Dialogic
Making Innovation Thrive™

Dialogic® TX Series SS7 Boards

CPI Library Developer's Reference Manual

July 2009 64-0458-01

www.dialogic.com

Copyright and legal notices

All contents of this document are furnished for informational use only and are subject to change without notice and do not represent a commitment on the part of Dialogic Corporation or its subsidiaries ("Dialogic"). Reasonable effort is made to ensure the accuracy of the information contained in the document. However, Dialogic does not warrant the accuracy of this information and cannot accept responsibility for errors, inaccuracies or omissions that may be contained in this document.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH DIALOGIC® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN A SIGNED AGREEMENT BETWEEN YOU AND DIALOGIC, DIALOGIC ASSUMES NO LIABILITY WHATSOEVER, AND DIALOGIC DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF DIALOGIC PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHT OF A THIRD PARTY.

Dialogic products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Due to differing national regulations and approval requirements, certain Dialogic products may be suitable for use only in specific countries, and thus may not function properly in other countries. You are responsible for ensuring that your use of such products occurs only in the countries where such use is suitable. For information on specific products, contact Dialogic Corporation at the address indicated below or on the web at www.dialogic.com.

It is possible that the use or implementation of any one of the concepts, applications, or ideas described in this document, in marketing collateral produced by or on web pages maintained by Dialogic may infringe one or more patents or other intellectual property rights owned by third parties. Dialogic does not provide any intellectual property licenses with the sale of Dialogic products other than a license to use such product in accordance with intellectual property owned or validly licensed by Dialogic and no such licenses are provided except pursuant to a signed agreement with Dialogic. More detailed information about such intellectual property is available from Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. Dialogic encourages all users of its products to procure all necessary intellectual property licenses required to implement any concepts or applications and does not condone or encourage any intellectual property infringement and disclaims any responsibility related thereto. These intellectual property licenses may differ from country to country and it is the responsibility of those who develop the concepts or applications to be aware of and comply with different national license requirements.

Any use case(s) shown and/or described herein represent one or more examples of the various ways, scenarios or environments in which Dialogic® products can be used. Such use case(s) are non-limiting and do not represent recommendations of Dialogic as to whether or how to use Dialogic products.

Dialogic, Dialogic Pro, Brooktrout, Diva, Cantata, SnowShore, Eicon, Eicon Networks, NMS Communications, NMS (stylized), Eiconcard, SIPcontrol, Diva ISDN, TruFax, Exnet, EXS, SwitchKit, N20, Making Innovation Thrive, Connecting to Growth, Video is the New Voice, Fusion, Vision, PacketMedia, NaturalAccess, NaturalCallControl, NaturalConference, NaturalFax and Shiva, among others as well as related logos, are either registered trademarks or trademarks of Dialogic Corporation or its subsidiaries. Dialogic's trademarks may be used publicly only with permission from Dialogic. Such permission may only be granted by Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. Any authorized use of Dialogic's trademarks will be subject to full respect of the trademark guidelines published by Dialogic from time to time and any use of Dialogic's trademarks requires proper acknowledgement.

Windows is a registered trademark of Microsoft Corporation in the United States and/or other countries. The names of actual companies and product mentioned herein are the trademarks of their respective owners.

This document discusses one or more open source products, systems and/or releases. Dialogic is not responsible for your decision to use open source in connection with Dialogic products (including without limitation those referred to herein), nor is Dialogic responsible for any present or future effects such usage might have, including without limitation effects on your products, your business, or your intellectual property rights.

Revision history

Revision	Release date	Notes
1.0		GJG
1.2	March 1999	GJG
1.3	November 2000	GJG, SS7 3.6
1.5	August 2001	GJG, SS7 3.8 beta
1.6	November 2003	MCM, SS7 4.0 beta
1.7	April 2004	MCM, SS7 4.0
1.8	June 2008	SRG, SS7 5.0
64-0458-01	July 2009	LBG, SS7 5.1
Last modified: July 7, 2009		

Refer to www.dialogic.com for product updates and for information about support policies, warranty information, and service offerings.

Table Of Contents

1 Introduction

The CPI library provides a consistent communications mechanism to the TX board, regardless of the operating system employed on the host (Windows or UNIX). The *Dialogic® TX Series SS7 Boards CPI Library Developer's Reference Manual* explains how to use the CPI library to facilitate application development on NMS Communications TX boards.

