BU-69090
Enhanced Mini-ACE Runtime Library Software Manual

MN-69090XX-001

The information provided in this Manual is believed to be accurate; however, no responsibility is assumed by Data Device Corporation for its use, and no license or rights are granted by implication or otherwise in connection therewith.

Specifications are subject to change without notice.

Please visit our Web site at **www.ddc-web.com** for the latest information

©Data Device Corporation

105 Wilbur Place, Bohemia, New York 11716-2482

For Technical Support - 1-800-DDC-5757 ext. 7771

Headquarters - Tel: (631) 567-5600 ext. 7771, Fax: (631) 567-7358
United Kingdom - Tel: +44-(0)1635-811140, Fax: +44-(0)1635-32264
France - Tel: +33-(0)1-41-16-3424, Fax: +33-(0)1-41-16-3425
Germany - Tel: +49-(0)89-150012-11, Fax: +49-(0)89-150012-22
Japan - Tel: +81-(0)3-3814-7688, Fax: +81-(0)3-3814-7689
World Wide Web - http://www.ddc-web.com

DATA DEVICE CORPORATION
REGISTERED TO ISO 9001:2000
FILE NO. A5976

REV L – February 2008

RECORD OF CHANGE

Revision	Date	Pages	Description
A	July, 2002	All	Original Release
B	October, 2002	Various	Miscellaneous typo errors corrected. Added Functional Group Listing. Corrections/clarifications made to the following functions: aceGetCoreVersion, aceGetTimeTagValue, aceSetIrqConditions, aceBCDecodeRawMsg, aceBCFrameCreate, aceBCGetHBufMsgDecoded, aceBCMsgCreateMode, aceBCOpCodeCreate, aceBCOpCodeCreate, aceRTMsgLegality, aceRTStktoHBuf, aceVxCreateISADevs, aceVxGetISADevInfo, and aceVxSetIOPort.
C	January, 2003	Various	Paragraph added to Host Buffers vs Stacks section. Corrections/clarifications made to the following functions: aceISQClear, aceISQRead, aceSetDecoderConfig, aceSetIrqConditions, aceSetTimeTagRes, aceBCGetHBufMsgDecoded, aceBCOpCodeCreate, aceRTInstallHBuf. Added: aceISQEnable function, Appendix B. Typo corrected for error –101 ACE_ERR_NODE_MEMBLOCK.
D	May, 2003	Various	Added eleven new functions: aceDioCtl, aceDioCtlBits, aceDioDir, aceDioDirBits, aceRTMTGetHBufMetric, aceRTMTGetHBufMsgCount, aceRTMTGetHBufMsgDecoded, aceRTMTGetHBufMsgsRaw, aceRTMTInstallHBuf, aceRTMTStkToHBuf, aceRTMTUninstallHBuf Updated aceBCGetHBufMetric, aceRTGetHBufMetric, aceMTGetHBufMetric Misc typos and corrections
E	August, 2003	Various	Added Appendix C: Integrity Release Information. Added twelve new functions: aceSetCANIsr, aceBCDataBlkRead32, aceBCFrmToHBuf32, aceRTRelatchAddr, aceRTStkToHBuf32, aceRTMTStkToHBuf32, aceMTStkToHBuf32, aceVxCreateEBRDevs, aceVxEnableSBMode, aceVxSetTaskPriority, aceDOSEnableSBMode, aceSetIntegrityIntPriority. Misc typos and corrections.
F	Nov., 2003	22, 252, 259	Text describing settings for Enhanced BC mode operation paragraph corrected. ACE_OPCODE_JMP description modified. Note modified.
G	Mar., 2004	Various	Added nine new functions: aceBCAsyncMsgCreateBcst, aceBCAsyncMsgCreateBcstMode, aceBCAsyncMsgCreateBcstRTtoRT, aceBCAsyncMsgCreateBCtoRT, aceBCAsyncMsgCreateMode, aceBCAsyncMsgCreateRTtoBC, aceBCAsyncMsgCreateRTtoRT, aceBCSendAsyncMsgHP, aceBCSendAsyncMsgLP. Updated aceBCConfigure.
H	Aug., 2004	Various	Added aceBCMsgModify Section Added StaticDynamic Section
J	Jan., 2006	329	Updated Description section of aceBCStart
K	Sept. 2007	Various	
L	Feb, 2008	Various	Added additional information to the sections referencing ACE_MT_DOUBLESTK

TABLE OF CONTENTS

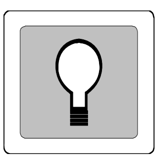
HOW TO USE THIS MANUAL

This manual uses typographical and iconic conventions to assist the reader in understanding the content. This section will define the text formatting and icons used in the rest of the manual. This manual is formatted with a ‘Scholar Margin’ where many tips, symbols or icons will be located.

Text Usage

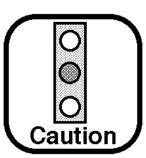
- **BOLD** – text that is written in bold letters indicates important information and table, figure, and chapter references.
- ***BOLD ITALIC*** – will designate DDC Part Numbers.
- `Courier New –` is used to indicate code examples.
- <…> - Indicates user entered text or commands.

Symbols and Icons



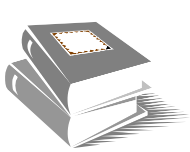
The Idea/Tip icon will be used to identify a handy bit of supplementary information that may be useful to the user.

The Note icon signifies important supplementary information that will be useful to the user.

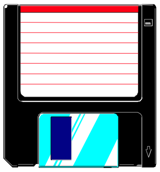


The Caution icon identifies important information that presents a possibility of damage to the product if not heeded.

Much stronger than a Caution, the Warning icon presents information pertaining to hazards that will cause damage to the product and possible injury to the user.



The Reference icon indicates that there is related material in this manual or in another specified document.



The Disk Icon describes information that is related to software.

Special Handling and Cautions

The ***BUS-69090*** is delivered on either a Compact Disk or a set of floppies, and proper care should be used to ensure that they are not damaged by heat or magnetic fields.

- Do not store disks in environments exposed to excessive heat, magnetic fields or radiation.

Trademarks

All trademarks are the property of their respective owners.

INTRODUCTION

Features

- Library of "C" Routines that are available for: DOS, Windows® 9x/2000/XP, Windows NT®, Linux, VxWorks, and Integrity operating systems
- Documentation Provided
- Provides Modular, Portable, Readable Code That Reduces Software Development Time
- "C" Structures Eliminate Need to Learn Detailed Address/Bit Maps and Data Formats
- Includes Routines to Create Binary Image File for Enhanced Mini-ACE, eliminating the need to validate Third Party Software.
- Includes Sample Programs and Compiled Libraries for Quick Startup
- Multi Environment/Compiler Support Included

Description

The intent of this runtime library is to provide the framework for developing "realtime" drivers and/or applications for the Enhanced Mini-ACE series of MIL-STD-1553 components and cards with minimal development time. The software contained within this runtime library was developed using Microsoft Visual C++ version 6.0.

This runtime library is written such that all low level access to the Enhanced Mini-ACE communication processor is performed through a set of functions in the interface and interrupt control modules. This allows the runtime library to be easily ported to any hardware and/or software platform by modifying these "low level" routines. The library includes a module specifically designed to allow portability among several existing Enhanced Mini-ACE interface's and operating environments, and is easily modified to include new hardware/software configurations.

If the Enhanced Mini-ACE is to be used in a mission critical embedded environment that requires software validation, it may be beneficial to use the binary image generation capability of the Enhanced Mini-ACE library. With this functionality, the user is able to create and test an Enhanced Mini-ACE configuration on a PC platform. After the functionality of the Enhanced Mini-ACE software is tested and stable, a library routine can be added to the software that will generate an exact image of the Enhanced Mini-ACE setup in a file. This file can then be downloaded into the Enhanced Mini-ACE in the embedded system, eliminating the necessity of having to use or validate the DDC library.

What is included in this manual?

This manual contains a complete listing of the software supplied with the card.

The library software provides a level of abstraction such that it is not necessary to understand the operation of the chip set. This manual provides a detailed description of the library.

System Requirements

- DOS, Windows 9x/2000/XP, Windows NT, Linux, VxWorks, or Integrity
- Tornado software development environment for VxWorks platforms
- MULTI 2000 Integrity Development Environment by Green Hills Software
- An appropriate compiler or development environment for Microsoft platforms

Technical Support

In the event that problems arise beyond the scope of this manual, you can visit the DDC web site and review the FAQ page. If you still have questions you can get in touch with DDC by calling:

U.S.A. toll free:	1-800-DDC-5757, ext. 7771
Outside U.S.A.:	(631) 567-5600, ext. 7771

Regional Technical Support:

Southeast, U.S.A.:	
(703) 450-7900	Fax: (703) 450-6610
Northeast and Mid-West, U.S.A.:	
(631) 567-5600, ext 7234	Fax: (631) 567-5758
West Coast, U.S.A.:	
(714) 895-9777	Fax: (714) 895-4988
Europe:	
44 (1635) 811140	Fax: 44 (1635) 32264
Asia/Pacific:	
81 (3) 3814-7688	Fax: 81 (3) 3814-7689

DDC also has an Internet World Wide Web site, which allows customers to easily download new revisions of software and documentation. The Internet address is **www.ddc-web.com**.

LIBRARY OVERVIEW

The Enhanced Mini-ACE RTL (Runtime Library) uses an Initialization-Setup-Run-Free programming paradigm. The RTL uses an internal state machine to make sure that the above paradigm is adhered to. During initialization the programmer specifies which mode of operation is to be performed by the Enhanced Mini-ACE device. Currently, there are five modes of operation supported by the RTL. They are as follows: ACE_MODE_BC, ACE_MODE_RT, ACE_MODE_MT, ACE_MODE_RTMT, and ACE_MODE_TEST. Each operating mode is initialized to its default state, but can be changed from this state by using the appropriate configuration function. In addition, there are two parameters that can be logically or'ed to any of the following modes: ACE_MODE_BC, ACE_MODE_RT, ACE_MODE_MT, or ACE_MODE_RTMT. The two parameters are ACE_NO_TT_RESET, and ACE_ADVANCED_MODE. If the ACE_NO_TT_RESET option is also chosen, the value of the time tag register will never be reset by the device and will run until the aceFree() function is called. The ACE_ADVANCED_MODE allows the user access to some advanced functions that are not regularly used. Please see the aceInitialize() function call for more information. Only functions for the mode of operation currently running, and global functions that can be used for all modes will be available after initialization.

The RTL uses an internal state machine to verify that functions are not called out of sequence. The internal states used by the library are: ACE_STATE_RESET, ACE_STATE_READY, and ACE_STATE_RUN. Before any device number is initialized, they are all set to the RESET state. Once the device has been successfully initialized the RTL modifies the state for the device and sets it to the READY state. If a device is run using one of the start functions the RTL will then modify the device to be in the RUN state.

The following sequence shows what state the RTL is in when functions are called in MT mode.

```
/* The device number DevNum is initially ACE_STATE_RESET */
aceInitialize(
      DevNum,           /* Device number of card */
      ACE_ACCESS_CARD,  /* Access Type: Card */
      ACE_MODE_MT,      /* Operating Mode: MT */
      0,                /* Memory Length: N/A */
      0,                /* Registers Address : N/A */
      0);               /* Memory Address : N/A */
/* Once initialized DevNum is ACE_STATE_READY */

aceMTStart(DevNum);
/* DevNum is running thus it is ACE_STATE_RUN */

aceMTStop(DevNum);
/* DevNum is stopped, DevNum is now ACE_STATE_READY again */

aceFree(DevNum);
/* DevNum is no longer initialized and is ACE_STATE_RESET */
```

Throughout this manual functions are described as being available during certain modes of operation and during certain device states.

Version Information

DDC versioning mask is defined as X.Y.Z. ‘X’ is the major version number and denotes a major change in design and functionality of the software. ‘Y’ is the minor revision and denotes an update or added functionality to the software. ‘Z’ is the interim release number. Any software with a version number where the third descriptor ‘Z’ is a number from 1 to 9 is interim software. Interim software is the latest version software available from DDC’s engineering department that has not been fully validated and officially released. If ‘Z’ is 0, this indicates that the software has been fully validated by DDC’s Software Quality Assurance Department and is officially released. If ‘Z’ is not present then this version was created before our X.Y.Z version mask was in place. Any version where ‘Z’ is not present should be considered an interim release software.

Naming Conventions

All functions in the RTL contain a prefix, which indicates how that function will be used in a program. All internal functions begin with an underscore and all external functions begin with 'ace'. Internal functions also contain 2 characters after the underscores that indicate what file they can be found in. The following is an example of an internal function:

```
_aceMemRead(DevNum,0); /* Internal function found in access.c
*/
```

The following is an example of an external function:

```
   aceBCDataBlkDelete(DevNum, DBLK1); /* BC external function
*/
```

External functions that can be used in all modes of operation do not have a mode prefix. The following is an example of a function that can be called in any mode of operation:

```
   aceSetRespTimeOut(DevNum, ACE_RESPTIME_18US);
```

External functions that can be used in BC mode of operation have a mode prefix of 'BC'. The following is an example of a function that can only be called in BC mode of operation:

```
   aceBCFrameDelete(DevNum, nFrameID);
```

External functions that can be used in RT mode of operation have a mode prefix of 'RT'. The following is an example of a function that can only be called in RT mode of operation:

```
   aceRTDataBlkDelete(DevNum, nDataBlkID);
```

External functions that can be used in MT mode of operation have a mode prefix of 'MT'. The following is an example of a function that can only be called in MT mode of operation:

```
   aceMTInstallHBuf(DevNum, dwHBufSize);
```

External functions that can be used in RTMT mode of operation have a mode prefix of ‘RTMT’. The following is an example of a function that can only be called in RTMT mode of operation:

```
aceRTMTGetHBufMsgCount(DevNum);
```

External functions that can be used in TEST mode of operation have a mode prefix of ‘Test’. The following is an example of a function that can only be called in TEST mode of operation:

```
aceTestRegisters(DevNum, pTest);
```

There are also external functions which are used to set and/or get the status of certain library features or registers. These external functions can be used in BC, RT, MT, or RTMT modes of operation and have a prefix of ‘Get’ or ‘Set’. Not every Get function has an associated Set function. The aceGetLibVersion function has no associated Set function because the library version cannot be set. The following is an example of a Get/Set function:

```
aceGetTimeTagValue(DevNum, &wTTValue);
aceSetTimeTagValue(DevNum, wTTValue);
```

Variable naming conventions

Whenever possible, DDC’s own form of “Hungarian notation” was used to describe variables. The following is a list of the most used prefixes that describe the properties of a variable:

dw – U32BIT, a 32-bit unsigned integer

w – U16BIT, a 16-bit unsigned integer

n – S32BIT, a 32-bit signed integer

p – pointer to a structure or address

pp – pointer to a pointer to a structure or address

b – variable is used as a flag or BOOLEAN

PROGRAMMING WITH THE ENHANCED MINI-ACE RTL

All programs that use the Enhanced Mini-ACE RTL must contain the external header file “stdemace.h”. This file includes all other external header files:

Bcop.h - contains BC prototypes and defines

Config.h - contains configuration/global prototypes and defines

Errordef.h - contains error definitions

Msgop.h - contains message operations and structures

Mtop.h - contains MT prototypes and defines

Rtmtop.h - contains RT/MT prototypes and defines

Rtop.h - contains RT prototypes and defines

Testop.h - contains Test mode prototypes and defines

Internal header files are also included with the RTL. The internal header files contain prototypes and defines that are not necessary for external programs to be compiled.

STATIC VS. DYNAMIC LINKS TO THE DLL

The Enhanced Mini-ACE Runtime Library API comes with a dll file that contains all of the function definitions for the API. This API interface allows you to include a library file (lib file) in your project and dynamically link into the dll. When you create an executable, this executable will look for a dll to link to each time you execute your program. The library file is called emace_32.lib and must be included in your project in order to have a dynamically linked executable. This is highly recommended because it allows you to update versions of DDC's Enhanced Mini-ACE software without rebuilding your application.

If you would like to standardize on a version of the Enhanced Mini-ACE and not upgrade then you can perform a static link to the dll. A static link will include all of the functionality of the API library into the executable of your program. This type of program is larger size but does not require a dll file on the computer. To update to a newer version you would have to recompile and build your application again. You can create an executable that has a static link where no dll file will be required by including the emace_32s.lib file in your project.

ENHANCED MINI-ACE REGISTERS

Each Enhanced Mini-ACE device contains 16-bit hardware registers which are used to initialize, setup, and operate the device. The Enhanced Mini-ACE library writes to these hardware registers to set up parameters according to the variables and functionality input to a function by the user. For some function calls the library will not write directly to a hardware register, but instead to a simulated software register. Internal functions simulate a write to a hardware register by writing to the REGSTATE simulated software register structure in the hardware device information structure. These simulated registers will be written to the actual hardware registers once a Start function has been called (i.e., aceBCStart, aceMTStart..etc). The following table represents the simulated registers that exist in software and their corresponding hardware registers.

Software Register Name	Hardware Register Name	Hardware Location
wIMR1	Interrupt Mask Register # 1	0x00
wCFG1	Configuration Register # 1	0x01
wCFG2	Configuration Register # 2	0x02
wBCRTCtrl	BC Control Word Register / RT Subaddress Control Word Register	0x04
wTT	Time Tag Register	0x05
wCFG3	Configuration Register # 3	0x07
wCFG4	Configuration Register # 4	0x08
wCFG5	Configuration Register # 5	0x09
wRTMTDataStkAddr	RT / MT Data Stack Address Register	0x0A
wBCFrmTimeRemain	BC Frame Time Remaining Register	0x0B
wBCRTMTMisc	BC Frame Time/RT Last Command/MT Trigger Register	0x0D
wCFG6	Configuration Register # 6	0x18
wCFG7	Configuration Register # 7	0x19
wBCGPF	BC General Purpose Flag Register	0x1B
wIMR2	Interrupt Mask Register # 2	0x1D
wBCRTMTQP	BC General Purpose Queue Pointer Register / RT, MT Interrupt Status Queue Pointer Register	0x1F
wRTBitWrd	RT Mode Block Status Word	N/A

LIBRARY INITIALIZATION

The Enhanced Mini-ACE RTL was created with the understanding that different programmers will have different software needs. There are three different ways the RTL can be initialized, and each method has its own specific function. The first function called in any program must be the aceInitialize() function. This function will allow the programmer to setup the access type, operation mode, and memory/register specific information if necessary.

Access Types

The library allows for three different access types specified in the aceInitialize() function which must always be the first function called when programming in the Enhanced Mini-ACE runtime library. This library allows for card, simulated, or user access. All three types of access are described in complete detail below.

Card Access

The basic operation of the runtime library allows a programmer to create extensive applications for DDC's card level Enhanced Mini-ACE products (including, but not limited to, the BU-65553 Enhanced Mini-ACE PCMCIA Card). This method uses the following initialization call:

```
aceInitialize(
   DevNum,            /* Device number of card */
   ACE_ACCESS_CARD,   /* Access Type: Card */
   ACE_MODE_BC,       /* Operating Mode: BC */
   0,                 /* Memory Length: N/A */
   0,                 /* Registers Address : N/A */
   0);                /* Memory Address : N/A */
```

The above function call uses the device number of a valid Enhanced Mini-ACE Card. Since the card's resources are setup by the operating system all resource specific information is not applicable here. In this particular example the device is being initialized to run in BC mode of operation.

Logical Device Numbers for Windows 9x/2000/XP and Windows NT

The Enhanced Mini-ACE Software Library comes with a DDC 1553 Card Manager (see Figure 1). This application provides the user with a graphical tool used to assign a logical device number to a card that has been enumerated on your computer's bus by the operating system. This is the logical device number that you will use to access the card in each of the functions provided in the Enhanced Mini-ACE Runtime Library. To assign a logical device number to a card you will have to open the DDC 1553 Card Manager which is located in your Windows control panel. You can create a shortcut to the DDC 1553 Card Manager on your desktop for your convenience if you wish to do so. For detailed information about assigning your card a logical device number please reference the Manual for your specific hardware card.

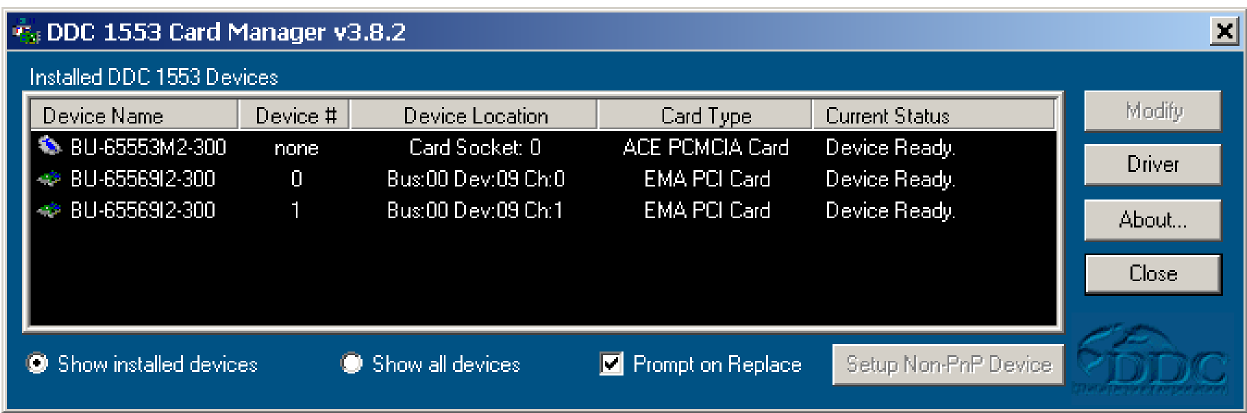


Figure 1. DDC 1553 Card Manager

Logical Device Numbers for VxWorks

The VxWorks Enhanced Mini-ACE Runtime Library provides you with the aceEnumCards() function to assign each device a logical device number. This program enumerates (finds and allocates resources for) all cards located on the single board computer's local PCI bus. This application enumerates all EMACE devices via a call to the aceVxCreateDevs API function. Once the devices are enumerated into the system the aceVxGetDevInfo() is called to retrieve and display all allocated device resources. Both functions are described in the API reference section for VxWorks in the back of this manual to enumerate the bus.

Logical Device Numbers for Linux

The Linux Enhanced Mini-ACE Runtime Library contains portable source code which can be re-compiled on the target platform and used with minimal change and allows for the creation of an Enhanced Mini-ACE memory image file with all the locations to structures within memory. These files can be created on platforms other than the one targeted for embedded design. This software package contains a card configuration file used to associate a logical device number to a particular card/channel installed in a system. Since a DDC control panel application was not yet available at the time of this package release, the user must edit the sample file included in the Linux distribution in order to access and use a card. The format of the file is documented in complete detail in the readme file included with the software package. Please reference this readme file for more information on assigning your card a logical device number in your Linux operating system.

Logical Device Numbers for Integrity

The Integrity Enhanced Mini-ACE Runtime Library provides automated device enumeration through the “EmacePciDriver.o” kernel driver. Upon operating system boot-up, any detected Enhanced Mini-ACE device will have its channels enumerated. A message will be displayed to the terminal window informing the user of any found devices. Device numbers are assigned consecutively beginning from device 0. (e.g. - a four channel device will be assigned device numbers 0 to 3.)

Integrity Virtual Memory Execution Mode

The Integrity Runtime Library is configured such that it places the controlling Single Board Computer (SBC) processor into virtual memory mode. This provides for added task code and data protection, and kernel process protection.

Any particular device will exist as an Integrity 'IODevice' object that is to be loaded 'statically' at boot time. This type of object possesses the capability to provide access to a card's physical memory mapped address region from the kernel process and a user task, a design requirement of the RTL regarding interrupt processing.

The card windows are mapped with the 'IOCoherent' option set to guarantee that no caching will occur on the region. Intertask data protection is accomplished by establishing a 'connection' between the individual tasks that are to share the data. Refer to the Green Hills Integrity online documents 'integrity.pdf' and 'bspguide.pdf' for full details and available functions.

Simulated Access

The RTL can be programmed without the need of an Enhanced Mini-ACE card. In Simulated Access mode the library will allocate host memory to be used as if it were the Enhanced Mini-ACE. Although the mode of operation being simulated cannot be run, files can be created that will allow the programmer to recreate the setup inside an embedded system. This method uses the following initialization call:

```
aceInitialize(
   DevNum,            /* Any unused device number */
   ACE_ACCESS_SIM,    /* Access Type: Simulated */
   ACE_MODE_BC,       /* Operating Mode: BC */
   0x10000,           /* Memory Length: 64k words */
   0,                 /* Registers Address : N/A */
   0);                /* Memory Address : N/A */
```

The above function call gets passed to it any currently unused device number. Since the simulated resources will be allocated by the system, the only resource specific information that must be passed to the RTL is the size of memory to be simulated as Enhanced Mini-ACE memory.

User Access

The RTL is an ANSI C library which is portable across many different platforms and compilers. A programmer may wish to port the library in its entirety to the target system. Once the RTL is compiled into the target system the programmer can initialize the embedded Enhanced Mini-ACE using the following function call.

```
aceInitialize(
   DevNum,            /* Any unused device number */
   ACE_ACCESS_USR,    /* Access Type: User */
```

```
   ACE_MODE_BC,       /* Operating Mode: BC */
   0x00010000,        /* Memory Length: 64k words */
   0x000D0000,        /* Registers Address : 0xD0000 */
   0x000D1000);       /* Memory Address : 0xD1000 */
```

The above function call gets passed to it any currently unused device number. The memory and register base addresses used above should match the actual location where the Enhanced Mini-ACE resides in the embedded memory map.

Modes of Operation

The Enhanced Mini-ACE runtime library allows for five modes of operation which are also selected with the aceInitialize() function. The five modes of operation are: BC, RT, MT, RTMT, and Test mode. The following parameters may be logically or'ed with BC, RT, MT, or RTMT modes of operation: ACE_NO_TT_RESET, and ACE_ADVANCED_MODE. If you optionally also choose the ACE_NO_TT_RESET option the value of the time tag register will never be reset by the device and will run until you call the aceFree() function. The ACE_ADVANCED_MODE allows the user access to some advanced functions that are not regularly used. The advanced functions, which can only be used if Advanced mode is logically or'ed with the operating mode, are as follows: aceSetIrqConfig(), aceSetClockFreq(), aceSetAddressMode(), and aceSetDecoderConfig(). In addition, the following functions can be run in the Reset, Ready, or Run state if the Advanced option has been logically or'ed in with the operating mode: aceRegRead(), aceRegWrite(), aceMemRead(), aceMemWrite(). If you did not choose the Advanced option then you will only be able to use the above four mentioned functions if your device is in a Ready state.

BC Mode

The Enhanced Mini-ACE device can be set up to run as a Bus Controller on the 1553 data bus by calling the aceInitialize() function with BC mode set. This method uses the following initialization call:

```
aceInitialize(
   DevNum,            /* Any unused device number */
   ACE_ACCESS_CARD,   /* Access Type: Card */
   ACE_MODE_BC,       /* Operating Mode: BC */
   0x0,               /* Memory Length: N/A */
   0x0,               /* Registers Address:  N/A */
   0x0);              /* Memory Address:  N/A
```

RT MODE

The Enhanced Mini-ACE device can be set up to run as a Remote Terminal on the 1553 data bus by calling the aceInitialize() function with RT mode set. This method uses the following initialization call:

```
aceInitialize(
   DevNum,            /* Any unused device number */
   ACE_ACCESS_CARD,   /* Access Type: Card */
   ACE_MODE_RT,       /* Operating Mode: RT */
   0x0,               /* Memory Length: N/A */
   0x0,               /* Registers Address:  N/A */
   0x0);              /* Memory Address:  N/A
```

MT MODE

The Enhanced Mini-ACE device can be set up to run as a Monitor on the 1553 data bus by calling the aceInitialize() function with MT mode set. This method uses the following initialization call:

```
aceInitialize(
   DevNum,            /* Any unused device number */
   ACE_ACCESS_CARD,   /* Access Type: Card */
   ACE_MODE_MT,       /* Operating Mode: MT */
   0x0,               /* Memory Length: N/A */
   0x0,               /* Registers Address:  N/A */
   0x0);              /* Memory Address:  N/A
```

RTMT MODE

The Enhanced Mini-ACE device can be set up to run as a combination Remote Terminal and Monitor on the 1553 data bus by calling the aceInitialize() function with RTMT mode set. This method uses the following initialization call:

```
aceInitialize(
   DevNum,            /* Any unused device number */
   ACE_ACCESS_CARD,   /* Access Type: Card */
   ACE_MODE_RTMT,     /* Operating Mode: RTMT */
   0x0,               /* Memory Length: N/A */
   0x0,               /* Registers Address:  N/A */
   0x0);              /* Memory Address:  N/A
```

TEST MODE

The Enhanced Mini-ACE device can be set up to run in Test mode on the 1553 data bus by calling the aceInitialize() function with Test mode set. This mode allows the user to run a series of tests on the card to make sure it is working properly. This method uses the following initialization call:

```
aceInitialize(
   DevNum,            /* Any unused device number */
   ACE_ACCESS_CARD,   /* Access Type: Card */
   ACE_MODE_TEST,     /* Operating Mode: Test */
   0x0,               /* Memory Length: N/A */
   0x0,               /* Registers Address:  N/A */
   0x0);              /* Memory Address:  N/A
```

ID'S AND THE ENHANCED MINI-ACE LIBRARY

Throughout the RTL, functions use identification numbers for creation, deletion and manipulation of certain data structures. These structures include Data Blocks (sometimes referred to as Data Tables), Message Blocks, OpCodes, Major Frames and Minor Frames. Each type of ID (Message Blocks, OpCodes…etc) are referenced independently of one another. You can assign the same ID to an OpCode as a previously existing Message Block since there is no relation between the ID's for different types of data structures.

The identification numbers are specified on the creation of a structure. The following code illustrates the creation of a 32 word Data Block with an ID of 1. This Data Block is not initialized.

```
aceRTDataBlkCreate(DevNum, 1 /*ID*/, 32, NULL, 0);
```

The ID of the Data Block can be used later to read or write.

```
aceRTDataBlkRead(DevNum, 1 /*ID*/, &wBuffer, 32, 0);
```

HOST BUFFERS VS STACKS

The Enhanced Mini-ACE RTL allows the programmer to log all messages processed in all 1553 modes of operation. When programming in a Real-Time OS such as VxWorks, simple logging techniques such as polling can be used to capture all messages. For example, in MT mode the functions aceMTGetStkMsgsRaw() and aceMTGetStkMsgsDecoded() can be used repeatedly in a loop to get all messages. The messages are obtained directly from message stacks residing on the Enhanced Mini-ACE hardware device itself.

An alternate method can be used in systems which do not have the luxury of Real-Time processing. In Windows, for example, the RTL allows for a Host Buffer to be installed. The Host Buffer (HBUF) is a circular memory buffer resident on the host that contains the log of all messages. Messages are transferred to the HBUF by means of interrupts which occur at time tag rollover, 50% rollover, and 100% rollover of the message stack on the Enhanced Mini-ACE hardware device in RT or MT mode. The functions aceMTGetHBufMsgsRaw() and/or aceMTGetHBufMsgsDecoded() can be used repeatedly in a loop to poll the HBUF on nondeterministic systems such as Windows.

When polling stacks, the following #defines can be used to convey to the RTL which stack should be read:

ACE_MT_STKLOC_ACTIVE
ACE_MT_STKLOC_INACTIVE
ACE_MT_STKLOC_STKA
ACE_MT_STKLOC_STKB

In BC mode a host buffer is implemented to provide the user with a log of all messages that have gone out onto the 1553 data bus. The host buffer will allow the user to see all messages that were sent by the BC as well as the status of those messages. This can be an extremely useful tool in post-analysis of data after the BC has sent the message. The format of the contents of the host buffer in BC mode of operation is described later on in this section of the manual. The user can periodically poll the host buffer and dequeue the buffer by reading messages in raw format with the aceBCGetHBufMsgsRaw() function or by reading a message in a decoded format with the aceBCGetHBufMsgDecoded() function. The aceBCGetHBufMsgsRaw() function will read all messages currently on the host buffer and purge them. The aceBCGetHBufMsgDecoded() function gives the user a choice to purge or not purge the message on the host buffer. If you use the aceBCGetHBufMsgDecoded() function and never purge the message, the host buffer may eventually overflow. If the host buffer overflows, older messages will be replaced with newer messages, since the host buffer has a circular buffer structure, and a count of the number of messages lost will be returned to the user.

