Board Management API for Windows and Linux Operating Systems

Library Reference

September 2005

05-1958-002

Publication Date: September 2005

Document Number: 05-1958-002

Intel Converged Communications, Inc.
1515 Route 10
Parsippany, NJ 07054

For **Technical Support**, visit the Intel Telecom Support Resources website at:
http://developer.intel.com/design/telecom/support

For **Products and Services Information**, visit the Intel Telecom and Compute Products website at:
http://www.intel.com/design/network/products/telecom

For **Sales Offices** and other contact information, visit the Buy Telecom Products page at:
http://www.intel.com/buy/networking/telecom.htm

Contents

Revision History

This revision history summarizes the changes made in each published version of this document.

Document No.	Publication Date	Description of Revisions
05-1958-002	September 2005	Updated for SR 6.1 GA on Linux* operating systems. brd_SendAlive( ) and brd_SendAliveEnable( ) functions:: Changed description from saying that “a protocol-specific out-of-service message” is sent to “an Alarm Indication Signal (AIS)” is sent, and in addition “a Q.931 maintenance messge SERVICE (out-of-service)” is sent for T1 ISDN protocols. Added details on what happens when the host failure occurs and the network interface is taken out of service, as well as alarm clearing after recovering from the failure.] Function Summary by Category: Added details to the Fault Monitoring functions description.
05-1958-001	November 2003	Initial version of document for SR 6.0 GA on Windows operating systems.

About This Publication

The following topics provide information about this publication.

Purpose

This publication provides reference information for all functions, parameters, data structures, values, events, and error codes in the Board Management API.

Intended Audience

This information is intended for:

- Distributors
- System Integrators
- Toolkit Developers
- Independent Software Vendors (ISVs)
- Value Added Resellers (VARs)
- Original Equipment Manufacturers (OEMs)
- End Users

How to Use This Publication

This publication assumes that you are familiar with and have prior experience with Windows* operating systems and the C programming language.

The information in this publication is organized as follows:

- Chapter 1, "Function Summary by Category" introduces the categories of functions and provides a brief description of each function.
- Chapter 2, "Function Information" provides an alphabetical reference to all the functions in the library.
- Chapter 3, "Error Codes" presents a listing of error codes that may be returned by the library functions.

Related Information

For related Intel® Dialogic® publications, see the product documentation (known as the on-line bookshelf) provided with the software release or at the following web site: *http://resource.intel.com/telecom/support/documentation/releases/index.htm*.

Function Summary by Category 1

This chapter contains an overview of the Board Management API functions and the categories into which they are grouped. Major topics include the following:

1.1 Board Management API Header File

The Board Management API contains functions that provide run-time fault monitoring and management of configurable boards. The Board Management API functions, parameters, data structures, values, events, and error codes are defined in the *devmgmt.h* header file. The Board Management API functions have a “brd_” prefix.

Note: The header file also contains other functions, such as those belonging to the Device Management API, which have a “dev_” prefix. The Device Management API functions and their associated data belong to a separate API category and are not addressed by this document. Their presence in the header file does not indicate that they are supported.

1.2 Fault Monitoring Functions

Board Management API Fault Monitoring functions manage the status of system resources. They provide the ability to monitor the host computer for a failure and take the network interface of the boards out of service. For example, in the event of an application program or host crash, channels can be set to out-of-service to prevent the Central Office or network switch from sending calls to a board if there is no program to process them. This prevents the acceptance of unwanted calls and thereby reduces the potential for being unnecessarily tariffed.. The Fault Monitoring functions include the following functions:

brd_SendAlive( )

The **brd_SendAlive( )** function sends a message (a “heartbeat” or “ping”) to the board specified by the device descriptor to indicate that the host is “alive” or it is “up” and running.

brd_SendAliveDisable( )

The **brd_SendAliveDisable( )** function disables host fault monitoring on the board specified by the device descriptor.

brd_SendAliveEnable( )
The **brd_SendAliveEnable( )** function enables host fault monitoring on the board specified by the device descriptor. When enabled, the board monitors the host computer for the presence of a repeated “heartbeat,” or “ping.”

1.3 Device Management Functions

Board Management API Device Management functions open and close board devices. They include the following functions

brd_Close( )
The **brd_Close( )** function closes a physical board.

brd_Open( )
The **brd_Open( )** function opens a physical board and returns a device descriptor that can be used in other Board Management API functions.

1.4 Configuration Functions

Board Management API Configuration functions provide system configuration information. They include the following functions:

brd_GetAllPhysicalBoards( )
The **brd_GetAllPhysicalBoards( )** function gets a list of the physical boards in the system.

brd_VirtualToPhysicalName( )
The **brd_VirtualToPhysicalName( )** function gets the physical board name for the specified virtual board name.

1.5 Error Processing Functions

Board Management API Error Processing functions provide error processing information. They include the following functions:

brd_ErrorMsg( )
The **brd_ErrorMsg( )** function provides an error message string based on the last error that occurred in the Board Management API.

brd_ErrorValue( )
The **brd_ErrorValue( )** function provides the last error that occurred in the Board Management API.

Function Information 2

This chapter is arranged in alphabetical order by function name and contains detailed information on each function in the Board Management API.

2.1 Function Syntax Conventions

The Board Management API functions use the following format:

```
brd_FunctionName (DeviceHandle, Parameter1, Parameter2, ..., ParameterN, mode)
```

where:

brd_FunctionName
: represents the name of the function. Functions in the Board Management API use the prefix “brd_” in the function name.