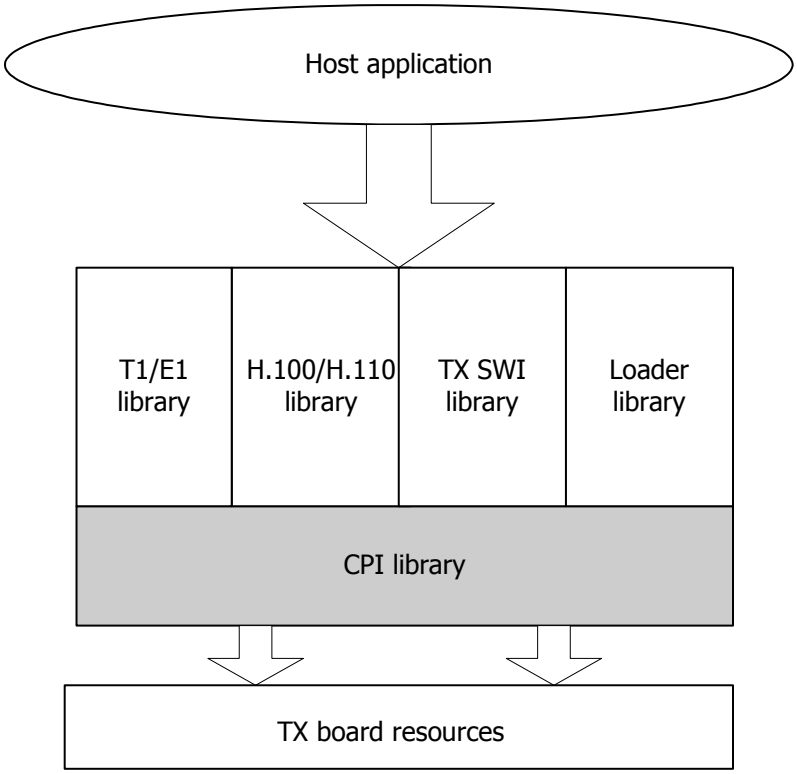
Note: The product(s) to which this document pertains is/are among those sold by NMS Communications Corporation ("NMS") to Dialogic Corporation ("Dialogic") in December 2008. Certain terminology relating to the product(s) has been changed, whereas other terminology has been retained for consistency and ease of reference. For the changed terminology relating to the product(s), below is a table indicating the "New Terminology" and the "Former Terminology". The respective terminologies can be equated to each other to the extent that either/both appear within this document.

Former terminology	Current terminology
NMS SS7	Dialogic® NaturalAccess™ Signaling Software
Natural Access	Dialogic® NaturalAccess™ Software

2 Overview of the CPI library

Development environment

The TX host application development environment (shown in the following illustration) consists of libraries that enable you to configure and control the protocol engines loaded on the TX board. This manual describes the CPI library.



CPI library definition

A channel is the basic unit of communication to the TX boards. The channel provides a multiplexing or de-multiplexing packet-based interface between the host operating system and one or more TX boards. The combination of board number (CP number) and channel is known as a logical port.

To implement multiplexing and de-multiplexing, a header is inserted on all packets transferred between the host and the TX boards. The header includes the following information:

Source channel	Source CP	Destination channel	Destination CP	Length	Data
1 byte	1 byte	1 byte	1 byte	2 bytes	1 through 1512 bytes

TX boards are numbered 1 through 16. The host is assigned the CP number of 0. The length field indicates the length of the entire data packet, including the header. Channels are numbered 0 through 255, and channel 0 is reserved.

A channel number is assigned to a task on the TX board by a prior arrangement, similar to the ports concept used in TCP/UDP. To minimize conflicts, NMS recommends the following channel usage:

Channel	Usage
0 through 31	Reserved
32 through 127	Available for use by applications
128 through 255	Reserved

The communications mechanism is similar to UDP datagrams. Tasks on the TX board register to receive all data from a particular channel. Host applications pick an unused channel to use and register to receive all packets on the chosen channel. Communications are accomplished through a connectionless datagram type of service. Due to the nature of such a service, most tasks on the TX board respond to requests from the host application by returning an indication of success or failure of the request. This response is at the application level, not at the CPI layer.

The following code sample provides a list of channels used by tasks on the TX board. These channels are defined in the *dpriface.h* file.

```
#define PT_MGR      0x00          /* Host Control Manager [$manager channel] */
#define PT_OACDRV   0x01          /* Open Access Interface to driver */
#define PT_SWI      0x03          /* Switching Control Channel */
#define PT_CONSOLE  0x06          /* Console Channel */
#define PT_LOADER   0x07          /* Loader Channel */
#define PT_DEBUG    0x08          /* Debug Channel */
#define PT_MVIP     0x09          /* MVIP Control Channel */
#define PT_T1E1C    0x0A          /* T1/E1 Control Channel */
#define PT_T1E1S    0x0B          /* T1/E1 Status Channel */
#define PT_INF      0x0C          /* Alarm Manager (raw alarm channel) */
#define PT_SS7MON   0x0F          /* SS7 Monitor API port #1 */
#define PT_ARP      0x12          /* ARP Protocol Channel */
#define PT_SS7MON2  0x13          /* SS7 Monitor API port #2 */
#define PT_ISUP     0x14          /* SS7 ISUP Task Channel */
#define PT_MTP      0x15          /* SS7 MTP Task Channel */
#define PT_TCAP     0x17          /* SS7 TCAP Task Channel */
#define PT_IUP      0x18          /* SS7 IUP Task Channel */
#define PT_TXMON    0x19          /* TX Monitor Task Channel */
#define PT_TUP      0x1D          /* SS7 TUP Task Channel */
#define PT_SCCP     0x1E          /* SS7 SCCP Task Channel */
```

The *txcpi.h* include file provides all CPI library function prototypes and literal definitions. Always use the structure packing compile option when compiling source code that uses functions from this library.

Accessing the TX device driver using Windows

The CPI library uses standard Windows routines to access the TX kernel mode device driver. The interface between the library and the driver is based on a Windows file handle. The library opens a channel like a file, reads from and writes to the driver like a file, and closes the channel like a file.