Decoded Vs Raw Messages

When polling stacks or Host Buffers, messages read can be returned in one of two forms: Decoded or Raw. Raw messages contain the binary information read directly from the Enhanced Mini-ACE. This binary form is specific to the 1553 mode of operation. In MT mode a raw message takes on the following form:

Word Location[0] = Block Status Word
Word Location[1] = Time Tag Word
Word Location[2] = (15-not all data rcv'd),
(14-8 n length)|MsgType
Word Location[3] = Command Word
Word Location[4] = Buffer Word n - 4
Word Location[n] = n
Word Location[40-n] = fixed length (filled with 0)

In RT mode a raw message takes on the following form:

Word Location[0] = Block Status Word
Word Location[1] = Time Tag Word
Word Location[2] = (15-not all data rcv'd), (14-8 n length)|MsgType
Word Location[3] = Command Word
Word Location[4] = Buffer Word n - 4
Word Location[n] = n
Word Location[36-n] = fixed length (filled with 0)

In BC mode a raw message takes on the following form:

Word Location[0] = BC Control Word
Word Location[1] = Command Word
Word Location[2] = (15-not all data rcv'd),
(14-8 n length)|MsgType
Word Location[3] = Time to Next Message
Word Location[4] = Time Tag Word
Word Location[5] = Block Status Word
Word Location[6] = Loopback Word
Word Location[7] = RT Status Word
Word Location[8] = 2nd (Tx)Command Word – RT-RT
Word Location[9] = 2nd RT Status Word – RT-RT
Word Location[10] = Buffer Word n - 10
Word Location[n] = n
Word Location[42-n] = fixed length (filled with 0)

For some applications the above raw binary form will be preferred when reading messages from the RTL. Alternatively, decoded messages with information parsed out in a easy to understand message structure may be needed. The decoded message structure is defined as follows:

```
typedef struct MSGSTRUCT
{
   U16BIT wType;            /* Contains the msg type */
   U16BIT wBlkSts;          /* Contains the block status word */
   U16BIT wTimeTag;         /* Time tag of message */
   U16BIT wCmdWrd1;         /* First command word */
   U16BIT wCmdWrd2;         /* Second command word (RT to RT) */
   U16BIT wCmdWrd1Flg;      /* Is command word 1 valid? */
   U16BIT wCmdWrd2Flg;      /* Is command word 2 valid? */
   U16BIT wStsWrd1;         /* First status word */
   U16BIT wStsWrd2;         /* Second status word */
   U16BIT wStsWrd1Flg;      /* Is status word 1 valid? */
   U16BIT wStsWrd2Flg;      /* Is status word 2 valid? */
   U16BIT wWordCount;       /* Number of valid data words */
   U16BIT aDataWrds[32];    /* An array of data words */

   /* The following are only applicable in BC mode messages */
   U16BIT wBCCtrlWrd;       /* Contains the BC control word */
   U16BIT wBCGapTime;       /* Message gap time word */
   U16BIT wBCLoopBack1;     /* First looped back word */
   U16BIT wBCLoopBack2;     /* Second looped back word */
   U16BIT wBCLoopBack1Flg; /* Is loop back 1 valid? */
   U16BIT wBCLoopBack2Flg; /* Is loop back 2 valid? */
}MSGSTRUCT;
```

BC OPERATION

The BC functionality for the Enhanced Mini-ACE includes two separate architectures: (1) the legacy non-Enhanced mode, which provides complete compatibility with the previous ACE and Mini-ACE (Plus) generation products; and (2) the newer, enhanced mode. The enhanced mode offers several new powerful architectural features. These include the incorporation of a highly autonomous BC message sequence control engine, which serves to offload the operation of the host CPU. The Enhanced Mini-ACE contains an autonomous BC architecture. The BC uses OpCodes to sequence through a frame of messages and perform non-message based operations. The Enhanced Mini-ACE RTL defaults to Enhanced mode when the user configures the device for BC, RT, MT, or RTMT mode.

Additionally, the Enhanced Mini-ACE Runtime Library provides support only for Enhanced BC mode of operation. This is accomplished by setting bit 15 of Configuration Register # 6 at memory location 0x18 to a logic 1. If this bit is set to a logic 0, the device will enter non-enhanced BC mode. In the non-enhanced BC mode, the Enhanced Mini-ACE opcodes, condition codes, and BC user defined interrupts are not used and this particular method of operation is not supported by the Enhanced Mini-ACE Runtime Library. You can, however, run in legacy non-Enhanced mode in conjunction with Enhanced BC mode to provide for compatibility with the previous ACE and Mini-ACE (Plus) generation products.

The following OpCodes are supported by the RTL:

ACE_S_OPCODE_AMSG
ACE_OPCODE_XEQ
ACE_OPCODE_JMP
ACE_OPCODE_CAL
ACE_OPCODE_RTN
ACE_OPCODE_IRQ
ACE_OPCODE_HLT
ACE_OPCODE_DLY
ACE_OPCODE_WFT
ACE_OPCODE_CFT
ACE_OPCODE_FLG
ACE_OPCODE_LTT
ACE_OPCODE_LFT
ACE_OPCODE_SFT
ACE_OPCODE_PTT
ACE_OPCODE_PBS
ACE_OPCODE_PSI
ACE_OPCODE_PSM
ACE_OPCODE_WTG

ACE_OPCODE_XQF
ACE_OPCODE_TRP

All opcodes are fully defined in the aceBCOpCodeCreate() function.

Example One

Many avionics applications involve the use of periodic polling. The Enhanced Mini-ACE bus controller provides a number of mechanisms for message timing control. This involves the implementation of minor and major frames and control of inter-message gaps. Minor and major frames may be used as a mechanism for regulating periodic, highly deterministic message traffic. (see Figure 2)

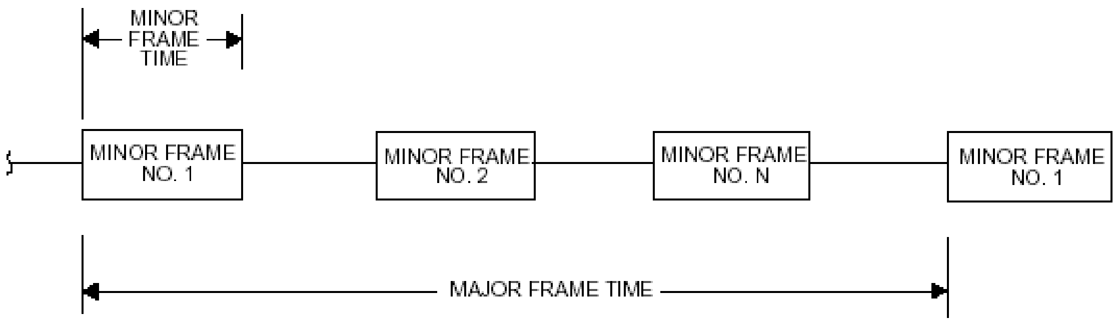


Figure 2. Minor/Major Frames

The BC is programmed by creating Data Blocks, Message Blocks, OpCodes, Minor Frames, and a Major Frame. A minor frame can contain multiple messages and each item created will contain an ID to be referenced by other functions. The ID's of each item are independent of each other. A minor frame contains a series of opcodes and each opcode is associated with a particular instruction and message block with the aceBCOpcodeCreate() function. A major frame will contain a CAL opcode for each minor frame. This is the method that inserts a minor frame into a major frame. A message block is associated to a previously defined data block by using one of the aceBCMsgCreate() functions.

The following source code shows how one minor frame that contains one message can be constructed using the Enhanced Mini-ACE Library in BC mode.

```
aceInitialize(
   DevNum,            /* Device number of card */
   ACE_ACCESS_CARD,   /* Access Type: Card */
   ACE_MODE_BC,       /* Operating Mode: BC */
   0,                 /* Memory Length: N/A */
   0,                 /* Registers Address : N/A */
   0);                /* Memory Address : N/A */

/* Create 1 data blocks – Not initialized */
aceBCDataBlkCreate(
   DevNum,            /* Device number of card */
   DBLK1,             /* Data Block ID to create */
   32,                /* Data Block Size */
   NULL,              /* N/A */
   0);                /* N/A */

/* Create message block – BC to RT message */
aceBCMsgCreateBCtoRT(
   DevNum,            /* Device number */
   MSG1,              /* Message ID to create */
   DBLK1,             /* Data Block of Message */
   RT1,               /* RT address */
   SA1,               /* RT subaddress */
   10,                /* Word count */
   0,                 /* Default message timer */
   ACE_BCCTRL_CHL_A);/* Use Bus A */

   /* Create XEQ opcode that will use message block 1 */
   aceBCOpCodeCreate(
   DevNum,            /* Device number */
   OP1,               /* Opcode ID to create */
   ACE_OPCODE_XEQ,    /* Execute message Opcode */
   ACE_CNDTST_ALWAYS,/* Always run message */
   MSG1,              /* Parameter 1: Message block ID */
   0,                 /* Parameter 2: N/A */
   0);                /* N/A */

/* Create CAL opcode that will call minor frame from major */
aceBCOpCodeCreate(
```

```
   DevNum,           /* Device Number */
   OP2,              /* Opcode ID to create */
   ACE_OPCODE_CAL,   /* Call subroutine opcode */
   ACE_CNDTST_ALWAYS,/* Always run minor frame */
   MNR1,             /* Parameter 1: Minor frame ID */
   0,                /* Parameter 2: N/A */
   0);               /* N/A */

/* Create array containing all created OpCodes */

aOpCodes[0] = OP1;

/* Create Minor Frame */
aceBCFrameCreate(
   DevNum,           /* Device Number */
   MNR1,             /* Frame ID to create */
   ACE_FRAME_MINOR,  /* Frame Type: Minor */
   aOpCodes,         /* Array of Opcodes */
   1,                /* Number of Opcodes in array */
   0,                /* N/A */
   0);               /* N/A */

aOpCodes[0] = OP2;

/* Create Major Frame */
aceBCFrameCreate(
   DevNum,           /* Device Number */
   MJR,              /* Frame ID to create */
   ACE_FRAME_MAJOR,  /* Frame Type: Major */
   aOpCodes,         /* Array of Opcodes */
   1,                /* Number of Opcodes in array */
   1000,             /* Minor Frame time, 100µs res */
   0);               /* N/A */
```

```
/* Start BC */
aceBCStart(
   DevNum,            /* Device Number */
   MJR,               /* Major frame ID to run */
   -1);               /* Run forever */

   /****** ENHANCED MINI-ACE WILL RUN BC FRAME ******/

/* Stop BC */
aceBCStop(DevNum);

/* Free RTL resources */
aceFree(DevNum);
```

GENERATING INTERRUPTS

The Enhanced Mini-ACE runtime library has robust interrupt generation support. The Enhanced Mini-ACE device has two Interrupt Status Registers which log all types of conditions that have occurred. The user can poll the Interrupt Status Register at memory offset 0x06 and Interrupt Status Register # 2 at memory offset 0x1E. In order for a user to generate an interrupt the Interrupt Mask Register must be set. When the Status, which is set by the hardware, and the Mask matches an interrupt will be generated. The user can set the respective Mask conditions by calling the aceSetIrqConditions() function. Some conditions may optionally require other register bits to be set in order for an interrupt to get generated. These conditions are specified in the definition of the aceSetIrqConditions() function.

Example One

For example, to generate an RT Subaddress End Of Message Interrupt the user must call the aceSetIrqConditions() function with the ACE_IMR1_RT_SUBADDR_EOM option to set the proper mask bit in Interrupt Mask Register # 1. In addition the user must set the proper bit in the RT Subaddress Control Word Register by calling the aceRTDataBlkMapToSA() function with the ACE_RT_DBLK_EOM_IRQ option. This function will set the appropriate bit in the RT Subaddress Control Word Register to generate an interrupt. All conditions that need other parameters or registers to be set other than the Interrupt Mask Registers are specified in the definition of the options for the aceSetIrqConditions() function.

```
/*
Map data block to sub address and set the RT Subaddress
Control Word register ACE_RT_DBLK_EOM_IRQ
*/
nResult=aceRTDataBlkMapToSA(
                     DevNum,
                     1,    //previously defined data block ID
                     1,    //sa
                     ACE_RT_MSGTYPE_RX,
                     ACE_RT_DBLK_EOM_IRQ,
                     TRUE);
nResult=aceSetIrqConditions(
                     DevNum,
                     TRUE,
                     ACE_IMR1_RT_SUBADDR_EOM,
                     myISR);


/* Definition of User Interrupt Service Routine */
void _DECL myISR(
         S16BIT DevNum,
         U32BIT Status)
{

   //user defined action
   int status = 0;

}
```

The contents of the status passed into the user defined interrupt service routine is the contents of the two Interrupt Status Registers. The most significant word will contain the contents of Interrupt Status Register # 2 and the least significant word will contain the contents of Interrupt Status Register # 1. The tables below show the definitions of the bit positions in the status registers.

BIT	DESCRIPTION
15(MSB)	MASTER INTERRUPT
14	RAM PARITY ERROR
13	TRANSMITTER TIMEOUT
12	BC/RT COMMAND STACK ROLLOVER
11	MT COMMAND STACK ROLLOVER
10	MT DATA STACK ROLLOVER
9	HANDSHAKE FAILURE
8	BC RETRY
7	RT ADDRESS PARITY ERROR
6	TIME TAG ROLLOVER
5	RT CIRCULAR BUFFER ROLLOVER
4	RT SUBADDRESS CONTROL WORD EOM
3	BC END OF FRAME
2	FORMAT ERROR
1	BC STATUS SET/RT MODE CODE/MT PATTERN TRIGGER
0(LB)	END OF MESSAGE

Table 1. Interrupt Status Register # 1

BIT	DESCRIPTION
15 (MSB)	MASTER INTERRUPT
14	BC OP CODE PARITY ERROR
13	RT ILLEGAL COMMAND/MESSAGE MONITOR MESSAGE RECEIVED
12	GENERAL PURPOSE QUEUE/INTERRUPT STATUS QUEUE ROLLOVER
11	CALL STACK POINTER REGISTER ERROR
10	BC TRAP OP CODE
9	RT COMMAND STACK 50% ROLLOVER
8	RT CIRCULAR BUFFER 50% ROLLOVER
7	MONITOR COMMAND STACK 50% ROLLOVER
6	MONITOR DATA STACK 50% ROLLOVER
5	ENHANCED BC IRQ3
4	ENHANCED BC IRQ2
3	ENHANCED BC IRQ1
2	ENHANCED BC IRQ0
1	BIT TEST COMPLETE
0 (LSB)	INTERRUPT CHAIN BIT

Table 2. Interrupt Status Register # 2

Example Two

To generate an interrupt at the end of every message in Monitor mode the user would only need to call the aceSetIrqConditions() function to set the appropriate bit in the Interrupt Mask Register. Whenever this condition is met an interrupt will be generated and the user defined interrupt service routine will be called.

```
nResult=aceInitialize(
               DevNum,
               ACE_ACCESS_CARD,
               ACE_MODE_MT,
               0,
               0,
               0);

nResult=aceSetIrqConditions(0,
                           TRUE,
                           ACE_IMR1_EOM,
                           myISR);

//see example one for the definition of myISR

nResult=aceMTStart(DevNum);
```

RELEASING LIBRARY RESOURCES

It is good programming practice to free up resources so they can be reused by the system. If an Enhanced Mini-ACE device is no longer needed by the program the function aceFree() can be called to perform this resource release.

INCLUDED DEMOS

Demo Programs

This software package is supplied with many examples of the use of the library and the capabilities of the hardware. The examples provided demonstrate the four main categories for use of the library. The four categories are BC mode, RT mode, MT mode, and RTMT mode.

In all cases the examples have been provided as source code with an appropriate Make file that may be used to build the executable.

BCDemo.c

This demonstration program creates a basic BC to RT message that uses a single data block of 32 words. Two opcodes are then created. One opcode is the XEQ (execute) opcode that will cause the BC message to be transmitted over the bus. The second opcode is created as a jump to minor frame by using the CAL (call) instruction.

The opcode 1 is then placed in a minor frame and opcode 2 is entered in the major frame. The major frame is run and will call the minor frame which contains the XEQ opcode. The XEQ opcode is tied to a message through the aceBCOpcodeCreate() function. One of the input parameters is the id of the message that was previously created with the aceBCMsgCreateBCtoRT() function.

When the major frame is run, it will run the major frame forever and will never halt. The frame will run forever because it was specified to do so in the aceBCStart() function.

During the execution, the operation of the program will be displayed on the screen.

BCDBuf.c

This demonstration program creates three messages that are included into one minor frame. The three messages are ‘BC to RT’, ‘RT to BC’ and ‘RT to RT’. A fourth opcode is created that will call minor frame 1. Message 1, 2, and 3 are attached to minor frame 1. The major frame will call minor frame 1 using opcode 4 and then run message 2 by attaching opcode 2.

When the major frame is run, it will run the major frame forever and will never halt. The frame will run forever because it was specified to do so in the aceBCStart() function.

While the frame is running, the software will look for keystroke input. If a ‘1’ is pressed, the program will set the General Purpose Flag (GPF) number 4. If a ‘2’ is pressed, the program will set the GPF number 5. If a ‘0’ is pressed, then the keystroke routine will quit.

The program will enter a data display mode until any key is pressed.

Finally, the BC will halt, and all allocated memory will be freed.

RTDBuf.c

This program creates an RT stack file in ASCII text using all messages read from the ACE.

A user buffer is created and initialized. The RT address is set to 5 and the buffer is attached to a double buffered data block. This data block is then attached to Sub Address 1 for transmit command and to Sub Address 2 for receive commands that the RT receives.

After these steps, the RT is set to the run state and the program enters a loop looking for keystrokes. If a ‘0’ is input, the program halts, if any other number (1 through 9) is typed, the number will be placed in the RT buffer for transmission.

RTMode.c

This demo program initializes the device to operate as an RT using the Logical Device number that is input in the command line parameters. It then sets the RT address to the value input by the user. After setting the address, the program attaches a double buffered data block to Sub Address 1.

The software will read the synchronize MODE code data by using the aceRTModeCodeReadData() function in a loop. The loop will exit and the device will close if a key is pressed.

RTBinFil.c

This program demonstrates the creation of an RT image file that can be downloaded to an embedded RT. RT mode is initialized in the normal fashion. The RT address is set to 5. A single buffer is created and mapped to a data block at sub-address 1. Finally, the image file is created.

RTMTDemo.c

This program demonstrates the minimum setup needed to run the device simultaneously as an RT and a Monitor. The program initializes the device, and sets the RT address to 6. It then creates a double buffer and maps it to sub-address 1. Finally, it starts the Remote Terminal /Monitor and then stops both. This program gets messages from the RT hardware stack and the MT hardware stack. In RT/MT mode of operation the monitor will monitor the entire 1553 data bus except for it's own RT address. The MT stack on the hardware will contain all contents of the data bus except anything received or sent by the Enhanced Mini-ACE's own RT address. This is a function of the device and cannot be changed. When using this Enhanced Mini-ACE runtime library some post processing is performed to combine the MT stack and the RT stack into this one RT/MT host buffer that will contain all monitored messages and data on the 1553 data bus.

If using this library in RT/MT mode without a host buffer installed the MT stack will contain all monitored data on the 1553 data bus except for the RT defined to be the Enhanced Mini-ACE device.

MTPoll.c

The mtpoll.c program demonstrates the setup and operation necessary to run the device in the Monitor mode. Interrupts are not used in this example; instead the software periodically polls the monitor stack and transfers the message data received to a user buffer. This program initializes the EMA to the monitor mode. Then it performs a self-test of the EMA that tests RAM, protocol, interrupt and self-test vectors. Following the self-test, the EMA is again initialized to monitor mode. This is needed to quickly restore the internal registers to a known state following self-test.

Monitor filtering is enabled and then disabled. This is not normally necessary, but this shows the use of the statements. Then monitor filtering is enabled for RT address 5, Receive messages, for all sub-addresses. Finally, the Monitor is started, and the monitored data is displayed on the terminal. If any key is pressed, the program will remove any dynamically allocated buffers and then quit.

MTIrq.c

This program demonstrates the setup and operation of the device as a Monitor operating with interrupts. The program will monitor the bus and display the data to the screen, and if a file is specified, the data will be stored to disk.

The device is initialized to Monitor Mode. After initialization, the storage file is opened if it is defined in the parameter list. The MT stack is created based on the memory size of the card. The software then allocates a host buffer based on the size of the Monitor stack which must be at least 3 times greater than the number of messages that can be stored in the command stack * 40 (fixed length messages). Finally the Monitor is started and the captured data will be displayed on the terminal and sent to a file based on the parameter list.

DEBUGGING USER APPLICATION CODE

This section explains some points to consider if you wish to debug an application created with the Enhanced Mini-ACE runtime library of functions. This section will not illustrate any debugging techniques nor does it teach the user how to debug an application.

Breakpoints

If desired, the user can debug an application by placing breakpoints in the code. Note that while in BC mode, if the BC is set to send out a specified number of major frames and a breakpoint is placed in the application after the aceBCStart() function is called, then the BC will run continuously. The Enhanced Mini-ACE runtime library loses synchronization with the number of frames sent out, and the counter never decrements to zero, at which point the HLT opcode instruction would normally stop the execution of the BC. The breakpoint halts the thread from executing and checking the number of major frames that were sent out, thus never decrementing the count.

Registers

The contents of the device's hardware registers are viewed by using the aceRegRead() function call. The user can also write to the device's hardware registers by using the aceRegWrite() function call. The Enhanced Mini-ACE runtime library has a set of simulated software registers that are copies of the physical hardware registers on the device itself. The actual hardware registers are not written to until the user calls one of the 'Start' function calls, such as aceBCStart() if in BC mode. To use the aceRegRead() function to view the actual contents of the physical hardware registers after calling one of the 'Start' function calls, the library software must be in advanced mode of operation. Advanced mode must be specified by logically or'ing ACE_ADVANCED_MODE to the mode of operation when calling the aceInitialize() function call. The aceInitialize() function definition has complete details about this mode of operation. Advanced mode gives the user access to some advanced functions which are typically not needed if using this library to program an Enhanced Mini-ACE card product from DDC.

DLL HIGH-LEVEL FUNCTION DEFINITIONS

This chapter contains high-level DLL functions. These high-level functions are defined as common instruction calls that will be utilized for board operations.

A detailed description of each DLL function, containing information about the routine's functionality, prototype, a formal parameter list, possible errors encountered, return codes, and example code will be contained in this section.

The include file described in each of the function sections is the file that contains the function prototype. It should be noted that when creating a program, only the STDEMACE.H file should be included. All other include files will be accessed through the inclusion of this file.

The macros specified are defined in the appropriate header files. In many cases the defined macro will take on different values based on the operating system or compiler. One of the macros that is dependent upon the operating system and the compiler used is the _DECL element of the prototypes. This is not something that the user will generally change, but it should be noted that this is not a fixed value for all systems.

It is wise practice to check return values for errors. If an error exists, then an appropriate action should be taken. The error condition should never be ignored as all operations to follow may not have a properly initialized state.

All of the VxWorks and DOS operating system specific functions are located after all the generic functions.

The following section is grouped by functional type (General, BC, RT, RTMT, MT, VxWorks, and DOS).

Table 3. Functional Grouping

Functional Group	Page
General Functions Listing	Page 40
BC Functions Listing	Page 127
RT Functions Listing	Page 334
RTMT Functions Listing	Page 443
MT Functions Listing	Page 475
VxWorks Functions Listing	Page 534
DOS Functions Listing	Page 549

General Functions

Table 4. General Functions Listing

aceCmdWordCreate

This function will create a command word from the input parts.

PROTOTYPE

```
#include "MsgOp.h"

S16BIT _DECL aceCmdWordCreate(U16BIT *pCmdWrd,
                              U16BIT wRT,
                              U16BIT wTR,
                              U16BIT wSA,
                              U16BIT wWC);
```

STATE

Not Applicable

MODE

Not Applicable

PARAMETERS

pCmdWrd
(output parameter)
Pointer to an U16BIT word that will contain the value of the command word

wRT
(input parameter)
Remote Terminal Address
Valid values:
0 - 31

wTR
(input parameter)
Transmit/Receive bit
Valid values:
0 – 1

wSA
(input parameter)
Sub-Address
Valid values:
0 – 31

wWC
(input parameter)
Data Word Count/Mode Code
Valid values:
0 – 31

aceCmdWordCreate (continued)

DESCRIPTION

This function creates a command word when given its individual parts.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_BUF	The pCmdWrd pointer is Null

EXAMPLE

```
U16BIT wRT, wTR, wSA, wWC;
U16BIT *pCmdWrd;
pRT = 5;
pTR = ACE_TX_CMD;
pSA = 10;
pWC = 4;

nResult = aceCmdWordCreate(pCmdWrd, wRT, wTR, wSA, wWC);

if(nResult)
{
      printf("Error in aceCmdWordCreate() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceCmdWordParse()

aceCmdWordParse

This function parses the given command word into its individual parts.

PROTOTYPE

```
#include "MsgOp.h"


S16BIT _DECL aceCmdWordParse(U16BIT wCmdWrd,
                             U16BIT *pRT,
                             U16BIT *pTR,
                             U16BIT *pSA,
                             U16BIT *pWC);
```

STATE

Not Applicable

MODE

Not Applicable

PARAMETERS

wCmdWrd	(input parameter) An U16BIT command word
pRT	(output parameter) Pointer to a value that will contain the Remote Terminal Address part of the command word
pTR	(output parameter) Pointer to a value that will contain the Transmit/Receive bit part of the command word
pSA	(output parameter) Pointer to a value that will contain the Sub-Address part of the command word
pWC	(output parameter) Pointer to a value that will contain the Word Count part of the command word

DESCRIPTION

This function parses the given command and outputs the individual parts of the command word into the output parameter pointers specified by the user.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_BUF	The pRT, pTR, pSA, and/or pWC output parameter specified by the user is/are Null

aceCmdWordParse (continued)

EXAMPLE

```
U16BIT wCmdWrd, pRT, pTR, pSA, pWC;
WCmdWrd = 0x0821;
S16BIT nResult = 0;

nResult = aceCmdWordParse(wCmdWrd, &pRT, &pTR, &pSA, &pWC);

if(nResult)
{
      printf("Error in aceCmdWordParse() function \n");
      PrintOutError(nResult);
      return;
}

else
{
      printf("RT Address = %d, T/R bit = %d", pRT, pTR,);
      printf("Sub-Address = %d, WordCount = %d\n", pSA, pWC);
 }
```

SEE ALSO

aceCmdWordCreate()

aceDioCtl

This function defines or determines the state of one of the discrete IO's on your PC/104 card.

PROTOTYPE

```
#include "DioOp.h"

S16BIT _DECL aceDioCtl (S16BIT DevNum,
                        U8BIT bPort,
                        U16BIT *wCmd);
```

STATE

Ready, Run

MODE

Not Applicable

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

bPort (input parameter)
The port you are accessing
Valid values:
DIO_1 to DIO_8

wCmd (input/output parameter)
Defines the command for this function
Valid values:

DIO_OFF
Turns the state of this port off

DIO_ON
Turns the state of this port on

DIO_READ
Reads the state of this port and outputs the state information to the wCmd parameter

aceDioCtl (continued)

DESCRIPTION

This function defines the state of one of the discrete IO's on your PC/104 card based on the value of the wCmd input parameter. The function can optionally read the state and output the value to the wCmd parameter if the user inputs DIO_READ to the wCmd parameter as an input. This function can only be used with the BU-65568CX and BU-65567CX series PC/104 cards.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_CARD	The device is not a BU-65567CX or BU-65568CX card
ACE_ERR_PARAMETER	The bPort input parameter does not contain a valid value or the wCmd input parameter does not contain a valid value

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT wCmd;
wCmd=DIO_OFF;

nResult = aceDioCtl(DevNum,
                    DIO_1,
                    &wCmd);

if(nResult)
{
      printf("Error in aceDioCtl() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceDioCtlBits()
aceDioDir()
aceDioDirBits()

aceDioCtlBits

This function defines or determines the state of all of the discrete IO's on your PC/104 card.

PROTOTYPE

```
#include "DioOp.h"

S16BIT _DECL aceDioCtlBits (S16BIT DevNum,
                            U8BIT *bState,
                            U8BIT bMask);
```

STATE

Ready, Run

MODE

Not Applicable

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

bState (input/output parameter)
Defines the state of each port
Valid values:
00000000 – 11111111(binary)
bit 0=0 => port 1 is off
bit 0=1 => port 1 is on
bit 1=0 => port 2 is off
bit 1=1 => port 2 is on
•
•
bit 7=0 => port 8 is off
bit 7=1 => port 8 is on

aceDioCtlBits (continued)

bMask (input parameter)
Enables or disables writing to the port
Valid values:
00000000 – 11111111(binary)
bit 0=1 => Write to port 1 enabled
bit 0=0 => Write to port 1 disabled
bit 1=1 => Write to port 2 enabled
bit 1=0 => Write to port 2 disabled
•
•
bit 7=1 => Write to port 8 enabled
bit 7=0 => Write to port 8 disabled
0x00
All writes disabled, read the status register and return contents in bState parameter

DESCRIPTION

This defines the states of some or all of the discrete IO's on your PC/104 card based on the value of the bState input parameter and the bMask input parameter. The function can optionally read the state and output the value to the bState parameter if the user inputs 0x00 to the bMask parameter as an input. This function can only be used with the BU-65568CX and BU-65567CX series PC/104 cards.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_CARD	The device is not a BU-65567CX or BU-65568CX card

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U8BIT bState;
U8BIT bMask;

bState=0x01;  //set port 1 active, all else inactive
bMask=0xFF;  //write to all ports

nResult = aceDioCtlBits(DevNum,
                        &bState,
                        bMask);
```

aceDioCtlBits (continued)

```
if(nResult)
{
      printf("Error in aceDioCtlBits() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceDioCtl () aceDioDir()
aceDioDirBits()

aceDioDir

This function defines or determines the direction of one of the discrete IO's on your PC/104 card.

PROTOTYPE

```
#include "DioOp.h"

S16BIT _DECL aceDioDir (S16BIT DevNum,
                        U8BIT bPort,
                        U8BIT *wCmd);
```

STATE

Ready, Run

MODE

Not Applicable

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

bPort (input parameter)
The port you are accessing
Valid values:
DIO_1 to DIO_8

wCmd (input/output parameter)
Defines the command for this function
Valid values:

DIO_IN
Makes this port an input

DIO_OUT
Makes this port an output

DIO_READ
Reads the state of this port and outputs the state information to the wCmd parameter

aceDioDir (continued)

DESCRIPTION

This function defines the direction of one of the discrete IO's on your PC/104 card. The function can also determine the direction of one of the discrete IO's on your PC/104 card if you command it to read in the wCmd input parameter. This function can only be used with the BU-65568CX and BU-65567CX series PC/104 cards.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_CARD	The device is not a BU-65567CX or BU-65568CX card
ACE_ERR_PARAMETER	The bPort input parameter does not contain a valid value or the wCmd input parameter does not contain a valid value

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT wCmd;
wCmd=DIO_IN;

nResult = aceDioDir(DevNum,
                    DIO_1,
                    &wCmd);

if(nResult)
{
      printf("Error in aceDioDir() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceDioCtl()
aceDioCtlBits()
aceDioDirBits()

aceDioDirBits

This function sets the direction of all IO ports on a card. A mask field is provided to select individual ports to operate on.

PROTOTYPE

```
#include "DioOp.h"

S16BIT _DECL aceDioDirBits (S16BIT DevNum,
                            U8BIT *bInOut,
                            U8BIT bMask);
```

STATE

Ready, Run

MODE

Not Applicable

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

bInOut (input/output parameter)
Defines the state of each port
Valid values:
00000000 – 11111111(binary)
bit 0=0 => port 1 is input
bit 0=1 => port 1 is output
bit 1=0 => port 2 is input
bit 1=1 => port 2 is output
•
•
bit 7=0 => port 8 is input
bit 7=1 => port 8 is output

bMask (input parameter)
Enables or disables writing to the port
Valid values:
00000000 – 11111111(binary)
bit 0=1 => Write to port 1 enabled
bit 0=0 => Write to port 1 disabled
bit 1=1 => Write to port 2 enabled
bit 1=0 => Write to port 2 disabled

aceDioDirBits (continued)

•
•
bit 7=1 => Write to port 8 enabled
bit 7=0 => Write to port 8 disabled

0x00
All writes disabled, read the status register and return contents in bInOut parameter

DESCRIPTION

This defines the direction of some or all of the discrete IO's on your PC/104 card based on the value of the bInOut input parameter and the bMask input parameter. The function can optionally read the state and output the value to the bInOut parameter if the user inputs 0x00 to the bMask parameter as an input. This function can only be used with the BU-65568CX and BU-65567CX series PC/104 cards.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_CARD	The device is not a BU-65567CX or BU-65568CX card

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U8BIT bInOut;
U8BIT bMask;

bInOut=0x01;  //set port 1 as output, all else inputs
bMask=0xFF;  //write to all ports

nResult = aceDioDirBits(DevNum,
                        &bInOut,
                        bMask);

if(nResult)
{
      printf("Error in aceDioDirBits() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceDioCtl ()
aceDioDir ()
aceDioCtlBits()

aceErrorStr

This function will pass general error information back to the user.

PROTOTYPE

```
#include "errordef.h"

S16BIT _DECL aceErrorStr(S16BIT nError,
                         char *pBuffer,
                         U16BIT wBufSize);
```

STATE

Not Applicable

MODE

Not Applicable

PARAMETERS

nError	(input parameter) The error number to return general information about
pBuffer	(output parameter) A pointer to a character buffer for the returned text string
wBufSize	(input parameter) Size of character buffer Valid values: >=80

DESCRIPTION

This function is used to pass an error information string back to the user. The string is passed using a character buffer. The user must pass in the error number and the size of the buffer.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_BUF	An invalid pointer to the character buffer was input by the user or the specified character buffer size is too small

EXAMPLE

```
S16BIT DevNum = 0;
char Buffer[80];
U16BIT wMemAddr = 0x03;
U16BIT wValue = 0xFFFF;
S16BIT nResult = 0;

nResult = aceMemWrite(DevNum, wMemAddr, wValue);
```

aceErrorStr (continued)

```
if(nResult)
{
      aceErrorStr(nResult, Buffer, 80);
      printf("RTL Function Failure-> %s.\n",pBuffer);
}
```

SEE ALSO

None

aceFree

Frees all resources used by the hardware based on the type of access used.

PROTOTYPE

```
#include "config.h"

S16BIT _DECL aceFree(S16BIT DevNum);
```

STATE

Run, Ready

MODE

Not Applicable

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

DESCRIPTION

This routine must be issued when device operation is complete. It will free all resources used by the device so that they are available for other programs or devices to use. After this function has successfully completed, the device will be in a Reset state.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_OS	This operating system is not one of the following supported types: DOS, Windows 9x/NT/2000/XP, Linux, or VxWorks

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceFree(DevNum);
if(nResult)
{
      //an error has occurred so notify the user
      printf("Error in aceFree() function \n");
      PrintOutError(nResult);
      return;
}
```

aceFree (continued)

SEE ALSO

aceInitialize()

aceGetBSWErrString

This function will return the block status word error string.

PROTOTYPE

```
#include "MsgOp.h"

char* _DECL aceGetBSWErrString(U16BIT wMode,
                               U16BIT wBlkSts);
```

STATE

Not Applicable

MODE

Not Applicable

PARAMETERS

wMode (input parameter)
Operating mode of EMA
Valid values:
ACE_MODE_BC
ACE_MODE_RT
ACE_MODE_MT

wBlkSts (input parameter)
The block status word (BSW) of the message

DESCRIPTION

This function returns a pointer to a character containing the given block status word error string.