DeviceHandle
: is an input parameter that specifies a valid handle obtained for a device when the device was opened

Parameter1, Parameter2, ..., ParameterN
: represent input or output parameters

mode
: is an input parameter that specifies how the function should be executed, typically either asynchronously or synchronously. Some functions can be executed in only one mode and so do not provide this parameter.

brd_Close( )

Name: int brd_Close(brdhndl)

Inputs: int brdhndl
- board device handle for the board that is to be closed

Returns: BRD_SUCCESS if successful
BRD_FAILURE if unsuccessful

Includes: devmgmt.h
srllib.h

Category: Device Management

Mode: synchronous

Description

The **brd_Close**( ) function closes a physical board.

Parameter	Description
brdhndl	specifies the board device handle for the board that is to be closed, which was originally obtained by the **brd_Open( )** function when the board was opened.

Cautions

- If you close a board that was enabled with the **brd_SendAliveEnable( )** function, the board stays enabled. If you want to disable the host computer fault monitoring on the board, use the **brd_SendAliveDisable( )** function before closing the board or open the board again and then disable it with **brd_SendAliveDisable**( ).
- Board Management API errors are thread-specific.

Errors

If this function returns BRD_FAILURE to indicate failure, use the Board Management API Error Processing functions **brd_ErrorMsg( )** and **brd_ErrorValue( )** to retrieve the error information. Possible errors for this function include:

EBRD_INVALIDDEVICEHANDLE
An invalid device descriptor was provided.

Example

The following example code shows how the function is used.

```
// Header files
#include <stdio.h>
#include "srllib.h"    // Standard Runtime Library API
#include "devmgmt.h"   // Board Mgt. API

// Defines
```

```
#define MAXBOARD   10   // Maximum number of physical boards
//---------------------------------------------------------------------
// Main Function
void main(void)
{
  int  physical_brd_handle[MAXBOARD];   // Store physical board handles
  int  ret;                             // Return value
  char temp_brd_name[MAXLENGTH];        // Store temp physical board name

  // Initialize
  memset(temp_brd_name, '\0', MAXLENGTH);

  // ------------------------------------
  // brd_Open()
  // Open the first physical board to get handle
  // "brdB1" is a digital network interface board
  sprintf(temp_brd_name, "brdB1");
  physical_brd_handle[1] = brd_Open(temp_brd_name, 0);
  if(physical_brd_handle[1] == BRD_FAILURE) // Function returns failure
  {
    printf("brd_Open(%s) failed. Error = %s <%d>\n", temp_brd_name, brd_ErrorMsg(),
      brd_ErrorValue());

    if(brd_ErrorValue() == EBRD_INVALIDPHYSICALNAME)
    {
      printf("Invalid physical board name was provided \n");
    }
    return;
  }
  else // Function returns succ
  {
    printf("brd_Open(%s) = %d\n", temp_brd_name, physical_brd_handle[1]);
  }

  // ------------------------------------
  // brd_Close()
  // Close the first physical board
  ret = brd_Close(physical_brd_handle[1]);
  if(ret != BRD_SUCCESS) // Function returns failure
  {
    printf("brd_Close() failed on board %s. Error = %s <%d>\n", temp_brd_name, brd_ErrorMsg(),
      brd_ErrorValue());
    return;
  }
  else // Function returns succ
  {
    printf("brd_Close(%d) = %d \n", physical_brd_handle[1], ret);
  }
} // End of function
```

See Also

- brd_Open( )

brd_ErrorMsg( )

Name: char* brd_ErrorMsg(void)

Inputs: none

Returns: an ASCIIZ error message string

Includes: devmgmt.h
srllib.h

Category: Error Processing

Mode: synchronous

Description

The **brd_ErrorMsg( )** function provides an ASCIIZ error message string based on the last error that occurred in the Board Management API.

The errors returned by this function are listed in Chapter 3, “Error Codes”.

Cautions

The Board Management API errors are thread-specific.

Errors

None.

Example

The following example code shows how the function is used.

```
// Header files
#include <stdio.h>
#include "srllib.h"    // Standard Runtime Library API
#include "devmgmt.h"   // Board Mgt. API

// Defines
#define MAXBOARD   10   // Maximum number of physical boards
//---------------------------------------------------------------------
// Main Function
void main(void)
{
  int  physical_brd_handle[MAXBOARD];   // Store physical board handles
  int  ret;                             // Return value
  char temp_brd_name[MAXLENGTH];        // Store temp physical board name

  // Initialize
  memset(temp_brd_name, '\0', MAXLENGTH);
```

```
  // ------------------------------------
  // brd_Open()
  // Open the first physical board to get handle
  // "brdB1" is a digital network interface board
  sprintf(temp_brd_name, "brdB1");
  physical_brd_handle[1] = brd_Open(temp_brd_name, 0);
  if(physical_brd_handle[1] == BRD_FAILURE) // Function returns failure
  {
    printf("brd_Open(%s) failed. Error = %s <%d>\n", temp_brd_name, brd_ErrorMsg(),
      brd_ErrorValue());

    if(brd_ErrorValue() == EBRD_INVALIDPHYSICALNAME)
    {
      printf("Invalid physical board name was provided \n");
    }
    return;
  }
  else // Function returns succ
  {
    printf("brd_Open(%s) = %d\n", temp_brd_name, physical_brd_handle[1]);
  }

  // ------------------------------------
  // brd_Close()
  // Close the first physical board
  ret = brd_Close(physical_brd_handle[1]);
  if(ret != BRD_SUCCESS) // Function returns failure
  {
    printf("brd_Close() failed on board %s. Error = %s <%d>\n", temp_brd_name, brd_ErrorMsg(),
      brd_ErrorValue());
    return;
  }
  else // Function returns succ
  {
    printf("brd_Close(%d) = %d \n", physical_brd_handle[1], ret);
  }
} // End of function
```

See Also

- brd_ErrorValue( )

brd_ErrorValue( )

Name: int brd_ErrorValue(void)

Inputs: none

Returns: an error value

Includes: devmgmt.h
srllib.h

Category: Error Processing

Mode: synchronous

Description

The **brd_ErrorValue( )** function provides the last error that occurred in the Board Management API.