The host can receive packets asynchronously. Windows provides standard mechanisms for receiving unsolicited packets. The library posts read calls to the driver that do not block. The application can then use Windows **WaitForSingleObject** or **WaitforMultipleObjects** to determine when those reads complete with a received packet from the TX device. Use **cpi_wait_obj** to retrieve the handle to pass to these Windows calls. Pass zero in the ***dwTimeout*** parameter, which is equivalent to polling for packets, to tell Windows calls to return immediately. The same parameter can be set to infinite, in which case it does not return until there is a packet (when using **WaitForSingleObject**) or one of the list of handles had something to report (when using **WaitForMultipleObjects**).

A flow control mechanism queues TX board messages on the board if the host-based application does not service received packets quickly enough. The flow control mechanism removes the possibility of the TX driver needing to drop received packets. A similar mechanism exists for packets sent from a host-based application to the TX board.

Accessing the TX device driver using UNIX

The TX driver for UNIX systems is a streams driver and is directly accessed through the standard system calls, **open***,* **close***,* **putmsg***,* **getmsg***,* and **ioctl**.

Because the driver communicates with applications using a specific driver-to-application protocol, direct access is not recommended.

The CPI library uses a TX_HANDLE type as an object on which all I/O is done. In UNIX systems, pass the TX_HANDLE to **cpi_wait_obj** to obtain a standard UNIX file descriptor. The host UNIX system can asynchronously receive packets from a TX board by using the **poll** system call or the **select** system call.

For example, to wait on both input and packets from a TX board, use **poll** on a UNIX system as follows:

```
.
        struct pollfd fds[2];
        .
        .

        cpi_init(0, &str);
        mode = CPIM_PORT;
        port = PORT((S16)board, (S16)chan);

        if ((txhandle = cpi_open(port, mode, NULL)) < 0)
        {
            < Error handling code >
        }
        fd = cpi_wait_obj (txhandle);
        for (;;)
        {
            fds[0].fd = 0;          /* fd for standard input */
            fds[0].events = POLLIN;
            fds[0].revents = 0;
            fds[1].fd = fd;         /* TX fd */
            fds[1].events = POLLIN;
            fds[1].revents = 0;

            if (poll(fds, 2, -1) < 0)
            {
                < Error handling code >
            }

            for (i = 0; i < 2; i++)
            {
                if (fds[i].revents & (POLLERR | POLLHUP | POLLNVAL))
                {
                    < Error handling code >
                }

                if (fds[i].revents & POLLIN)
                {
                    /* TX receive */
                    if (fds[i].fd == fd)
                    {   len = sizeof (CPIPKT);
                        if (ret = cpi_get_data(txhandle,&inbuf,&len))
                        {
                            < Error handling code >
                        }
                        .
                        .
                        < Code to process data >
                        .
                        .
                    }
                    /* Terminal input */
                    else if (fds[i].fd == 0)
                    {
                    }
                }

            } /* for i */

        } /* for ever */
```

3 Function reference

Function summary

The CPI library provides synchronous and asynchronous interfaces to the TX board:

Interface	Description
Synchronous	• Can allow packet loss and can introduce data overload conditions. • Adequate for simple applications but not as efficient as asynchronous interfaces. • Can stall multiple-threaded calling applications until a response is received from the TX host-based driver. • Does not include a flow control mechanism, causing dropped packets and resource depletion during heavy packet traffic, whether from host to TX board or from TX board to host. • Used when opening a CPI channel using **cpi_open**.
Asynchronous	• Recommended for all development because it is more efficient than synchronous interfaces. • Does not stall the calling application because all responses from the driver are handled as independent events. For certain operating systems, multiple-threaded applications must use an asynchronous interface. For operating systems that do not impose this restriction, an asynchronous interface is still recommended. • Includes flow control mechanisms to ensure that no packets are dropped and no depletion conditions are introduced due to host traffic. • Used when opening a CPI channel with **cpia_open**.

Not all CPI library functions can handle asynchronous I/O and synchronous I/O functions. Mixed-mode (synchronous and asynchronous) I/O on the same handle is not allowed. The following table summarizes the CPI functions and their modality. An asterisk (*) indicates a user-supplied function.

Function	Synchronous	Asynchronous
cpia_chkey	No	Yes
cpia_get_data	No	Yes
cpia_intr	No	Yes
cpia_open	No	Yes
cpia_rxnotify*	No	Yes
cpia_send	No	Yes
cpia_txnotify*	No	Yes
cpi_check_bs	Yes	Yes
cpi_close	Yes	Yes
cpi_cptoh_l	Yes	Yes

Function	Synchronous	Asynchronous
cpi_cptoh_s	Yes	Yes
cpi_force_bs	Yes	Yes
cpi_get_board	Yes	Yes
cpi_get_data	Yes	No
cpi_get_dev_info	Yes	Yes
cpi_get_error_str	Yes	Yes
cpi_get_last_error	Yes	Yes
cpi_get_resources	Yes	Yes
cpi_htocp_l	Yes	Yes
cpi_htocp_s	Yes	Yes
cpi_init	Yes	Yes
cpi_nmi	Yes	Yes
cpi_open	Yes	No
cpi_read_control	Yes	Yes
cpi_read_dpr	Yes	Yes
cpi_send	Yes	No
cpi_set_cpid	Yes	Yes
cpi_show_stats	Yes	Yes
cpi_stats	Yes	Yes
cpi_wait_msg	Yes	No
cpi_wait_obj	Yes	Yes
cpi_write_control	Yes	Yes
cpi_write_dpr	Yes	Yes