RETURN VALUE

Char * If (wMode == ACE_MODE_BC || ACE_MODE_RT || ACE_MODE_MT):

"INVWD"
This indicates that a RT responded with one or more words containing one or more of the following error types: sync field error, Manchester encoding error, parity error, and/or bit count error

"INSYN"
An RT responded with a Data sync in a Status Word and/or a Command/Status sync in a Data Word

"WDCNT"
The responding RT did not transmit the correct number of data words

aceGetBSWErrString (continued)

"NOGAP"

The RT address field of a responding RT does not match the RT address in the Command Word and/or bit 8 of Configuration Register #5 at memory location 0x09 is set to logic 1 and a responding RT responds with a response time of less than 4 µs as per the MIL-STD-1553B standard.

"LPTST"

A loopback test is performed on the transmitted portion of every message in BC mode. A validity check is performed on the received version of every word transmitted by the Enhanced Mini-ACE BC. In addition, a bit-by-bit comparison is performed on the last word transmitted by the BC for each message. If either the received version of any transmitted word is invalid (sync, encoding, bit count, and/or parity error) and/or the received version of the last word transmitted by the Enhanced Mini-ACE BC does not match the transmitted version of this word, this error will occur.

"NORES"

This indicates that an RT has either not responded or has responded later than the BC No Response Timeout time. The Enhanced Mini-ACE 's No Response Timeout Time is defined as per the MIL-STD-1553B standard as the time from the mid-bit crossing of the parity bit to the mid-sync crossing of the RT Status Word. In the ENHANCED MODE DISABLE, the value of the BC Response Timeout is 17.5 to 19.5 µs. If ENHANCED MODE ENABLED is logic 1, the value of the No Response Timeout value is programmable from among the nominal values 18.5, 22.5, 50.5, and 130 µs (±1 µs) by means of bits 10 and 9 of Configuration Register #5 at memory location 0x09. The library sets ENHANCED MODE ENABLE to a value of 1 by default.

"FORMT"

This indicates that the received portion of a message contained one or more violations of the 1553 message validation criteria (sync, encoding, parity, bit count, word count, etc.), or the RT's status word received from a responding RT contained an incorrect RT address field.

If (wMode == ACE_MODE_RT):

aceGetBSWErrString (continued)

"ILCMD"

This indicates that the message has been illegalized. A message is illegalized if (ENHANCE MODE ENABLE, bit 15 of Configuration Register #3 is logic "0" or ILLEGALIZATION DISABLE, bit 7 of Configuration Register #3 is logic "0") and the appropriate bit for the respective Bcst/Tx/Rx-Subaddress-Word Count/Mode Code combination is set in the illegalization table, address locations 0300-03FF in the shared RAM. The library will legalize a message when the user calls the aceRTMsgLegalityEnable() function or the aceRTDataBlkMapToSA() function.

If (wMode==ACE_MODE_RT || ACE_MODE_MT)

"RTRTG"

Indicates the BC (or transmitting RT in an RT-to-RT transfer) transmitted one or more words containing one or more of the following error types: sync field error, Manchester encoding error, parity error, and/or bit count error.

"RTRTC"

If the Enhanced Mini-ACE is the receiving RT for an RT-to-RT transfer, this indicates one or more of the following error conditions in the transmit Command Word: (1) T/R* bit = logic "0"; (2)subaddress = 00000 or 11111; and/or (3) same RT address field as the receive.

"CMDER"

Indicates a received command word is not defined in accordance with the MIL-STD-1553B spec.

EXAMPLE

```
//get a decoded message from the EMA and display the error code in the
//BSW

aceBCGetMsgFromIDDecoded(DevNum,MSG1,&pMsg,TRUE);

// Display Error information

if(pMsg->wBlkSts & 0x170f)
{
      printf("\n ERROR: %s",
      aceGetBSWErrString(ACE_MODE_BC,pMsg->wBlkSts);
}
```

SEE ALSO

None

aceGetCoreVersion

This function will get the core version of the Enhanced Mini ACE software package.

PROTOTYPE

```
#include "config.h"

U16BIT _DECL aceGetCoreVersion(void);
```

STATE

Not Applicable

MODE

Not Applicable

PARAMETERS

None

DESCRIPTION

This function returns an unsigned 16-bit word containing the version information of the core library. The response contains a version number X.Y.Z. The core version is the part of the Enhanced Mini-ACE runtime library that is common to all platforms and operating systems.

The high byte contains the major version and the low byte contains the minor version to 2 decimal places.
(Example: 0x0101-> Version 1.01, 0x0244-> Version 2.44)

RETURN VALUE

EMACE_CORE_VERSION	U16BIT value that contains the version number

EXAMPLE

```
wLibVer =aceGetCoreVersion();

//temp contains MSB (major version)
temp= wLibVer >>8;
//temp2 contains Most Significant nibble (minor version)
temp2=wLibVer&0x00F0;
temp2=temp2>>4;
//wLibVer now contains the minor version Least Sig. nibble
wLibVer = wLibVer &0x000F;
printf("EMA Software package version is %x", temp);
printf(".%x", temp2);
printf(".%x", wLibVer);
```

aceGetCoreVersion (continued)

SEE ALSO

None

aceGetLibVersion

This function will get the version of the library.

PROTOTYPE

#include “config.h”

U16BIT _DECL aceGetLibVersion(void);

STATE

Not Applicable

MODE

Not Applicable

PARAMETERS

None

DESCRIPTION

This function returns an unsigned 16-bit word containing the version information of this library. The response contains a version number X.Y.Z. Any software where the third descriptor “Z” is a number between 1 and 9 is interim release software. Interim software is the latest version software from DDC’s engineering department that has not been fully validated and officially released. If “Z” is 0, this indicates that this software has been fully validated by DDC’s Software Quality Assurance Department and is officially released.

The high byte contains the major version (X) and the low byte contains the minor version (Y.Z) split by each nibble.
(Example: 0x0101-> Version 1.0.1, 0x0244-> Version 2.4.4)

RETURN VALUE

EMACE_RTL_VERSION	U16BIT value that contains the version number

EXAMPLE

```
wLibVer =aceGetLibVersion();

//temp contains MSB (major version)
temp= wLibVer >>8;
//temp2    contains    Most    Significant    nibble    (minor    version)
temp2=retVal&0x00F0;
temp2=temp2>>4;
//wLibVer now contains the minor version Least Sig. nibble
wLibVer = wLibVer &0x000F;
printf("EMA Run Time Library version is %x", temp);
printf(".%x", temp2);
printf(".%x", wLibVer);
```

SEE ALSO

None

aceGetMemRegInfo

This function will get the locations of memory and registers.

PROTOTYPE

```
#include "config.h"

S16BIT _DECL aceGetMemRegInfo(S16BIT DevNum,
                              U32BIT *pRegAddr,
                              U32BIT *pMemAddr);
```

STATE

Not Applicable

MODE

Not Applicable

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
dwRegAddr	(output parameter) This value is a pointer to the mapped base address of EMA register space
dwMemAddr	(output parameter) This value is a pointer to the mapped base address of EMA memory space

DESCRIPTION

This function will get the locations of registers and memory and put the location in the dwRegAddr parameter and the dwMemAddr parameter respectively. These are the addresses that the operating system or user has assigned for the base memory window. These are not valid memory locations to pass into the aceInitialize() function in ACE_MODE_USR.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully or an invalid device number was input.

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U32BIT *pRegAddr;
U32BIT *pMemAddr;

nResult = aceGetMemRegInfo(DevNum, &pRegAddr, &pMemAddr);
```

aceGetMemRegInfo (continued)

```
if (DevNum <0 || DevNum >31)
{
      printf("An invalid device number has been input \n");
}

else
{
      printf("Function completed successfully \n");
}
```

SEE ALSO

aceRegRead()
aceRegWrite()
aceMemRead()
aceMemWrite()
aceISQRead()

aceGetMsgTypeString

This function will return the given message type string.

PROTOTYPE

```
#include "MsgOp.h"

char* _DECL aceGetMsgTypeString(U16BIT wMsgType);
```

STATE

Not Applicable

MODE

Not Applicable

PARAMETERS

wMsgType (input parameter)
An U16BIT word representation of the message type
Valid values:

- ACE_MSG_BCTORT (0)
- ACE_MSG_RTTOBC (1)
- ACE_MSG_ RTTORT (2)
- ACE_MSG_MODENODATA (5)
- ACE_MSG_MODEDATARX (6)
- ACE_MSG_MODEDATATX (7)
- ACE_MSG_BRDCST (8)
- ACE_MSG_BRDCSTRTTORT (10)
- ACE_MSG_BRDCSTMODENODATA (13)
- ACE_MSG_BRDCSTMODEDATA (14)
- ACE_MSG_ ACE_MSG_INVALID (15)

DESCRIPTION

This function returns a pointer to a character buffer that will contain the given message type string.

RETURN VALUE

Char *

- "BC to RT"
- "RT to BC"
- "RT to RT"
- "Invalid"
- "Mode No Data"
- "Mode Rx Data"
- "Mode Tx Data"
- "Bcst"
- "Bcst RT to RT"
- "Bcst Mode No Data"
- "Bcst Mode Data"

aceGetMsgTypeString (continued)

EXAMPLE

```
Char *msg_type;
msg_type = aceGetMsgTypeString(ACE_MSG_BCTORT);

printf("Message type %d = %s\n" ACE_MSG_BCTORT, msg_type);
```

SEE ALSO

None

aceGetTimeTagValue

This function retrieves the time tag value.

PROTOTYPE

#include “config.h”

S16BIT _DECL aceGetTimeTagValue(S16BIT DevNum,
U16BIT *wTTValue);

STATE

Reset, Ready, Run

MODE

Not Applicable

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31	
wTTValue	(output parameter) Time Tag Value read from the register Valid values: 0x0000 – 0xFFFF	

DESCRIPTION

This function gets the value of the Time Tag Register in any state.

RETURN VALUE

ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_SUCCESS	The function completed successfully

EXAMPLE

```
//To get the time tag value.
  S16BIT DevNum = 0;
  S16BIT nResult = 0;
  U16BIT \pwTTValue;

  nResult = aceGetTimeTagValue (DevNum, &pwTTValue);
  if(nResult)
  {
  printf(“Error      in      aceGetTimeTagValue()      function      \n”);
  PrintOutError(nResult);
        return;
  }
```

aceGetTimeTagValue (continued)

SEE ALSO

None

aceInitialize

This function initializes hardware resources such as memory and register space for a particular mode of operation (BC/RT/MT/RTMT/TEST).

PROTOTYPE

```
#include "config.h"

S16BIT _DECL aceInitialize(S16BIT DevNum,
                           U16BIT wAccess,
                           U16BIT wMode,
                           U32BIT dwMemWrdSize,
                           U32BIT dwRegAddr,
                           U32BIT dwMemAddr);
```

STATE

Reset, Ready

MODE

Set by wMode parameter after this function is called.

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 - 31

wAccess (input parameter)
This parameter specifies the type of access to be used by the device
Valid values:
ACE_ACCESS_CARD
ACE_ACCESS_SIM
ACE_ACCESS_USR

wMode (input parameter)
This parameter specifies the mode of operation that this device is to be initialized to.
Valid Values:
ACE_MODE_TEST
Sets up the device to run in test mode.
ACE_MODE_BC
Sets up the device to run as a bus controller
ACE_MODE_RT
Sets up the device to run as a remote terminal
ACE_MODE_MT
Sets up the device to run as a monitor
ACE_MODE_RTMT
Sets up the device to run in combination remote terminal and monitor mode

aceInitialize (continued)

The above inputs can be logically or'ed with the following values:

ACE_NO_TT_RESET
Results in the time tag register never getting reset

ACE_ADVANCED_MODE
Advanced mode gives the user access to some advanced functions which are typically not needed if using this library to program an Enhanced Mini-ACE card from DDC. In this mode of operation you have access to the following functions:

aceRegRead
aceRegWrite
aceMemRead
aceMemWrite
aceSetIrqConfig
aceSetClockFreq
aceSetAddressMode
aceSetDecoderconfig

wMemWrdSize (input parameter)
This parameter specifies the amount of ACE memory to be allocated for use when using the ACE_ACCESS_SIM or the ACE_ACCESS_USR access type, otherwise the value is not used
Valid Values:
4K - 64K

dwRegAddr (input parameter)
This parameter specifies the registry address to be used by the device when the access type is selected to be ACE_ACCESS_USR

dwMemAddr (input parameter)
base memory address for the device to use if the access type is selected to be ACE_ACCESS_USR

DESCRIPTION

This function initializes hardware resources such as memory and register space for a particular mode of operation. The user can select the mode of operation with this function and the wMode parameter. The Enhanced Mini-ACE can be programmed using 3 methods of access. Simulated memory allows the user to allocate a 4K to 64K chunk of host memory by using the malloc command and manipulate the memory as if it were hardware memory on the device until an image file is created. Simulated memory can be selected by inputting ACE_ACCESS_SIM into the wAccess parameter of this function. Device memory should be used if you are using a DDC card. This memory allows the memory on the device to be accessed through a Win32 device driver. This memory can be selected by inputting ACE_ACCESS_CARD into the wAccess parameter of this function. User memory allows memory and register addresses to be passed directly to the library. User memory can be selected by inputting ACE_ACCESS_USR into the wAccess parameter of this function. Any access method can produce a binary image file but only the device memory access method (ACE_ACCESS_CARD) and the user memory access method (ACE_ACCESS_USR) can actually run the binary image file.

aceInitialize (continued)

RETURN VALUE

ACE_ERR_SUCCESS	The device has been initialized successfully
ACE_ERR_INVALID_DEVNUM	An incorrect device number was input
ACE_ERR_INVALID_ACCESS	The access type specified is invalid
ACE_ERR_INVALID_MODE	The mode of operation selected is invalid
ACE_ERR_INVALID_MEMSIZE	The memory size specified in wMemWrdSize is not valid
ACE_ERR_INVALID_ADDRESS	One or all of the addresses specified in dwRegAddr and dwMemAddr are invalid
ACE_ERR_INVALID_MALLOC	The proper amount of memory required for the Win32 information structure definition and initialization failed to be allocated
ACE_ERR_REG_ACCESS	Failed to open the Windows registry to get the device ID and logical device number
ACE_ERR_INVALID_CARD	Card type is not recognized as a supported card
ACE_ERR_INVALID_OS	This operating system is not one of the following supported types: DOS, Windows 9x/NT/2000/XP, Linux, or VxWorks
ACE_ERR_DRIVER_OPEN	Failed to open the device driver for this card

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nresult = 0;

nResult = aceInitialize(DevNum,ACE_ACCESS_CARD,ACE_MODE_BC,0,0,0);
if(nResult)
{
      //an error has occurred so notify the user
      printf("Error in aceInitialize() function \n");
      PrintOutError(nResult);
      return;
            }
```

SEE ALSO

aceFree()

aceInt80Enable

This function is used to turn on / off interrupts on a BU-65580 PC./104 EBR device.

PROTOTYPE

```
#include "config.h"


S16BIT _DECL aceInt80Enable(S16BIT DevNum,
                            U16BIT bEnable) ;
```

STATE

Ready

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
bEnable	(input parameter) Switch to enable / disable interrupts on the BU-65580 Valid values: FALSE This will disable interrupts TRUE This will enable interrupts

DESCRIPTION

This function is used to turn interrupts on after the card is properly initialized. This function is called internally when the RTL has determined the proper initialization state has been reached. The aceInt80Enable function may also be called by the user if they wish to disable / enable a BU-65580 device's interrupts on their own.

This function is only used with the BU-6558xCx series of cards.

RETURN VALUE

ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_SUCCESS	The function completed successfully

EXAMPLE

```
//Enable BU-65580 device #0's interrupt
  nResult = aceInt80Enable (DevNum, TRUE);
```

aceInt80Enable (continued)

SEE ALSO

None

aceISQClear

This function clears the ISQ.

PROTOTYPE

#include “config.h”

S16BIT _DECL aceISQRead(S16BIT DevNum);

STATE

Ready, Run

MODE

MT, RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function clears all values on the ISQ and resets the pointer register to the starting position queue. The return value will report the success of the operation.

RETURN VALUE

ACE_ERR_ISQ_DISABLED	The Interrupt Status Queue is disabled and must be enabled by calling the aceISQEnable() function before calling this function
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_MODE	The device is not in MT, RT, or RTMT mode
ACE_ERR_INVALID_STATE	The device is not in the Ready or Run state
ACE_ERR_SUCCESS	The function completed successfully

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceISQClear(DevNum);

if(nResult)
{
      printf(“Error in aceISQClear() function \n”);
      aceErrorStr(nResult, Buffer, 80);
      printf("RTL Function Failure-> %s.\n",pBuffer);
}
```

aceISQClear (continued)

SEE ALSO

aceISQEnable()
aceISQRead()
Appendix B

aceISQEnable

This function allows the user to enable or disable the Interrupt Status Queue.

PROTOTYPE

```
#include "config.h"

S16BIT _DECL aceISQEnable(S16BIT DevNum,
                          U16BIT bEnable);
```

STATE

Ready

MODE

RT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

bEnable (input parameter)
FALSE
This will disable the interrupt status queue

TRUE
This will enable the interrupt status queue

DESCRIPTION

This function enables the interrupt status queue for the user to utilize in RT or RTMT mode of operation. The interrupt status queue can also be disabled with this function but is disabled by default so there is no need to call this function if you do not wish to use the interrupt status queue.

aceISQEnable (continued)

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_MODE	The device is not in RT, or RTMT mode
ACE_ERR_INVALID_STATE	The device is not in the Ready state
ACE_WRN_RT_CFG_INVALID	Operation of your device may be problematic because one or more of the following interrupts have been enabled: Time Tag Rollover, RT Address Parity Error, and/or Ram Parity Error along with the Interrupt Status Queue. See Appendix B for details

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceISQEnable(DevNum, TRUE);

if (nResult)
{
      PrintOutError(nResult);
      return;
}

else
{
      printf("The function completed successfully \n");
}
```

SEE ALSO

aceISQClear()   aceISQRead()
Appendix B

aceISQRead

This function reads the next unread entry off of the interrupt status queue.

PROTOTYPE

```
#include "config.h"


S16BIT _DECL aceISQRead(S16BIT DevNum,
                        ISQENTRY *pISQEntry);
```

STATE

Ready, Run

MODE

MT, RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
pISQEntry	(output parameter) Pointer to the entry read off of the interrupt status queue

DESCRIPTION

This function reads the next unread entry off of the interrupt status queue. The return value will report the success of the operation.

RETURN VALUE

ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_MODE	The device is not in MT, RT, or RTMT mode
ACE_ERR_INVALID_STATE	The device is not in the Ready or Run state
ACE_ERR_PARAMETER	The pISQEntry parameter is NULL
ACE_ERR_ISQ_DISABLED	The Interrupt Status Queue is disabled and must be enabled by calling the aceISQEnable() function before calling this function
‘0’(zero)	No entries were read off of the interrupt status queue
‘1’(one)	One entry was read off of the interrupt status queue
‘2’(two)	One entry was read off of the interrupt status queue with an overrun condition

aceISQRead (continued)

EXAMPLE

```
S16BIT DevNum = 0;
ISQENTRY *pISQEntry;
S16BIT nResult = 0;

nResult = aceISQRead(DevNum, &pISQEntry);

if (nResult < 0)
{
      PrintOutError(nResult);
      return;
}

else if (nResult == 0)
{
      printf(“No entries were read from the ISQ \n”);
}

else if (nResult == 1)
{
      printf(“One entry was successfully read from the ISQ \n”);
}

else if (nResult == 2)
{
      printf(“One entry was read from the ISQ with overrun cond. \n”);
}

else
{
      printf(“An undefined error occurred \n”);
}
```

SEE ALSO

aceISQEnable()
aceISQClear()
Appendix B

aceMemRead

This function reads memory.

PROTOTYPE

```
#include "config.h"

U16BIT _DECL aceMemRead(S16BIT DevNum,
                        U16BIT wMemAddr);
```

STATE

Ready if not in advanced mode of operation
Any state if in advanced mode of operation

MODE

Advanced plus one of the following BC, RT, MT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wMemAddr	This parameter specifies the memory location to be read

DESCRIPTION

This function reads the memory location specified by the wMemAddr input parameter. This function is available for the advanced user that would like to know the contents of a hardware memory location and can only be used while in the ready state if the Enhanced Mini-ACE runtime library is not in advanced mode of operation. If the library is in advanced mode of operation then this library function can be called in any state.

RETURN VALUE

0x0000 – 0xFFFF	The contents of the memory location that was read
0	An invalid device number, and/or the device is not in a Ready state, and/or the wMemAddr is invalid
ACE_ERR_NOT_SUPPORTED	This function can only be used in Advanced mode of operation unless you are in a Ready state

EXAMPLE

```
S16BIT DevNum = 0;
U16BIT wMemAddr = 0x03;
U16BIT wMemValue = 0;

wMemValue = aceMemRead(DevNum, wMemAddr);
```

aceMemRead (continued)

```
printf(“Memory location %x \n”, wMemAddr);

if (wMemValue != 0)
{
      printf(“contains the following value: %x”, wMemValue);
}

else
{
      printf(“The function returned an error or memory contained 0’s”);
}
```

SEE ALSO

aceRegRead()
aceMemWrite()
aceISQRead()
aceRegWrite()
aceGetMemRegInfo()

aceMemWrite

This function writes to a specified memory location.

PROTOTYPE

```
#include "config.h"


S16BIT _DECL aceMemWrite(S16BIT DevNum,
                         U16BIT wMemAddr,
                         U16BIT wValue);
```

STATE

Ready if not in advanced mode of operation
Any state if in advanced mode of operation

MODE

Advanced plus one of the following BC, RT, MT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wMemAddr	(input parameter) This parameter specifies the memory location to write to
wValue	(input parameter) This is the value to be written to the memory location Valid values: 0x0000 – 0xFFFF

DESCRIPTION

This function writes a 16-bit value input by the user to a specified memory location on the device. This function is available for the advanced user that would like to write the contents of a hardware memory location and can only be used while in the ready state if the Enhanced Mini-ACE runtime library is not in advanced mode of operation. If the library is in advanced mode of operation then this library function can be called in any state. Caution must be taken as to not write invalid data to a memory address and corrupt the operation of the device.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in a Ready state and the function could not be completed

aceMemWrite (continued)

ACE_ERR_INVALID_ADDRESS	An invalid memory address was input to this function
ACE_ERR_NOT_SUPPORTED	This function can only be used in Advanced mode of operation unless you are in a Ready state

EXAMPLE

```
S16BIT DevNum = 0;
U16BIT wMemAddr = 0x02;
U16BIT wValue = 0xFFFF;
S16BIT nResult = 0;

nResult = aceMemWrite(DevNum, wMemAddr, wValue);

if(nResult)
{
printf("Error in aceMemWrite() function \n");  PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRegRead()
aceMemRead()
aceISQRead()
aceRegWrite()
aceGetMemRegInfo()

aceRegRead

This function reads a register on the device at the specified memory location.

PROTOTYPE

```
#include "config.h"

U16BIT _DECL aceRegRead(S16BIT DevNum,
                        U16BIT wRegAddr);
```

STATE

Ready if not in advanced mode of operation
Any state if in advanced mode of operation

MODE

Advanced plus one of the following BC, RT, MT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wRegAddr	(input parameter) This parameter specifies the register address to be read Valid values: 0 – 63

DESCRIPTION

This function reads a register on the device at the specified memory location. This function should be used when the user would like to know the contents of a specific Enhanced Mini-ACE register. This function is available for the advanced user that would like to know the contents of the hardware registers and can only be used while in the ready state if the Enhanced Mini-ACE runtime library is not in advanced mode of operation. If the library is in advanced mode of operation then this library function can be called in any state.

RETURN VALUE

0x0000 – 0xFFFF	The contents of the register that was read
0	An invalid device number, and/or the wRegAddr parameter is greater than 63, and/or the device is not in a Ready state
ACE_ERR_NOT_SUPPORTED	This function can only be used in Advanced mode of operation unless you are in a Ready state

aceRegRead (continued)

EXAMPLE

```
//The following example will read Configuration Reg. #2
S16BIT DevNum = 0;
U16BIT wRegContents = 0;
U16BIT wRegAddr = 0x02;

wRegContents = aceRegRead(DevNum, wRegAddr);

printf("Register %x \n", wRegAddr);

if (wRegContents!=0)
{
      printf("Contains the following value: %x \n, wRegContents);
}

else
{
      printf("The register is all zeros or an error occurred \n");
}
```

SEE ALSO

aceRegWrite()
aceMemWrite()
aceISQRead()
aceMemRead()
aceGetMemRegInfo()

aceRegWrite

This function will write to the specified register location.

PROTOTYPE

```
#include "config.h"


S16BIT _DECL aceRegWrite(S16BIT DevNum,
                         U16BIT wRegAddr,
                         U16BIT wValue);
```

STATE

Ready if not in advanced mode of operation
Any state if in advanced mode of operation

MODE

Advanced plus one of the following BC, RT, MT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wRegAddr	(input parameter) This parameter specifies the register to write to Valid values: 0 - 63
wValue	(input parameter) The value that should be written to the register Valid values: 0x0000 – 0xFFFF

DESCRIPTION

This function will write to the specified register location. This function should be used when data must be written to a specific Enhanced Mini-ACE register that the user is concerned with. This function is available for the advanced user that would like to write the contents of the hardware registers and can only be used while in the ready state if the Enhanced Mini-ACE runtime library is not in advanced mode of operation. If the library is in advanced mode of operation then this library function can be called in any state.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in a Ready state

aceRegWrite (continued)

ACE_ERR_INVALID_ADDRESS	An invalid register address was input
ACE_ERR_NOT_SUPPORTED	This function can only be used in Advanced mode of operation unless you are in a Ready state

EXAMPLE

```
S16BIT DevNum = 0;
U16BIT wRegAddr = 0x02;
U16BIT wValue = 0x0800;
S16BIT nResult = 0;

nResult = aceRegWrite(DevNum, wRegAddr, wValue);

if(nResult)
{
      printf("Error in aceRegWrite() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRegRead()
aceMemRead()
aceMemWrite()
aceGetMemRegInfo()
aceISQRead()

aceResetTimeTag

This function allows the user to reset the value of the Time Tag Register to 0x0000.

PROTOTYPE

#include “config.h”

S16BIT _DECL aceResetTimeTag (S16BIT DevNum);

STATE

Reset, Ready, Run

MODE

Not Applicable

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function accesses bit 3 of the Start/Reset Register, which causes the value of the Time Tag Register to reset to zero (0x0000).

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user

EXAMPLE

```
//To reset the time tag register to 0x0000.
  S16BIT DevNum = 0;
  S16BIT nResult = 0;

  nResult = aceResetTimeTag (DevNum);
  if(nResult)
  {
  printf(“Error      in      aceResetTimeTag()      function      \n”);
  PrintOutError(nResult);
        return;
  }
```

SEE ALSO

None

aceSetAddressMode

This function defines the word addressing based on the host computer architecture.

PROTOTYPE

```
include "config.h"


S16BIT _DECL aceSetAddressMode(S16BIT DevNum,
                               U16BIT wAddrMode);
```

STATE

Ready

MODE

Advanced plus one of the following BC, RT, MT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

wAddrMode (input parameter)
Defines hardware access to EMA
Valid values:
ACE_ADDRMODE_A0_A0
ACE_ADDRMODE_A1_A0
ACE_ADDRMODE_A2_A0

DESCRIPTION

This function sets the way in which hardware registers and memory will be addressed. It is important to note that with respect to the Enhanced Mini ACE's address bus all internal address mapping is word orientated rather than byte orientated. Most standard microprocessors are byte orientated. This difference must be taken into account and is handled internally in this library by calling this function. The default mode (ACE_ADDRMODE_A1_A0) assumes that the host's microprocessor is byte orientated so it will take two increments of the host's address to access the next word location on the Enhanced Mini ACE.

The valid values for wAddrMode are:

ACE_ADDRMODE_A0_A0 - Incrementing the host address by 1 will access the next word on the Mini-ACE.

ACE_ADDRMODE_A1_A0 - (Default) Incrementing the host address by 2 will access the next word on the Mini-ACE.

ACE_ADDRMODE_A2_A0 - Incrementing the host address by 4 will access the next word on the Mini-ACE.

aceSetAddressMode (continued)

For users running an Enhanced Mini-ACE device on a DDC card product there is no need to call this function. This function is only available in advanced mode of operations for those that are designing their own system or card.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_ADMODE	The wAddrMode input parameter contains an incorrect value
ACE_ERR_NOT_SUPPORTED	This function can only be used in Advanced mode of operation

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceSetAddressMode(DevNum,
                            ACE_ADDRMODE_A2_A0);
if(nResult)
{
      //an error has occurred so notify the user
      printf("Error in aceSetAddressMode() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

None

aceSetCANIsr

This function is used to turn on / off interrupts on a BU-65580 PC./104 EBR device.

PROTOTYPE

```
#include "config.h"


S16BIT _DECL aceSetCANIsr(S16BIT DevNum,
                          void(_DECL *funcCANIsr)
                           (S16BIT DevNum,U16BIT wIrqStatus));
```

STATE

Ready

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
funcCANIsr	(input parameter) Pointer to function handle for CAN interrupts.
funcExternalISR	(input parameter) This is the user designated ISR callback function written by the user. This function will be called when the device generates an interrupt.

DESCRIPTION

This function is used to assign a user interrupt handler to any received CANbus interrupt. This function is only used with the BU-6558xCx series of cards.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in a Ready state and the function could not be completed
ACE_ERR_PARAMETER	The designated funcCANIsr contains an invalid handle
ACE_ERR_INVALID_CARD	The device being assigned an ISR is not one of the BU-6558xCx series of cards

aceSetCANIsr (continued)

EXAMPLE

```
//The following code will assign a user ISR handler of “myISR” to any CAN
//interrupt
  S16BIT DevNum = 0;

  //user defined callback function (myISR) sample
  void _DECL myISR (S16BIT DevNum, U16BIT Status)
  {
        //ISR implementation user defined callback routine
        printf(“An interrupt has occurred \n”);
  }
```

SEE ALSO

None

aceSetClockFreq

This function sets the hardware clock input to the EMA.

PROTOTYPE

```
#include "config.h"

S16BIT _DECL aceSetClockFreq(S16BIT DevNum,
                             U16BIT wClockIn);
```

STATE

Ready

MODE

Advanced plus one of the following BC, RT, MT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wClockIn	(input parameter) Defines the clock frequency input Valid Values: ACE_CLOCK_16MHZ ACE_CLOCK_12MHZ ACE_CLOCK_20MHZ ACE_CLOCK_10MHZ

DESCRIPTION

This function sets the type of clock input the hardware should be expecting. The valid values for wClockIn are:

Legacy Mode and Enhanced Mini-ACE Mode
ACE_CLOCK_16MHZ -> Clock running at 16 MHz
ACE_CLOCK_12MHZ -> Clock running at 12 MHz
Enhanced Mini-ACE Mode Only
ACE_CLOCK_20MHZ -> Clock running at 20 MHz
ACE_CLOCK_10MHZ -> Clock running at 10 MHz

For card level products the default clock frequency is 16 MHz and is set up when you configure a BC, RT, or MT for operation. This clock should not be changed if you are using a card level product. To use this function you must be in advanced mode of operation.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function

aceSetClockFreq (continued)

ACE_ERR_INVALID_STATE	The device is in not in a Ready state and this function could not complete
ACE_ERR_CLOCKIN	The device is in legacy mode and you have selected the clock frequency to be either 10 MHz or 20 MHz
ACE_ERR_NOT_SUPPORTED	This function can only be used in Advanced mode of operation

EXAMPLE

```
//If the hardware is an EMA and it is a card, the clock supplies 16Mhz
//to the clock input.

  S16BIT DevNum = 0;
  S16BIT nResult = 0;

  nresult = aceSetClockFreq(DevNum, ACE_CLOCK_16MHz);

  if(nResult)
  {
        //an error has occurred so notify the user
        printf(“Error in aceSetClockFreq() function \n”);
        PrintOutError(nResult);
        return;
  }
```

SEE ALSO

None

aceSetDecoderConfig

This function allows the user to set the decoder configuration.

PROTOTYPE

```
#include "config.h"

S16BIT _DECL aceSetDecoderConfig(S16BIT DevNum,
                                 U16BIT wDoubleOrSingle,
                                 U16BIT wExpXingEnable);
```

STATE

Ready

MODE

Advanced plus one of the following BC, RT, MT, RTMT

PARAMETERS

DevNum
(input parameter)
Logical Device Number
Valid values:
0 – 31

wDoubleOrSingle
(input parameter)
This parameter indicates the use of the inputs to the Manchester II decoder. It can run as either a single-ended or double-ended input device. The Manchester II decoders for the standard BU-61740/3/5, BU-61840/3/5, BU-61860/4/5, BU-64743, BU-64843, BU-64863, BU-65743, BU-65843 and BU-65863 versions of the Enhanced Mini-ACE device are all configured for double-ended (MIL-STD-1553 receiver) type of inputs. For these products bit 14 of Config. Reg. #5 is read only and will always return a 0. A write to this bit by calling this function with the ACE_SINGLE_ENDED or ACE_DOUBLE_ENDED options will have no effect. All DDC cards use the ACE_DOUBLE_ENDED option.
Valid values:
ACE_SINGLE_ENDED
(bit 14 of Config. Reg. #5 is set to 1)

ACE_DOUBLE_ENDED (default)
(bit 14 of Config. Reg. #5 is set to 0)

wexpandedXingEnable
(input parameter)
The EMA can detect zero crossing of the bus signal using either one edge of the input clock or both edges of the input clock. If both edges are used, the accuracy of the detection increases.
Valid values:
ACE_DISABLE_EXPANDED_XING
(bit 11 of Config. Reg. #5 is set to 0)

aceSetDecoderConfig (continued)

This option will disable any type of legacy support for older generation ACE devices.

ACE_ENABLE_EXPANDED_XING (default)
(bit 11 of Config. Reg. #5 is set to 1)
This option will provide legacy compatibility to the previous ACE and Mini-ACE generations.