The error codes returned by this function are listed in Chapter 3, “Error Codes”.

Cautions

The Board Management API errors are thread-specific.

Errors

None.

Example

The following example code shows how the function is used.

```
// Header files
#include <stdio.h>
#include "srllib.h"    // Standard Runtime Library API
#include "devmgmt.h"   // Board Mgt. API

// Defines
#define MAXBOARD   10   // Maximum number of physical boards
//---------------------------------------------------------------------
// Main Function
void main(void)
{
  int  physical_brd_handle[MAXBOARD];   // Store physical board handles
  int  ret;                             // Return value
  char temp_brd_name[MAXLENGTH];        // Store temp physical board name

  // Initialize
  memset(temp_brd_name, '\0', MAXLENGTH);
```

```
// -----------------------------------
// brd_Open()
// Open the first physical board to get handle
// "brdB1" is a digital network interface board
sprintf(temp_brd_name, "brdB1");
physical_brd_handle[1] = brd_Open(temp_brd_name, 0);
if(physical_brd_handle[1] == BRD_FAILURE) // Function returns failure
{
  printf("brd_Open(%s) failed. Error = %s <%d>\n", temp_brd_name, brd_ErrorMsg(),
    brd_ErrorValue());

  if(brd_ErrorValue() == EBRD_INVALIDPHYSICALNAME)
  {
    printf("Invalid physical board name was provided \n");
  }
  return;
}
else // Function returns succ
{
  printf("brd_Open(%s) = %d\n", temp_brd_name, physical_brd_handle[1]);
}

// -----------------------------------
// brd_Close()
// Close the first physical board
ret = brd_Close(physical_brd_handle[1]);
if(ret != BRD_SUCCESS) // Function returns failure
{
  printf("brd_Close() failed on board %s. Error = %s <%d>\n", temp_brd_name, brd_ErrorMsg(),
    brd_ErrorValue());
  return;
}
else // Function returns succ
{
  printf("brd_Close(%d) = %d \n", physical_brd_handle[1], ret);
}
} // End of function
```

See Also

- brd_ErrorMsg( )

brd_GetAllPhysicalBoards( )

Name: int brd_GetAllPhysicalBoards(physicalDevs, count)

Inputs:

SRLDEVICEINFO *physicalDevs	• pointer to array of SRLDEVICEINFO structure for obtaining a list of physical board devices
int *count	• pointer to length of **physicalDevs** array

Returns: BRD_SUCCESS if successful
BRD_FAILURE if failure

Includes: devmgmt.h
srllib.h

Category: Configuration

Mode: synchronous

Description

The **brd_GetAllPhysicalBoards( )** function gets a list of the physical boards in the system. Upon successful completion of the function, the array contains a list of boards that includes the device name for each board. All detected boards in the system are listed regardless of whether they are initialized (started and successfully downloaded) or are disabled. The **brd_Open( )** function uses the physical board name in its operation.

Parameter	Description
physicalDevs	pointer to array of SRLDEVICEINFO structure for obtaining list of physical board devices. The device type returned in the SRLDEVICEINFO **iDevType** field is TYPE_R4_PHYSICAL_BOARD. See the *Standard Runtime Library API Library Reference* for details on SRLDEVICEINFO.
count	pointer to length of **physicalDevs** array (i.e., equal to the maximum number of physical boards). If you point to a value of zero, the function generates an EBRD_BUFFERTOOSMALL error and updates **count** to the maximum number of boards.

Cautions

Board Management API errors are thread-specific.

Errors

If this function returns BRD_FAILURE to indicate failure, use the Board Management API Error Processing functions **brd_ErrorMsg( )** and **brd_ErrorValue( )** to retrieve the error information. Possible errors for this function include:

EBRD_BUFFERTOOSMALL
The size of the buffer provided is too small.

EBRD_NULLPOINTERARGUMENT
One of the arguments provided is a null-pointer.

EBRD_DEVICEMAPPERFAILED
Internal error indicating the SRL device mapping failed.

Example

The following example code shows how the function is used.

```
// Header files
#include <stdio.h>
#include "srllib.h"    // Standard Runtime Library API
#include "devmgmt.h"   // Board Mgt. API

// Defines
#define MAXBOARD   10   // Maximum number of physical boards
#define MAXLENGTH  12   // Maximum length of physical board name

//---------------------------------------------------------------------
// Main Function
void main(void)
{
  int  count;                          // Physical board counter
  int  index;                          // Index of boards
  int  length;                         // The length of board name
  int  physical_brd_handle[MAXBOARD];  // Store physical board handles
  int  ret;                            // Return value
  SRLDEVICEINFO physicalDevs[MAXBOARD]; // Physical device data structure

  // Initialize
  length    = MAXLENGTH; // The length of board name

  // ------------------------------------
  // brd_GetAllPhysicalBoards()
  // Get all physical boards in the system
  count = MAXBOARD;
  ret = brd_GetAllPhysicalBoards(physicalDevs, &count);
  if(ret != BRD_SUCCESS) // Function returns failure
  {
    printf("brd_GetAllPhysicalBoards() failed; error = %s <%d>\n", brd_ErrorMsg(),
      brd_ErrorValue());
    return;
  }
  else // Function returns succ
  {
    // Print out all the board names in the system
    for(index = 0; index < count; index++)
    {
      printf("Board %d name is %s\n", index+1, physicalDevs[index].szDevName);
    }
  }
} // End of function
```

See Also

- **brd_VirtualToPhysicalName( )**
- **brd_Open( )**

brd_Open( )

Name: int brd_Open (physical, mode)

Inputs: char *physical — a physical board name

long mode — ignored (reserved for future use); 0 is recommended setting

Returns: a device descriptor if successful
BRD_FAILURE if failure

Includes: devmgmt.h
srllib.h

Category: Device Management

Mode: synchronous

Description

The **brd_Open( )** function opens a physical board and returns a device descriptor that can be used in the **brd_SendAliveEnable( )**, **brd_SendAlive( )**, **brd_SendAliveDisable( )**, and **brd_Close( )** functions.