Using the function reference

This section provides an alphabetical reference to the CPI library functions. A typical function definition includes the following:

Prototype	The prototype is shown followed by a listing of the function arguments. Dialogic data types include: • U8 (8-bit unsigned) • S8 (8-bit signed) • U16 (16-bit unsigned) • S16 (16-bit signed) • U32 (32-bit unsigned) • S32 (32-bit signed) If a function argument is a data structure, the complete data structure is defined.
Return values	The return value for a function is either CPI_SUCCESS or an error code. For asynchronous functions, a return value of CPI_SUCCESS (zero) indicates that the function was initiated. Subsequent events indicate the status of the operation.

cpia_chkey

Returns the user-provided key associated with the specified ***handle***.

Prototype

#include *txcpi.h*

void ***cpia_chkey*** (TX_HANDLE ***handle***)

Argument	Description
handle	TX_HANDLE associated with the channel.

Return values

Return value	Description
NULL	Provided TX handle is not a handle to an asynchronous channel.

Details

One of the parameters provided to **cpia_open** is a user-controlled key named ***chkey***. Applications can use **cpia_chkey** to get the key associated with the open. For asynchronous receive and transmit complete notifications, it is not necessary to call **cpia_chkey** since the user's key is provided as a parameter to **cpia_rxnotify** and **cpia_txnotify**.

cpia_get_data

Obtains a packet of data from the specified channel.

Prototype

#include *txcpi.h*

S16 **cpia_get_data** (TX_HANDLE ***handle***, CPIPKT ***buffer***, S16***len**)

Argument	Description
handle	TX_HANDLE associated with the asynchronous transmit completion.
buffer	Pointer to the CPIPKT buffer to store the received packet.
len	Pointer to the size of the buffer on input and the length of the received packet on output.

Return values

Return value	Description
CPI_SUCCESS	Packet successfully received.
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.
CPI_INVALID_MODE	Handle is not open for asynchronous I/O.
CPI_TRUNCATED	Received packet was larger than the passed buffer.

Details

cpia_get_data obtains a packet of data from the channel. On entry, the passed length parameter is checked. If the length is less than the received message, then ***len*** bytes of the message are copied to buffer and CPI_TRUNCATED is returned. The length of the received packet is placed in ***len***.

Call **cpia_get_data** from within **cpia_rxnotify**. Calling **cpia_get_data** from outside **cpia_rxnotify** can result in communication errors.

cpia_intr

Drains the asynchronous transmit acknowledgements and checks for any waiting received packets.

Prototype

#include *txcpi.h*

CPI_ERR_TYPE **cpia_intr** (TX_HANDLE ***handle***)

Argument	Description
handle	TX_HANDLE that has had an I/O event.

Return values

Return value	Description
CPI_SUCCESS	Asynchronous processing completed successfully.
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.
CPI_INVALID_MODE	Handle is not open for asynchronous I/O.

Details

Call **cpia_intr** when an I/O event is detected. Detecting such events is operating system-specific (**WaitForMultipleObjects** for Windows or **poll** for UNIX).

Note: Asynchronous transmit complete messages are processed before received messages.

cpia_open

Opens a channel for asynchronous I/O on the host.

Prototype

#include *txcpi.h*

TX_HANDLE **cpia_open** (void ***chkey**, U16 ***board**, U16 ***channel**, void ((***rxnotify**)(TX_HANDLE ***handle**, void ***chkey**)), void ((***txnotify**)(TX_HANDLE ***handle**, void***chkey**, CPIPKT ****buffer**, void***user_tx_key**, CPI_ERR_TYPE, ***ccode**)))

Argument	Description
chkey	User-controlled key passed back on all callback functions.
board	Board number from which to receive packets.
channel	DPR channel from which to receive packets.
rxnotify	Pointer to a receive notification callback function.
txnotify	Pointer to a transmit notification callback function.

Return values

Return value	Description
CPI_INVALID_HANDLE	Unable to open the channel.

Details

Use **cpia_open** to open a channel for aynchronous I/O. Use **cpi_open** to open a channel for synchronous I/O. Mixed mode I/O on a given channel is not possible, either with a single TX_HANDLE or multiple TX_HANDLEs. If successful, TX_HANDLE is returned.

See also

cpia_intr, **cpia_rxnotify**, **cpia_txnotify**

cpia_rxnotify

Notifies upper layers of messages to be received.

Prototype

void **cpia_rxnotify** (TX_HANDLE ***handle***, void ****chkey***)

Argument	Description
handle	TX_HANDLE on which the message was received.
chkey	Channel key provided when the handle was opened.

Details

Provide **cpia_rxnotify** as a parameter to **cpia_open**. The CPI library calls this function as a result of a call to **cpia_intr** when receive packets are pending for the given channel.

cpia_rxnotify calls **cpia_get_data** to receive the incoming message.

cpia_send

Asynchronously sends a packet of data over the specified channel.