DESCRIPTION

This function allows the user to set the decoder configuration of the device to allow for different inputs. The Enhanced Mini-ACE library will call this function with ACE_DOUBLE_ENDED and ACE_ENABLE_EXPANDED_XING as the parameters by default when the user configures a BC, RT, or MT. This is the desired configuration for all of DDC's card level products. The user can change these options after the BC, RT, or MT has been configured with a call to this function. For all DDC card level products this function should not be called and is only available in advanced mode of operation. Some devices may not allow write access to set bit 14 of Configuration Register #5 for ACE_SINGLE_ENDED or ACE_DOUBLE_ENDED.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in a Ready state and the function could not be completed
ACE_ERR_NOT_SUPPORTED	This function can only be used in Advanced mode of operation

EXAMPLE

```
//If the EMA hardware is being used with an interface to a transceiver
//that uses singled ended I/O and the usage of the device will require
//greater accuracy in detecting the zero crossing of the bus signal, //the
following code could be used.

  S16BIT DevNum = 0;

  nResult = aceSetDecoderConfig(DevNum, ACE_SINGLE_ENDED,
                                ACE_ENABLE EXPANDED_XING);
  if(nResult)
  {
        printf("Error in aceSetDecoderConfig() function \n");
        PrintOutError(nResult);
        return;
  }
```

SEE ALSO

None

aceSetIrqConditions

This function enables the selected interrupts as specified by the settings of the interrupt mask.

PROTOTYPE

```
#include "config.h"

S16BIT _DECL aceSetIrqConditions(S16BIT DevNum,
                                 U16BIT bEnable,
                                 U32BIT dwIrqMask,
void(_DECL *funcExternalIsr)(S16BIT DevNum, U32BIT dwIrqStatus));
```

STATE

Ready

MODE

Not Applicable

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

bEnable (input parameter)
Enable or disable interrupts defined in interrupt mask Register 1 & 2. Interrupt Mask Register #1 is at memory location 0x00, and Interrupt Mask Register #2 is at memory location 0x1D.
Valid values:
FALSE (disable)
TRUE (enable)

dwIrqMask (input parameter)
Bit masks to select the interrupt register bits. Each #define correlates to a bit in one of the interrupt mask registers.
Valid Values:

ACE_IMR1_EOM
(bit 0 of Interrupt Mask Reg. #1 is set to 0 or 1)
This mask will result in an interrupt in BC, RT, and selective Monitor modes at the completion of every message.

ACE_IMR1_BC_STATUS_SET
(bit 1 of Interrupt Mask Reg. #1 is set to 0 or 1)

aceSetIrqConditions (continued)

This mask will result in an interrupt in BC mode when an RT status word is received with an incorrect RT address field or one of the 8 non-reserved status bits contain an unexpected bit value.

ACE_IMR1_RT_MODE_CODE
(bit 1 of Interrupt Mask Reg. #1 is set to 0 or 1)
This mask will result in a mode code interrupt in RT mode when an enabled mode code message is received.

ACE_IMR1_MT_PATTERN_TRIG
(bit 1 of Interrupt Mask Reg. #1 is set to 0 or 1)
This mask will result in a pattern trigger interrupt in the word Monitor mode if a valid command word that matches the bit pattern programmed in the Monitor Trigger Register at memory location 0x0D is received. The Monitor Trigger Register is set to a value of 0x0000 when the aceInitialize() function is called.

ACE_IMR1_FORMAT_ERR
(bit 2 of Interrupt Mask Reg. #1 is set to 0 or 1)
This mask will result in an interrupt if a loop test failure or a message error is encountered.

ACE_IMR1_BC_END_OF_FRM
(bit 3 of Interrupt Mask Reg. #1 is set to 0 or 1)
This mask will result in an interrupt in non-enhanced BC mode if an entire programmed BC message frame has been processed.

ACE_IMR1_BC_MSG_EOM
(bit 4 of Interrupt Mask Reg. #1 is set to 0 or 1)
This mask will result in an interrupt at the end of the current message as long as ACE_BCCTRL_EOM_IRQ is selected in one of the aceBCMsgCreate() functions to set bit 4 of the BC Control Word to 1 and CFG4_BC_ENH_CTRL_WORD is selected as an input parameter to the aceRegWrite() function to set bit 12 of Configuration Register #4 to 1. Bit 12 can be set by using the aceRegWrite() function. Caution must be taken as to not overwrite any of the other bit values that are already in Configuration Register #4 when writing to bit 12.

ACE_IMR1_RT_SUBADDR_EOM
(bit 4 of Interrupt Mask Reg. #1 is set to 0 or 1)
This mask will result in an interrupt at the end of the current message as long as ACE_RT_DBLK_EOM_IRQ is

aceSetIrqConditions (continued)

selected as an input parameter to the aceRTDataBlkMapToSA() function to set one of the Interrupt at End of Message bits (bit 4, 9, or 14) of the Subaddress Control Word to 1. Bit 4 is set to 1 for a Broadcast End of Message Interrupt. Bit 9 is set to 1 for a Receive End of Message Interrupt. Bit 14 is set to 1 for a Transmit End of Message Interrupt. For this interrupt to occur the device must also have Enhanced Memory Management enabled by setting bit 1 of Configuration Register #2 at memory location 0x02 to 1. This is done by default internally by the library when an RT is configured using the aceRTConfigure() function.

ACE_IMR1_RT_CIRCBUF_ROVER
(bit 5 of Interrupt Mask Reg. #1 is set to 0 or 1)
This mask will result in an interrupt if the circular buffer has rolled over. For this interrupt to occur the device must have Enhanced Memory Management enabled by setting bit 1 of Configuration Register #2 at memory location 0x02 to 1. This is done by default internally by the library when an RT is configured using the aceRTConfigure() function. Another condition that must be set for this interrupt to occur is that the aceRTDataBlkMapToSA() function must be called with the ACE_RT_DBLK_CIRC_IRQ input parameter to set one of the Circular Buffer Interrupts (bit 3, 8, or 13) in the Subaddress Control Word to 1. Bit 3 is set to 1 for a Broadcast Circular Buffer Interrupt. Bit 8 is set to 1 for a Receive Circular Buffer Interrupt. Bit 13 is set to 1 for a Transmit Circular Buffer Interrupt.

ACE_IMR1_TT_ROVER
(bit 6 of Interrupt Mask Reg. #1 is set to 0 or 1)
This mask will result in an interrupt if the 16-bit Time Tag Register at memory location 0x05 rolls over from 0xFFFF to 0x0000.

ACE_IMR1_RT_ADDR_PAR_ERR
(bit 7 of Interrupt Mask Reg. #1 is set to 0 or 1)
This mask will result in an interrupt if an RT Address parity error is encountered.

ACE_IMR1_BC_RETRY
(bit 8 of Interrupt Mask Reg. #1 is set to 0 or 1)
This mask will result in an interrupt if a message retry has occurred in BC mode. The interrupt will occur regardless of whether the retry attempt was successful or unsuccessful.

aceSetIrqConditions (continued)

ACE_IMR1_HSHAKE_FAIL
(bit 9 of Interrupt Mask Reg. #1 is set to 0 or 1)
This mask will result in an interrupt following a handshake timeout during a transfer between the 1553 protocol section and the RAM.

ACE_IMR1_MT_DATASTK_ROVER
(bit 10 of Interrupt Mask Reg. #1 is set to 0 or 1)
This mask will result in an interrupt following a rollover of the Word Monitor or Message Monitor data stack. This interrupt will only occur if Enhanced Mode is selected by setting bit 15 of Configuration Register #3 at memory location 0x07 to 1. This is done by default internally by the library when a MT is configured using the aceMTConfigure() function.

ACE_IMR1_MT_CMDSTK_ROVER
(bit 11 of Interrupt Mask Reg. #1 is set to 0 or 1)
This mask will result in an interrupt following a rollover of the Message Monitor Command Stack. This interrupt will only occur if Enhanced Mode is selected by setting bit 15 of Configuration Register #3 at memory location 0x07 to 1. This is done by default internally by the library when a MT is configured using the aceMTConfigure() function.

ACE_IMR1_BCRT_CMDSTK_ROVER
(bit 12 of Interrupt Mask Reg. #1 is set to 0 or 1)
This mask will result in an interrupt following a rollover of the BC or the RT command stack. This interrupt will only occur if Enhanced Mode is selected by setting bit 15 of Configuration Register #3 at memory location 0x07 to 1. This is done by default internally by the library when a BC/RT/MT is configured.

ACE_IMR1_BCRT_TX_TIMEOUT
(bit 13 of Interrupt Mask Reg. #1 is set to 0 or 1)
This mask will result in an interrupt if a timeout condition has occurred. This interrupt will only occur in BC or RT mode if the device's encoder attempts to transmit for longer than 660.5µs.

ACE_IMR1_RAM_PAR_ERR
(bit 14 of Interrupt Mask Reg. #1 is set to 0 or 1)
This mask will result in an interrupt if a RAM parity error occurs during a read access. This interrupt will only occur if Enhanced Mode is selected by setting bit 15 of Configuration Register #3 at memory location 0x07 to 1. This is done by default internally by the library when a

aceSetIrqConditions (continued)

BC/RT/MT is configured. The RAM Parity Enable option must also be set by setting bit 14 of Configuration Register #2 at memory location 0x02 to 1 for this interrupt to occur. This can be set by calling the aceSetRamParityChecking() function.

ACE_IMR2_BIT_COMPLETE
(bit 1 of Interrupt Mask Reg. #2 is set to 0 or 1)
This mask will result in an interrupt if protocol built-in self-test or the RAM built-in self-test has been completed. Bit 0 of Interrupt Mask Reg. #2 at memory location 0x1D is not used.

ACE_IMR2_BC_UIRQ0
ACE_IMR2_BC_UIRQ1
ACE_IMR2_BC_UIRQ2
ACE_IMR2_BC_UIRQ3
(bits 2 - 5 of Interrupt Mask Reg. #2 are set to 0 or 1)
This mask will result in an interrupt if the device is in Enhanced Bus Controller Mode and an IRQ (Generate Interrupt) hardware instruction is generated by the device. The Interrupt Status Register at memory location 0x1E will contain the value of the lower 4 bits of the parameter associated with the IRQ instruction in bits 2 – 5.

ACE_IMR2_MT_DSTK_50P_ROVER
(bit 6 of Interrupt Mask Reg. #2 is set to 0 or 1)
For selective monitor mode, this mask will result in an interrupt if the data stack is more than half full. This interrupt will occur at the end of the message in which the 50% rollover occurred.

ACE_IMR2_MT_CSTK_50P_ROVER
(bit 7 of Interrupt Mask Reg. #2 is set to 0 or 1)
For selective monitor mode, this mask will result in an interrupt if the command stack is more than half full.

ACE_IMR2_RT_CIRC_50P_ROVER
(bit 8 of Interrupt Mask Reg. #2 is set to 0 or 1)
This mask will result in an interrupt if the circular buffer has rolled over. For this interrupt to occur the device must have Enhanced Memory Management enabled by setting bit 1 of Configuration Register #2 at memory location 0x02 to 1. This is done by default internally by the library when an RT is configured using the aceRTConfigure() function. Another condition that must be set for this interrupt to occur is that the aceRTDataBlkMapToSA() function must be called with the ACE_RT_DBLK_CIRC_IRQ input parameter to set one

aceSetIrqConditions (continued)

of the Circular Buffer Interrupts (bit 3, 8, or 13) in the Subaddress Control Word to 1. Bit 3 is set to 1 for a Broadcast Circular Buffer Interrupt. Bit 8 is set to 1 for a Receive Circular Buffer Interrupt. Bit 13 is set to 1 for a Transmit Circular Buffer Interrupt.

ACE_IMR2_RT_CSTK_50P_ROVER
(bit 9 of Interrupt Mask Reg. #2 is set to 0 or 1)
For RT mode, this mask will result in an interrupt if the command stack is more than half full.

ACE_IMR2_BC_TRAP
(bit 10 of Interrupt Mask Reg. #2 is set to 0 or 1)
For BC mode, this mask will result in an interrupt if the BC has fetched an illegal opcode or if the BC watchdog timer (frame timer) has timed out. An illegal opcode is one that is not defined, fails it's parity check, and/or has an incorrect value for one or more of bits 9 through 5.

ACE_IMR2_BC_CALLSTK_ERR
(bit 11 of Interrupt Mask Reg. #2 is set to 0 or 1)
For BC mode, this mask will result in an interrupt if there has been a violation of the BC's subroutine stack depth. This error occurs if either: (1) a call stack overflow condition or (2) a call stack underflow condition occurred.

ACE_IMR2_GPQ_ISQ_ROVER
(bit 12 of Interrupt Mask Reg. #2 is set to 0 or 1)
For BC mode, this mask will result in an interrupt if the General Purpose Queue has rolled over. For RT mode, this mask will result in an interrupt if the interrupt status queue rolls over.

ACE_IMR2_RT_ILL_CMD
(bit 13 of Interrupt Mask Reg. #2 is set to 0 or 1)
For RT mode, this mask will result in an interrupt if an illegal message has been received by the RT.

ACE_IMR2_BC_OPCODE_PARITY
(bit 14 of Interrupt Mask Reg. #2 is set to 0 or 1)
For BC mode, this mask will generate an interrupt if the opcode word for a BC instruction fails it's parity check.

funcExternalISR (input parameter)
This is the user designated ISR callback function written by the user. This function will be called when the device generates an interrupt.

aceSetIrqConditions (continued)

DESCRIPTION

This function enables the selected interrupts as specified by the settings of interrupt mask registers 1 and 2 if bEnable is set to TRUE. If bEnable is set to FALSE then the selected interrupts are disabled by writing a 0 to the appropriate bit in interrupt mask registers 1 and 2. The selected interrupts may be logically or'ed together in order to combine the operations into one statement and enable/disable multiple interrupt conditions.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in a Ready state and the function could not be completed
ACE_WRN_RT_CFG_INVALID	Operation of your device may be problematic because one or more of the following interrupts have been enabled: Time Tag Rollover, RT Address Parity Error, and/or Ram Parity Error along with the Interrupt Status Queue. See Appendix B for details.

EXAMPLE

```
//The following code would be used to enable only the End-Of-Message
//interrupt.
  S16BIT DevNum = 0;
  U16BIT bEnable = TRUE;
  S16BIT nResult = 0;

  nResult = aceSetIrqConditions(DevNum, bEnable, ACE_IMR1_EOM,
                                funcExternalISR);
  if(nResult)
  {
        printf("Error in aceSetIrqConditions() function \n");
        PrintOutError(nResult);
        return;
  }

  //user defined callback function (funcExternalISR) sample
  void _DECL funcExternalISR(S16BIT DevNum,
                             U32BIT Status)
  {
        //ISR implementation user defined callback routine
        printf("An interrupt has occurred \n");
  }
```

SEE ALSO

aceSetIrqConfig()

aceSetIrqConfig

This function sets the type of interrupt signal generated by the hardware.

PROTOTYPE

```
#include "config.h"

S16BIT _DECL aceSetIrqConfig(S16BIT DevNum,
                             U16BIT wLvlOrPulse,
                             U16BIT wAutoClear);
```

STATE

Ready

MODE

Advanced plus one of the following BC, RT, MT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wLvlOrPulse	(input parameter) Define hardware interrupt operation Valid values: ACE_IRQ_LEVEL ACE_IRQ_PULSE
wAutoClear	(input parameter) Define register read and clear operation Valid Values: ACE_IRQ_AUTO_CLR ACE_IRQ_NO_AUTO_CLR

DESCRIPTION

This function sets the type of interrupt signal generated by the hardware, and whether or not to auto clear status registers after they have been read. There is no need to call this function since it is automatically called when you set up your card as a BC, RT, or MT. The function is called with the wLvlOrPulse parameter set for ACE_IRQ_LEVEL, and the wAutoClear parameter set for ACE_IRQ_AUTO_CLR. If you would like to change the parameters you may do so by calling this function while you are in advanced mode of operation. The function can not be used unless you are in advanced mode of operation. If you are using a card product from DDC you should never call this function.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully

aceSetIrqConfig (continued)

ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in a Ready state and the function could not be completed
ACE_ERR_NOT_SUPPORTED	This function is only supported in Advanced mode of operation

EXAMPLE

```
//If it is desirable to have the EMA to not clear the interrupt
//registers once they have been read and the

//EMA is configured in the system for a level interrupt, then the
following example would be used.

  S16BIT DevNum = 0;

  nResult = aceSetIrqConfig (DevNum, ACE_IRQ_LEVEL, ACE_IRQ_NO_AUTO_CLR);

  if(nResult)
  {
  printf("Error in aceSetIrqConfig() function \n");
  PrintOutError(nResult);
        return;
  }
```

SEE ALSO

aceSetIrqConditions()

aceSetMetrics

This function allows the user to enable built-in performance metrics for informative purposes.

PROTOTYPE

```
#include "Config.h"

S16BIT _DECL aceSetMetrics (S16BIT DevNum,
                            U16BIT bEnable);
```

STATE

Ready

MODE

BC, RT, RTMT, MT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
bEnable	(input parameter) Enable/Disable System Metrics Valid values: FALSE (0) Disable system metrics TRUE (1) Enable system metrics

DESCRIPTION

This function allows the user to enable built-in performance metrics for informative purposes. Built-in test metrics can report the number of messages in the host buffer, the total number of messages lost since the host buffer was installed, the current percentage of the host buffer that is used, the highest percentage of the host buffer used since it was installed, the total number of messages lost on the device's hardware stack, the current percentage of the stack that is used, and the highest percentage of the stack used. In addition, while in BC mode, built-in test metrics can report the number of messages lost on the GPQ, the current percentage of the GPQ that is used, and the highest percentage of the GPQ that is used.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in BC, RT, MT, or RTMT mode

aceSetMetrics (continued)

EXAMPLE

```
S16BIT DevNum = 0;
U16BIT bEnable = 1;
S16BIT nResult = 0;

nResult = aceSetMetrics(DevNum, bEnable);

if(nResult)
{
      printf("Error in aceSetMetrics() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceMTGetHBufMetric()
aceRTGetHBufMetric()
aceBCGetHBufMetric()
aceMTGetStkMetric()
aceRTGetStkMetric()
aceBCGetGPQMetric()

aceSetRamParityChecking

This function will be used to enable or disable RAM parity checking.

PROTOTYPE

```
#include "config.h"

S16BIT _DECL aceSetRamParityChecking(S16BIT DevNum,
                                     U16BIT wRamParityEnable);
```

STATE

Ready

MODE

Not Applicable

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wRamParityEnable	(input parameter) This parameter specifies the desired RAM checking operation Valid values: ACE_ENABLE_PARITY_CHECK (bit 14 of Configuration Reg. #2 is set to 1) ACE_DISABLE_PARITY_CHECK (bit 14 of Configuration Reg. #2 is set to 0)

DESCRIPTION

This function will be used to enable or disable RAM parity checking. If the hardware design includes 17-bit RAM, this option can be used to employ the automatic RAM parity checking. If the RAM is 16-bit, this option has no effect. The function will set bit 14 of Configuration Register #2 to a 0 or a 1 to respectively disable or enable the RAM parity checking. The RAM parity checking is disabled by default internally in the library when a BC, RT, or MT is configured and can be enabled with a call to this function after the BC, RT or MT has been configured.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in a Ready state and the function could not be completed

aceSetRamParityChecking (continued)

EXAMPLE

```
//The following code can be used for hardware that incorporates 17-bit
//RAM.

  S16BIT DevNum = 0;

  nResult = aceSetRamParityChecking(DevNum, ACE_ENABLE_PARITY_CHECK)

  if(nResult)
  {
        printf("Error in aceSetRamParityChecking() function \n");
        PrintOutError(nResult);
        return;
  }
```

SEE ALSO

None

aceSetRespTimeOut

This function sets the message RT response timeout timer on the hardware.

PROTOTYPE

```
#include "config.h"

S16BIT _DECL aceSetRespTimeOut(S16BIT DevNum,
                               U16BIT wRespTimeOut);
```

STATE

Ready

MODE

Not Applicable

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

WrespTimeOut (input parameter)
Message RT Response Timeout
Valid values:
ACE_RESPTIME_18US - 18.5 μs timeout (default)
ACE_RESPTIME_22US - 22.5 μs timeout
ACE_RESPTIME_50US - 50.5 μs timeout
ACE_RESPTIME_130US - 130 μs timeout

DESCRIPTION

This function sets the device's response timeout timer on the hardware by configuring bits 9 and 10 of Configuration Register # 5 at location 0x09. This timer is used in BC mode, for RT mode (for messages in which the device is the receiving RT in an RT-RT transfer), and in the message MT mode. If a RT is fairly slow to respond to messages from a Bus Controller, as might be the case for a very long bus length between the two terminals, it might be necessary to increase the timeout to 50.5 μs. This will cause the BC to wait a little longer for a response before declaring an error and continuing with the next message.

RETURN VALUE

ACE_ERR_SUCCESS	This function has successfully set the response timeout value
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in a Ready state and the function could not complete
ACE_ERR_RESPTIME	The wRespTimeOut input parameter contains an incorrect value

aceSetRespTimeOut (continued)

EXAMPLE

```
//sets the response timeout value
  S16BIT DevNum = 0;
  S16BIT nResult = 0;

  nResult = aceSetRespTimeOut (DevNum, ACE_RESPTIME_50US);

  if(nResult)
  {

  printf("Error      in      aceSetRespTimeOut()      function      \n");
  PrintOutError(nResult);
        return;
  }
```

SEE ALSO

None

aceSetTimeTagRes

This function sets the time tag resolution of the device.

PROTOTYPE

```
#include "config.h"

S16BIT _DECL aceSetTimeTagRes(S16BIT DevNum,
                              U16BIT wTTRes);
```

STATE

Ready

MODE

Not Applicable

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wTTRes	(input parameter) Time Tag Resolution Valid values: ACE_TT_64US – 64 μs resolution ACE_TT_32US – 32 μs resolution ACE_TT_16US – 16 μs resolution ACE_TT_8US - 8 μs resolution ACE_TT_4US - 4 μs resolution ACE_TT_2US - 2 μs resolution ACE_TT_TEST - Increment manually ACE_TT_EXT – Use External Clock

DESCRIPTION

This function sets the resolution of the Time Tag Register by setting bits 7-9 of Configuration Register #2 at memory offset 0x02. The Time Tag Resolution is initially set to a resolution of 2μs when the aceInitialize() function is first called. The user can then use this function to change the time tag resolution to any of the valid values mentioned above.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number has been input to this function
ACE_ERR_INVALID_STATE	The device is not in the Ready state and the function could not be completed

aceSetTimeTagRes (continued)

ACE_ERR_TIMETAG_RES	The wTTRes input by the user does not contain a valid value

EXAMPLE

```
//To set the time tag resolution to 16 µs.
  S16BIT DevNum = 0;
  S16BIT nResult = 0;

  nResult = aceSetTimeTagRes (DevNum, ACE_TT_16_US);

  if(nResult)
  {
  printf(“Error      in      aceSetTimeTagRes()      function      \n”);
  PrintOutError(nResult);
        return;
  }
```

SEE ALSO

None

aceSetTimeTagValue

This function allows the user to modify the value of the time tag register.

PROTOTYPE

#include “config.h”

S16BIT _DECL aceSetTimeTagValue(S16BIT DevNum,
U16BIT wTTValue);

STATE

Reset, Ready, Run

MODE

Not Applicable

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wTTValue	(input parameter) Time Tag Value Valid values: 0x0000 – 0xFFFF

DESCRIPTION

This function sets the value of the Time Tag Register.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully

EXAMPLE

```
//To set the time tag value.
  S16BIT DevNum = 0;
  S16BIT nResult = 0;

  nResult = aceSetTimeTagValue (DevNum, 0x0023);
  if(nResult)
  {
  printf(“Error      in      aceSetTimeTagValue()      function      \n”);
  PrintOutError(nResult);
        return;
  }
```

SEE ALSO

None

aceTestIrqs

This function will verify that interrupts are working.

PROTOTYPE

```
#include "testop.h"

S16BIT _DECL aceTestIrqs(S16BIT DevNum,
                         TESTRESULT *pTest);
```

STATE

Ready

MODE

Test

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
pTest	(output parameter) Pointer to a TESTRESULT structure that will contain the results of the interrupt tests

DESCRIPTION

This function will reset the device, and set bits 7 – 9 of Configuration Register #2 to a value of 011 which sets the time tag resolution to Test Mode. The interrupt is selected to be a level type interrupt by setting bit 3 of Configuration Register #2 to a 1, unless you are using a PC/104 card with DOS or VxWorks. The time tag rollover is enabled by setting bit 6 in Interrupt Mask Register #1. This function then generates an interrupt by loading the time tag register with a value of 0xFFFF and then incrementing it to generate a time tag rollover interrupt. The function then checks that it was captured by the internal Win32 interrupt handler.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in Test mode
ACE_ERR_TEST_BADSTRUCT	The pTest pointer to a TESTRESULT structure input by the user is Null
ACE_ERR_INVALID_ACCESS	The device is not a card because it is not in ACE_ACCESS_CARD mode set by the aceInitialize() function

aceTestIrqs (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

TESTRESULT *pTest;

aceInitialize(DevNum,ACE_ACCESS_CARD,ACE_MODE_TEST,0,0,0);

nResult = aceTestIrqs(DevNum, pTest);

if(nResult)
{
      printf("Error in aceTestIrqs() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceTestRegisters()
aceTestMemory()
aceTestProtocol()
aceTestVectors()

aceTestMemory

This function tests the hardware's memory.

PROTOTYPE

#include "testop.h"

```
S16BIT _DECL aceTestMemory(S16BIT DevNum,
                           TESTRESULT *pTest,
                           U16BIT wValue);
```

STATE

Ready

MODE

Test

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
pTest	(output parameter) Pointer to a TESTRESULT structure that will contain the results of the memory tests
wValue	(input parameter) Test value to be written and read during memory tests

DESCRIPTION

This function tests hardware memory. It fills all of the memory with wValue and verifies by reading each location back and comparing to make sure the values are the same.

RETURN VALUE

ACE_ERR_SUCCESS	The function successfully completed and wrote any memory errors to the TESTRESULT structure
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in Test mode
ACE_ERR_TEST_BADSTRUCT	The pTest pointer to the TESTRESULT structure input by the user is Null

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
TESTRESULT *pTest;
U16BIT wValue = 0xA5A5;
```

aceTestMemory (continued)

```
aceInitialize(DevNum,ACE_ACCESS_CARD,ACE_MODE_TEST,0,0,0);

nResult = aceTestMemory(DevNum, pTest, wValue);

if(nResult)
{
      printf("Error in aceTestMemory() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceTestRegisters()
aceTestIrqs()
aceTestProtocol()
aceTestVectors()

aceTestProtocol

This function performs a test on the hardware protocol functions.

PROTOTYPE

```
#include "testop.h"

aceTestProtocol(S16BIT DevNum,
                TESTRESULT *pTest);
```

STATE

Ready

MODE

Test

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
pTest	(output parameter) Pointer to a TESTRESULT structure that will contain the results of the protocol tests

DESCRIPTION

The function will reset the device, and set bits 8 – 11 in Configuration Register #1 to a value of 1 to configure the device. The function will then set the time tag resolution to 2 µs by setting bits 7 – 9 of Configuration Register # 2 to a value of 101. This function then performs a series of tests on the hardware protocol functions to make sure that the device is working properly.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was passed to this function
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in Test mode
ACE_ERR_TEST_BADSTRUCT	The pTest pointer to the TESTRESULT structure input by the user is Null

EXAMPLE

```
S16BIT DevNum = 0;
```

aceTestProtocol (continued)

```
TESTRESULT *pTest;

aceInitialize(DevNum,ACE_ACCESS_CARD,ACE_MODE_TEST,0,0,0);

nResult = aceTestProtocol(DevNum, pTest);

if(nResult)
{
      printf(“Error in aceTestProtocol() function \n”);
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceTestRegisters()
aceTestIrqs()
aceTestMemory()
aceTestVectors()

aceTestRegisters

This function tests hardware registers.

PROTOTYPE

#include “testop.h”

S16BIT _DECL aceTestRegisters(S16BIT DevNum,
TESTRESULT *pTest);

STATE

Ready

MODE

Test

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
pTest	(output parameter) Pointer to a TESTRESULT structure that will return the results of the register tests.

DESCRIPTION

This function tests hardware registers by performing a series of reads and writes to the hardware registers. This test can only be performed on an Enhanced Mini-ACE device.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in Test mode
ACE_ERR_TEST_BADSTRUCT	An invalid pointer to a TESTRESULT structure was input to this function

EXAMPLE

```
S16BIT DevNum = 0;
TESTRESULT *pTest;

aceInitialize(DevNum,ACE_ACCESS_CARD,ACE_MODE_TEST,0,0,0);
nResult = aceTestRegisters(DevNum, pTest);

if(nResult)
{
      printf(“Error in aceTestRegisters() function \n”);
      PrintOutError(nResult);
```

aceTestRegisters (continued)

```
        return;
    }
```

SEE ALSO

aceTestMemory()
aceTestIrqs()
aceTestProtocol()
aceTestVectors()

aceTestVectors

This function will test the hardware using a vector file.

PROTOTYPE

```
#include "testop.h"


S16BIT _DECL aceTestVectors(S16BIT DevNum,
                            TESTRESULT *pTest,
                            char *pFileName);
```

STATE

Ready

MODE

Test

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
pTest	(output parameter) Pointer to a TESTRESULT structure that will contain the results of the register tests
pFileName	(input parameter) A pointer to the Test Vector source file.

DESCRIPTION

This function tests hardware using a vector file. Test vectors will be retrieved from the source file one at a time and applied to the EMA hardware. After applying a complete group of vectors, registers and memory will be read to ensure the test group passed.

RETURN VALUE

ACE_ERR_SUCCESS	The function has completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in a ready state
ACE_ERR_INVALID_MODE	The device is not in a test mode
ACE_ERR_TEST_BADSTRUCT	The pTest pointer to a TESTRESULT structure input by the user is Null
ACE_ERR_TEST_FILE	The pFileName pointer to the Test Vector Source file is Null

aceTestVectors (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
TESTRESULT *pTest;
char *pFileName = “c:\\acetest.vec”;

aceInitialize(DevNum,ACE_ACCESS_CARD,ACE_MODE_TEST,0,0,0);

nResult = aceTestVectors(DevNum, pTest, pFileName);

if(nResult)
{
      printf(“Error in aceTestVectors() function \n”);
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceTestRegisters()
aceTestIrqs()
aceTestMemory()
aceTestProtocol()

BC Functions

Table 5. BC Functions Listing

aceBCAsyncMsgCreateBcst

This function creates an asynchronous broadcast message.

PROTOTYPE

```
#include "BCOp.h"

S16BIT _DECL aceBCAsyncMsgCreateBcst(S16BIT DevNum,
                                     S16BIT nMsgBlkID,
                                     S16BIT nDataBlkID,
                                     U16BIT wSA,
                                     U16BIT wWC,
                                     U16BIT wMsgGapTime
                                     U32BIT dwMsgOptions
                                     U16BIT *pBuffer)
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgBlkID	(input parameter) Unique ID number for the new message block Valid values: >0
nDataBlkID	(input parameter) Unique ID number for this block of data Valid values: >0
wSA	(input parameter) Remote Terminal Subaddress
wWC	(input parameter) Data word count for this message
wMsgGapTime	(input parameter) The time to next message in µseconds

aceBCAsyncMsgCreateBcst (continued)

dwMsgOptions	(input parameter) See aceBCMsgCreateBcst for a list of options
pBuffer	(input parameter) Buffer containing the data words for this message

DESCRIPTION

This function creates an asynchronous broadcast message. All parameters will be set up for the Broadcast message based on the dwMsgOptions input parameter. This function calls the aceBCMsgCreateBcst() function and then asynchronously inserts your message when you call one of the asynchronous send routines.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_NOT_ASYNC_MODE	The software is not in asynchronous mode. This state must be selected in the aceBCConfigure() function.
ACE_ERR_BC_DBLK_EXISTS	The message block specified by the nMsgBlkID input parameter already exists
ACE_ERR_NODE_NOT_FOUND	The data block specified by the nDataBlkID can not be created. Make sure that this data block has not been previously created with the aceBCDataBlkCreate() function. Asynchronouse messages will call the aceBCDataBlkCreate() function internally and this function does not need to be called by the user.
ACE_ERR_BC_DBLK_SIZE	The data block size is not valid
ACE_ERR_BC_DBLK_ALLOC	The message block could not be allocated
ACE_ERR_MEMMGR_FAIL	The required memory for the message block could not be allocated

aceBCAsyncMsgCreateBcst (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT pBuffer[32];

for (int count=0; count<32; count++)
{
      pBuffer [count]=0x1111;
}

nResult = aceBCAsyncMsgCreateBcst(S16BIT DevNum,
                                  1,
                                  1,
                                  1,
                                  32,
                                  0,
                                  ACE_BCCTRL_CHL_A,
                                  pBuffer);

if(nResult)
{
      printf("Error occurred in aceBCAsyncMsgCreateBcst function call
      \n");

      PrintError(nResult);
      return;
}
```

SEE ALSO

aceBCMsgCreateBcst()

aceBCAsyncMsgCreateBcstMode

This function creates an asynchronous broadcast mode code message.