Parameter	Description
physical	specifies a physical board name to open. The board name must follow the format “brdBn”, where n represents an integer equal to or greater than 1.
mode	ignored (reserved for future use); 0 is recommended setting

Cautions

- The board must be initialized (downloaded); otherwise, the function generates a EBRD_COMMANDNOTSUPPORTED error.
- Board Management API errors are thread-specific.

Errors

If this function returns BRD_FAILURE to indicate failure, use the Board Management API Error Processing functions **brd_ErrorMsg( )** and **brd_ErrorValue( )** to retrieve the error information. Possible errors for this function include:

EBRD_INVALIDPHYSICALNAME
Invalid physical board name was provided.

EBRD_OUTOFMEMORY
Internal error indicating unable to allocate memory.

EBRD_COMMANDNOTSUPPORTED
The operation is not supported on the board or the protocol, or the board is not initialized.

EBRD_FAILEDOPENINGDTILIB
Internal error indicating the network interface library cannot be opened.

EBRD_DEVICEMAPPERFAILED
Internal error indicating the SRL device mapping failed.

Example

The following example code shows how the function is used.

```
// Header files
#include <stdio.h>
#include "srllib.h"    // Standard Runtime Library API
#include "devmgmt.h"   // Board Mgt. API

// Defines
#define MAXBOARD   10   // Maximum number of physical boards
//---------------------------------------------------------------------
// Main Function
void main(void)
{
  int  physical_brd_handle[MAXBOARD];   // Store physical board handles
  int  ret;                             // Return value
  char temp_brd_name[MAXLENGTH];        // Store temp physical board name

  // Initialize
  memset(temp_brd_name, '\0', MAXLENGTH);

  // ------------------------------------
  // brd_Open()
  // Open the first physical board to get handle
  // "brdB1" is a digital network interface board
  sprintf(temp_brd_name, "brdB1");
  physical_brd_handle[1] = brd_Open(temp_brd_name, 0);
  if(physical_brd_handle[1] == BRD_FAILURE) // Function returns failure
  {
    printf("brd_Open(%s) failed. Error = %s <%d>\n", temp_brd_name, brd_ErrorMsg(),
      brd_ErrorValue());

    if(brd_ErrorValue() == EBRD_INVALIDPHYSICALNAME)
    {
      printf("Invalid physical board name was provided \n");
    }
    return;
  }
  else // Function returns succ
  {
    printf("brd_Open(%s) = %d\n", temp_brd_name, physical_brd_handle[1]);
  }

  // ------------------------------------
  // brd_Close()
  // Close the first physical board
  ret = brd_Close(physical_brd_handle[1]);
  if(ret != BRD_SUCCESS) // Function returns failure
  {
    printf("brd_Close() failed on board %s. Error = %s <%d>\n", temp_brd_name, brd_ErrorMsg(),
      brd_ErrorValue());
    return;
  }
  else // Function returns succ
  {
    printf("brd_Close(%d) = %d \n", physical_brd_handle[1], ret);
  }
} // End of function
```

See Also

- **brd_Close( )**

brd_SendAlive( )

Name:	int brd_SendAlive (brdhndl, mode)	
Inputs:	int brdhndl	• board device handle
	long mode	• ignored (reserved for future use); 0 is recommended setting
Returns:	BRD_SUCCESS if successful BRD_FAILURE if failure	
Includes:	devmgmt.h srllib.h	
Category:	Fault Monitoring	
Mode:	synchronous	

Description

The **brd_SendAlive( )** function sends a message (a "heartbeat" or "ping") to the board specified by the device descriptor to indicate that the host is "alive" or it is "up" and running. If the board does not receive a **brd_SendAlive( )** message (the "heartbeat" or "ping") within the time parameters defined by the **brd_SendAliveEnable( )** function, the board treats it as a host failure. If the heartbeat stops, for whatever reason, and the time parameters are exceeded, from the board's perspective, the host is "down." If this "host failure" occurs, the board will remove its network interface from service and respond to any calls with an out-of-service message, thus preventing the network from offering calls to the failed system. For example, in the event of an application program or host crash, the channels are set to out-of-service to prevent the Central Office or network switch from sending calls to a board when there is no program to process them. This prevents the acceptance of unwanted calls and thereby reduces the potential for being unnecessarily tariffed.

If the board receives a **brd_SendAlive( )** heartbeat from the host or a new **brd_SendAliveEnable( )**, the timer is reset to zero.

Parameter	Description
brdhndl	specifies a board device handle obtained using a **brd_Open( )** function
mode	ignored (reserved for future use); 0 is recommended setting

The following happens when the host failure occurs and the network interface is taken out of service: The Global Call line devices are reset on the board, which includes releasing or dropping all active calls existing on the board, freeing all associated memory, taking the board's network interface out of service, and resetting the network interface protocol. Although the host application program may not be able to receive it, the unsolicited Global Call event, GCEV_D_CHAN_STATUS, is also generated.

The network interface is taken out-of-service by sending an Alarm Indication Signal (AIS) toward the network. This is the ITU recommended mechanism for informing the CO or network that the trunk is not available. In addition, for the T1 ISDN protocols that support it, the Q.931 maintenance

message SERVICE (Out-Of-Service) is also used to inform the network that the channels are no longer available.

