigned char	input	The buffer that will hold the data read
size	int	input	The size of the buffer

Description

This function transfers a single report from the VCO/4K system into the buffer pointed to by “*message.” “network” should indicate either NET_SIDE_A or NET_SIDE_B.

Returns

This function returns one of the following values:

- The number of bytes read
- –1 (LINK_IS_BROKEN) if the read fails
- 0 if nothing is read

Send Message to VCO

Module: network.c

Name

int pass_to_network(int network, unsigned char *message, int length)

Parameter	Type	Use	Description
network	int	input	The network side to which to write
*message	unsigned char	input	The buffer containing the data to write to the VCO/4K system
length	int	input	The number of bytes to write to the message buffer

Description

This function writes a command to the VCO/4K system indicated by “network.” Before sending the message, this function checks the length of the message. If the message is too long for the buffer, the message is not sent. The function does not perform any other tasks on the message; it is sent unchanged. “side” should indicate either NET_SIDE_A or NET_SIDE_B.

Returns

This function returns one of the following values:

- 0 if the write is successful
- “LINK_IS_BROKEN” if the write fails
- “MSG_2LONG” if the message is greater than “MAX_MSG_LEN”

Check for Report

Module: network.c

Name

int check_report(int network, unsigned char *msg_data, int size)

Parameter	Type	Use	Description
network	int	input	The network side from which the report data is collected
*msg_data	unsigned char	input	The buffer which will hold the report
size	int	input	The size of the msg_data buffer

Description

This function checks the VCO/4K system pointed to by “network” to see if data is available. “side” should indicate either NET_SIDE_A or NET_SIDE_B.

Returns

If this function is successful, it returns the number of bytes received from the VCO/4K. If an error occurs, the function returns a 0 or a –1.

ENET Utilities Data Structures and Constants

This module uses the same data structures and constants as the ENET Link Manager.

Ethernet Utilities Functions

The utilities module, enet_util.c, contains seven functions which are used by both the ENET Link Manager and ENET communications driver.

Open and Write to File

Name

int logMsg(char *log_file, char *msg)

Parameter	Type	Use	Description
*log_file	char	input	A valid UNIX file name
*msg	char	input	Text string to write to file

Description

This function attempts to open the file name passed in “*log_file” and then writes into the file the string pointed to by “*msg.” This function is included for compatibility with the ADLC product.

Returns

If this function completes successfully, it returns 0. Otherwise, it returns –1.

Convert Hex to ASCII

Name

unchar hex2ascii(unchar hexval)

Parameter	Type	Use	Description
hexval	unchar	input	A hexadecimal value

Description

This function returns the ASCII character equivalent to the value passed to it in “hexval.” This function is included for compatibility with the ADLC product.

Returns

The ASCII equivalent of “hexval.”

Convert ASCII to Hex

Name

unchar ascii2hex(int asciival)

Parameter	Type	Use	Description
asciival	int	input	An ASCII character

Description

This function returns the hexadecimal value that is equivalent to the character passed to it in “asciival.” This function is included for compatibility with the ADLC product.

Returns

The hexadecimal value of “asciival.”

Convert ASCII String into Hexadecimal Byte Stream

Name

int str2hex(char *str, uchar *buf)

Parameter	Type	Use	Description
*str	char	input	A string of ASCII characters
*buf	uchar	input	Returns the hexadecimal byte stream

Description

This function translates the ASCII string pointed to by “*str” into a hexadecimal byte stream. This function is included for compatibility with the ADLC product.

Returns

The number of bytes in the “buf” array.

Convert Hexadecimal Byte Stream into ASCII String

Name

char *hex2str(uchar *buf, int buflen)

Parameter	Type	Use	Description
*buf	uchar	input	The hexadecimal byte array
buflen	int	input	Number of bytes to translate from “buf”

Description

This function translates the hexadecimal byte stream located in the buffer pointed to by “*buf” into an ASCII string. “buflen” specifies how many bytes in “buf” to translate. It inserts a blank character between every two digits in the output string. It calls the “malloc()” function to allocate space for the ASCII string.

Returns

This function returns a pointer to the ASCII string.

Display Q_PKT Contents

Name

void show_qpkt(Q_PKT *qpktp)

Parameter	Type	Use	Description
*qpktp	Q_PKT	input	Packet to display

Description

This function displays the contents of the Q_PKT packet (pointed to by “*qpktp” and identified by the pkt_type member). It displays a detailed breakdown of the packet on the standard output.

Output ASCII Representation of Hexadecimal Byte Stream

Name

void displayHex(uchar buf, int buflen)

Parameter	Type	Use	Description
buf	uchar	input	Buffer of byte stream to display
buflen	int	input	Number of bytes to display

Description

This function displays on the standard output the ASCII representation of the hexadecimal byte stream pointed to by “buf” up to “buflen” bytes. It inserts a blank character after each displayed byte. This function is standard output to check its contents.