PROTOTYPE

```
#include "BCOp.h"

S16BIT _DECL aceBCAsyncMsgCreateBcstMode (S16BIT DevNum,
                                          S16BIT nMsgBlkID,
                                          S16BIT nDataBlkID,
                                          U16BIT wTR,
                                          U16BIT wModeCmd,
                                          U16BIT wMsgGapTime,
                                          U32BIT dwMsgOptions,
                                          U16BIT *pBuffer);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

nMsgBlkID (input parameter)
Unique ID number for the new message block
Valid values:
>0

nDataBlkID (input parameter)
Unique ID number for this block of data
Valid values:
>0

wTR (input parameter)
Message Transmit/Receive bit
Valid Values:
ACE_RX_CMD
ACE_TX_CMD

aceBCAsyncMsgCreateBcstMode (continued)

wModeCmd	(input parameter) Message Mode Code Command Valid Values: 0 - 31
wMsgGapTime	(input parameter) The time to next message in µseconds
dwMsgOptions	(input parameter) See aceBCMsgCreateBcstMode for a list of options
pBuffer	(input parameter) Buffer containing the data words for this message

DESCRIPTION

This function creates an asynchronous broadcast mode code message. All parameters will be set up for the Broadcast mode code message based on the dwMsgOptions input parameter. This function calls the aceBCMsgCreateBcstMode() function and then asynchronously inserts your message when you call one of the asynchronous send routines.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_NOT_ASYNC_MODE	The software is not in asynchronous mode. This state must be selected in the aceBCConfigure() function.
ACE_ERR_BC_DBLK_EXISTS	The message block specified by the nMsgBlkID input parameter already exists
ACE_ERR_NODE_NOT_FOUND	The data block specified by the nDataBlkID can not be created. Make sure that this data block has not been previously created with the aceBCDataBlkCreate() function. Asynchronouse messages will call the aceBCDataBlkCreate() function internally and this function does not need to be called by the user.
ACE_ERR_BC_DBLK_SIZE	The data block size is not valid
ACE_ERR_BC_DBLK_ALLOC	The message block could not be allocated
ACE_ERR_MEMMGR_FAIL	The required memory for the message block could not be allocated

aceBCAsyncMsgCreateBcstMode (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT pBuffer[32];

for (int count=0; count<32; count++)
{
      pBuffer [count]=0x1111;
}

nResult = aceBCAsyncMsgCreateBcstMode (S16BIT DevNum,
                                       1,
                                       1,
                                       ACE_RX_CMD,
                                       1,
                                       0,
                                       ACE_BCCTRL_CHL_A);

if(nResult)
{
      printf("Error occurred in aceBCAsyncMsgCreateBcstMode function
      call \n");

      PrintError(nResult);
      return;
}
```

SEE ALSO

aceBCMsgCreateBcstMode()

aceBCAsyncMsgCreateBcstRTtoRT

This function creates an asynchronous RT to RT message.

PROTOTYPE

```
#include "BCOp.h"

S16BIT _DECL aceBCAsyncMsgCreateBcstRTtoRT
                                (S16BIT DevNum,
                                S16BIT nMsgBlkID,
                                S16BIT nDataBlkID,
                                U16BIT wSARx,
                                U16BIT wWC,
                                U16BIT wRTTx,
                                U16BIT wSATx,
                                U16BIT wMsgGapTime,
                                U32BIT dwMsgOptions,
                                U16BIT *pBuffer);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum
(input parameter)
Logical Device Number
Valid values:
0 – 31

nMsgBlkID
(input parameter)
Unique ID number for the new message block
Valid values:
>0

nDataBlkID
(input parameter)
Unique ID number for this block of data
Valid values:
>0

wSARx
(input parameter)
RT SA of receiving RT

wWC
(input parameter)
The word count for the message

aceBCAsyncMsgCreateBcstRTtoRT (continued)

wRTTx	(input parameter) RT address of transmitting RT
wSATx	(input parameter) RT SA of transmitting RT
wMsgGapTime	(input parameter) The time to next message in μseconds
dwMsgOptions	(input parameter) See aceBCMsgCreateBcstMode for a list of options
pBuffer	(input parameter) Buffer containing the data words for this message

DESCRIPTION

This function creates an asynchronous broadcast RT to RT message. All parameters will be set up for the message based on the dwMsgOptions input parameter. This function calls the aceBCMsgCreateBcstRTtoRT () function and then asynchronously inserts your message when you call one of the asynchronous send routines.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_NOT_ASYNC_MODE	The software is not in asynchronous mode. This state must be selected in the aceBCConfigure() function.
ACE_ERR_BC_DBLK_EXISTS	The message block specified by the nMsgBlkID input parameter already exists
ACE_ERR_NODE_NOT_FOUND	The data block specified by the nDataBlkID can not be created. Make sure that this data block has not been previously created with the aceBCDataBlkCreate() function. Asynchronouse messages will call the aceBCDataBlkCreate() function internally and this function does not need to be called by the user.
ACE_ERR_BC_DBLK_SIZE	The data block size is not valid
ACE_ERR_BC_DBLK_ALLOC	The message block could not be allocated
ACE_ERR_MEMMGR_FAIL	The required memory for the message block could not be allocated

aceBCAsyncMsgCreateBcstRTtoRT (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT pBuffer[32];

for (int count=0; count<32; count++)
{
      pBuffer [count]=0x1111;
}

nResult = aceBCAsyncMsgCreateBcstRTtoRT(S16BIT DevNum,
                                        1,
                                        1,
                                        1,
                                        32,
                                        1,
                                        1,
                                        0,
                                        ACE_BCCTRL_CHL_A,
                                        pBuffer);

if(nResult)
{
      printf("Error occurred in aceBCAsyncMsgCreateBcstRTtoRT function
      call \n");

      PrintError(nResult);
      return;
}
```

SEE ALSO

aceBCMsgCreateBcstRTtoRT()

aceBCAsyncMsgCreateBCtoRT

This function creates an asynchronous BC to RT message.

PROTOTYPE

```
#include "BCOp.h"

S16BIT _DECL aceBCAsyncMsgCreateBCtoRT(S16BIT DevNum,
                                       S16BIT nMsgBlkID,
                                       S16BIT nDataBlkID,
                                       U16BIT wRT,
                                       U16BIT wSA,
                                       U16BIT wWC,
                                       U16BIT wMsgGapTime,
                                       U32BIT dwMsgOptions,
                                       U16BIT *pBuffer);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgBlkID	(input parameter) Unique ID number for the new message block Valid values: >0
nDataBlkID	(input parameter) Unique ID number for this block of data Valid values: >0
wRT	(input parameter) RT address
wSA	(input parameter) RT SA
wWC	(input parameter) The word count for the message

aceBCAsyncMsgCreateBCtoRT (continued)

wMsgGapTime	(input parameter) The time to next message in µseconds
dwMsgOptions	(input parameter) See aceBCMsgCreateBCtoRT for a list of options
pBuffer	(input parameter) Buffer containing the data words for this message

DESCRIPTION

This function creates an asynchronous BC to RT message. All parameters will be set up for the message based on the dwMsgOptions input parameter. This function calls the aceBCMsgCreateBCtoRT () function and then asynchronously inserts your message when you call one of the asynchronous send routines.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_NOT_ASYNC_MODE	The software is not in asynchronous mode. This state must be selected in the aceBCConfigure() function.
ACE_ERR_BC_DBLK_EXISTS	The message block specified by the nMsgBlkID input parameter already exists
ACE_ERR_NODE_NOT_FOUND	The data block specified by the nDataBlkID can not be created. Make sure that this data block has not been previously created with the aceBCDataBlkCreate() function. Asynchronouse messages will call the aceBCDataBlkCreate() function internally and this function does not need to be called by the user.
ACE_ERR_BC_DBLK_SIZE	The data block size is not valid
ACE_ERR_BC_DBLK_ALLOC	The message block could not be allocated
ACE_ERR_MEMMGR_FAIL	The required memory for the message block could not be allocated

aceBCAsyncMsgCreateBCtoRT (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT pBuffer[32];

for (int count=0; count<32; count++)
{
      pBuffer [count]=0x1111;
}

nResult = aceBCAsyncMsgCreateBCtoRT(DevNum,
                                    1,
                                    1,
                                    1,
                                    1,
                                    32,
                                    0,
                                    ACE_BCCTRL_CHL_A,
                                    pBuffer);

if(nResult)
{
      printf("Error occurred in aceBCAsyncMsgCreateBCtoRT function call
      \n");

      PrintError(nResult);
      return;
}
```

SEE ALSO

aceBCMsgCreateBCtoRT()

aceBCAsyncMsgCreateMode

This function creates an asynchronous mode code message.

PROTOTYPE

```
#include "BCOp.h"

S16BIT _DECL aceBCAsyncMsgCreateMode   (S16BIT DevNum,
                                        S16BIT nMsgBlkID,
                                        S16BIT nDataBlkID,
                                        U16BIT wRT,
                                        U16BIT wTR,
                                        U16BIT wModeCmd,
                                        U16BIT wMsgGapTime,
                                        U32BIT dwMsgOptions,
                                        U16BIT *pBuffer)
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgBlkID	(input parameter) Unique ID number for the new message block Valid values: >0
nDataBlkID	(input parameter) Unique ID number for this block of data Valid values: >0
wRT	(input parameter) RT address
wTR	(input parameter) Message Transmit/Receive bit Valid Values: ACE_RX_CMD ACE_TX_CMD

aceBCAsyncMsgCreateMode (continued)

wModeCmd	(input parameter) The mode command to be issued
wMsgGapTime	(input parameter) The time to next message in µseconds
dwMsgOptions	(input parameter) See aceBCMsgCreateMode for a list of options
pBuffer	(input parameter) Buffer containing the data words for this message

DESCRIPTION

This function creates an asynchronous mode code message. All parameters will be set up for the message based on the dwMsgOptions input parameter. This function calls the aceBCMsgCreateMode () function and then asynchronously inserts your message when you call one of the asynchronous send routines.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_NOT_ASYNC_MODE	The software is not in asynchronous mode. This state must be selected in the aceBCConfigure() function.
ACE_ERR_BC_DBLK_EXISTS	The message block specified by the nMsgBlkID input parameter already exists
ACE_ERR_NODE_NOT_FOUND	The data block specified by the nDataBlkID can not be created. Make sure that this data block has not been previously created with the aceBCDataBlkCreate() function. Asynchronouse messages will call the aceBCDataBlkCreate() function internally and this function does not need to be called by the user.
ACE_ERR_BC_DBLK_SIZE	The data block size is not valid
ACE_ERR_BC_DBLK_ALLOC	The message block could not be allocated
ACE_ERR_MEMMGR_FAIL	The required memory for the message block could not be allocated

aceBCAsyncMsgCreateMode (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT pBuffer[32];

for (int count=0; count<32; count++)
{
      pBuffer [count]=0x1111;
}

nResult = aceBCAsyncMsgCreateMode(DevNum,
                                  1,
                                  1,
                                  1,
                                  1,
                                  1,
                                  0,
                                  ACE_BCCTRL_CHL_A,
                                  pBuffer);

if(nResult)
{
      printf("Error occurred in aceBCAsyncMsgCreateMode function call
      \n");

      PrintError(nResult);
      return;
}
```

SEE ALSO

aceBCMsgCreateMode()

aceBCAsyncMsgCreateRTtoBC

This function creates an asynchronous BC to RT message.

PROTOTYPE

```
#include "BCOp.h"

S16BIT _DECL aceBCAsyncMsgCreateRTtoBC(S16BIT DevNum,
                                       S16BIT nMsgBlkID,
                                       S16BIT nDataBlkID,
                                       U16BIT wRT,
                                       U16BIT wSA,
                                       U16BIT wWC,
                                       U16BIT wMsgGapTime,
                                       U32BIT dwMsgOptions,
                                       U16BIT *pBuffer);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgBlkID	(input parameter) Unique ID number for the new message block Valid values: >0
nDataBlkID	(input parameter) Unique ID number for this block of data Valid values: >0
wRT	(input parameter) RT address
wSA	(input parameter) RT SA
wWC	(input parameter) The word count for the message

aceBCAsyncMsgCreateRTtoBC (continued)

wMsgGapTime	(input parameter) The time to next message in µseconds
dwMsgOptions	(input parameter) See aceBCMsgCreateBCtoRT for a list of options
pBuffer	(input parameter) Buffer containing the data words for this message

DESCRIPTION

This function creates an asynchronous RT to BC message. All parameters will be set up for the message based on the dwMsgOptions input parameter. This function calls the aceBCMsgCreateRTtoBC () function and then asynchronously inserts your message when you call one of the asynchronous send routines.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_NOT_ASYNC_MODE	The software is not in asynchronous mode. This state must be selected in the aceBCConfigure() function.
ACE_ERR_BC_DBLK_EXISTS	The message block specified by the nMsgBlkID input parameter already exists
ACE_ERR_NODE_NOT_FOUND	The data block specified by the nDataBlkID can not be created. Make sure that this data block has not been previously created with the aceBCDataBlkCreate() function. Asynchronouse messages will call the aceBCDataBlkCreate() function internally and this function does not need to be called by the user.
ACE_ERR_BC_DBLK_SIZE	The data block size is not valid
ACE_ERR_BC_DBLK_ALLOC	The message block could not be allocated
ACE_ERR_MEMMGR_FAIL	The required memory for the message block could not be allocated

aceBCAsyncMsgCreateRTtoBC (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT pBuffer[32];

for (int count=0; count<32; count++)
{
      pBuffer [count]=0x1111;
}

nResult = aceBCAsyncMsgCreateRTtoBC(DevNum,
                                    1,
                                    1,
                                    1,
                                    1,
                                    32,
                                    0,
                                    ACE_BCCTRL_CHL_A,
                                    pBuffer);

if(nResult)
{
      printf("Error occurred in aceBCAsyncMsgCreateRTtoBC function call
      \n");

      PrintError(nResult);
      return;
}
```

SEE ALSO

aceBCMsgCreateRTtoBC()

aceBCAsyncMsgCreateRTtoRT

This function creates an asynchronous BC to RT message.

PROTOTYPE

#include “BCOp.h”

```
S16BIT _DECL aceBCAsyncMsgCreateRTtoRT(S16BIT DevNum,
                                       S16BIT nMsgBlkID,
                                       S16BIT nDataBlkID,
                                       U16BIT wRTRx,
                                       U16BIT wSARx,
                                       U16BIT wWC,
                                       U16BIT wRTTx,
                                       U16BIT wSATx
                                       U16BIT wMsgGapTime,
                                       U32BIT dwMsgOptions,
                                       U16BIT *pBuffer);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgBlkID	(input parameter) Unique ID number for the new message block Valid values: >0
nDataBlkID	(input parameter) Unique ID number for this block of data Valid values: >0
wRTRx	(input parameter) RT address for receiving RT

aceBCAsyncMsgCreateRTtoRT (continued)

wSARx	(input parameter) RT SA for receiving RT
wWC	(input parameter) The word count for the message
wRTTx	(input parameter) RT address for transmitting RT
wSATx	(input parameter) RT SA for transmitting RT
wMsgGapTime	(input parameter) The time to next message in µseconds
dwMsgOptions	(input parameter) See aceBCMsgCreateRTtoRT for a list of options
pBuffer	(input parameter) Buffer containing the data words for this message

DESCRIPTION

This function creates an asynchronous RT to RT message. All parameters will be set up for the message based on the dwMsgOptions input parameter. This function calls the aceBCMsgCreateRTtoRT () function and then asynchronously inserts your message when you call one of the asynchronous send routines.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_NOT_ASYNC_MODE	The software is not in asynchronous mode. This state must be selected in the aceBCConfigure() function.
ACE_ERR_BC_DBLK_EXISTS	The message block specified by the nMsgBlkID input parameter already exists
ACE_ERR_NODE_NOT_FOUND	The data block specified by the nDataBlkID can not be created. Make sure that this data block has not been previously created with the aceBCDataBlkCreate() function. Asynchronouse messages will call the aceBCDataBlkCreate() function internally and this function does not need to be called by the user.

aceBCAsyncMsgCreateRTtoRT (continued)

ACE_ERR_BC_DBLK_SIZE	The data block size is not valid
ACE_ERR_BC_DBLK_ALLOC	The message block could not be allocated
ACE_ERR_MEMMGR_FAIL	The required memory for the message block could not be allocated

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT pBuffer[32];

for (int count=0; count<32; count++)
{
      pBuffer [count]=0x1111;
}

nResult = aceBCAsyncMsgCreateRTtoRT(DevNum,
                                    1,
                                    1,
                                    1,
                                    1,
                                    32,
                                    2,
                                    1,
                                    0,
                                    ACE_BCCTRL_CHL_A,
                                    pBuffer);

if(nResult)
{
      printf("Error occurred in aceBCAsyncMsgCreateRTtoRT function call
      \n");

      PrintError(nResult);
      return;
}
```

SEE ALSO

aceBCMsgCreateRTtoRT()

aceBCConfigure

This function configures the Bus Controller operation.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCConfigure(S16BIT DevNum,
                            U32BIT dwOptions);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
dwOptions	BC operation options Valid values: 0 No Special Options Selected ACE_BC_ASYNC_LMODE This option selects low priority asynchronous messaging mode. This mode allows you to create a queue of messages that can be sent at the end of the current frame only if frame time permits with the aceBCSendAsyncMsgLP() function. ACE_BC_ASYNC_HMODE This option selects high priority asynchronous messaging mode. This mode allows you to create asynchronous messages and send them onto the bus one at a time at the end of the current synchronous message with the aceBCSendAsyncMsgHP() function.

aceBCConfigure (continued)

ACE_BC_ASYNC_BOTH
This options selects high and low priority messaging. This allows you to mix high and low priority messages. Please note that when asynchronous messages are create in this mode there is no high or low priority designation. If you call the aceBCSendAsyncMsgLP() function then all previously created asynchronous messages will be sent at the end of the current frame if the frame timer permits. The aceBCSendAsyncMsgHP() function allows you to specify a specific message by inputting the unique message id number.

DESCRIPTION

This function configures the Bus Controller operation. Data structures and memory allocations are returned to an initialized state. All existing messages, data blocks, minor frames and frames are deleted from the specified EMA device.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_INVALID_MALLOC	The device failed to allocate memory for the BC structure
ACE_ERR_MEMMGR_FAIL	Memory allocation could not be completed

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceBCConfigure(DevNum,0);

if(nResult)
{
      printf("Error in aceBCConfigure() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

None

aceBCCreateImageFiles

This function allows users to create image files for their code.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCCreateImageFiles(S16BIT DevNum,
                                   S16BIT nMjrFrameID,
                                   char *pszIFile,
                                   char *pszHFile);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMjrFrameID	(input parameter) Handle to the major frame operation structures
pszIFile	(input parameter) Pointer to a null terminated string designating the image file name to be generated
pszHFile	(input parameter) Pointer to a null terminated string designating the header file name to be generated

DESCRIPTION

This function outputs 2 files. The first is a binary image of the Enhanced Mini-ACE's memory. The second is a 'C' header file that contains all offsets and sample functions that allow memory to be accessed easily in an embedded system.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in the Ready state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	The pszIFile pointer or the pszHFile pointer input by the user is Null

aceBCCreateImageFiles (continued)

ACE_ERR_NODE_NOT_FOUND	A major frame specified by nMjrFrameID could not be found
ACE_ERR_FRAME_NOT_MAJOR	The frame specified by the nMjrFrameID parameter is not a major frame
ACE_ERR_UNRES_DATABLK	The function failed to resolve the addresses for all of the message data pointers

EXAMPLE

```
// after full setup of messages and frames
S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceBCCreateImageFiles(DevNum, 4, "bcimage.bin", "bcimage.h");

if(nResult)
{
      printf("Error in aceBCCreateImageFiles() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

None

aceBCDataBlkCreate

This function creates a data block to be used by a message.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCDataBlkCreate(S16BIT DevNum,
                                S16BIT nDataBlkID,
                                U16BIT wDataBlkSize,
                                U16BIT *pBuffer,
                                U16BIT wBufferSize);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nDataBlkID	(input parameter) Unique S16BIT value representing the block id. Arbitrary number supplied by the user Valid values: Must be > 0
wDataBlkSize	(input parameter) A U16BIT word value representing the size of the data block Valid values: ACE_BC_DBLK_DOUBLE ACE_BC_DBLK_SINGLE Arbitrary size specified by user between 1 and 32
pBuffer	(input parameter) A pointer to data to be loaded into the data block at creation Valid values: Address of user buffer

aceBCDataBlkCreate (continued)

wBufferSize	(input parameter) Number of words to be copied from user buffer to newly created data block Valid values: wBufferSize must be less than or equal to the size of the buffer and the size of the data block.

DESCRIPTION

This function allocates a data block to be used by a message. The wDataBlkSize value can be 1-32 words long, single or double buffered.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	The nDataBlkID and/or the nDataBlkSize and/or the wBufferSize parameter input by the user contains an incorrect value
ACE_ERR_BC_DBLK_EXISTS	The specified ID input by the user in the nDataBlkID parameter already exists
ACE_ERR_BC_DBLK_ALLOC	The data block could not be created
ACE_ERR_MEMMGR_FAIL	Memory required for the data block could not be allocated

EXAMPLE

```
// Create 1 data block, 32 words, double buffered
S16BIT DevNum = 0;
S16BIT DBLK1 = 1;
S16BIY nResult = 0;
U16BIT pBuffer[32] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0xA,0xB, 0xC, 0xD,
0xE, 0xF, 0x1001, 0x1002, 0x1003, 0x1004, 0x1005, 0x1006, 0x1007,
0x1008, 0x1009, 0x100A, 0x100B, 0x100C, 0x100D, 0x100E, 0x100F};

nResult =
aceBCDataBlkCreate(DevNum,DBLK1,ACE_BC_DBLK_DOUBLE,pBuffer,32);
if(nResult)
{
      printf(“Error in aceBCDataBlkCreate() function \n”);
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCDataBlkDelete()

aceBCDataBlkDelete

This function deletes a previously defined data block.

PROTOTYPE

```
#include "bcop.h"
S16BIT _DECL aceBCDataBlkDelete(S16BIT DevNum,
                                S16BIT nDataBlkID);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nDataBlkID	(input parameter) A unique S16BIT word value representing the block id to be deleted. Valid values: A previously defined ID >0

DESCRIPTION

This function removes a data block from memory and frees all resources associated with it.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	The nDataBlkID parameter contains an incorrect value
ACE_ERR_NODE_NOT_FOUND	The data block specified by nDataBlkID does not exist

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nDataBlkID = 1;
S16BIT nResult = 0;

aceBCDataBlkDelete(DevNum,nDataBlkID);

if(nResult)
```

aceBCDataBlkDelete (continued)

```
{
      printf("Error in aceBCDataBlkDelete() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCDataBlkCreate()

aceBCDataBlkRead

This function reads a data block.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCDataBlkRead(S16BIT DevNum,
                              S16BIT nDataBlkID,
                              U16BIT *pBuffer,
                              U16BIT wBufferSize,
                              U16BIT wOffset);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nDataBlkID	(input parameter) A unique S16BIT word value representing the data block id to be read Valid values: A previously defined data block ID >0
pBuffer	(output parameter) Pointer to a buffer that will hold the data read from the data block
wBufferSize	(input parameter) Number of words to be read from the data block Valid values: Must be < buffer size and data block size
wOffset	(input parameter) Word number offset to start reading from the data block Valid values: Must be < data block size

aceBCDataBlkRead (continued)

DESCRIPTION

This function reads a data block to a buffer given the data block ID. The number of words will be read starting at an offset from the beginning of the data block. The buffer will contain the raw unformatted 16-bit words contained in the data block.

RETURN VALUE

S16BIT	The number of data words read
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	The nDataBlkID, wBufferSize, wOffset, and/or pBuffer parameter(s) input by the user contain an incorrect value
ACE_ERR_NODE_NOT_FOUND	The data block ID specified by the nDataBlkID parameter input by the user does not exist

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nDataBlkID = 1;
S16BIT nResult = 0;
U16BIT pBuffer[32];
U16BIT wBufferSize = 20;
U16BIT wOffset = 10;

NResult = aceBCDataBlkRead(DevNum, nDataBlkID, pBuffer, wBufferSize,
                           wOffset);

if(nResult)
{
      printf("Error in aceBCDataBlkRead() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCDataBlkWrite()

aceBCDataBlkRead32

This function reads a data block 32-bits at a time from the EMACE device.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCDataBlkRead32(S16BIT DevNum,
                                S16BIT nDataBlkID,
                                U16BIT *pBuffer,
                                U16BIT wBufferSize,
                                U16BIT wOffset);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nDataBlkID	(input parameter) A unique S16BIT word value representing the data block id to be read Valid values: A previously defined data block ID >0
pBuffer	(output parameter) Pointer to a buffer that will hold the data read from the data block
wBufferSize	(input parameter) Number of words to be read from the data block Valid values: Must be < buffer size and data block size
wOffset	(input parameter) Word number offset to start reading from the data block Valid values: Must be < data block size

aceBCDataBlkRead32 (continued)

DESCRIPTION

This function reads a data block to a buffer given the data block ID. The number of words will be read starting at an offset from the beginning of the data block. The buffer will contain the raw unformatted 16-bit words contained in the data block. This function will read memory 32-bits at a time. When compared to aceBCDataBlkRead, this function makes more efficient use of PCI transfers. This is beneficial for real time systems, having the effect of lowering the demand on the host CPU.

RETURN VALUE

S16BIT	The number of data words read
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	The nDataBlkID, wBufferSize, wOffset, and/or pBuffer parameter(s) input by the user contain an incorrect value
ACE_ERR_NODE_NOT_FOUND	The data block ID specified by the nDataBlkID parameter input by the user does not exist

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nDataBlkID = 1;
S16BIT nResult = 0;
U16BIT pBuffer[32];
U16BIT wBufferSize = 20;
U16BIT wOffset = 10;

NResult = aceBCDataBlkRead32(DevNum, nDataBlkID, pBuffer, wBufferSize,
                             wOffset);

if(nResult)
{
      printf("Error in aceBCDataBlkRead() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCDataBlkRead() aceBCDataBlkWrite()

aceBCDataBlkWrite

This function will write data to a data block.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCDataBlkWrite(S16BIT DevNum,
                               S16BIT nDataBlkID,
                               U16BIT *pBuffer,
                               U16BIT wBufferSize,
                               U16BIT wOffset);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nDataBlkID	(input parameter) A unique S16BIT word value representing the block id to be written to Valid values: A previously defined ID >0
pBuffer	(input parameter) Pointer to a buffer that holds the data that will get written to the data block
wBufferSize	(input parameter) Number of words to be written to the data block Valid values: Must be < buffer size and data block size
wOffset	(input parameter) Data block word number offset to start writing into the data block Valid values: Must be < data block size

aceBCDataBlkWrite (continued)

DESCRIPTION

This function writes data to a data block from a user provided buffer given the data block ID. The number of words will be written starting at the user specified offset from the beginning of the data block.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	The nDataBlkID, wBufferSize, wOffset, and/or pBuffer parameters input by the user contain an incorrect value
ACE_ERR_NODE_NOT_FOUND	The data block ID specified by the nDataBlkID parameter input by the user does not exist

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nDataBlkID = 1;
S16BIT nResult = 0;
U16BIT wBufferSize = 32;
U16BIT wOffset = 10;
U16BIT pBuffer1[32] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0xA, 0xB, 0xC, 0xD,
0xE,  0xF,  0x1001,  0x1002,  0x1003,  0x1004,  0x1005,  0x1006,  0x1007,
0x1008, 0x1009, 0x100A, 0x100B, 0x100C, 0x100D, 0x100E, 0x100F};

nResult = aceBCDataBlkWrite(DevNum, nDataBlkID, pBuffer1, wBufferSize,
                            wOffset);

if(nResult)
{
      printf("Error in aceBCDataBlkWrite() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCDataBlkRead()

aceBCDecodeRawMsg

This function converts raw 16-bit messages to formatted fields in a MSGSTRUCT structure.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCDecodeRawMsg(S16BIT DevNum,
                               U16BIT *pBuffer,
                               MSGSTRUCT *pMsg);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum
(input parameter)
Logical Device Number
Valid values:
0 – 31

pBuffer
(input parameter)
This is a pointer to the buffer that contains the raw 16-bit stack message information.

pMsg
(output parameter)
This is a pointer to a MSGSTRUCT structure that will contain the decoded message information formatted in the fields contained in the MSGSTRUCT structure. The table below lists all member variables that exist in the MSGSTRUCT structure along with their definition.

Member Variable Name	Definition
wBlkSts	Contains the block status word of the message
wTimeTag	Contains the time tag of the message
wCmdWrd1	Contains the command word
wCmdWrd2	Contains the second command word for RT to RT transfers
wCmdWrd1Flg	Indicates the validity of the first command word (1 = valid)
wCmdWrd2Flg	Indicates the validity of the second command word (1 = valid)
wStsWrd1	Contains first status word
wStsWrd2	Contains second status word

aceBCDecodeRawMsg (continued)

Member Variable Name	Definition
wStsWrd1Flg	Indicates the validity of the first status word (1 = valid)
wStsWrd2Flg	Indicates the validity of the second status word (1 = valid)
wWordCount	Contains the number of valid data words
adataWrds[32]	An array that will contain the data words
wBCCtrlWrd	Contains the BC Control Word for BC mode messages only
wBCGapTime	Contains the message gap time only for BC
wBCLoopBack1	Contains the first looped back word for BC mode messages only
wBCLoopBack2	Contains the second looped back word for BC mode messages only
wBCLoopBack1Flg	Indicates validity of first loop back word for BC mode message only
wBCLoopBack2Flg	Indicates validity of second loop back word for BC mode message only

DESCRIPTION

This function takes a buffer and decodes the raw message it contains into a decoded MSGSTRUCT structure. The raw message would be the data read directly from the Bus Controller messages. The MSGSTRUCT structure contains easily addressable elements of the message.

RETURN VALUE

ACE_ERR_SUCCESS	This function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_MSGSTRUCT	The pMsg parameter input by the user is Null
ACE_ERR_INVALID_BUF	The pBuffer parameter input by the user is Null

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
MSGSTRUCT *pMsg;
U16BIT *pBuffer;

nResult = aceBCDecodeRawMsg(DevNum, pBuffer, pMsg);

if(nResult)
{
      printf("Error in aceBCDataBlkWrite() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCGetMsgFromIDRaw()   aceBCGetMsgFromIDDecoded()

aceBCFrameCreate

This function will create a frame for the BC.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCFrameCreate(S16BIT DevNum,
                              S16BIT nFrameID,
                              U16BIT wFrameType,
                              S16BIT aOpCodeIDs,
                              U16BIT wOpCodeCount,
                              U16BIT wMnrFrmTime,
                              U16BIT wFlags);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

nFrameID (input parameter)
A unique S16BIT word provided by the user that is used to identify the frame that is being created
Valid values:
Must be >= 0

wFrameType (input parameter)
Type of frame item being created
Valid values:
ACE_FRAME_MAJOR
ACE_FRAME_MINOR

aOpCodeIDs (input parameter)
The address of an S16BIT word array that contains the handles of each of the opcodes to be used for the frame

wOpCodeCount (input parameter)
A count of the number of opcodes in the aOpCodeIDs list input by the user
Valid values:
Must be > 0

aceBCFrameCreate (continued)

wMnrFrmTime (input parameter)
This is the time in μs that the frame will take to complete.
The least significant value is 100 μSec.

wFlags (input parameter)
Special Options
Valid values:

0
Allows the library to use default options.

NOTE: *If a value other than zero is input to the wMnrfrmTime and/or the wFlags parameter and the frame is specified to be a major frame by inputting ACE_FRAME_MAJOR to the wFrameType input parameter, then these values will be used for all frames (major and minor).*

ACE_BC_MNRFRM_IRQ_DISABLE
Disables the Enhanced Mini-ACE runtime library from generating an interrupt to call the internal ISR and dequeue the GPQ at the end of this frame.

DESCRIPTION

This function creates a frame from an array of opcode Ids input by the user. The frame is not totally resolved (IDs are written to memory instead of addresses). The resolution of addresses occurs at run time or during creation of the binary image files.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_PARAMETER	The nFrameID, wFrameType, and/or wOpCodeCount parameter(s) contains an incorrect value
ACE_ERR_BC_DBLK_EXISTS	The frame specified by the nFrameID parameter input by the user already exists
ACE_ERR_BC_DBLK_ALLOC	The frame failed to be configured
ACE_ERR_UNRES_OPCODE	The opcode input by the user in the nOpCodeIDs parameter does not exist and must be created by calling the aceBCOpcodeCreate() function
ACE_ERR_MEMMGR_FAIL	Memory for the frame could not be allocated

aceBCFrameCreate (continued)

EXAMPLE

```
//define data blocks
#define DBLK1     1
#define DBLK2     2
#define DBLK3     3

//define message constants
#define MSG1      1

//define opcodes
#define OP1       1
#define OP2       2
#define OP3       3

//define frame constants
#define MNR1      1
#define MJR       3
S16BIT nResult = 0;
S16BIT DevNum = 0;

U16BIT pBuffer[32] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0xA, 0xB, 0xC, 0xD,
0xE,  0xF,  0x1001,  0x1002,  0x1003,  0x1004,  0x1005,  0x1006,  0x1007,
0x1008, 0x1009, 0x100A, 0x100B, 0x100C, 0x100D, 0x100E, 0x100F};


// create data block
aceBCDataBlkCreate(DevNum,DBLK1,32,pBuffer,32);
aceBCDataBlkCreate(DevNum,DBLK2,32,NULL,0);
aceBCDataBlkCreate(DevNum,DBLK3,32,NULL,0);

// Create message block
aceBCMsgCreateBCtoRT(DevNum,        // Device number
                     MSG1,          // Message ID to create
                     DBLK1,         // Message will use this data block
                     5,             // RT address
                     1,             // RT subaddress
                     10,            // Word count
                     0,             // Default message timer
                     ACE_BCCTRL_CHL_A);   // use chl A options
```

aceBCFrameCreate (continued)

```
// Create XEQ opcode that will use msg block
aceBCOpCodeCreate(DevNum,OP1,ACE_OPCODE_XEQ,ACE_CNDTST_ALWAYS,MSG1,0,
                  0);

// Create CAL opcode that will call mnr frame from major
aceBCOpCodeCreate(DevNum,OP2,ACE_OPCODE_CAL,ACE_CNDTST_ALWAYS,MNR1,0,
                  0);

// create a minor frame
aOpCodes[0] = OP1;
nResult = aceBCFrameCreate(DevNum, MNR1, ACE_FRAME_MINOR, aOpCodes, 1,
                           0, 0);

if(nResult)
{
      printf("Error in aceBCFrameCreate() function \n");
      PrintOutError(nResult);
      return;
}

/* create a major frame to call the minor frame 2 times */
aOpCodes[0] = OP2;
aOpCodes[1] = OP2;

nResult = aceBCFrameCreate(DevNum, MJR, ACE_FRAME_MAJOR, aOpCodes, 2,
                           10, 0);

if(nResult)
{
      printf("Error in aceBCFrameCreate() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCFrameDelete()

aceBCFrameDelete

This function will delete a frame.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCFrameDelete(S16BIT DevNum,
                              S16BIT nFrameID);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nFrameID	(input parameter) A unique S16BIT user supplied ID number identifying the frame to delete Valid values: Must be > 0

DESCRIPTION

This function deletes a frame. The frame must have been previously created for this device by a call to the aceBCFrameCreate() function. All data structures and memory resources required for the frame and opcodes will be released.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_PARAMETER	The nFrameID parameter input by the user contains a value less than zero
ACE_ERR_NODE_NOT_FOUND	The frame specified by the nFrameID parameter input by the user does not exist

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
S16BIT MNR1 = 1;
```

aceBCFrameDelete (continued)

```
U16BIT aOpCodes[1];

// create a minor frame
aOpCodes[0] = OP1;

nResult = aceBCFrameCreate(DevNum, MNR1, ACE_FRAME_MINOR, aOpCodes, 1,
                           0, 0);

nResult = aceBCFrameDelete(DevNum, MNR1);

if(nResult)
{
      printf("Error in aceBCFrameDelete() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCFrameCreate()

aceBCFrmToHBuf

This function writes messages to a Host Buffer from a minor frame.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCFrmToHBuf(S16BIT DevNum);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function reads the latest messages from the minor frame and writes them out to a Host Buffer provided by the user. The routine will determine which minor frame needs servicing, and handle the transfer. The host buffer will be created using a library routine, and then linked to the device. This routine will retrieve the buffer from the device structure, so the only parameter required is the device number.