Prototype

#include *txcpi.h*

S16 **cpia_send** (TX_HANDLE ***handle***, CPIPKT ****buffer***, void ****user_tx_key***)

Argument	Description
handle	TX_HANDLE associated with the channel.
buffer	Pointer to a CPIPKT structure containing data to send.
user_tx_key	Pointer to a user-defined key returned when I/O completes.

Return values

Return value	Description
CPI_SUCCESS	Packet send successfully started. Completes when **cpia_txnotify** is called.
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.
CPI_INVALID_MODE	Handle is not open for asynchronous I/O.
CPI_QUEUE_FULL	Maximum number of pending asynchronous I/O requests already in progress.

Details

The value returned by **cpia_send** reflects the result of enqueing the packet for transmission. The ultimate disposition of the packet is passed back as a parameter to **cpia_txnotify**.

Once sent, a packet cannot be unsent (that is, there is no **cpia_cancel**).

Final I/O result notification is handled by **cpia_intr** and **cpia_txnotify** callback.

The CPIPKT structure pointed to by the buffer parameter cannot be freed, re-used, or re-allocated until the final disposition of the packet is determined with **cpia_intr** and **cpia_txnotify**. Failure to adhere to this requirement causes unreliable and unpredictable results.

See also

cpia_open

cpia_txnotify

Processes an asynchronous transmit completion message received from the TX board.

Prototype

void **cpia_txnotify** (TX_HANDLE ***handle***, void ****chkey***, CPIPKT ****buffer***, void ****user_tx_key***, CPI_ERR_TYPE ***ccode***)

Argument	Description
handle	TX_HANDLE associated with the asynchronous transmit completion.
chkey	Pointer to the channel key provided when the handle was opened.
buffer	CPIPKT buffer pointer provided when **cpia_send** was called.
user_tx_key	Pointer to the user key provided when **cpia_send** was called.
ccode	I/O completion code.

Details

Provide **cpia_txnotify** as a parameter to **cpia_open**. The CPI library calls this function as a result of a call to **cpia_intr** when previously issued transmit requests (with **cpia_send**) complete for the given channel. When **cpia_txnotify** is called, or any time thereafter, the application can free the corresponding CPIPKT passed in on **cpia_send**. Failure to adhere to this rule results in communications errors.

cpi_check_bs

Determines whether the TX board specified by ***handle*** is in the boot state.

Prototype

#include *txcpi.h*

S16 **cpi_check_bs** (TX_HANDLE ***handle***, CPIBS ****bsp***)

Argument	Description
handle	TX handle number of the board to check.

bsp	Pointer to the location where the boot state is to be returned:
	`typedef struct _CPIBS`
	`{`
	`  U16  state;`
	`  U8   reg[5];`
	`} CPIBS;`
	Refer to the Details section for valid boot states.

Return values

Return value	Description
CPI_SUCCESS	Boot state determined.
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.

Details

The bsp.state is loaded with the boot state (low byte) and the CSR (high byte). The boot state can be one of the following values:

State	Description
BS_BOOT	Waiting to begin PREBOOT.
BS_READY	KERNEL loaded, initialized, and ready
BS_INIT	KERNEL is initializing.
BS_DOWN	KERNEL not responding.
BS_BERR	Bus error indicated by KERNEL.
BS_LOADING	Loading block of KERNEL.
BS_PREBOOTING	PREBOOT running, not ready for KERNEL.
BS_WAIT_KERNEL	PREBOOT complete, waiting for KERNEL.

The reg element in the structure is unused.

See also

cpi_force_bs

cpi_close

Closes the channel associated with the specified ***handle***.

Prototype

#include *txcpi.h*

S16 **cpi_close** (TX_HANDLE ***handle***)

Argument	Description
handle	TX handle associated with the channel, returned from **cpi_open** or **cpia_open**.

Return values

Return value	Description
CPI_SUCCESS	Channel successfully closed.
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.

Details

Applications that open CPI channels must close all channels before the application terminates. Failing to close a channel can leave resources in an indeterminate state.

cpi_cptoh_l

Converts the ***src*** value from the TX board native format to the host format.

Prototype

#include *txcpi.h*

U32 **cpi_cptoh_l** (U32 ***src***)

Argument	Description
src	Value in TX native format to be converted.

Details

The value of ***src*** is converted to the host format and placed in the return value.

Note: This function performs no operation on a host system that uses the same native format as the TX board (TX boards use the Motorola native format). However, for code portability, NMS recommends that you always use the conversion functions, even with host systems that are already in Motorola format.

cpi_cptoh_s

Converts the ***src*** value from the TX board native format to the host format.

Prototype

#include *txcpi.h*

U16 **cpi_cptoh_s** (U16 ***src***)

Argument	Description
src	Value in TX native format to be converted.

Details

The value of ***src*** is converted to the host format and placed in the return value.

Note: This function performs no operation on a host system that uses the same native format as the TX board (TX boards use the Motorola native format). However, for code portability, NMS recommends that you always use the conversion functions, even with host systems that are already in Motorola format.

cpi_force_bs

Boots the TX device indicated by the specified ***handle***. The board performs a complete reset, including loading the operating system kernel from on-board flash memory.

Prototype

#include *txcpi.h*

S16 **cpi_force_bs** (TX_HANDLE ***handle***)

Argument	Description
handle	TX handle number.