After recovering from the failure (e.g., the host or application crash), the AIS alarm, and the SERVICE message if applicable, is cleared automatically when the trunk is put in service using **gc_OpenEx( )** or **gc_Open( )** on the trunk device (dtiBn) or channel (dtiBnTm) in any given trunk. In the event that an AIS alarm was being transmitted on some other trunks prior to the crash, then the AIS alarm on those trunks will not be cleared when the other trunks are put back in service. In this case, the application needs to clear the alarm using the Global Call Alarm Management System (GCAMS) functions; see the Alarm Handling section in the *Global Call API Programming Guide* for information.

Cautions

- Host fault monitoring must be enabled on the board with the **brd_SendAliveEnable( )** function before using the **brd_SendAlive( )** function; otherwise, it causes a EBRD_SENDALIVENOTENABLED error.
- Board Management API errors are thread-specific.

Errors

If this function returns BRD_FAILURE to indicate failure, use the Board Management API Error Processing functions **brd_ErrorMsg( )** and **brd_ErrorValue( )** to retrieve the error information. Possible errors for this function include:

EBRD_SENDALIVENOTENABLED
: Host fault monitoring must be enabled on the board with the **brd_SendAliveEnable( )** function before using the **brd_SendAlive( )** function.

EBRD_INVALIDDEVICEHANDLE
: An invalid device descriptor was provided.

EBRD_COMMANDNOTSUPPORTED
: The operation is not supported on the board or the protocol, the board is not initialized.

Example

```
// Header files
#include <stdio.h>
#include "srllib.h"    // Standard Runtime Library API
#include "devmgmt.h"   // Board Mgt. API

// Defines
#define MAXBOARD   10   // Maximum number of physical boards

//--------------------------------------------------------------------
// Main Function
void main(void)
{
  int  physical_brd_handle[MAXBOARD];   // Store physical board handles
  int  ret;                             // Return value
  int  end;                             // Flag of sending alive message
  int  send_nums;                       // Messages number of sending
  char temp_brd_name[MAXLENGTH];        // Store temp physical board name
  unsigned short interval;              // The interval time value between two sending messages
  unsigned short threshold;             // The number of missing messages
```

```
// Initialize
memset(temp_brd_name, '\0', MAXLENGTH);
interval  = 5;          // 5 seconds
threshold = 3;          // Can miss 3 times
end       = 0;          // Not stop sending messages

// -----------------------------------
// brd_Open()
// Open the first physical board to get handle
// “brdB1” is a digital network interface board
sprintf(temp_brd_name, "brdB1");
physical_brd_handle[1] = brd_Open(temp_brd_name, 0);
if(physical_brd_handle[1] == BRD_FAILURE) // Function returns failure
{
  printf("brd_Open(%s) failed. Error = %s <%d>\n", temp_brd_name, brd_ErrorMsg(),
    brd_ErrorValue());

  if(brd_ErrorValue() == EBRD_INVALIDPHYSICALNAME)
  {
    printf("Invalid physical board name was provided \n");
  }
  return;
}
else // Function returns succ
{
  printf("brd_Open(%s) = %d\n", temp_brd_name, physical_brd_handle[1]);
}

// -----------------------------------
// brd_SendAliveEnable()
// Enable Send Alive Feature on the first physical board
ret = brd_SendAliveEnable(physical_brd_handle[1], interval, threshold, 0);
if(ret != BRD_SUCCESS) // Function returns failure
{
  printf("brd_SendAliveEnable(%d, %d, %d, 0) failed on board %s. Error = %s <%d>\n",
    physical_brd_handle[1], interval, threshold, temp_brd_name, brd_ErrorMsg(),
    brd_ErrorValue());
  return;
}
else // Function returns succ
{
  printf("brd_SendAliveEnable(%d, %d, %d, 0) succ\n", physical_brd_handle[1], interval,
    threshold);
}

// -----------------------------------
// brd_SendAlive()
// Sending alive messages to the first physical board
send_nums = 0;
while(!end)
{
  ret = brd_SendAlive(physical_brd_handle[1], 0);
  if(ret != BRD_SUCCESS) // Function returns failure
  {
    printf("brd_SendAlive() failed on board %s. Error = %s <%d>\n", temp_brd_name,
      brd_ErrorMsg(), brd_ErrorValue());
    return;
  }
  else // Function returns succ
  {
    printf("brd_SendAlive(%d, 0) = %d\n", physical_brd_handle[1], ret);
  }

  // Count the number of sending messages
  send_nums++;
  if(send_nums == 10)
  {
```

```
        printf("After sending 10 times, stop sending alive messages!\n");
        end = 1;
      }
    } // End of SendAlive

    } // Additional Processing ...

} // End of function
```

See Also

- **brd_Open( )**
- **brd_SendAliveEnable( )**
- **brd_SendAliveDisable( )**

brd_SendAliveDisable( )

Name: int brd_SendAliveDisable (brdhndl, mode)

Inputs:

int brdhndl	• board device handle
long mode	• ignored (reserved for future use); 0 is recommended setting

Returns: BRD_SUCCESS if successful
BRD_FAILURE if failure

Includes: devmgmt.h
srllib.h

Category: Fault Monitoring

Mode: synchronous

Description

The **brd_SendAliveDisable( )** function disables host fault monitoring on the board specified by the device descriptor. Host fault monitoring is enabled by the **brd_SendAliveEnable( )** function and stays enabled until disabled by a **brd_SendAliveDisable( )** or until the board is reset. If you call the **brd_SendAliveDisable( )** function multiple times for the same board, after the board is disabled with the first call, the additional calls to the function will have no effect and will be ignored.

Parameter	Description
brdhndl	specifies a board device handle obtained using a **brd_Open( )** function
mode	ignored (reserved for future use); 0 is recommended setting

Cautions

Board Management API errors are thread-specific.