The Enhanced Mini-ACE runtime library contains an internal Interrupt Service Routine that will get triggered if a time tag rollover has occurred when running in bus controller mode or if the end of a minor frame has been reached. The end of minor frame interrupt can be disabled in the aceBCFrameCreate() function. The internal Interrupt Service Routine will call the aceBCFrmToHBuf () function.

The aceBCFrmToHBuf () function is used to read all messages from a frame ID to the BC host buffer. This function can be called by the user in BC mode or can be left as an operation performed by the Enhanced Mini-ACE runtime library to reliably transfer data to the host buffer. DDC recommends that the user **not** call the aceBCFrmToHBuf() function and allow the library to internally generate interrupts and transfer data to the host buffer by calling the aceBCFrmToHBuf() function. This function is made available to the advanced user that would like to transfer data to the host buffer more often and is aware of the internal mechanisms that the Enhanced Mini-ACE runtime library is performing or for systems where interrupts are not present.

aceBCFrmToHBuf (continued)

Each call to aceBCFrmToHBuf () contains an internal function that is used to read all entries currently on the hardware General Purpose Queue to the internal GPQ buffers. This transfer is done inside of a critical section. During this read all entries are parsed into their correct buffer (User or Lib). Once an entry is read from the hardware General Purpose Queue it is taken off of the queue.

NOTE: *If no buffer is created and attached to the device, this routine will still return with the ACE_ERR_SUCCESS* value*.*

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state

EXAMPLE

```
DevNum = 0;

// Create and install host buffer
aceBCInstallHBuf(DevNum,8*1024);

nResult = aceBCFrmToHBuf(DevNum);

if(nResult)
{
      printf("Error in aceBCFrmToHBuf() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCFrmToHBuf32()

aceBCInstallHBuf()

aceBCFrmToHBuf32

This function writes messages to a Host Buffer from a minor frame 32-bits at a time.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCFrmToHBuf32(S16BIT DevNum);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function reads the latest messages from the minor frame and writes them out to a Host Buffer provided by the user. The routine will determine which minor frame needs servicing, and handle the transfer. The host buffer will be created using a library routine, and then linked to the device. This routine will retrieve the buffer from the device structure, so the only parameter required is the device number.

The Enhanced Mini-ACE runtime library contains an internal Interrupt Service Routine that will get triggered if a time tag rollover has occurred when running in bus controller mode or if the end of a minor frame has been reached. The end of minor frame interrupt can be disabled in the aceBCFrameCreate() function. The internal Interrupt Service Routine will call the aceBCFrmToHBuf32 () function for all devices except for the BU-65567/68 PC/104 cards and the BU-65553 cards because these cards are ISA devices that will use the aceBCFrmToHBuf() function to create 16-bit memory accesses.

The aceBCFrmToHBuf32 () function is used to read all messages from a frame ID to the BC host buffer. This function can be called by the user in BC mode or can be left as an operation performed by the Enhanced Mini-ACE runtime library to reliably transfer data to the host buffer. DDC recommends that the user **not** call the aceBCFrmToHBuf32() function and allow the library to internally generate interrupts and transfer data to the host buffer by calling the aceBCFrmToHBuf32() function. This function is made available to the advanced user that would like to transfer data to the host buffer more often and is aware of the internal mechanisms that the Enhanced Mini-ACE runtime library is performing or for systems where interrupts are not present.

aceBCFrmToHBuf32 (continued)

Each call to aceBCFrmToHBuf () contains an internal function that is used to read all entries currently on the hardware General Purpose Queue to the internal GPQ buffers. This transfer is done inside of a critical section. During this read all entries are parsed into their correct buffer (User or Lib). Once an entry is read from the hardware General Purpose Queue it is taken off of the queue.

NOTE: *If no buffer is created and attached to the device, this routine will still return with the ACE_ERR_SUCCESS* value.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state

EXAMPLE

```
DevNum = 0;

// Create and install host buffer
aceBCInstallHBuf(DevNum,8*1024);

nResult = aceBCFrmToHBuf(DevNum);

if(nResult)
{
      printf("Error in aceBCFrmToHBuf() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCFrmToHBuf() aceBCInstallHBuf()

aceBCGetConditionCode

This function allows the user to read the BC Condition Code register to obtain the current condition.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCGetConditionCode(S16BIT DevNum,
                                   U16BIT wConditionCode,
                                   U16BIT *pCurrentState);
```

STATE

Run

MODE

BC

PARAMETERS

DevNum
(input parameter)
Logical Device Number
Valid values:
0 – 31

wConditionCode
(input parameter)
Condition code for which the current state is desired.
Valid values:

ACE_CND_LT_GT
This will check bit 0 which is the equal flag indicator.

ACE_CND_EQ_NE
This parameter will check bit 1 which is the less than flag indicator.

ACE_CND_GPF0
ACE_CND_GPF1
ACE_CND_GPF2
ACE_CND_GPF3
ACE_CND_GPF4
ACE_CND_GPF5
ACE_CND_GPF6
ACE_CND_GPF7
These parameters individually check the general purpose flag bits 0-7.

aceBCGetConditionCode (continued)

ACE_CND_NORES
This parameter will check the no response bit 8. If bit 8 is logic 1, this indicates that the RT did not respond or did not respond on time causing a timeout condition.

ACE_CND_FMT_ERR
This parameter will check the format error bit 9. If bit 9 is logic 1, this indicates that the received portion of the most recent message contained one or more violations of the 1553 validation criteria or the status word received from a responding RT contained an incorrect RT address field.

ACE_CND_GD_DATA
This parameter will check the good block transfer bit 10. For the most recent message, this bit will be set to logic 1 following completion of a valid (error-free) RT-to-BC transfer, RT-to-RT transfer, or transmit mode code with data message.

ACE_CND_MSKED_STS
This parameter will check the masked status set bit 11. This bit will be set if one or both of the following conditions have occurred on the most recent message: (1) If one (or more) of the Status Mask bits (14 through 9) in the BC Control Word is logic 0 and the corresponding bit(s) is/are set to logic 1 in the received RT Status Word. In the case of the RESERVED BITS MASK (bit 9) set to logic "0," any or all of the 3 Reserved status word bits being set will result in a MASKED STATUS SET condition; and/or (2) If BROADCAST MASK ENABLED/XOR* (bit 11 of Configuration Register #4) is logic "0" and the logic sense of the MASK BROADCAST bit of the message's BC Control Word and the BROADCAST COMMAND RECEIVED bit in the received RT Status Word are opposite; or (3) If BROADCAST MASK ENABLED/XOR* (bit 11 of Configuration Register #4) is logic "1" and the MASK BROADCAST bit of the message's BC Control Word is logic "0" and the BROADCAST COMMAND RECEIVED bit in the received RT Status Word is logic "1."

ACE_CND_BAD_MSG
This parameter will check the status of the bad message bit 12. If this bit is set to a logic 1, this indicates either a format error, loop test fail, or no response error for the most recent message.

aceBCGetConditionCode (continued)

	ACE_CND_RETRY This parameter will check the retry bits 14 and 13. These two bits reflect the retry status of the most recent message. The number of times that the message was retried is delineated by these two bits. A 00 represents 0 retry messages, a 01 represents 1 retry message, a 11 represents 2 retry messages, and a 10 is not applicable.
	ACE_CND_ALWAYS This parameter will check the always bit 15. This bit will always return a value of logic 1.
PCurrentState	The bit read off of the BC Condition Code Register at memory location 0x1B Valid values: 0 – 1

DESCRIPTION

This function allows the user to read the BC Condition Code Register at memory location 0x1B to obtain the current state of a condition code. Each condition code input by the user checks a specific bit location in the BC Condition Code Register to return the appropriate value of the condition. The function will first check bit 2 of Configuration Register # 1 at memory location 0x01 to make sure that the BC is enabled. If the BC is not enabled, ACE_ERR_INVALID_STATE will be returned back to the user. If the BC is enabled the function will check the appropriate bit location specified by the wConditionCode parameter input by the user.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Run state and/or the BC is not enabled
ACE_ERR_PARAMETER	The wConditionCode parameter contains a value greater than 15 and/or the pCurrentState parameter is NULL

EXAMPLE

```
DevNum = 0;
U16BIT pCurrentState;

nResult = aceBCGetConditionCode(DevNum, ACE_CND_NORES,
                                &pCurrentState);
if(nResult)
{
      printf("Error in aceBCGetConditionCode() function \n");
      PrintOutError(nResult);
      return;
}
```

aceBCGetConditionCode (continued)

SEE ALSO

None

aceBCGetGPQMetric

This function allows the user to get metrics for the GPQ.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCGetGPQMetric(S16BIT DevNum,
                               GPQMETRIC *pGPQMetric,
                               U16BIT bReset);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
pMetric	(output parameter) Pointer to a GPQMETRIC structure to be filled in with metrics. The GPQMETRIC structure contains the following members: dwLost, wPctFull, and wHighPct. The dwLost member parameter contains the total number of messages lost in the GPQ. The wPctFull member parameter contains the current percentage of the GPQ being used at one snapshot in time. The wHighPct parameter contains the highest percentage of the GPQ used over an extended period of time.
bReset	(input parameter) This will specify if the highest percentage should be reset after this function returns. Valid values: FALSE (0) Do not reset the highest percentage value TRUE (1) Reset the highest percentage value

DESCRIPTION

This function returns performance information about the General Purpose Queue. The built in metrics can report the total number of messages lost in the General Purpose Queue, the current percentage of the General Purpose Queue that is used, and the highest percentage of the General Purpose Queue used over an extended period of time.

aceBCGetGPQMetric (continued)

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input to this function
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state and the function could not be completed
ACE_ERR_PARAMETER	The pMetric pointer is Null.
ACE_ERR_METRICS_NOT_ENA	Metrics are not enabled and should be set by calling the aceSetMetrics() function

EXAMPLE

```
//This will get performance metrics for the GPQ

  S16BIT DevNum = 0;
  S16BIT nResult = 0;
  GPQMETRIC sGPQMetric;

  nResult = aceBCGetGPQMetric(DevNum, &sGPQMetric, 0);

  if(nResult)
  {
        printf("Error in aceBCGetGPQMetric() function \n");
        PrintOutError(nResult);
        return;
  }

  else
  {
        printf("GPQ pct full: %d \n", sGPQMetric.wPctFull);
        printf("GPQ highest pct full: %d \n", sGPQMetric.wHighPct);
        printf("GPQ lost messages: %d \n", sGPQMetric.dwLost);
  }
```

SEE ALSO

aceSetMetrics()
aceBCGetHBufMetric()
aceRTGetHBufMetric()
aceRTGetStkMetric()
aceMTGetHBufMetric()
aceMTGetStkMetric()

aceBCGetHBufMetric

This function returns performance information about the host buffer.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCGetHBufMetric (S16BIT DevNum,
                                 BUFMETRIC *pMetric,
                                 U16BIT bReset);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

pMetric (output parameter)
Pointer to an HBUFMETRIC structure to be filled in with metrics. The HBUFMETRIC structure contains the following members: dwCount, dwLost, dwPctFull, and dwHighPct. The dwCount member parameter contains the total number of messages in the host buffer. The dwLost member parameter contains the total number of messages lost in the host buffer. The dwPctFull parameter contains the percentage of the host buffer used at one snapshot in time. The dwHighPct parameter contains the highest percentage of the host buffer used over an extended period of time.

Member Variable Name	Definition
dwCount	The number of messages in the host buffer
dwLost	The total number of messages lost since the host buffer was installed
dwPctFull	The current percentage of host buffer used
dwHighPct	The highest percentage of the host buffer used since the host buffer was installed or metrics were reset

bReset (input parameter)
This will specify if the highest percentage should be reset after this function returns.

aceBCGetHBufMetric (continued)

Valid values:

FALSE (0)
Do not reset the highest percentage value

TRUE (1)
Reset the highest percentage value

DESCRIPTION

This function returns performance information about the BC Host Buffer. Built-in test metrics can report the number of messages in the host buffer, the total number of messages lost since the host buffer was installed, the current percentage of the host buffer that is used, and the highest percentage of the host buffer used since it was installed.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	The pMetric pointer input by the user is NULL
ACE_ERR_METRICS_NOT_ENA	Metrics are not enabled and should be set by calling the aceSetMetrics() function

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
S16BIT bReset = 1;
HBUFMETRIC *pMetric;

nResult = aceBCGetHBufMetric(DevNum, pMetric, bReset)

if(nResult)
{
      printf(“Error in aceBCGetHBufMetric() function \n”);
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceSetMetrics()
aceRTGetHBufMetric()
aceMTGetHBufMetric()
aceBCGetGPQMetric()
aceRTGetStkMetric()
aceMTGetStkMetric()

aceBCGetHBufMsgCount

This function returns the number of msgs in the host buffer.

PROTOTYPE

#include “bcop.h”

S16BIT _DECL aceBCGetHBufMsgCount(S16BIT DevNum);

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function returns the number of msgs in the host buffer.

RETURN VALUE

ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
S16BIT dwCount	The number of messages that are currently in the host buffer

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT dwCount = 0;

dwCount = aceBCGetHBufMsgCount(DevNum);

if(nResult)
{
      printf(“Error in aceBCGetHBufMsgCount() function \n”);
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

None

aceBCGetHBufMsgDecoded

This function will read a decoded message from the host buffer if one exists.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCGetHBufMsgDecoded(S16BIT DevNum,
                                    MSGSTRUCT *pMsg,
                                    U32BIT *pdwMsgCount,
                                    U32BIT *pdwMsgLostHBuf,
                                    U16BIT wMsgLoc);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

pMsg (output parameter)
Pointer to the message structure (MSGSTRUCT) into which the decoded message should be returned. The table below lists all member variables that exist in the MSGSTRUCT structure along with their definition.

Member Variable Name	Definition
wBlkSts	Contains the block status word of the message
wTimeTag	Contains the time tag of the message
wCmdWrd1	Contains the command word
wCmdWrd2	Contains the second command word for RT to RT transfers
wCmdWrd1Flg	Indicates the validity of the first command word
wCmdWrd2Flg	Indicates the validity of the second command word
wStsWrd1	Contains first status word
wStsWrd2	Contains second status word

aceBCGetHBufMsgDecoded (continued)

Member Variable Name	Definition
wStsWrd1Flg	Indicates the validity of the first status word
wStsWrd2Flg	Indicates the validity of the second status word
wWordCount	Contains the number of valid data words
adataWrds[32]	An array that will contain the data words
wBCCtrlWrd	Contains the BC Control Word for BC mode messages only
wBCGapTime	Contains the message gap time only for BC
wBCLoopBack1	Contains the first looped back word for BC mode messages only
wBCLoopBack2	Contains the second looped back word for BC mode messages only
wBCLoopBack1Flg	Indicates validity of first loop back word for BC mode message only
wBCLoopBack2Flg	Indicates validity of second loop back word for BC mode message only

pdwMsgCount (output parameter)
Pointer to the variable that will contain the message count returned
Valid values:
- 1 – One message was returned
- 0 – No messages were returned

pdwMsgLostHBuf (output parameter)
The number of times that a buffer full condition was encountered

wMsgLoc (input parameter)
Defined macro describing the location the desired message should be read from. Next indicates the next unread message on the host buffer. Latest will read the latest message just processed by the BC. All messages between the last read message and the latest message will be skipped. Purge indicates the message will be taken off of the host buffer. Npurge indicates that the message will remain on the host buffer.
Valid values:

ACE_BC_MSGLOC_NEXT_PURGE
Retrieves the next message and takes it off of the host buffer

ACE_BC_MSGLOC_NEXT_NPURGE
Retrieves the next message and leaves it on the host buffer

ACE_BC_MSGLOC_LATEST_PURGE
Retrieves the current message and takes it off of the host buffer

aceBCGetHBufMsgDecoded (continued)

ACE_BC_MSGLOC_LATEST_NPURGE
Retrieves the current message and leaves it on the host buffer

DESCRIPTION

This function reads a decoded message from the host buffer if it is present. While the BC is running, messages can be moved from the EMA memory to a host buffer in raw BC format. These messages may be read and decoded into a user buffer using this routine. The decoded messages are in the form of the MSGSTRUCT structure which allows the user to easily access the relevant part of the message in an easy to read format.

NOTE: *If no buffer is created and attached to the device, the function will still return with the ACE_ERR_SUCCESS code.*

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_PARAMETER	The wMsgLoc parameter contains a value greater than three or the pdwMsgLostHBuf parameter is NULL

EXAMPLE

```
DevNum = 0;
MSGSTRUCT pMsg;
U32BIT pdwMsgCount, pdwMsgLostHBuf;
U16BIT wMsgLoc;
// Create and install host buffer
aceBCInstallHBuf(DevNum,8*1024);

// move data from stack to host buffer
aceBCFrmToHBuf(DevNum);

// read the next unread message on stack and then purge its existence
// from the stack

wMsgLoc = ACE_BC_MSGLOC_NEXT_PURGE;
nResult = aceBCGetHBufMsgDecoded(DevNum, &pMsg, &pdwMsgCount,
                                 &pdwMsgLostHBuf, wMsgLoc)
if(nResult)
{
      printf("Error in aceBCGetHBufMsgDecoded() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCInstallHBuf() aceBCFrmToHBuf()
aceBCGetHBufMsgsRaw()

aceBCGetHBufMsgsRaw

This function will read raw messages from the host buffer if they exist.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCGetHBufMsgsRaw(S16BIT DevNum,
                                 U16BIT *pBuffer,
                                 U16BIT wBufferSize,
                                 U32BIT *pdwMsgCount,
                                 U32BIT *pdwMsgLostHBuf);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
pBuffer	(output parameter) Pointer to a word buffer for message information
wBufferSize	(input parameter) Size of buffer in words
pdwMsgCount	(output parameter) Pointer to a double word buffer to be filled with the message count
pdwMsgLostHBuf	(output parameter) Pointer to a double word buffer to be filled with the number of messages lost

DESCRIPTION

This function reads as many messages as possible off of the host buffer. If no errors occur the amount of messages will be returned. The limiting factor when copying messages to the local buffer is the local buffer size and the number of messages available on the host buffer.

aceBCGetHBufMsgsRaw (continued)

NOTE: *Each message is a fixed length of ACE_MSGSIZE_BC (42) words.*

NOTE: *If no buffer is created and attached to the device, the routine will still return with the ACE_ERR_SUCCESS code.*

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully or no host buffer exists.
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_PARAMETER	An invalid parameter was input by the user

EXAMPLE

```
DevNum = 0;
MSGSTRUCT pMsg;
U32BIT pdwMsgCount, pdwMsgLostHBuf;
U16BIT pBuffer[1024], wBufferSize = 1024;
// Create and install host buffer
aceBCInstallHBuf(DevNum,8*1024);

// move data from stack to host buffer
aceBCFrmToHBuf(DevNum);

// read the next 1024 words from the host buffer
nResult = aceBCGetHBufMsgsRaw(DevNum, &pBuffer, BufferSize,
                              &pdwMsgCount, &pdwMsgLostHBuf);
if(nResult)
{
      printf("Error in aceBCGetHBufMsgsRaw() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCInstallHBuf()
aceBCFrmToHBuf()
aceBCGetHBufMsgDecoded()

aceBCGetMsgFromIDDecoded

This function will read a message on the stack based on the message ID.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCGetMsgFromIDDecoded(S16BIT DevNum,
                                      S16BIT nMsgBlkID,
                                      MSGSTRUCT *pMsg,
                                      U16BIT bPurge);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum
: (input parameter)
Logical Device Number
Valid values:
0 – 31

nMsgBlkID
: (input parameter)
The message ID that will be read and decoded
Valid values:
A previously defined message block ID >=0

pMsg
: Pointer to a MSGSTRUCT to be filled with the decoded message. The table below lists all member variables that exist in the MSGSTRUCT structure along with their definition.

Member Variable Name	Definition
wBlkSts	Contains the block status word of the message
wTimeTag	Contains the time tag of the message
wCmdWrd1	Contains the command word
wCmdWrd2	Contains the second command word for RT to RT transfers
wCmdWrd1Flg	Indicates the validity of the first command word
wCmdWrd2Flg	Indicates the validity of the second command word
wStsWrd1	Contains first status word
wStsWrd2	Contains second status word

aceBCGetMsgFromIDDecoded (continued)

Member Variable Name	Definition
wStsWrd1Flg	Indicates the validity of the first status word
wStsWrd2Flg	Indicates the validity of the second status word
wWordCount	Contains the number of valid data words
adataWrds[32]	An array that will contain the data words
wBCCtrlWrd	Contains the BC Control Word for BC mode messages only
wBCGapTime	Contains the message gap time only for BC
wBCLoopBack1	Contains the first looped back word for BC mode messages only
wBCLoopBack2	Contains the second looped back word for BC mode messages only
wBCLoopBack1Flg	Indicates validity of first loop back word for BC mode message only
wBCLoopBack2Flg	Indicates validity of second loop back word for BC mode message only

bPurge — Indicates that the message should be purged after reading
Valid values:
TRUE
FALSE

DESCRIPTION

This function reads either the next unread message or the latest msg received on the stack based on the message block ID input by the user. The function then decodes the message by placing all the info into a MSGSTRUCT structure pointed to by the pMsg parameter by calling the aceBCDecodeRawMsg function.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_MSGSTRUCT	The pMsg pointer to the MSGSTRUCT structure contains a NULL value

EXAMPLE

```
S16BIT DevNum = 0;
MSGSTRUCT pMsg;
S16BIT      nMsgBlkID = 100;

nResult = aceBCGetMsgFromIDDecoded(DevNum, nMsgBlkID,
                                   &pMsg, TRUE)

if(nResult)
{
      printf("Error in aceBCGetMsgFromIDDecoded() function \n");
      PrintOutError(nResult);
      return;
}
```

aceBCGetMsgFromIDDecoded (continued)

SEE ALSO

aceBCFrmToHBuf()

aceBCGetHBufMsgDecoded()

aceBCGetMsgFromIDRaw

This function will read a raw message based on the message block ID.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCGetMsgFromIDRaw(S16BIT DevNum,
                                  S16BIT nMsgBlkID,
                                  U16BIT *pBuffer,
                                  U16BIT bPurge);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgBlkID	(input parameter) The message ID that will be read and decoded
pBuffer	(output parameter) Pointer to a buffer that will contain the raw message from the BC
bPurge	(input parameter) Indicates that the message should be purged after reading Valid values: TRUE FALSE

DESCRIPTION

This function reads a message given its ID into the given buffer. The message is written in its raw format as read from the BC message data block structures.

NOTE: Buffer must be at least ACE_MSGSIZE_BC words long.

RETURN VALUE

0	The function failed
1	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user

aceBCGetMsgFromIDRaw (continued)

ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_BUF	The pBuffer pointer contains a NULL value

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nMsgBlkID = 15;
U16BIT pBuffer[ACE_MSGSIZE_BC];

nResult = aceBCGetMsgFromIDRaw(DevNum, nMsgBlkID, &pBuffer, TRUE)
{
      printf("Error in aceBCGetMsgFromIDDecoded() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCFrmToHBuf()
aceBCGetHBufMsgDecoded()

aceBCGPQGetCount

This function will return the number of entries currently in the general purpose queue.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCGPQGetCount(S16BIT DevNum);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function returns the number of entries currently in the general purpose queue.

Each entry consists of two unsigned 16-bit words. One 16-bit word is a wGPQHeader that contains the information on what the data is. The other 16-bit word is a wGPQData parameter that contains the data for the entry. When the user places an entry on the GPQ the wGPQHeader entry is the first entry on the queue and can be any value except for 0xFFFF or 0xFFF8. This header value will be a unique identifier for the data that the user will place on the queue. The library uses the GPQ internally and will push the 0xFFF8 or 0xFFFF header values onto the queue at the end of each minor frame or major frame respectively in BC mode of operation.

NOTE: *Library versions less than or equal to 1.5.4 returned the total number of messages in the GPQ when this function was called. Library versions greater than or equal to 1.5.5 return the total number of messages in the GPQ input only by the user. You can use Metrics to find out the total number of messages in the host buffer, the total number of messages lost since the host buffer was installed, the current percentage of the host buffer that is used, the highest percentage of the host buffer used since it was installed, the number of messages lost in the GPQ, the current percentage of the GPQ that is used, and the highest percentage of the GPQ used over an extended period of time.*

aceBCGPQGetCount (continued)

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully or there were no more entries in the general purpose queue
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceBCGPQGetCount(DevNum);

printf("number of entries on the GP Queue = %d \n", nResult);
```

SEE ALSO

None

aceBCGPQRead

This function will read the next unread entry off of the general purpose queue.

PROTOTYPE

```
#include "bcop.h"


S16BIT _DECL aceBCGPQRead(S16BIT DevNum,
                          GPQENTRY* pGPQEntry);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
pGPQEntry	(output parameter) A pointer to a GPQENTRY structure which will contain the entry read off of the General Purpose queue

DESCRIPTION

This function reads off the next unread entry off of the general purpose queue.

NOTE: Library versions less than or equal to 1.5.4 returned all messages in the GPQ when this function was called. Library versions greater than or equal to 1.5.5 return the messages in the GPQ input only by the user. You can use Metrics to find out the total number of messages in the host buffer, the total number of messages lost since the host buffer was installed, the current percentage of the host buffer that is used, the highest percentage of the host buffer used since it was installed, the number of lost messages in the GPQ, the percentage of the GPQ full, and the highest percentage of the GPQ full since the BC started.

RETURN VALUE

0	No entries were read
1	An entry was read
2	An entry was read with an overrun condition
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_PARAMETER	The pGPQEntry pointer to a GPQENTRY structure contains a NULL value

aceBCGPQRead (continued)

EXAMPLE

```
S16BIT DevNum = 0;
GPQENTRY pGPQEntry;

nResult = aceBCGPQRead(DevNum, &pGPQEntry)

if(nResult < 0)
{
      printf(“Error in aceBCGPQRead() function \n”);
      PrintOutError(nResult);
      return;
}

else
{
      switch (nResult)
      {
            case 0:  //no entries read from GPQ
                  break;
            case 1:  //One entry read from GPQ
                  break;
            case 2:  //entry read with GPQ overrun
                  break;
      }
}
```

SEE ALSO

None

aceBCInstallHBuf

This function will allocate a host buffer.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCInstallHBuf(S16BIT DevNum,
                              U32BIT dwHBufSize);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
dwHBufSize	(input parameter) Size of new buffer in 16-bit words. Valid value: 4K – 5000K 16- bit words

DESCRIPTION

This function allocates a host buffer based on the size parameter. For this function to succeed the size must be at least 4K 16-bit words and can not exceed 5000K 16-bit words. This function will enable IRQ conditions so that they may be used.

NOTE: The dwHBufSize parameter is in 16-bit words.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_HBUFSIZE	The dwHBufSize parameter is less than 4096 (4K)
ACE_ERR_HBUF	The requested memory could not be allocated for the host buffer

aceBCInstallHBuf (continued)

EXAMPLE

```
S16BIT DevNum = 0;
U32BIT dwHBufSize 8192; // 8K words buffer size

nResult = aceBCInstallHBuf(DevNum, dwHBufSize)

if(nResult)
{
      printf(“Error in aceBCInstallHBuf() function \n”);
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

None

aceBCMsgCreate

This function creates a message block to be used inside of frames.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCMsgCreate(S16BIT DevNum,
                            S16BIT nMsgBlkID,
                            S16BIT nDataBlkID1,
                            U16BIT wBCCtrlWrd1,
                            U16BIT wCmdWrd1_1,
                            U16BIT wCmdWrd1_2,
                            U16BIT wMsgGapTime1,
                            S16BIT nDataBlkID2,
                            U16BIT wBCCtrlWrd2,
                            U16BIT wCmdWrd2_1,
                            U16BIT wCmdWrd2_2,
                            U16BIT wMsgGapTime2,
                            U32BIT dwMsgOptions);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgBlkID	(input parameter) Unique ID number for the new msg block Valid values: >0
nDataBlkID1	(input parameter) Unique ID number of previously created data block Valid values: A previously created id number >0

aceBCMsgCreate (continued)

wBCCtrlWrd1 (input parameter)

The BC control word of the msg block

Valid values:

Bitwise OR'd combination of the following that will set bits in the BC Control Word Register at memory location 0x04:

ACE_BCCTRL_1553A

This parameter will write a 1 to 1553 A/B Select bit 3 to select 1553A. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 and Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for BC, RT, MT, or RTMT mode.

ACE_BCCTRL_EOM_IRQ

This parameter will write a 1 to EOM Interrupt Enable bit 4 to result in an interrupt request at the end of a message if bit 4 of Interrupt Mask Register # 1 at memory location 0x00 is set to a logic 1 by calling the aceSetIrqConditions() function. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for BC, RT, MT, or RTMT mode.

aceBCMsgCreate (continued)

ACE_BCCTRL_BCST_MSK
This parameter will write a 1 to Mask Broadcast bit 5.

If Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 is logic 0, then the "expected value" of the Broadcast Command Received bit becomes 1, rather than 0 if this parameter is chosen as an input to this function. That is, a value of logic "**0**" (rather than logic "1") for the Broadcast Command Received bit in the received RT Status Word will result in a "Status Set" condition. Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 is 0 by default.

If the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1, **and** Expanded BC Control Word Enable bit 12 of Configuration Register # 4 is a logic 1 then this parameter will set the Mask Broadcast bit 5 to a logic 1 to be used as a mask bit, rather than performing an "XOR" operation with the Broadcast Received Status Word bit.

The Expanded BC Control Word bit 12 of Configuration Register # 4 must also be programmed to a logic 1. This bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function. The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode. In this instance, a Status Set condition arising from the Broadcast Command Received RT Status bit occurs when the Mask Broadcast bit 5 is logic 0 and the Broadcast Command Received RT Status Word bit is logic 1. If Broadcast Mask ENA/XOR* is logic 1 **and** this parameter is chosen, then the value of the Broadcast Command Received bit in the received RT Status Word becomes "don't care" in affecting a "Status Set" condition.

aceBCMsgCreate (continued)

ACE_BCCTRL_SELFTST
This parameter will set the Off-Line Self-Test bit 6. If this bit is set, it enables the off-line self-test for the respective message. In an off-line self-test message, the 1553 transmitter is inhibited; there is no activity on the external 1553 bus. The off-line self-test exercises the digital protocol portion of the Enhanced Mini-ACE by routing the output of the Manchester II serial encoder directly to the decoder input of the selected bus channel. After the message has been processed, the user can determine the success or failure of the off-line self-test by reading the Loopback Word and the LOOP TEST FAIL bit of the Block Status Word.

ACE_BCCTRL_CHL_A
This parameter will set the Bus Channel A/B bit 7 to a logic 1. If this bit is set to a logic 1 the messages will be processed on 1553 bus channel A.

ACE_BCCTRL_RETRY_ENA
This parameter will set the Retry Enabled bit 8 to a logic 1. If this bit is set to a logic 1 and Retry Enabled bit 4 of Configuration Register # 1 at memory location 0x01 is set to a logic 1 and ACE_BCCTRL_CHL_A is OR'ed with this parameter then the device will attempt a message retry as the result of the Message Error bit being set in the RT Status Word. Read Enabled bit 4 of Configuration Register # 1 can be set to a logic 1 by calling the aceBCSetMsgRetry() function.

Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function. The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

aceBCMsgCreate (continued)

ACE_BCCTRL_RES_MSK
This parameter will set the Reserved Bits Mask bit 9. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If the Reserved Bits Mask bit 9 is logic 0, a Status Set condition will occur if the one or more of the 3 Reserved bits are logic 1 in the received RT Status Word.

If the Reserved Bits Mask bit 9 is logic 1, by choosing this parameter, the value of the 3 Reserved bits in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_TFLG_MSK
This parameter will set Terminal Flag bit 10 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

aceBCMsgCreate (continued)

If this parameter is selected, the value of the Terminal Flag bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SSFLG_MSK
This parameter will set Subsystem Flag Mask bit 11 to a logic 1.

Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in, the value of the Subsystem Flag bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SSBSY_MSK
This parameter will set Busy Mask bit 12 to a logic 1.
Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

aceBCMsgCreate (continued)

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true, the value of the Busy bit in the received RT Status Word becomes "don't care”, in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SREQ_MSK
This parameter will set Busy Mask bit 13 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true then the value of the Service Request bit in the received RT Status Word becomes "don't care”, in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_ME_MSK
This parameter will set Message Error Mask bit 14 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

aceBCMsgCreate (continued)

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true then the value of the Message Error bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

wCmdWrd1_1	(input parameter) Command Word 1 of the message
wCmdWrd1_2	(input parameter) Command Word 2 of msg - if RT-to-RT
wMsgGapTime1	(input parameter) The time to next msg in µseconds Valid values: >=0
nDataBlkID2	(input parameter) Dual Mode, Unique ID number of previously created data block Valid values: A previously created id number > 0
wBCCtrlWrd2	(input parameter) Dual Mode, msg #2 BC control word Valid values: Same as wBCCtrlWrd1
wCmdWrd2_1	(input parameter) Dual Mode, msg #2 Command Word 1
wCmdWrd2_2	(input parameter) Dual Mode, msg #2 Command Word 2
wMsgGapTime2	(input parameter) Dual Mode, msg #2 msg gap time in µseconds Valid values: >=0
dwMsgOptions	(input parameter) Additional options. Valid values: ACE_MSG_OPT_DOUBLE_BUFFER Dual Mode ACE_MSG_OPT_STAY_ON_ALT Stay on alternate bus after failure

DESCRIPTION

This function creates a msg block to be used inside of frames. Dual mode is used when creating a message for an execute and flip opcode. This allows both messages of the opcode to be created and encapsulated under one message.
This message is kept internal to the EMA and the only method of accessing this message after creation is by way of the ‘nMsgBlkID’.

aceBCMsgCreate (continued)

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_BC_DBLK_EXISTS	The message block specified by nMsgBlkID already exists
ACE_ERR_NODE_NOT_FOUND	The data block specified by nDataBlkID1 does not exist and should be created using the aceBCDataBlkCreate() function
ACE_ERR_BC_DBLK_SIZE	The data block size is incorrect as was specified in the aceBCDataBlkCreate() function
ACE_ERR_BC_DBLK_ALLOC	The message block could not be allocated
ACE_ERR_MEMMGR_FAIL	The required memory for the message block could not be allocated

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nMsgBlkID = 10; // unique user message id
S16BIT nDataBlkID1 = MDBLK1; // handle for data block

U16BIT wCmdWrd1_1, wBCCtrlWrd1, wCmdWrd1_2;

U16BIT wMsgGapTime1 = 100;    // 100 µsec gap time
S16BIT nDataBlkID2= MDBLK2; // handle for data block
U16BIT wBCCtrlWrd2, wCmdWrd2_1, wCmdWrd2_2);

U16BIT wMsgGapTime2= 100;     // 100 usec gap time
U32BIT dwMsgOptions = ACE_MSGOPT_DOUBLE_BUFFER;

aceCmdWordCreate (&wCmdWrd1_1, 5, ACE_TX_CMD, 11, 23);
aceCmdWordCreate (&wCmdWrd1_2, 5, ACE_TX_CMD, 11, 23);
aceCmdWordCreate (&wCmdWrd2_1, 5, ACE_TX_CMD, 12, 23);
aceCmdWordCreate (&wCmdWrd2_2, 5, ACE_TX_CMD, 12, 23);

nResult = aceBCMsgCreate(DevNum, nMsgBlkID, nDataBlkID1,
                         wBCCtrlWrd1, wCmdWrd1_1, wCmdWrd1_2,
                         wMsgGapTime1, nDataBlkID2, wBCCtrlWrd2,
                         wCmdWrd2_1, wCmdWrd2_2, wMsgGapTime2,
                         dwDualType);
if(nResult)
{
      printf("Error in aceBCMsgCreate() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCMsgDelete()

aceBCMsgCreateBcst

This function will create a Broadcast message.

PROTOTYPE

```
#include "bcop.h"


S16BIT _DECL aceBCMsgCreateBcst(S16BIT DevNum,
                                S16BIT nMsgBlkID,
                                S16BIT nDataBlkID,
                                U16BIT wSA,
                                U16BIT wWC,
                                U16BIT wMsgGapTime,
                                U32BIT dwMsgOptions);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgBlkID	(input parameter) Unique ID number for the new message block Valid values: >0
nDataBlkID	(input parameter) Unique ID number of previously created data block Valid values: >0
wSA	(input parameter) Remote Terminal subaddress
wWC	(input parameter) Message word count of message
wMsgGapTime	The time to next msg in µseconds

aceBCMsgCreateBcst (continued)

dwMsgOptions (input parameter)
The BC control word of the msg block
Valid values:

Bitwise OR'd combination of the following that will set bits in the BC Control Word Register at memory location 0x04:

ACE_BCCTRL_1553A
This parameter will write a 1 to 1553 A/B Select bit 3 to select 1553A. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for BC, RT, MT, or RTMT mode.

ACE_BCCTRL_EOM_IRQ
This parameter will write a 1 to EOM Interrupt Enable bit 4 to result in an interrupt request at the end of a message if bit 4 of Interrupt Mask Register # 1 at memory location 0x00 is set to a logic 1 by calling the aceSetIrqConditions() function. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for BC, RT, MT, or RTMT mode.

ACE_BCCTRL_BCST_MSK
This parameter will write a 1 to Mask Broadcast bit 5.

aceBCMsgCreateBcst (continued)

If Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 is logic 0, then the "expected value" of the Broadcast Command Received bit becomes 1, rather than 0 if this parameter is chosen as an input to this function. That is, a value of logic "**0**" (rather than logic "1") for the Broadcast Command Received bit in the received RT Status Word will result in a "Status Set" condition.

Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 is 0 by default.

If the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1, **and** Expanded BC Control Word Enable bit 12 of Configuration Register # 4 is a logic 1 then this parameter will set the Mask Broadcast bit 5 to a logic 1 to be used as a mask bit, rather than performing an "XOR" operation with the Broadcast Received Status Word bit.

The Expanded BC Control Word bit 12 of Configuration Register # 4 must also be programmed to a logic 1. This bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function. The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode. In this instance, a Status Set condition arising from the Broadcast Command Received RT Status bit occurs when the Mask Broadcast bit 5 is logic 0 and the Broadcast Command Received RT Status Word bit is logic 1. If Broadcast Mask ENA/XOR* is logic 1 **and** this parameter is chosen, then the value of the Broadcast Command Received bit in the received RT Status Word becomes "don't care" in affecting a "Status Set" condition.

ACE_BCCTRL_SELFTST
This parameter will set the Off-Line Self-Test bit 6. If this bit is set, it enables the off-line self-test for the respective message. In an off-line self-test message, the 1553 transmitter is inhibited; there is no activity on the external 1553 bus. The off-line self-test exercises the digital protocol portion of the Enhanced Mini-ACE by routing the output of the Manchester II serial encoder directly to the decoder input of the selected bus channel. After the

aceBCMsgCreateBcst (continued)

message has been processed, the user can determine the success or failure of the off-line self-test by reading the Loopback Word and the LOOP TEST FAIL bit of the Block Status Word.

ACE_BCCTRL_CHL_A
This parameter will set the Bus Channel A/B bit 7 to a logic 1. If this bit is set to a logic 1 the messages will be processed on 1553 bus channel A.

ACE_BCCTRL_RETRY_ENA
This parameter will set the Retry Enabled bit 8 to a logic 1. If this bit is set to a logic 1 and Retry Enabled bit 4 of Configuration Register # 1 at memory location 0x01 is set to a logic 1 and ACE_BCCTRL_CHL_A is OR'ed with this parameter then the device will attempt a message retry as the result of the Message Error bit being set in the RT Status Word. Read Enabled bit 4 of Configuration Register # 1 can be set to a logic 1 by calling the aceBCSetMsgRetry() function.

Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

ACE_BCCTRL_RES_MSK
This parameter will set the Reserved Bits Mask bit 9. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

aceBCMsgCreateBcst (continued)

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If the Reserved Bits Mask bit 9 is logic 0, a Status Set condition will occur if the one or more of the 3 Reserved bits are logic 1 in the received RT Status Word.

If the Reserved Bits Mask bit 9 is logic 1, by choosing this parameter, the value of the 3 Reserved bits in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_TFLG_MSK
This parameter will set Terminal Flag bit 10 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is selected, the value of the Terminal Flag bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SSFLG_MSK
This parameter will set Subsystem Flag Mask bit 11 to a logic 1.

aceBCMsgCreateBcst (continued)

Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in, the value of the Subsystem Flag bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SSBSY_MSK
This parameter will set Busy Mask bit 12 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true, the value of the Busy bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SREQ_MSK
This parameter will set Busy Mask bit 13 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set

aceBCMsgCreateBcst (continued)

to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true then the value of the Service Request bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_ME_MSK
This parameter will set Message Error Mask bit 14 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true then the value of the Message Error bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

DESCRIPTION

This function creates a Broadcast message. The function first calls the aceCmdWordCreate() function to create a command word to be passed to this function. All parameters will be set up for the Broadcast message based on the dwOptions input parameter and the aceBCMsgCreate() function is called to create the Broadcast message.

aceBCMsgCreateBcst (continued)

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_BC_DBLK_EXISTS	The message block specified by nMsgBlkID already exists
ACE_ERR_NODE_NOT_FOUND	The data block specified by nDataBlkID does not exist and should be created using the aceBCDataBlkCreate() function
ACE_ERR_BC_DBLK_SIZE	The data block size is incorrect as was specified in the aceBCDataBlkCreate() function
ACE_ERR_BC_DBLK_ALLOC	The message block could not be allocated
ACE_ERR_MEMMGR_FAIL	The required memory for the message block could not be allocated

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
S16BIT nMsgBlkID = 1;
S16BIT nDataBlkID = 13;

U32BIT dwMsgOptions = (ACE_BCCTRL_EOM_IRQ | ACE_BCCTRL_RETRY_ENA);
U16BIT wMsgGapTime = 150, wWC = 10, wSA = 0, wRT = 5;

nResult = aceBCMsgCreateBcst(DevNum, nMsgBlkID, nDataBlkID, wRT,
                             wSA, wWC, wMsgGapTime, dwMsgOptions)

if(nResult)
{
      printf("Error in aceBCMsgCreateBcst() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCMsgCreateBCtoRT()	aceBCMsgCreateRTtoRT()
aceBCMsgCreateRTtoRT()	aceBCMsgCreateMode()
aceBCMsgCreateBcstMode()	aceBCMsgCreate()

aceBCMsgCreateBcstMode

This function will create a Broadcast Mode Code message.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCMsgCreateBcstMode(S16BIT DevNum,
                                    S16BIT nMsgBlkID,
                                    S16BIT nDataBlkID,
                                    U16BIT wTR,
                                    U16BIT wModeCmd,
                                    U16BIT wMsgGapTime,
                                    U32BIT dwMsgOptions);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum
(input parameter)
Logical Device Number
Valid values:
0 – 31

nMsgBlkID
(input parameter)
Unique ID number for the new message block
Valid values:
>0

nDataBlkID
(input parameter)
Unique ID number of previously created data block
Valid values:
>0

wTR
(input parameter)
Message Transmit / Receive bit
Valid values:
ACE_RX_CMD
ACE_TX_CMD

wModeCmd
(input parameter)
Message Mode Code command
Valid values:
0 – 31

aceBCMsgCreateBcstMode (continued)

wMsgGapTime (input parameter)
The time to next msg in μseconds

dwOptions (input parameter)
The BC control word of the msg block
Valid values:

Bitwise OR'd combination of the following that will set bits in the BC Control Word Register at memory location 0x04:

ACE_BCCTRL_1553A
This parameter will write a 1 to 1553 A/B Select bit 3 to select 1553A. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for BC, RT, MT, or RTMT mode.

ACE_BCCTRL_EOM_IRQ
This parameter will write a 1 to EOM Interrupt Enable bit 4 to result in an interrupt request at the end of a message if bit 4 of Interrupt Mask Register # 1 at memory location 0x00 is set to a logic 1 by calling the aceSetIrqConditions() function. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for BC, RT, MT, or RTMT mode.

aceBCMsgCreateBcstMode (continued)

ACE_BCCTRL_BCST_MSK
This parameter will write a 1 to Mask Broadcast bit 5.

If Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 is logic 0, then the "expected value" of the Broadcast Command Received bit becomes 1, rather than 0 if this parameter is chosen as an input to this function. That is, a value of logic "**0**" (rather than logic "1") for the Broadcast Command Received bit in the received RT Status Word will result in a "Status Set" condition.

Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 is 0 by default.

If the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1, **and** Expanded BC Control Word Enable bit 12 of Configuration Register # 4 is a logic 1 then this parameter will set the Mask Broadcast bit 5 to a logic 1 to be used as a mask bit, rather than performing an "XOR" operation with the Broadcast Received Status Word bit.

The Expanded BC Control Word bit 12 of Configuration Register # 4 must also be programmed to a logic 1. This bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function. The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode. In this instance, a Status Set condition arising from the Broadcast Command Received RT Status bit occurs when the Mask Broadcast bit 5 is logic 0 and the Broadcast Command Received RT Status Word bit is logic 1. If Broadcast Mask ENA/XOR* is logic 1 **and** this parameter is chosen, then the value of the Broadcast Command Received bit in the received RT Status Word becomes "don't care" in affecting a "Status Set" condition.

ACE_BCCTRL_SELFTST
This parameter will set the Off-Line Self-Test bit 6. If this bit is set, it enables the off-line self-test for the respective message. In an off-line self-test message, the 1553 transmitter is inhibited; there is no activity on the external

aceBCMsgCreateBcstMode (continued)

1553 bus. The off-line self-test exercises the digital protocol portion of the Enhanced Mini-ACE by routing the output of the Manchester II serial encoder directly to the decoder input of the selected bus channel. After the message has been processed, the user can determine the success or failure of the off-line self-test by reading the Loopback Word and the LOOP TEST FAIL bit of the Block Status Word.

ACE_BCCTRL_CHL_A
This parameter will set the Bus Channel A/B bit 7 to a logic 1. If this bit is set to a logic 1 the messages will be processed on 1553 bus channel A.

ACE_BCCTRL_RETRY_ENA
This parameter will set the Retry Enabled bit 8 to a logic 1. If this bit is set to a logic 1 and Retry Enabled bit 4 of Configuration Register # 1 at memory location 0x01 is set to a logic 1 and ACE_BCCTRL_CHL_A is OR'ed with this parameter then the device will attempt a message retry as the result of the Message Error bit being set in the RT Status Word. Read Enabled bit 4 of Configuration Register # 1 can be set to a logic 1 by calling the aceBCSetMsgRetry() function.

Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 and Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

ACE_BCCTRL_RES_MSK
This parameter will set the Reserved Bits Mask bit 9. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

aceBCMsgCreateBcstMode (continued)

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If the Reserved Bits Mask bit 9 is logic 0, a Status Set condition will occur if the one or more of the 3 Reserved bits are logic 1 in the received RT Status Word.

If the Reserved Bits Mask bit 9 is logic 1, by choosing this parameter, the value of the 3 Reserved bits in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_TFLG_MSK
This parameter will set Terminal Flag bit 10 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is selected, the value of the Terminal Flag bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SSFLG_MSK
This parameter will set Subsystem Flag Mask bit 11 to a logic 1.

aceBCMsgCreateBcstMode (continued)

Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in, the value of the Subsystem Flag bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SSBSY_MSK
This parameter will set Busy Mask bit 12 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true, the value of the Busy bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

aceBCMsgCreateBcstMode (continued)

ACE_BCCTRL_SREQ_MSK
This parameter will set Busy Mask bit 13 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true then the value of the Service Request bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_ME_MSK
This parameter will set Message Error Mask bit 14 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true then the value of the Message Error bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

aceBCMsgCreateBcstMode (continued)

DESCRIPTION

This function creates a Broadcast Mode Code command. The function first calls the aceCmdWordCreate() function to create a command word to be passed to this function. All parameters will be set up for the Broadcast Mode Code command based on the dwOptions input parameter and the aceBCMsgCreate() function is called to create the Broadcast Mode Code command.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_BC_DBLK_EXISTS	The message block specified by nMsgBlkID already exists
ACE_ERR_NODE_NOT_FOUND	The data block specified by nDataBlkID does not exist and should be created using the aceBCDataBlkCreate() function
ACE_ERR_BC_DBLK_SIZE	The data block size is incorrect as was specified in the aceBCDataBlkCreate() function
ACE_ERR_BC_DBLK_ALLOC	The message block could not be allocated
ACE_ERR_MEMMGR_FAIL	The required memory for the message block could not be allocated

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
S16BIT nMsgBlkID = 1;
S16BIT nDataBlkID = 13;
U32BIT dwMsgOptions = (ACE_BCCTRL_EOM_IRQ | ACE_BCCTRL_RETRY_ENA);
U16BIT wModeCmd = 0x0001;
U16BIT wMsgGapTime = 150;

nResult = aceBCMsgCreateBcstMode(DevNum, nMsgBlkID, nDataBlkID,
                                 ACE_RX_CMD, wModeCmd, wMsgGapTime,
                                 dwMsgOptions);

if(nResult)
{
      printf("Error in aceBCMsgCreateBcstMode() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCMsgCreateBCtoRT()
aceBCMsgCreateRTtoBC()
aceBCMsgCreateBcst()
aceBCMsgCreate()
aceBCMsgCreateMode()
aceBCMsgCreateRTtoRT()
aceBCMsgCreateBcstRTtoRT()

aceBCMsgCreateBcstRTtoRT

This function will create a Broadcast RT to multiple RT's message.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCMsgCreateBcstRTtoRT(S16BIT DevNum,
                                      S16BIT nMsgBlkID,
                                      S16BIT nDataBlkID,
                                      U16BIT wSARx,
                                      U16BIT wWC,
                                      U16BIT wRTTx,
                                      U16BIT wSATx,
                                      U16BIT wMsgGapTime,
                                      U32BIT dwMsgOptions);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgBlkID	(input parameter) Unique ID number for the new message block Valid values: >0
nDataBlkID	(input parameter) Unique ID number of previously created data block Valid values: >0
wSARx	(input parameter) Receiving Remote Terminal subaddress
wWC	(input parameter) Message word count of message
wRTTx	(input parameter) Transmitting Remote Terminal address of RT

aceBCMsgCreateBcstRTtoRT (continued)

wSATx (input parameter)
Transmitting Remote Terminal subaddress

wMsgGapTime (input parameter)
The time to next msg in μseconds

dwOptions (input parameter)
The BC control word of the msg block
Valid values:

Bitwise OR'd combination of the following that will set bits in the BC Control Word Register at memory location 0x04:

ACE_BCCTRL_1553A
This parameter will write a 1 to 1553 A/B Select bit 3 to select 1553A. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for BC, RT, MT, or RTMT mode.

ACE_BCCTRL_EOM_IRQ
This parameter will write a 1 to EOM Interrupt Enable bit 4 to result in an interrupt request at the end of a message if bit 4 of Interrupt Mask Register # 1 at memory location 0x00 is set to a logic 1 by calling the aceSetIrqConditions() function. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

aceBCMsgCreateBcstRTtoRT (continued)

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for BC, RT, MT, or RTMT mode.

ACE_BCCTRL_BCST_MSK
This parameter will write a 1 to Mask Broadcast bit 5.

If Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 is logic 0, then the "expected value" of the Broadcast Command Received bit becomes 1, rather than 0 if this parameter is chosen as an input to this function. That is, a value of logic "**0**" (rather than logic "1") for the Broadcast Command Received bit in the received RT Status Word will result in a "Status Set" condition.

Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 is 0 by default.

If the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1, **and** Expanded BC Control Word Enable bit 12 of Configuration Register # 4 is a logic 1 then this parameter will set the Mask Broadcast bit 5 to a logic 1 to be used as a mask bit, rather than performing an "XOR" operation with the Broadcast Received Status Word bit.

The Expanded BC Control Word bit 12 of Configuration Register # 4 must also be programmed to a logic 1. This bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function. The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode. In this instance, a Status Set condition arising from the Broadcast Command Received RT Status bit occurs when the Mask Broadcast bit 5 is logic 0 and the Broadcast Command Received RT Status Word bit is logic 1. If Broadcast Mask ENA/XOR* is logic 1 **and** this parameter is chosen, then the value of the Broadcast Command Received bit in the received RT Status Word becomes "don't care" in affecting a "Status Set" condition.

aceBCMsgCreateBcstRTtoRT (continued)

ACE_BCCTRL_SELFTST
This parameter will set the Off-Line Self-Test bit 6. If this bit is set, it enables the off-line self-test for the respective message. In an off-line self-test message, the 1553 transmitter is inhibited; there is no activity on the external 1553 bus. The off-line self-test exercises the digital protocol portion of the Enhanced Mini-ACE by routing the output of the Manchester II serial encoder directly to the decoder input of the selected bus channel. After the message has been processed, the user can determine the success or failure of the off-line self-test by reading the Loopback Word and the LOOP TEST FAIL bit of the Block Status Word.

ACE_BCCTRL_CHL_A
This parameter will set the Bus Channel A/B bit 7 to a logic 1. If this bit is set to a logic 1 the messages will be processed on 1553 bus channel A.

ACE_BCCTRL_RETRY_ENA
This parameter will set the Retry Enabled bit 8 to a logic 1. If this bit is set to a logic 1 and Retry Enabled bit 4 of Configuration Register # 1 at memory location 0x01 is set to a logic 1 and ACE_BCCTRL_CHL_A is OR'ed with this parameter then the device will attempt a message retry as the result of the Message Error bit being set in the RT Status Word. Read Enabled bit 4 of Configuration Register # 1 can be set to a logic 1 by calling the aceBCSetMsgRetry() function.

Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

aceBCMsgCreateBcstRTtoRT (continued)

ACE_BCCTRL_RES_MSK
This parameter will set the Reserved Bits Mask bit 9. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If the Reserved Bits Mask bit 9 is logic 0, a Status Set condition will occur if the one or more of the 3 Reserved bits are logic 1 in the received RT Status Word.

If the Reserved Bits Mask bit 9 is logic 1, by choosing this parameter, the value of the 3 Reserved bits in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_TFLG_MSK
This parameter will set Terminal Flag bit 10 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

aceBCMsgCreateBcstRTtoRT (continued)

If this parameter is selected, the value of the Terminal Flag bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SSFLG_MSK
This parameter will set Subsystem Flag Mask bit 11 to a logic 1.

Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in, the value of the Subsystem Flag bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SSBSY_MSK
This parameter will set Busy Mask bit 12 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

aceBCMsgCreateBcstRTtoRT (continued)

If this parameter is passed in and the above holds true, the value of the Busy bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SREQ_MSK
This parameter will set Busy Mask bit 13 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true then the value of the Service Request bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_ME_MSK
This parameter will set Message Error Mask bit 14 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

aceBCMsgCreateBcstRTtoRT (continued)

If this parameter is passed in and the above holds true then the value of the Message Error bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

DESCRIPTION

This function creates a Broadcast RT to multiple RTs message. The function first calls the aceCmdWordCreate() function to create a command word to be passed to this function. All parameters will be set up for the Broadcast message based on the dwOptions input parameter and the aceBCMsgCreate() function is called to create the Broadcast RT to multiple RTs message.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_BC_DBLK_EXISTS	The message block specified by nMsgBlkID already exists
ACE_ERR_NODE_NOT_FOUND	The data block specified by nDataBlkID does not exist and should be created using the aceBCDataBlkCreate() function
ACE_ERR_BC_DBLK_SIZE	The data block size is incorrect as was specified in the aceBCDataBlkCreate() function
ACE_ERR_BC_DBLK_ALLOC	The message block could not be allocated
ACE_ERR_MEMMGR_FAIL	The required memory for the message block could not be allocated

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
S16BIT nMsgBlkID = 1;
S16BIT nDataBlkID = 13;
U32BIT dwMsgOptions = (ACE_BCCTRL_EOM_IRQ | ACE_BCCTRL_RETRY_ENA);
U16BIT wMsgGapTime = 150;
U16BIT wWC = 32;
U16BIT wSARx = 1;
U16BIT wSATx = 2;
U16BIT wRTTx = 11;

nResult = aceBCMsgCreateBcstRTtoRT(DevNum, nMsgBlkID, nDataBlkID,
                                   wSARx, wWC, wRTTx, wSATx,
                                   wMsgGapTime, dwMsgOptions);

if(nResult)
{
      printf(“Error in aceBCMsgCreateBcstRTtoRT () function \n”);
      PrintOutError(nResult);
      return;
}
```

aceBCMsgCreateBcstRTtoRT (continued)

SEE ALSO

aceBCMsgCreateBCtoRT()	aceBCMsgCreateBCtoRT()
aceBCMsgCreateRTtoRT()	aceBCMsgCreateMode()
aceBCMsgCreateBcst()	aceBCMsgCreateBcstMode()
aceBCMsgCreate()	

aceBCMsgCreateBCtoRT

This function creates a BC to RT message.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCMsgCreateBCtoRT(S16BIT DevNum,
                                  S16BIT nMsgBlkID,
                                  S16BIT nDataBlkID,
                                  U16BIT wRT,
                                  U16BIT wSA,
                                  U16BIT wWC,
                                  U16BIT wMsgGapTime,
                                  U32BIT dwMsgOptions);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgBlkID	(input parameter) Unique ID number for the new message block Valid values: >0
nDataBlkID	(input parameter) Unique ID number of previously created data block Valid values: >0
wRT	(input parameter) Remote Terminal address of destination RT
wSA	(input parameter) Remote Terminal subaddress
wWC	(input parameter) Message word count of message

aceBCMsgCreateBCtoRT (continued)

wMsgGapTime (input parameter)
The time to next msg in μseconds

dwOptions (input parameter)
The BC control word of the msg block
Valid values:

Bitwise OR'd combination of the following that will set bits in the BC Control Word Register at memory location 0x04:

ACE_BCCTRL_1553A
This parameter will write a 1 to 1553 A/B Select bit 3 to select 1553A. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for BC, RT, MT, or RTMT mode.

ACE_BCCTRL_EOM_IRQ
This parameter will write a 1 to EOM Interrupt Enable bit 4 to result in an interrupt request at the end of a message if bit 4 of Interrupt Mask Register # 1 at memory location 0x00 is set to a logic 1 by calling the aceSetIrqConditions() function. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for BC, RT, MT, or RTMT mode.

aceBCMsgCreateBCtoRT (continued)

ACE_BCCTRL_BCST_MSK
This parameter will write a 1 to Mask Broadcast bit 5.

If Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 is logic 0, then the "expected value" of the Broadcast Command Received bit becomes 1, rather than 0 if this parameter is chosen as an input to this function. That is, a value of logic "**0**" (rather than logic "1") for the Broadcast Command Received bit in the received RT Status Word will result in a "Status Set" condition. Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 is 0 by default.

If the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1, **and** Expanded BC Control Word Enable bit 12 of Configuration Register # 4 is a logic 1 then this parameter will set the Mask Broadcast bit 5 to a logic 1 to be used as a mask bit, rather than performing an "XOR" operation with the Broadcast Received Status Word bit.

The Expanded BC Control Word bit 12 of Configuration Register # 4 must also be programmed to a logic 1. This bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function. The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode. In this instance, a Status Set condition arising from the Broadcast Command Received RT Status bit occurs when the Mask Broadcast bit 5 is logic 0 and the Broadcast Command Received RT Status Word bit is logic 1. If Broadcast Mask ENA/XOR* is logic 1 **and** this parameter is chosen, then the value of the Broadcast Command Received bit in the received RT Status Word becomes "don't care" in affecting a "Status Set" condition.

ACE_BCCTRL_SELFTST
This parameter will set the Off-Line Self-Test bit 6. If this bit is set, it enables the off-line self-test for the respective message. In an off-line self-test message, the 1553 transmitter is inhibited; there is no activity on the external 1553 bus. The off-line self-test exercises the digital protocol portion of the Enhanced Mini-ACE by routing the

aceBCMsgCreateBCtoRT (continued)

output of the Manchester II serial encoder directly to the decoder input of the selected bus channel. After the message has been processed, the user can determine the success or failure of the off-line self-test by reading the Loopback Word and the LOOP TEST FAIL bit of the Block Status Word.

ACE_BCCTRL_CHL_A
This parameter will set the Bus Channel A/B bit 7 to a logic 1. If this bit is set to a logic 1 the messages will be processed on 1553 bus channel A.

ACE_BCCTRL_RETRY_ENA
This parameter will set the Retry Enabled bit 8 to a logic 1. If this bit is set to a logic 1 and Retry Enabled bit 4 of Configuration Register # 1 at memory location 0x01 is set to a logic 1 and ACE_BCCTRL_CHL_A is OR'ed with this parameter then the device will attempt a message retry as the result of the Message Error bit being set in the RT Status Word. Read Enabled bit 4 of Configuration Register # 1 can be set to a logic 1 by calling the aceBCSetMsgRetry() function.

Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

ACE_BCCTRL_RES_MSK
This parameter will set the Reserved Bits Mask bit 9. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

aceBCMsgCreateBCtoRT (continued)

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If the Reserved Bits Mask bit 9 is logic 0, a Status Set condition will occur if the one or more of the 3 Reserved bits are logic 1 in the received RT Status Word.

If the Reserved Bits Mask bit 9 is logic 1, by choosing this parameter, the value of the 3 Reserved bits in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_TFLG_MSK
This parameter will set Terminal Flag bit 10 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is selected, the value of the Terminal Flag bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SSFLG_MSK
This parameter will set Subsystem Flag Mask bit 11 to a logic 1.

aceBCMsgCreateBCtoRT (continued)

Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in, the value of the Subsystem Flag bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SSBSY_MSK
This parameter will set Busy Mask bit 12 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true, the value of the Busy bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

aceBCMsgCreateBCtoRT (continued)

ACE_BCCTRL_SREQ_MSK
This parameter will set Busy Mask bit 13 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true then the value of the Service Request bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_ME_MSK
This parameter will set Message Error Mask bit 14 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true then the value of the Message Error bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

aceBCMsgCreateBCtoRT (continued)

DESCRIPTION

This function creates a BC to RT message by setting all of the appropriate parameters based on the dwOptions input by the user and calling the aceBCMsgCreate() function. The options set in the dwOptions parameter will write to the BC Control Word Register of the specified device at memory offset 0x04. The default option set by this function is 1553B. If the user would like to use 1553A, the ACE_BCCTRL_1553A option must be used. If 1553A is input by the user the BC will only consider a RT responding with only a status word to be valid otherwise a valid response from a RT will consist of a status word plus a data word.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_BC_DBLK_EXISTS	The message block specified by nMsgBlkID already exists
ACE_ERR_NODE_NOT_FOUND	The data block specified by nDataBlkID1 does not exist and should be created using the aceBCDataBlkCreate() function
ACE_ERR_BC_DBLK_SIZE	The data block size is incorrect as was specified in the aceBCDataBlkCreate() function
ACE_ERR_BC_DBLK_ALLOC	The message block could not be allocated
ACE_ERR_MEMMGR_FAIL	The required memory for the message block could not be allocated

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
S16BIT nMsgBlkID = 1, nDataBlkID = 13;
U32BIT dwMsgOptions = (ACE_BCCTRL_EOM_IRQ | ACE_BCCTRL_RETRY_ENA);
U16BIT wMsgGapTime = 150, wWC = 32, wSA = 1, wRT = 5;

nResult = aceBCMsgCreateBCtoRT(S16BIT DevNum, nMsgBlkID, nDataBlkID,
                               wRT, wSA, wWC, wMsgGapTime,
                               dwMsgOptions);

if(nResult)
{
      printf("Error in aceBCMsgCreateBCtoRT() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCMsgCreate
aceBCMsgCreateRTtoRT()
aceBCMsgCreateMode()
aceBCMsgCreateBcst()
aceBCMsgCreateBcstMode()
aceBCMsgCreateRTtoBC()

aceBCMsgCreateMode

This function will create a Mode Code message.