Return values

Return value	Description
CPI_SUCCESS	Reset of the TX board successfully started.
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.

Details

cpi_force_bs triggers the board to reboot from the TX operating system that is stored in on-board flash memory. All current processing on the board is aborted. When the board reset completes, **cpi_check_bs** returns a state of BS_READY.

cpi_get_board

Returns the board number and channel number associated with the specified ***handle***.

Prototype

#include *txcpi.h*

S16 **cpi_get_board** (TX_HANDLE ***handle***, U8 ****board***, U8 ****chan***)

Argument	Description
handle	TX handle number.
board	Pointer to a location to return the TX board number.
chan	Pointer to a location to return the channel number.

Return values

Return value	Description
CPI_SUCCESS	Board and channel numbers returned (as ***board*** and ***chan***).
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.

cpi_get_data

Recovers received packets from the channel associated with the specified ***handle***.

Prototype

#include *txcpi.h*

S16 **cpi_get_data** (TX_HANDLE ***handle***, CPIPKT ****buffer***, S16 ****len***)

Argument	Description
handle	TX handle associated with the channel.
buffer	Pointer to a location to store the received packet.
len	Pointer to the length of the buffer on input and the length of the received packet on output.

Return values

Return value	Description
CPI_SUCCESS	Packet successfully received.
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.
CPI_TRUNCATED	Received length is longer than the specified buffer length.

Details

Specify the length of the buffer in the ***len*** parameter in the call to **cpi_get_data**. If there is no packet to receive, **cpi_get_data** returns CPI_SUCCESS and ***len*** is set to zero. If there is a packet, **cpi_get_data** returns CPI_SUCCESS, the length is placed in ***len***, and the packet is copied into the specified ***buffer***.

cpi_get_dev_info

Retrieves device information for available TX boards.

Prototype

#include *txcpi.h*

CPI_ERR_TYPE **cpi_get_dev_info** (CPI_DEV_INFO ****devinfo***, U16 ****numdevs***)

Argument	Description
devinfo	Pointer to an array of device information structures.
numdevs	Pointer to the number of entries in the ***devinfo*** array on input and the number of entries populated on output (number of detected TX devices).

Return values

Return value	Description
CPI_SUCCESS	Information about the set of detected TX devices provided in ***devinfo***.
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.

Details

Use **cpi_get_dev_info** to determine the PCI bus and slot for each installed TX board when assigning CP numbers to the detected boards.

See also

cpi_set_cpid

cpi_get_error_str

Returns an ASCII string associated with the ***errnum*** passed to the function.

Prototype

#include *txcpi.h*

S8 ***cpi_get_error_str** (CPI_ERR_TYPE ***errnum***)

Argument	Description
errnum	CPI library error number.

Return values

Return value	Description
Unknown error ***nnn***	No match for the ***errnum*** parameter.
NULL	NULL terminated string containing a description of the ***errnum*** passed.

Details

When a CPI library function returns CPI_ERROR, use **cpi_get_last_error** to determine the error code. Then use **cpi_get_error_str** to convert this ***errnum*** into an ASCII string describing the error.

cpi_get_last_error

Returns the error code for the most recent error that occurred in the library.

Prototype

#include *txcpi.h*

CPI_ERR_TYPE **cpi_get_last_error()**

Details

When a CPI library function returns CPI_ERROR, use **cpi_get_last_error** to determine the error code. Then use **cpi_get_error_str** to convert this ***errnum*** into an ASCII string describing the error.

cpi_get_resources

Identifies the available TX boards.

Prototype

#include *txcpi.h*

S16 **cpi_get_resources** (S16 ***max_cps***, S32 ****cps[]***)

Argument	Description
max_cps	Maximum CP number for which to return resource information.
cps	Pointer to an array of entries where CP types are returned.

Return values

Return value	Description
CPI_SUCCESS	Board types identified in ***cps*** array.
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.

Details

If a CP of the corresponding index does not exist, each element of the array ***cps*** is filled with 0. If a CP for that index does exist, the array is filled with the TX board types. The parameter ***max_cps*** indicates the number of CPs to check. The ***cps*** array should have ***max_cps*** + 1 elements since the array is filled according to board number. There is no board number 0 and this element is not used by this routine.

cpi_htocp_l

Converts the ***src*** value from the host format to the TX board native format.

Prototype

#include *txcpi.h*

U32 **cpi_htocp_l** (U32 ***src***)

Argument	Description
src	Value in host native format to be converted.

Details

The value of ***src*** is converted to the TX board format and placed in the return value.

Note: This function performs no operation on a host system that uses the same native format as the TX board (TX boards use the Motorola native format). However, for code portability, NMS recommends that you always use the conversion functions, even with host systems that are already in Motorola format.

cpi_htocp_s

Converts the ***src*** value from the host format to the TX board native format.

Prototype

#include *txcpi.h*

U16 **cpi_htocp_s** (U16 ***src***)

Argument	Description
src	Value in host native format to be converted.

Details

The value of ***src*** is converted to the TX board format and placed in the return value.

Note: This function performs no operation on a host system that uses the same native format as the TX board (TX boards use the Motorola native format). However, for code portability, NMS recommends that you always use the conversion functions, even with host systems that are already in Motorola format.

cpi_init

Initializes the CPI library.