Errors

If this function returns BRD_FAILURE to indicate failure, use the Board Management API Error Processing functions **brd_ErrorMsg( )** and **brd_ErrorValue( )** to retrieve the error information. Possible errors for this function include:

EBRD_INVALIDDEVICEHANDLE
An invalid device descriptor was provided.

EBRD_COMMANDNOTSUPPORTED
The operation is not supported on the board or the protocol, or the board is not initialized.

Example

The following example code shows how the function is used.

```
// Header files
#include <stdio.h>
#include "srllib.h"    // Standard Runtime Library API
#include "devmgmt.h"   // Board Mgt. API

// Defines
#define MAXBOARD   10   // Maximum number of physical boards

//---------------------------------------------------------------------
// Main Function
void main(void)
{
  int  physical_brd_handle[MAXBOARD];   // Store physical board handles
  int  ret;                             // Return value
  int  end;                             // Flag of sending alive message
  int  send_nums;                       // Messages number of sending
  char temp_brd_name[MAXLENGTH];        // Store temp physical board name
  unsigned short interval;              // The interval time value between two sending messages
  unsigned short threshold;             // The number of missing messages

  // Initialize
  memset(temp_brd_name, '\0', MAXLENGTH);
  interval  = 5;         // 5 seconds
  threshold = 3;         // Can miss 3 times
  end       = 0;         // Not stop sending messages

  // -------------------------------------
  // brd_Open()
  // Open the first physical board to get handle
  // "brdB1" is a digital network interface board
  sprintf(temp_brd_name, "brdB1");
  physical_brd_handle[1] = brd_Open(temp_brd_name, 0);
  if(physical_brd_handle[1] == BRD_FAILURE) // Function returns failure
  {
    printf("brd_Open(%s) failed. Error = %s <%d>\n", temp_brd_name, brd_ErrorMsg(),
      brd_ErrorValue());

    if(brd_ErrorValue() == EBRD_INVALIDPHYSICALNAME)
    {
      printf("Invalid physical board name was provided \n");
    }
    return;
  }
  else // Function returns succ
  {
    printf("brd_Open(%s) = %d\n", temp_brd_name, physical_brd_handle[1]);
  }

  // -------------------------------------
  // brd_SendAliveEnable()
  // Enable Send Alive Feature on the first physical board
  ret = brd_SendAliveEnable(physical_brd_handle[1], interval, threshold, 0);
  if(ret != BRD_SUCCESS) // Function returns failure
  {
    printf("brd_SendAliveEnable(%d, %d, %d, 0) failed on board %s. Error = %s <%d>\n",
      physical_brd_handle[1], interval, threshold, temp_brd_name, brd_ErrorMsg(),
      brd_ErrorValue());
    return;
  }
  else // Function returns succ
  {
    printf("brd_SendAliveEnable(%d, %d, %d, 0) succ\n", physical_brd_handle[1], interval,
      threshold);
  }
```

```
  // ------------------------------------
  // brd_SendAliveDisable()
  // Disable Send Alive Feature on the first physical board
  ret = brd_SendAliveDisable(physical_brd_handle[1], 0);
  if(ret != BRD_SUCCESS) // Function returns failure
  {
    printf("brd_SendAliveDisable() failed on board %s. Error = %s <%d>\n", temp_brd_name,
      brd_ErrorMsg(), brd_ErrorValue());
    return;
  }
  else // Function returns succ
  {
    printf("brd_SendAliveDisable(%d, 0) = %d\n", physical_brd_handle[1], ret);
  }

  // ------------------------------------
  // brd_Close()
  // Close the first physical board
  ret = brd_Close(physical_brd_handle[1]);
  if(ret != BRD_SUCCESS) // Function returns failure
  {
    printf("brd_Close() failed on board %s. Error = %s <%d>\n", temp_brd_name, brd_ErrorMsg(),
      brd_ErrorValue());
    return;
  }
  else // Function returns succ
  {
    printf("brd_Close(%d) = %d \n", physical_brd_handle[1], ret);
  }
} // End of function
```

See Also

- **brd_Open( )**
- **brd_SendAliveEnable( )**
- **brd_SendAlive( )**

brd_SendAliveEnable( )

Name: int brd_SendAliveEnable (brdhndl, interval, threshold, mode)

Inputs:

int brdhndl	• board device handle
unsigned short interval	• the duration in seconds in which the board expects to receive a “heartbeat”
unsigned short threshold	• the number of missed “heartbeats” allowed
long mode	• ignored (reserved for future use); 0 is recommended setting

Returns: BRD_SUCCESS if successful
BRD_FAILURE if failure

Includes: devmgmt.h
srllib.h

Category: Fault Monitoring

Mode: synchronous

Description

The **brd_SendAliveEnable( )** function enables host fault monitoring on the board specified by the device descriptor. When enabled, the board monitors the host computer for the presence of a repeated “heartbeat,” or “ping.” The heartbeat is sent to the board by the **brd_SendAlive( )** function from an application on the host computer. If the board does not receive a **brd_SendAlive( )** message (the “heartbeat” or “ping”) within the required parameters defined in the **interval** and **threshold**, the board treats it as a host failure. When this occurs, the board takes its network interface out of service, thus preventing the network from offering calls to the failed system. This prevents the acceptance of unwanted calls and thereby reduces the potential for being unnecessarily tariffed.

For details on what happens when the host failure occurs and the network interface is taken out of service, see the **brd_SendAlive( )** function.

Parameter	Description
brdhndl	specifies a board device handle obtained using a **brd_Open( )** function
interval	specifies the duration in seconds in which the board expects to receive a “heartbeat.” The valid range is 1 to 65535.
threshold	specifies the number of missed “heartbeats” allowed (i.e., the number of **intervals** allowed). The valid range is 0 to 255.
mode	ignored (reserved for future use); 0 is recommended setting

Once enabled, if the board receives a **brd_SendAlive( )** heartbeat from the host or a new **brd_SendAliveEnable( )**, the timer is reset to zero.