Prototype

#include *txcpi.h*

S16 **cpi_init** (S16 ***dummy***, S8 ****idstring***)

Argument	Description
dummy	Unused and retained for compatibility.
idstring	Unused and retained for compatibility.

Return values

Return value	Description
CPI_SUCCESS	CPI library successfully initialized.
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.

Details

Call **cpi_init** once per application.

cpi_nmi

Controls the non-maskable interrupt (NMI) state on the TX board.

Prototype

#include *txcpi.h*

S16 **cpi_nmi** (TX_HANDLE ***handle***, U32 ***state***)

Argument	Description
handle	TX handle number.
state	Desired state of the NMI signal. Valid values: CPI_NMI_ASSERT Assert NMI signal. CPI_NMI_DEASSERT Deassert NMI signal.

Return values

Return value	Description
CPI_SUCCESS	NMI signal state set as requested.
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.

Details

Use the NMI to halt all standard processing on the TX board and to place the board into a state where diagnostic information can be read from the board. An application should assert NMI first, and then deassert NMI to cause the TX board to begin executing in this diagnostic state.

cpi_open

Opens a channel for synchronous I/O on the host.

Note: NMS recommends that you use **cpia_open** to open all channels to TX boards.

Prototype

#include *txcpi.h*

TX_HANDLE **cpi_open** (U16 ***port***, S16 ***mode***, S16 ****rcvr*** (S16 ***handle***, S16 ***len***))

Argument	Description
port	Combination of the TX board number and the channel number. Use the PORT macro to combine a board number and channel number into a port.
mode	Unused and retained for backwards compatibility.
rcvr	Unused and retained for backwards compatibility.

Return values

Return value	Description
CPI_INVALID_HANDLE	Unable to open the channel.

Details

TX_HANDLE is operating-system specific. Since this return value is passed back only to other CPI calls, the type is not important to the application. When the handle is required for a wait call (**WaitForSingleObject** in Windows, **poll** in UNIX), use **cpi_wait_obj (*handle*)** to access the proper element for each operating system, as follows:

```
WaitForSingleObject (cpi_wait_obj (handle), 0);
```

For multiple-threaded applications, the thread that opens a channel should be the same thread that processes all I/O for that channel. Otherwise, unpredictable behavior can result.

See also

cpi_get_data, **cpi_wait_msg**

cpi_read_control

Reads a set of control registers from a TX board.

Prototype

#include *txcpi.h*

S16 **cpi_read_control** (TX_HANDLE ***handle***, U16 ***basereg***, U16 ***numreg***, U32 ****regarray***, U16 ****actcnt***)

Argument	Description
handle	TX handle number.
basereg	Number of the base register to read (0 through max-1).
numreg	Count of registers to read.
regarray	Pointer to an array to hold register values.
actcnt	Pointer to the location where the actual number of registers read are stored.

Return values

Return value	Description
CPI_SUCCESS	Requested set of registers successfully read from TX board.
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.

Details

In addition to the dual-ported RAM shared between the host processor and the TX board, a set of registers is used for communication control. Certain low-level diagnostics on the TX board use the control registers to pass status information to the host.

All control register access should be restricted to diagnostic applications. Do not use this function for normal data transfer situations.

See also

cpi_write_control

cpi_read_dpr

Reads from the dual-ported RAM of the TX board specified by ***handle***.

Prototype

#include *txcpi.h*

S16 **cpi_read_dpr** (TX_HANDLE ***handle***, S8 ****buffer***, U32 ***off***, S16 ***len***)

Argument	Description
handle	TX handle number.
buffer	Pointer to a location to which the data is read.
off	Offset into the dual-ported RAM from which data is to be read.
len	Number of bytes to be read.

Return values

Return value	Description
CPI_SUCCESS	DPR successfully read into the buffer.
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.

Details

The read starts at ***off*** in the DPR and reads ***len*** number of bytes.

All dual-ported RAM is used for messaging and therefore should not be read directly. Do not use this function for normal data transfer situations.

cpi_send

Synchronously sends a packet of data over the channel indicated by the specified ***handle***.

Prototype

#include *txcpi.h*

S16 **cpi_send** (TX_HANDLE ***handle***, CPIPKT ****buffer***)

Argument	Description
handle	TX handle associated with the channel.
buffer	Pointer to a CPIPKT structure containing data to send.

Return values

Return value	Description
CPI_SUCCESS	Packet successfully sent to the TX board.
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.

Details

buffer should point to a properly formatted CP packet. The application sets the destination board and channel number in the header portion of the buffer. The function does not return until the board acknowledges the sent packet.

cpi_set_cpid

Assigns a CP number to the TX board at the given PCI bus and slot number.

Prototype

#include *txcpi.h*

S16 **cpi_set_cpid** (S16 ***type***, U32 ***param1***, U32 ***param2***, U32 ***cpId***)

Argument	Description
type	Type of board. The only supported ***type*** is CPI_PCI_BUS = PCI board.
param1	Bus number.
param2	Slot number.
cpId	TX board number to associate with the board at the given bus and slot.

Return values

Return value	Description
CPI_SUCCESS	CP number successfully assigned to the given PCI bus and slot number.
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.

Details

cpi_set_cpid assigns a board number (CP number) to the TX board at the indicated PCI bus and slot number. After a TX board is assigned a CP number, the board can be accessed by other CPI functions.

cpi_show_stats

Displays common statistics to stdout using a series of printf calls.

Prototype

#include *txcpi.h*

S16 **cpi_show_stats** (TX_STATS_COMMON ***stats**)

Argument	Description
stats	Pointer to a location where common statistics information is written.

Return values

Return value	Description
CPI_SUCCESS	Statistics successfully displayed to stdout.
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.

Details

cpi_show_stats enables an application to display all common statistics in a standardized format. All statistics definitions can be found in the *txstats.h* include file.

See also

cpi_stats

cpi_stats

Obtains per-channel statistics synchronously.

Prototype

#include *txcpi.h*

CPI_ERR_TYPE **cpi_stats** (TX_HANDLE ***handle***, U32 ***options***, TX_STATS_COMMON ****stats***)

Argument	Description
handle	TX_HANDLE associated with the channel.
options	Statistics collection operation. Refer to the Details section for valid values.
stats	Pointer to a location where statistics information is written.

Return values

Return value	Description
CPI_SUCCESS	Statistics request successfully completed.
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.

Details

cpi_stats enables an application to collect the per-channel statistics maintained by the CPI library. All statistics definitions can be found in the *txstats.h* include file.

The CPI layer maintains a set of common statistics and optionally a set of layer-specific statistics. The common statistics are defined by the TX_STATS_COMMON structure. Use the TX_STATS_NAME operation to get ASCII names of the common statistics.

Use the ***options*** parameter to describe the type of statistics to return. The following table lists the valid values for the ***options*** parameter:

#include *txstats.h*

Use this value...	To return...
TX_STATS_GET	Current statistics.
TX_STATS_ZERO	Current statistics, then zero the statistics.
TX_STATS_NAME	Names of common statistics.
TX_STATS_NAME_LAYER	Names of layer-specific statistics.
TX_STATS_DESC	A description of common statistics.
TX_STATS_DESC_LAYER	A description of layer-specific statistics.

See also

cpi_show_stats

cpi_wait_msg

Waits a specified amount of time (in milliseconds) and returns a packet if one is received.

Prototype

#include *txcpi.h*

S16 **cpi_wait_msg** (TX_HANDLE ***handle***, CPIPKT ****buffer***, S16****len***, S32 ***millisecs***)

Argument	Description
handle	TX handle associated with the channel.
buffer	Pointer to the address to which to copy the received buffer.
len	Pointer to the length of the buffer on input and the length of the received packet on output.
millisecs	Amount of time to wait before returning the packet.

Return values

Return value	Description
CPI_SUCCESS	Packet successfully received in ***buffer***.
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.
CPI_TIMEOUT	No packet to receive.
CPI_TRUNCATED	Received length is longer than the specified buffer length.

Details

cpi_wait_msg recovers received packets from the channel associated with the specified ***handle***. Upon entry, ***len*** contains the size of the buffer. If there is a packet to receive, the length is placed in ***len*** and the packet is placed in the specified buffer.

cpi_wait_obj

Returns the wait object for the channel associated with the specified ***handle***.

Prototype

#include *txcpi.h*

CPI_WAIT_TYPE **cpi_wait_obj** (TX_HANDLE ***handle***)

Argument	Description
handle	TX handle associated with the channel.

Return values

Return value	Description
CPI_INVALID_WAIT_HANDLE	Invalid TX handle.

Details

Use the wait object when calling the host operating system's native wait routine, such as **WaitForSingleObject** in Windows or **poll** for UNIX.

cpi_write_control

Writes a set of control registers to a TX board.

Prototype

#include *txcpi.h*

S16 **cpi_write_control** (TX_HANDLE ***handle***, U16 ***basereg***, U16 ***numreg***, U32 ****regarray***, U16 ****actcnt***)

Argument	Description
handle	TX handle number.
basereg	Number of the base register to write (0 through max-1).
numreg	Count of registers to write.
regarray	Pointer to an array holding the register values to be written.
actcnt	Pointer to the location to store the number of registers written.

Return values

Return value	Description
CPI_SUCCESS	Provided set of registers successfully written to the TX board.
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.

Details

In addition to the dual-ported RAM shared between the host processor and the TX board, a set of registers is used for communication control. Certain low-level diagnostics on the TX board use the control registers to control low-level, on-board diagnostics.

All control register access should be restricted to diagnostic applications. Do not use this function for normal data transfer situations.

See also

cpi_read_control

cpi_write_dpr

Writes to the dual-ported RAM of the CP indicated by the specified ***handle***.

Prototype

#include *txcpi.h*

S16 **cpi_write_dpr** (TX_HANDLE ***handle***, S8 ****buffer***, U32 ***off***, S16 ***len***)

Argument	Description
handle	TX handle number.
buffer	Pointer to a location to which the data is written.
off	Offset into the dual-ported RAM where data is written.
len	Number of bytes to be written.

Return values

Return value	Description
CPI_SUCCESS	Provided ***buffer*** successfully written to DPR.
CPI_ERROR	Call **cpi_get_last_error** to obtain the error code.

Details

cpi_write_dpr writes from ***buffer*** for ***len*** number of bytes starting at ***off*** in the DPR.

All dual-ported RAM is used for messaging and should not be written directly. Do not use this function for normal data transfer situations.

Index