The board must receive a **brd_SendAlive( )** message within the period specified by the **interval** and **threshold** parameters. The combination of these parameters determines the maximum time within which the board must receive the heartbeat. The maximum time allowed = **interval** * (**threshold** + 1). For example, if **interval** = 60 seconds, and **threshold** = 1, the board must receive a heartbeat within 60 * (1 + 1) = 120 seconds, or 2 minutes. If the board receives the heartbeat within this time, the timer starts at zero and the board must receive another heartbeat within 2 minutes. It does not matter at what point within the 2 minutes the board receives the heartbeat, as long as it receives the message, it will restart the timer at 0 and wait for another heartbeat. Similarly, if **interval** = 60 seconds, and **threshold** = 0, the board must receive a heartbeat within 60 seconds.

The combination of the **interval** and **threshold** parameters allows you to set a very long time (hours or even days).

Once enabled, the fault monitoring stays enabled until a **brd_SendAliveDisable( )** function disables it or until the board is reset.

You can call the **brd_SendAliveEnable( )** function any number of times to reset the **interval** and **threshold** parameters to new values without calling **brd_SendAliveDisable( )** in between. If you reset the **interval** and **threshold** parameters to new values with **brd_SendAliveEnable( )**, it overwrites the previous values and resets the timer to zero.

The design and implementation of the host computer application program determines what constitutes a host failure. As long as the heartbeat continues within the time parameters enabled for the board, the board treats the host as a healthy system, and from the board's perspective, the host is "up" and running. If the heartbeat stops, for whatever reason, and the time parameters are exceeded, the board treats it as a host failure, and from the board's perspective, the host is "down;" In this case, the board will remove its network interface from service and respond to any calls with an out-of-service message.

Cautions

- This function enables the timer with the specified parameters and starts the timer.
- The host computer application program requires but a single thread to ping all the boards in the system.
- Board Management API errors are thread-specific.

Errors

If this function returns BRD_FAILURE to indicate failure, use the Board Management API Error Processing functions **brd_ErrorMsg( )** and **brd_ErrorValue( )** to retrieve the error information. Possible errors for this function include:

EBRD_INVALIDINTERVAL
: An invalid value of zero was specified for the timer **interval**.

EBRD_INVALIDTHRESHOLD
: An invalid value for the timer **threshold** was provided.

EBRD_INVALIDDEVICEHANDLE
: An invalid device descriptor was provided.

EBRD_COMMANDNOTSUPPORTED
The operation is not supported on the board or the protocol, or the board is not initialized.

Example

The following example code shows how the function is used.

```
// Header files
#include <stdio.h>
#include "srllib.h"    // Standard Runtime Library API
#include "devmgmt.h"   // Board Mgt. API

// Defines
#define MAXBOARD   10   // Maximum number of physical boards

//---------------------------------------------------------------------
// Main Function
void main(void)
{
  int  physical_brd_handle[MAXBOARD];   // Store physical board handles
  int  ret;                             // Return value
  int  end;                             // Flag of sending alive message
  int  send_nums;                       // Messages number of sending
  char temp_brd_name[MAXLENGTH];        // Store temp physical board name
  unsigned short interval;              // The interval time value between two sending messages
  unsigned short threshold;             // The number of missing messages

  // Initialize
  memset(temp_brd_name, '\0', MAXLENGTH);
  interval  = 5;         // 5 seconds
  threshold = 3;         // Can miss 3 times
  end       = 0;         // Not stop sending messages

  // ------------------------------------
  // brd_Open()
  // Open the first physical board to get handle
  // "brdB1" is a digital network interface board
  sprintf(temp_brd_name, "brdB1");
  physical_brd_handle[1] = brd_Open(temp_brd_name, 0);
  if(physical_brd_handle[1] == BRD_FAILURE) // Function returns failure
  {
    printf("brd_Open(%s) failed. Error = %s <%d>\n", temp_brd_name, brd_ErrorMsg(),
      brd_ErrorValue());

    if(brd_ErrorValue() == EBRD_INVALIDPHYSICALNAME)
    {
      printf("Invalid physical board name was provided \n");
    }
    return;
  }
  else // Function returns succ
  {
    printf("brd_Open(%s) = %d\n", temp_brd_name, physical_brd_handle[1]);
  }

  // ------------------------------------
  // brd_SendAliveEnable()
  // Enable Send Alive Feature on the first physical board
  ret = brd_SendAliveEnable(physical_brd_handle[1], interval, threshold, 0);
  if(ret != BRD_SUCCESS) // Function returns failure
  {
    printf("brd_SendAliveEnable(%d, %d, %d, 0) failed on board %s. Error = %s <%d>\n",
      physical_brd_handle[1], interval, threshold, temp_brd_name, brd_ErrorMsg(),
      brd_ErrorValue());
    return;
```

```
  }
  else // Function returns succ
  {
    printf("brd_SendAliveEnable(%d, %d, %d, 0) succ\n", physical_brd_handle[1], interval,
      threshold);
  }

  // ------------------------------------
  // brd_SendAlive()
  // Sending alive messages to the first physical board
  send_nums = 0;
  while(!end)
  {
    ret = brd_SendAlive(physical_brd_handle[1], 0);
    if(ret != BRD_SUCCESS) // Function returns failure
    {
      printf("brd_SendAlive() failed on board %s. Error = %s <%d>\n", temp_brd_name,
        brd_ErrorMsg(), brd_ErrorValue());
      return;
    }
    else // Function returns succ
    {
      printf("brd_SendAlive(%d, 0) = %d\n", physical_brd_handle[1], ret);
    }

    // Count the number of sending messages
    send_nums++;
    if(send_nums == 10)
    {
      printf("After sending 10 times, stop sending alive messages!\n");
      end = 1;
    }
  } // End of SendAlive

} // End of function
```

See Also

- **brd_Open( )**
- **brd_SendAlive( )**
- **brd_SendAliveDisable( )**

brd_VirtualToPhysicalName( )

Name: int brd_VirtualToPhysicalName(virtual, physical, len)

Inputs:

char *virtual	• pointer to virtual board name
char *physical	• pointer to physical board name returned
int *len	• pointer to maximum length allowed for physical board name

Returns: BRD_SUCCESS if successful
BRD_FAILURE if failure

Includes: devmgmt.h
srllib.h

Category: Configuration

Mode: synchronous

Description

The **brd_VirtualToPhysicalName( )** function gets the physical board name for the specified virtual board name. This is useful for obtaining a physical board name (e.g., "brdB2") that is associated with a virtual network interface board (e.g., "dtiB29"). The **brd_Open( )** function uses the physical board name in its operation.

Parameter	Description
virtual	specifies a pointer to a virtual board name that is used to obtain the physical board name. The virtual board name follows the format "xxxBn", where xxx represents a board identifier for a specific technology (as in "dti") and n represents an integer equal to or greater than 1.
physical	specifies a pointer to a physical board name that is returned by the function. The board name follows the format "brdBn", where n represents an integer equal to or greater than 1.
len	specifies a pointer to the maximum length allowed for the physical board name.

Cautions

- The length of the physical board name is limited to the specified **len**.
- Board Management API errors are thread-specific.

Errors

If this function returns BRD_FAILURE to indicate failure, use the Board Management API Error Processing functions **brd_ErrorMsg( )** and **brd_ErrorValue( )** to retrieve the error information. Possible errors for this function include:

EBRD_INVALIDVIRTUALNAME
: Invalid virtual board name was provided.

EBRD_BUFFERTOOSMALL
: The size of the buffer provided is too small.

EBRD_NULLPOINTERARGUMENT
: One of the arguments provided is a null-pointer.

EBRD_DEVICEMAPPERFAILED
: Internal error indicating the SRL device mapping failed.

Example

The following example code shows how the function is used.

```
// Header files
#include <stdio.h>
#include "srllib.h"    // Standard Runtime Library API
#include "devmgmt.h"   // Board Mgt. API

// Defines
#define MAXBOARD   10   // Maximum number of physical boards
#define MAXLENGTH  12   // Maximum length of physical board name

// Main Function ---------------------------------------------------------------
void main(void)
{
  int  length;                          // The length of board name
  int  ret;                             // Return value
  char temp_brd_name[MAXLENGTH];        // Store temp physical board name

  // Initialize
  memset(temp_brd_name, '\0', MAXLENGTH);
  length    = MAXLENGTH; // The length of board name

  // brd_VirtualToPhysicalName() -----------------------
  // From Virtual board name to physical board name
  ret = brd_VirtualToPhysicalName("dtiB1", temp_brd_name, &length);
  if(ret != BRD_SUCCESS) // Function returns failure
  {
    printf("brd_VirtualToPhysicalName() failed. Error = %s <%d>\n", brd_ErrorMsg(),
      brd_ErrorValue());
    return;
  }
  else // Function returns succ
  {
    printf("Physical board name of virtual board dtiB1 is %s\n", temp_brd_name);
  }
} // End of function
```

See Also

- **brd_GetAllPhysicalBoards( )**
- **brd_Open( )**

Error Codes 3

This chapter describes the error codes supported by the Board Management API.

The functions return a value indicating the outcome of the function operation. In most cases, the function returns the value BRD_SUCCESS for a successful outcome and BRD_FAILURE for an unsuccessful outcome or an error. (Some functions return data upon success, rather than the BRD_SUCCESS value, such as the **brd_Open( )** function returning a device descriptor upon success.)

If a function returns BRD_FAILURE to indicate failure, use the Board Management API Error Processing functions **brd_ErrorMsg( )** and **brd_ErrorValue( )** to retrieve the error information.

Note: Board Management API errors are thread-specific.

The library contains the following error codes, listed in alphabetical order.

EBRD_BUFFERTOOSMALL
: The size of the buffer provided is too small. Occurs with the **brd_GetAllPhysicalBoards( )** or the **brd_VirtualToPhysicalName( )** function.

EBRD_COMMANDNOTSUPPORTED
: The operation is not supported on the board or the protocol, or the board is not initialized.

EBRD_DEVICEMAPPERFAILED
: Internal error indicating the SRL device mapping failed.

EBRD_FAILEDOPENINGDTILIB
: Internal error indicating the network interface library cannot be opened.

EBRD_INVALIDDEVICEHANDLE
: An invalid device descriptor was provided.

EBRD_INVALIDINTERVAL
: An invalid value of zero was specified for the timer **interval** in the **brd_SendAliveEnable( )** function.

EBRD_INVALIDPHYSICALNAME
: Invalid physical board name was provided for the **brd_Open( )** function.

EBRD_INVALIDTHRESHOLD
: An invalid value for the timer **threshold** was provided for the **brd_SendAliveEnable( )** function.

EBRD_INVALIDVIRTUALNAME
: Invalid virtual board name was provided for the **brd_VirtualToPhysicalName( )** function.

EBRD_NULLPOINTERARGUMENT
: One of the arguments provided is a null-pointer.

EBRD_OUTOFMEMORY
: Internal error indicating unable to allocate memory.

EBRD_SENDALIVENOTENABLED

Host fault monitoring must be enabled on the board with the **brd_SendAliveEnable( )** function before using the **brd_SendAlive( )** function.