PROTOTYPE

```
#include "bcop.h"


S16BIT _DECL aceBCMsgCreateMode(S16BIT DevNum,
                                S16BIT nMsgBlkID,
                                S16BIT nDataBlkID,
                                U16BIT wRT,
                                U16BIT wTR,
                                U16BIT wModeCmd,
                                U16BIT wMsgGapTime,
                                U32BIT dwMsgOptions);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

nMsgBlkID (input parameter)
Unique ID number for the new message block
Valid values:
>0

nDataBlkID (input parameter)
Unique ID number of previously created data block
Valid values:
>0

wRT (input parameter)
Remote Terminal address of destination RT

wTR (input parameter)
Message Transmit / Receive bit
Valid values:
ACE_RX_CMD
ACE_TX_CMD

aceBCMsgCreateMode (continued)

wModeCmd (input parameter)
Message Mode Code command
Valid values:
0 – 31

wMsgGapTime (input parameter)
The time to next msg in µseconds

dwOptions (input parameter)
The BC control word of the msg block
Valid values:
Bitwise OR'd combination of the following that will set bits in the BC Control Word Register at memory location 0x04:

ACE_BCCTRL_1553A
This parameter will write a 1 to 1553 A/B Select bit 3 to select 1553A. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for BC, RT, MT, or RTMT mode.

ACE_BCCTRL_EOM_IRQ
This parameter will write a 1 to EOM Interrupt Enable bit 4 to result in an interrupt request at the end of a message if bit 4 of Interrupt Mask Register # 1 at memory location 0x00 is set to a logic 1 by calling the aceSetIrqConditions() function. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

aceBCMsgCreateMode (continued)

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for BC, RT, MT, or RTMT mode.

ACE_BCCTRL_BCST_MSK
This parameter will write a 1 to Mask Broadcast bit 5.

If Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 is logic 0, then the "expected value" of the Broadcast Command Received bit becomes 1, rather than 0 if this parameter is chosen as an input to this function. That is, a value of logic "**0**" (rather than logic "1") for the Broadcast Command Received bit in the received RT Status Word will result in a "Status Set" condition.

Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 is 0 by default.

If the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1, **and** Expanded BC Control Word Enable bit 12 of Configuration Register # 4 is a logic 1 then this parameter will set the Mask Broadcast bit 5 to a logic 1 to be used as a mask bit, rather than performing an "XOR" operation with the Broadcast Received Status Word bit.

The Expanded BC Control Word bit 12 of Configuration Register # 4 must also be programmed to a logic 1. This bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function. The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode. In this instance, a Status Set condition arising from the Broadcast Command Received RT Status bit occurs when the Mask Broadcast bit 5 is logic 0 and the Broadcast Command Received RT Status Word bit is logic 1. If Broadcast Mask ENA/XOR* is logic 1 **and** this parameter is chosen, then the value of the Broadcast Command Received bit in the received RT Status Word becomes "don't care" in affecting a "Status Set" condition.

aceBCMsgCreateMode (continued)

ACE_BCCTRL_SELFTST
This parameter will set the Off-Line Self-Test bit 6. If this bit is set, it enables the off-line self-test for the respective message. In an off-line self-test message, the 1553 transmitter is inhibited; there is no activity on the external 1553 bus. The off-line self-test exercises the digital protocol portion of the Enhanced Mini-ACE by routing the output of the Manchester II serial encoder directly to the decoder input of the selected bus channel. After the message has been processed, the user can determine the success or failure of the off-line self-test by reading the Loopback Word and the LOOP TEST FAIL bit of the Block Status Word.

ACE_BCCTRL_CHL_A
This parameter will set the Bus Channel A/B bit 7 to a logic 1. If this bit is set to a logic 1 the messages will be processed on 1553 bus channel A.

ACE_BCCTRL_RETRY_ENA
This parameter will set the Retry Enabled bit 8 to a logic 1. If this bit is set to a logic 1 and Retry Enabled bit 4 of Configuration Register # 1 at memory location 0x01 is set to a logic 1 and ACE_BCCTRL_CHL_A is OR'ed with this parameter then the device will attempt a message retry as the result of the Message Error bit being set in the RT Status Word. Read Enabled bit 4 of Configuration Register # 1 can be set to a logic 1 by calling the aceBCSetMsgRetry() function.

Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

aceBCMsgCreateMode (continued)

ACE_BCCTRL_RES_MSK
This parameter will set the Reserved Bits Mask bit 9. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If the Reserved Bits Mask bit 9 is logic 0, a Status Set condition will occur if the one or more of the 3 Reserved bits are logic 1 in the received RT Status Word.

If the Reserved Bits Mask bit 9 is logic 1, by choosing this parameter, the value of the 3 Reserved bits in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_TFLG_MSK
This parameter will set Terminal Flag bit 10 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

aceBCMsgCreateMode (continued)

If this parameter is selected, the value of the Terminal Flag bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SSFLG_MSK
This parameter will set Subsystem Flag Mask bit 11 to a logic 1.

Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in, the value of the Subsystem Flag bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SSBSY_MSK
This parameter will set Busy Mask bit 12 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

aceBCMsgCreateMode (continued)

If this parameter is passed in and the above holds true, the value of the Busy bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SREQ_MSK
This parameter will set Busy Mask bit 13 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true then the value of the Service Request bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_ME_MSK
This parameter will set Message Error Mask bit 14 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

aceBCMsgCreateMode (continued)

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true then the value of the Message Error bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

DESCRIPTION

This function creates a Mode Code message. The function first calls the aceCmdWordCreate() function to create a command word to be passed to this function. All parameters will be set up for the BC Control Word based on the dwOptions input parameter and the aceBCMsgCreate() function is called to create the Mode Code message.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_BC_DBLK_EXISTS	The message block specified by nMsgBlkID already exists
ACE_ERR_NODE_NOT_FOUND	The data block specified by nDataBlkID does not exist and should be created using the aceBCDataBlkCreate() function
ACE_ERR_BC_DBLK_SIZE	The data block size is incorrect as was specified in the aceBCDataBlkCreate() function
ACE_ERR_BC_DBLK_ALLOC	The message block could not be allocated
ACE_ERR_MEMMGR_FAIL	The required memory for the message block could not be allocated

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nMsgBlkID = 1;
S16BIT nDataBlkID = 13;
U32BIT dwMsgOptions = ACE_BCCTRL_EOM_IRQ | ACE_BCCTRL_RETRY_ENA;
U16BIT wMsgGapTime = 150;
U16BIT wModeCmd = 0x0001;
U16BIT wRT = 5;

nResult = aceBCMsgCreateMode(DevNum, nMsgBlkID, nDataBlkID, wRT,
                             ACE_TX_CMD, wModeCmd, wMsgGapTime,
                             dwMsgOptions)

if(nResult)
{
      printf("Error in aceBCMsgCreateMode() function \n");
      PrintOutError(nResult);
      return;
}
```

aceBCMsgCreateMode (continued)

SEE ALSO

aceBCMsgCreateBCtoRT()	aceBCMsgCreateRTtoRT()
aceBCMsgCreateRTtoRT()	aceBCMsgCreateBcst()
aceBCMsgCreateBcstMode()	aceBCMsgCreate()

aceBCMsgCreateRTtoBC

This function creates a RT to BC message.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCMsgCreateRTtoBC(S16BIT DevNum,
                                  S16BIT nMsgBlkID,
                                  S16BIT nDataBlkID,
                                  U16BIT wRT,
                                  U16BIT wSA,
                                  U16BIT wWC,
                                  U16BIT wMsgGapTime,
                                  U32BIT dwMsgOptions);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgBlkID	(input parameter) Unique ID number for the new message block Valid values: >0
nDataBlkID	(input parameter) Unique ID number of previously created data block Valid values: >0
wRT	(input parameter) Remote Terminal address of destination RT
wSA	(input parameter) Remote Terminal subaddress
wWC	(input parameter) Message word count of message

aceBCMsgCreateRTtoBC (continued)

wMsgGapTime (input parameter)
The time to next msg in µseconds

dwOptions (input parameter)
The BC control word of the msg block
Valid values:

Bitwise OR'd combination of the following that will set bits in the BC Control Word Register at memory location 0x04:

ACE_BCCTRL_1553A
This parameter will write a 1 to 1553 A/B Select bit 3 to select 1553A. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for BC, RT, MT, or RTMT mode.

ACE_BCCTRL_EOM_IRQ
This parameter will write a 1 to EOM Interrupt Enable bit 4 to result in an interrupt request at the end of a message if bit 4 of Interrupt Mask Register # 1 at memory location 0x00 is set to a logic 1 by calling the aceSetIrqConditions() function. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for BC, RT, MT, or RTMT mode.

aceBCMsgCreateRTtoBC (continued)

ACE_BCCTRL_BCST_MSK
This parameter will write a 1 to Mask Broadcast bit 5.

If Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 is logic 0, then the "expected value" of the Broadcast Command Received bit becomes 1, rather than 0 if this parameter is chosen as an input to this function. That is, a value of logic "**0**" (rather than logic "1") for the Broadcast Command Received bit in the received RT Status Word will result in a "Status Set" condition.

Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 is 0 by default.

If the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1, **and** Expanded BC Control Word Enable bit 12 of Configuration Register # 4 is a logic 1 then this parameter will set the Mask Broadcast bit 5 to a logic 1 to be used as a mask bit, rather than performing an "XOR" operation with the Broadcast Received Status Word bit.

The Expanded BC Control Word bit 12 of Configuration Register # 4 must also be programmed to a logic 1. This bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function. The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode. In this instance, a Status Set condition arising from the Broadcast Command Received RT Status bit occurs when the Mask Broadcast bit 5 is logic 0 and the Broadcast Command Received RT Status Word bit is logic 1. If Broadcast Mask ENA/XOR* is logic 1 **and** this parameter is chosen, then the value of the Broadcast Command Received bit in the received RT Status Word becomes "don't care" in affecting a "Status Set" condition.

ACE_BCCTRL_SELFTST
This parameter will set the Off-Line Self-Test bit 6. If this bit is set, it enables the off-line self-test for the respective message. In an off-line self-test message, the 1553 transmitter is inhibited; there is no activity on the external

aceBCMsgCreateRTtoBC (continued)

1553 bus. The off-line self-test exercises the digital protocol portion of the Enhanced Mini-ACE by routing the output of the Manchester II serial encoder directly to the decoder input of the selected bus channel. After the message has been processed, the user can determine the success or failure of the off-line self-test by reading the Loopback Word and the LOOP TEST FAIL bit of the Block Status Word.

ACE_BCCTRL_CHL_A
This parameter will set the Bus Channel A/B bit 7 to a logic 1. If this bit is set to a logic 1 the messages will be processed on 1553 bus channel A.

ACE_BCCTRL_RETRY_ENA
This parameter will set the Retry Enabled bit 8 to a logic 1. If this bit is set to a logic 1 and Retry Enabled bit 4 of Configuration Register # 1 at memory location 0x01 is set to a logic 1 and ACE_BCCTRL_CHL_A is OR'ed with this parameter then the device will attempt a message retry as the result of the Message Error bit being set in the RT Status Word. Read Enabled bit 4 of Configuration Register # 1 can be set to a logic 1 by calling the aceBCSetMsgRetry() function.

Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

ACE_BCCTRL_RES_MSK
This parameter will set the Reserved Bits Mask bit 9. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

aceBCMsgCreateRTtoBC (continued)

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If the Reserved Bits Mask bit 9 is logic 0, a Status Set condition will occur if one or more of the 3 Reserved bits are logic 1 in the received RT Status Word.

If the Reserved Bits Mask bit 9 is logic 1, by choosing this parameter, the value of the 3 Reserved bits in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_TFLG_MSK
This parameter will set Terminal Flag bit 10 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is selected, the value of the Terminal Flag bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SSFLG_MSK
This parameter will set Subsystem Flag Mask bit 11 to a logic 1.

aceBCMsgCreateRTtoBC (continued)

Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in, the value of the Subsystem Flag bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SSBSY_MSK
This parameter will set Busy Mask bit 12 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true, the value of the Busy bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

aceBCMsgCreateRTtoBC (continued)

ACE_BCCTRL_SREQ_MSK
This parameter will set Busy Mask bit 13 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true then the value of the Service Request bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_ME_MSK
This parameter will set Message Error Mask bit 14 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true then the value of the Message Error bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

aceBCMsgCreateRTtoBC (continued)

DESCRIPTION

This function creates a RT to BC message by setting all of the appropriate parameters based on the dwOptions input by the user and calling the aceBCMsgCreate() function. The options set in the dwOptions parameter will write to the BC Control Word Register of the specified device at memory offset 0x04. The default option set by this function is 1553B. If the user would like to use 1553A, the ACE_BCCTRL_1553A option must be used. If 1553A is input by the user the BC will only consider a RT responding with only a status word to be valid otherwise a valid response from a RT will consist of a status word plus a data word.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_BC_DBLK_EXISTS	The message block specified by nMsgBlkID already exists
ACE_ERR_NODE_NOT_FOUND	The data block specified by nDataBlkID1 does not exist and should be created using the aceBCDataBlkCreate() function
ACE_ERR_BC_DBLK_SIZE	The data block size is incorrect as was specified in the aceBCDataBlkCreate() function
ACE_ERR_BC_DBLK_ALLOC	The message block could not be allocated
ACE_ERR_MEMMGR_FAIL	The required memory for the message block could not be allocated

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
S16BIT nMsgBlkID = 1, nDataBlkID = 13;
U32BIT dwMsgOptions = ACE_BCCTRL_EOM_IRQ |      ACE_BCCTRL_RETRY_ENA;
U16BIT wMsgGapTime = 150, wWC = 32, wSA = 1, wRT = 5;

nResult = aceBCMsgCreateRTtoBC(S16BIT DevNum, nMsgBlkID, nDataBlkID,
                                wRT, wSA, wWC, wMsgGapTime,
                                dwMsgOptions)

if(nResult)
{
      printf("Error in aceBCMsgCreateRTtoBC() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCMsgCreate()	aceBCMsgCreateBCtoRT()
aceBCMsgCreateRTtoRT()	aceBCMsgCreateMode()
aceBCMsgCreateBcst()	aceBCMsgCreateBcstMode()

aceBCMsgCreateRTtoRT

This function creates an RT to RT message.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCMsgCreateRTtoRT(S16BIT DevNum,
                                  S16BIT nMsgBlkID,
                                  S16BIT nDataBlkID,
                                  U16BIT wRTRx,
                                  U16BIT wSARx,
                                  U16BIT wWC,
                                  U16BIT wRTTx,
                                  U16BIT wSATx,
                                  U16BIT wMsgGapTime,
                                  U32BIT dwMsgOptions);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

nMsgBlkID (input parameter)
Unique ID number for the new message block
Valid values:
>0

nDataBlkID (input parameter)
Unique ID number of previously created data block
Valid values:
>0

wRTRx (input parameter)
Receiving Remote Terminal address of destination RT

wSARx (input parameter)
Receiving Remote Terminal subaddress

aceBCMsgCreateRTtoRT (continued)

wWC (input parameter)
Message word count of message

wRTTx (input parameter)
Transmitting Remote Terminal address of RT

wSATx (input parameter)
Transmitting Remote Terminal subaddress

wMsgGapTime (input parameter)
The time to next msg in µseconds

dwOptions (input parameter)
The BC control word of the msg block
Valid values:

Bitwise OR'd combination of the following that will set bits in the BC Control Word Register at memory location 0x04:

ACE_BCCTRL_1553A
This parameter will write a 1 to 1553 A/B Select bit 3 to select 1553A. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for BC, RT, MT, or RTMT mode.

ACE_BCCTRL_EOM_IRQ
This parameter will write a 1 to EOM Interrupt Enable bit 4 to result in an interrupt request at the end of a message if bit 4 of Interrupt Mask Register # 1 at memory location 0x00 is set to a logic 1 by calling the aceSetIrqConditions() function. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

aceBCMsgCreateRTtoRT (continued)

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for BC, RT, MT, or RTMT mode.

ACE_BCCTRL_BCST_MSK
This parameter will write a 1 to Mask Broadcast bit 5.

If Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 is logic 0, then the "expected value" of the Broadcast Command Received bit becomes 1, rather than 0 if this parameter is chosen as an input to this function. That is, a value of logic "**0**" (rather than logic "1") for the Broadcast Command Received bit in the received RT Status Word will result in a "Status Set" condition.

Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 is 0 by default.

If the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Broadcast Mask ENA/XOR* bit 11 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1, **and** Expanded BC Control Word Enable bit 12 of Configuration Register # 4 is a logic 1 then this parameter will set the Mask Broadcast bit 5 to a logic 1 to be used as a mask bit, rather than performing an "XOR" operation with the Broadcast Received Status Word bit.

The Expanded BC Control Word bit 12 of Configuration Register # 4 must also be programmed to a logic 1. This bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function. The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode. In this instance, a Status Set condition arising from the Broadcast Command Received RT Status bit occurs when the Mask Broadcast bit 5 is logic 0 and the Broadcast Command Received RT Status Word bit is logic 1. If

aceBCMsgCreateRTtoRT (continued)

Broadcast Mask ENA/XOR* is logic 1 **and** this parameter is chosen, then the value of the Broadcast Command Received bit in the received RT Status Word becomes "don't care" in affecting a "Status Set" condition.

ACE_BCCTRL_SELFTST
This parameter will set the Off-Line Self-Test bit 6. If this bit is set, it enables the off-line self-test for the respective message. In an off-line self-test message, the 1553 transmitter is inhibited; there is no activity on the external 1553 bus. The off-line self-test exercises the digital protocol portion of the Enhanced Mini-ACE by routing the output of the Manchester II serial encoder directly to the decoder input of the selected bus channel. After the message has been processed, the user can determine the success or failure of the off-line self-test by reading the Loopback Word and the LOOP TEST FAIL bit of the Block Status Word.

ACE_BCCTRL_CHL_A
This parameter will set the Bus Channel A/B bit 7 to a logic 1. If this bit is set to a logic 1 the messages will be processed on 1553 bus channel A.

ACE_BCCTRL_RETRY_ENA
This parameter will set the Retry Enabled bit 8 to a logic 1. If this bit is set to a logic 1 and Retry Enabled bit 4 of Configuration Register # 1 at memory location 0x01 is set to a logic 1 and ACE_BCCTRL_CHL_A is OR'ed with this parameter then the device will attempt a message retry as the result of the Message Error bit being set in the RT Status Word. Read Enabled bit 4 of Configuration Register # 1 can be set to a logic 1 by calling the aceBCSetMsgRetry() function.

Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 and Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

aceBCMsgCreateRTtoRT (continued)

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

ACE_BCCTRL_RES_MSK
This parameter will set the Reserved Bits Mask bit 9. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 and Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If the Reserved Bits Mask bit 9 is logic 0, a Status Set condition will occur if the one or more of the 3 Reserved bits are logic 1 in the received RT Status Word.

If the Reserved Bits Mask bit 9 is logic 1, by choosing this parameter, the value of the 3 Reserved bits in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_TFLG_MSK
This parameter will set Terminal Flag bit 10 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 and Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

aceBCMsgCreateRTtoRT (continued)

If this parameter is selected, the value of the Terminal Flag bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SSFLG_MSK
This parameter will set Subsystem Flag Mask bit 11 to a logic 1.

Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 and Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in, the value of the Subsystem Flag bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SSBSY_MSK
This parameter will set Busy Mask bit 12 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

aceBCMsgCreateRTtoRT (continued)

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true, the value of the Busy bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_SREQ_MSK
This parameter will set Busy Mask bit 13 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 and Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true then the value of the Service Request bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

ACE_BCCTRL_ME_MSK
This parameter will set Message Error Mask bit 14 to a logic 1. Applicable only if the Enhanced Mode Enabled bit 15 of Configuration Register # 3 at memory location 0x07 is set to a logic 1 **and** Expanded BC Control Word bit 12 of Configuration Register # 4 at memory location 0x08 is also programmed to a logic 1.

The Expanded BC Control Word bit can be programmed to a logic 1 by reading the contents of Configuration Register # 4 at memory location 0x08 with the aceRegRead() function and logically OR the value with the CFG4_BC_ENH_CTRL_WORD word. You can then write this value to the register with the aceRegWrite() function.

aceBCMsgCreateRTtoRT (continued)

The Enhanced Mode Enabled bit 15 of Configuration Register # 3 is set to a logic 1 by default when you configure the device for RT, MT, or RTMT mode.

If this parameter is passed in and the above holds true then the value of the Message Error bit in the received RT Status Word becomes "don't care", in terms of affecting the occurrence of a "Status Set" condition.

DESCRIPTION

This function creates an RT to RT message by setting all of the appropriate parameters based on the dwOptions input by the user and calling the aceBCMsgCreate() function. The options set in the dwOptions parameter will write to the BC Control Word Register of the specified device at memory offset 0x04. The default option set by this function is 1553B. If the user would like to use 1553A, the ACE_BCCTRL_1553A option must be used. If 1553A is input by the user the BC will only consider a RT responding with only a status word to be valid otherwise a valid response from a RT will consist of a status word plus a data word.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_BC_DBLK_EXISTS	The message block specified by nMsgBlkID already exists
ACE_ERR_NODE_NOT_FOUND	The data block specified by nDataBlkID does not exist and should be created using the aceBCDataBlkCreate() function
ACE_ERR_BC_DBLK_SIZE	The data block size is incorrect as was specified in the aceBCDataBlkCreate() function
ACE_ERR_BC_DBLK_ALLOC	The message block could not be allocated
ACE_ERR_MEMMGR_FAIL	The required memory for the message block could not be allocated

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
S16BIT nMsgBlkID = 1, nDataBlkID = 13;
U32BIT dwMsgOptions = (ACE_BCCTRL_EOM_IRQ | ACE_BCCTRL_RETRY_ENA);
U16BIT wMsgGapTime = 150;
U16BIT wWC = 32, wSARx = 1, wRTRx = 5;
U16BIT wSATx = 2, wRTTx = 11;
nResult = aceBCMsgCreateRTtoRT(DevNum, nMsgBlkID, nDataBlkID, wRTRx,
                               wSARx, wWC, wRTTx, wSATx, wMsgGapTime,
                               dwMsgOptions);
if(nResult)
{
      printf("Error in aceBCMsgCreateRTtoBC() function \n");
      PrintOutError(nResult);
      return;
}
```

aceBCMsgCreateRTtoRT (continued)

SEE ALSO

aceBCMsgCreateBCtoRT()	aceBCMsgCreateRTtoBC()
aceBCMsgCreateMode()	aceBCMsgCreateBcst()
aceBCMsgCreateBcstMode()	aceBCMsgCreate()

aceBCMsgDelete

This function deletes a previously defined message block.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCMsgDelete(S16BIT DevNum,
                            S16BIT nMsgBlkID);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgBlkID	(input parameter) A unique message ID that identifies the message to be deleted. This is the same id that was provided during the creation of the message with the aceBCMsgCreate() function Valid values: >0

DESCRIPTION

This function removes a message block from memory and frees all resources associated with it.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	The nMsgBlkID parameter contains a value less than zero
ACE_ERR_NODE_NOT_FOUND	The specified message block could not be found or it could not be deleted

aceBCMsgDelete (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nMsgBlkID = 42;

nResult = aceBCMsgDelete(DevNum, nMsgBlkID)

if(nResult)
{
      printf(“Error in aceBCMsgDelete() function \n”);
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCMsgCreate()

aceBCMsgGapTimerEnable

This function enables/disables the message gap time field for all messages.

PROTOTYPE

```
#include "bcop.h"


S16BIT _DECL aceBCMsgGapTimerEnable(S16BIT DevNum,
                                    U16BIT bEnable);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
bEnable	(input parameter) Enables global inter-message gap timer Valid values: TRUE FALSE

DESCRIPTION

This function enables the message gap time field for all messages by setting bit 5 of Configuration Register # 1 at memory location 0x01 to a 1 if the bEnable input parameter is TRUE. If the bEnable input parameter is FALSE the message gap time field will be disabled for all messages. If disabled the default message gap time of approximately 8-11 µseconds is used.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ER_INVALID_MODE	The device is not in BC mode

aceBCMsgGapTimerEnable (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceBCMsgGapTimerEnable(DevNum, TRUE);

if(nResult)
{
      printf("Error in aceBCMsgGapTimerEnable() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

None

aceBCMsgModify

This function creates an asynchronous message.

PROTOTYPE

```
#include "BCOp.h"

S16BIT _DECL aceBCMsgModify(S16BIT DevNum,
                            S16BIT nMsgBlkID,
                            S16BIT nDataBlkID1,
                            U16BIT wBCCtrlWrd1,
                            U16BIT wCmdWrd1_1,
                            U16BIT wCmdWrd1_2,
                            U16BIT wMsgGapTime1,
                            S16BIT nDataBlkID2,
                            U16BIT wBCCtrlWrd2,
                            U16BIT wCmdWrd2_1,
                            U16BIT wCmdWrd2_2,
                            U16BIT wMsgGapTime2,
                            U16BIT wModFlags);
```

STATE

Ready, Run

MODE

BC

aceBCMsgModify (continued)

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgBlkID	(input parameter) Unique ID number for the new message block Valid values: >0
nDataBlkID	(input parameter) Unique ID number for this block of data Valid values: >0
wBCCtrlWrd1	(input parameter) The BC control word of the msg block Valid values: See aceBCMsgCreate()
wCmdWrd1_1	(input parameter) Command word 1 of the message
wCmdWrd1_2	(input parameter) Command word 2 of the message if RT to RT
wMsgGapTime1	(input parameter) The time to next message in μseconds Valid values: >=0
nDataBlkID2	(input parameter) Dual Mode, Unique ID number of previously created data block Valid values: A previously created id number > 0
wBCCtrlWrd2	(input parameter) Dual Mode, msg #2 BC control word Valid values: Same as wBCCtrlWrd1

aceBCMsgModify (continued)

wCmdWrd2_1 (input parameter)
Dual Mode, msg #2 Command Word 1

wCmdWrd2_2 (input parameter)
Dual Mode, msg #2 Command Word 2

wMsgGapTime2 (input parameter)
Dual Mode, msg #2 msg gap time in ∝seconds
Valid values:
>=0

wModFlags (input parameter)
The specific parts of the message to modify
Valid values:
ACE_BC_MOD_DBLK1
ACE_BC_MOD_BCCTRL1
ACE_BC_MOD_CMDWRD1_1
ACE_BC_MOD_CMDWRD1_2
ACE_BC_MOD_GAPTIME1
ACE_BC_MOD_DBLK2
ACE_BC_MOD_BCCTRL2
ACE_BC_MOD_CMDWRD2_1
ACE_BC_MOD_CMDWRD2_2
ACE_BC_MOD_GAPTIME2

aceBCMsgModify (continued)

DESCRIPTION

This function modifies an existing msg block's parameters and options. The existing message specified by the messag block id input to this function should have previously been created using the aceBCMsgCreate() or any of the other message creation functions.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	One or more of the input parameters is/are invalid
ACE_ERR_UNRES_MSGBLK	The message id specified in this function does not exist
ACE_ERR_UNRES_DATABLK	The data id specified in this function does not exist
ACE_ERR_MEMMGR_FAIL	The new message is an RT-to-RT message and the original was not so there is not enough memory to complete this operation
ACE_ERR_BC_DBLK_SIZE	The data block size is invalid
ACE_ERR_NOT_ASYNC_MODE	The software is not in asynchronous mode. This state must be selected in the aceBCConfigure() function.

aceBCMsgModify (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nMsgBlkID = 10; // unique user message id
S16BIT nDataBlkID1 = MDBLK1; // handle for data block
U16BIT wCmdWrd1_1, wBCCtrlWrd1, wCmdWrd1_2;
U16BIT wMsgGapTime1 = 100; // 100 µsec gap time
S16BIT nDataBlkID2= MDBLK2; // handle for data block
U16BIT wBCCtrlWrd2, wCmdWrd2_1, wCmdWrd2_2);
U16BIT wMsgGapTime2= 100; // 100 usec gap time
ACE_MSGOPT_DOUBLE_BUFFER;
wBCControlWord = ACE_BCCTRL_CHL_A;

aceCmdWordCreate(&wCmdWrd1_1, 5, ACE_TX_CMD, 11, 23);

nResult = aceBCMsgModify(DevNum, nMsgBlkID, nDataBlkID1,
                         wBCControlWord, wCmdWrd1_1, 0, wMsgGapTime1,
                         0, 0, 0, 0, 0, ACE_MSGOPT_DOUBLE_BUFFER);

if(nResult)
{
printf(“Error in aceBCAsyncMsgCreate() function \n”);
PrintOutError(nResult);
return;
}
```

SEE ALSO

aceBCMsgCreate()

aceBCMsgModifyBcst

This function modifies an asynchronous broadcast message.

PROTOTYPE

```
#include "BCOp.h"

S16BIT _DECL aceBCMsgModifyBcst   (S16BIT DevNum,
                                    S16BIT nMsgBlkID,
                                    S16BIT nDataBlkID,
                                    U16BIT wSA,
                                    U16BIT wWC,
                                    U16BIT wMsgGapTime,
                                    U32BIT dwMsgOptions,
                                    U16BIT wModFlags);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgBlkID	(input parameter) Unique ID number for the new message block Valid values: >0
nDataBlkID	(input parameter) Unique ID number for this block of data Valid values: >0
wSA	(input parameter) RT subaddress

aceBCMsgModifyBcst (continued)

wWC	(input parameter) Message data word count
wMsgGapTime	(input parameter) The time to next message in µseconds Valid values: >=0
dwMsgOptions	(input parameter) The BC Control word of the message block Valid values: See aceBCMsgCreateBcst() function
wModFlags	(input parameter) The specific parts of the message to modify Valid values: ACE_BC_MOD_DBLK1 ACE_BC_MOD_BCCTRL1 ACE_BC_MOD_CMDWRD1_1 ACE_BC_MOD_CMDWRD1_2 ACE_BC_MOD_GAPTIME1 ACE_BC_MOD_DBLK2 ACE_BC_MOD_BCCTRL2 ACE_BC_MOD_CMDWRD2_1 ACE_BC_MOD_CMDWRD2_2 ACE_BC_MOD_GAPTIME2

aceBCMsgModifyBcst (continued)

DESCRIPTION

This function modifies an existing message block's parameters and options. The existing message specified by the message block id input to this function should have previously been created using the aceBCMsgCreate(), aceBCMsgCreateBcst(), or aceBCAsyncMsgCreateBcst() function calls. This message must be a broadcast message.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	One or more of the input parameters is/are invalid
ACE_ERR_UNRES_MSGBLK	The message id specified in this function does not exist
ACE_ERR_UNRES_DATABLK	The data id specified in this function does not exist
ACE_ERR_MEMMGR_FAIL	The new message is an RT-to-RT message and the original was not so there is not enough memory to complete this operation
ACE_ERR_BC_DBLK_SIZE	The data block size is invalid
ACE_ERR_NOT_ASYNC_MODE	The software is not in asynchronous mode. This state must be selected in the aceBCConfigure() function.

aceBCMsgModifyBcst (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nMsgBlkID = 10; // unique user message id
S16BIT nDataBlkID1 = MDBLK1; // handle for data block
U16BIT wSA = 1; //RT SA 1
U16BIT wWc = 32; //32 data words
U16BIT wMsgGapTime = 100; // 100 µsec gap time

nResult = aceBCMsgModifyBcst(DevNum, nMsgBlkID, nDataBlkID, wSA,
                             wWC,wMsgGapTime, ACE_MSGOPT_DOUBLE_BUFFER,
                             ACE_BC_MOD_DBLK1);

if(nResult)
{
      printf(“Error in aceBCMsgModifyBcst() function \n”);
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCMsgCreateBcst()

aceBCMsgModifyBcstMode

This function modifies an asynchronous broadcast message mode code.

PROTOTYPE

```
#include "BCOp.h"

S16BIT _DECL aceBCMsgModifyBcstMode  (S16BIT DevNum,
                                      S16BIT nMsgBlkID,
                                      S16BIT nDataBlkID,
                                      U16BIT wTR,
                                      U16BIT wModeCmd,
                                      U16BIT wMsgGapTime,
                                      U32BIT dwMsgOptions,
                                      U16BIT wModFlags);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgBlkID	(input parameter) Unique ID number for the new message block Valid values: >0
nDataBlkID	(input parameter) Unique ID number for this block of data Valid values: >0
wTR	(input parameter) The transmit receive bit of message Valid values: 0 or 1

aceBCMsgModifyBcstMode (continued)

wModeCmd	(input parameter) The mode command to be changed
wMsgGapTime	(input parameter) The time to next message in µseconds Valid values: >=0
dwOptions	(input parameter) The BC Control word of the message block Valid values: See aceBCMsgCreateBcstMode() function
wModFlags	(input parameter) The specific parts of the message to modify Valid values: ACE_BC_MOD_DBLK1 ACE_BC_MOD_BCCTRL1 ACE_BC_MOD_CMDWRD1_1 ACE_BC_MOD_CMDWRD1_2 ACE_BC_MOD_GAPTIME1 ACE_BC_MOD_DBLK2 ACE_BC_MOD_BCCTRL2 ACE_BC_MOD_CMDWRD2_1 ACE_BC_MOD_CMDWRD2_2 ACE_BC_MOD_GAPTIME2

aceBCMsgModifyBcstMode (continued)

DESCRIPTION

This function modifies an existing message block's parameters and options. The existing message specified by the message block id input to this function should have previously been created using the aceBCMsgCreate(), aceBCMsgCreateBcstMode(), or aceBCAsyncMsgCreateBcstMode() function calls. This message must be a broadcast mode code message.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	One or more of the input parameters is/are invalid
ACE_ERR_UNRES_MSGBLK	The message id specified in this function does not exist
ACE_ERR_UNRES_DATABLK	The data id specified in this function does not exist
ACE_ERR_MEMMGR_FAIL	The new message is an RT-to-RT message and the original was not so there is not enough memory to complete this operation
ACE_ERR_BC_DBLK_SIZE	The data block size is invalid
ACE_ERR_NOT_ASYNC_MODE	The software is not in asynchronous mode. This state must be selected in the aceBCConfigure() function.

aceBCMsgModifyBcstMode (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nMsgBlkID = 10; // unique user message id
S16BIT nDataBlkID1 = MDBLK1; // handle for data block
U16BIT wSA = 1; //RT SA 1
U16BIT wWc = 32; //32 data words
U16BIT wMsgGapTime = 100; // 100 µsec gap time

nResult = aceBCMsgModifyBcstMode(DevNum, nMsgBlkID, nDataBlkID, wSA,
                                 wWC,wMsgGapTime,
                                 ACE_MSGOPT_DOUBLE_BUFFER,
                                 ACE_BC_MOD_DBLK1);

if(nResult)
{
      printf("Error in aceBCMsgModifyBcstMode() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCMsgCreateBcstMode()

aceBCMsgModifyBcstRTtoRT

This function modifies an asynchronous broadcast RT to RT message.

PROTOTYPE

```
#include "BCOp.h"
S16BIT _DECL aceBCMsgModifyBcstRTtoRT  (S16BIT DevNum,
                                         S16BIT nMsgBlkID,
                                         S16BIT nDataBlkID,
                                         U16BIT wSARx,
                                         U16BIT wWC,
                                         U16BIT wRTTx,
                                         U16BIT wSATx,
                                         U16BIT wMsgGapTime,
                                         U32BIT dwMsgOptions,
                                         U16BIT wModFlags);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum
: (input parameter)
Logical Device Number
Valid values:
0 – 31

nMsgBlkID
: (input parameter)
Unique ID number for the new message block
Valid values:
>0

nDataBlkID
: (input parameter)
Unique ID number for this block of data
Valid values:
>0

wSARx
: (input parameter)
RT subaddress for receiving RT

aceBCMsgModifyBcstRTtoRT (continued)

wWC	(input parameter) The data word count
wRTTx	(input parameter) RT address for transmitting RT
wSATx	(input parameter) RT subaddress for transmitting RT
wMsgGapTime	(input parameter) The time to next message in µseconds Valid values: >=0
dwOptions	(input parameter) The BC Control word of the message block Valid values: See aceBCMsgCreateBcstRTtoRT() function
wModFlags	(input parameter) The specific parts of the message to modify Valid values: ACE_BC_MOD_DBLK1 ACE_BC_MOD_BCCTRL1 ACE_BC_MOD_CMDWRD1_1 ACE_BC_MOD_CMDWRD1_2 ACE_BC_MOD_GAPTIME1 ACE_BC_MOD_DBLK2 ACE_BC_MOD_BCCTRL2 ACE_BC_MOD_CMDWRD2_1 ACE_BC_MOD_CMDWRD2_2 ACE_BC_MOD_GAPTIME2

aceBCMsgModifyBcstRTtoRT (continued)

DESCRIPTION

This function modifies an existing message block's parameters and options. The existing message specified by the message block id input to this function should have previously been created using the aceBCMsgCreate(), aceBCMsgCreateBcstRTtoRT(), or aceBCAsyncMsgCreateBcstRTtoRT() function calls. This message must be a broadcast RT to RT message.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	One or more of the input parameters is/are invalid
ACE_ERR_UNRES_MSGBLK	The message id specified in this function does not exist
ACE_ERR_UNRES_DATABLK	The data id specified in this function does not exist
ACE_ERR_MEMMGR_FAIL	The new message is an RT-to-RT message and the original was not so there is not enough memory to complete this operation
ACE_ERR_BC_DBLK_SIZE	The data block size is invalid
ACE_ERR_NOT_ASYNC_MODE	The software is not in asynchronous mode. This state must be selected in the aceBCConfigure() function.

aceBCMsgModifyBcstRTtoRT (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nMsgBlkID = 10; // unique user message id
S16BIT nDataBlkID1 = MDBLK1; // handle for data block
U16BIT wSA = 1; //RT SA 1
U16BIT wWc = 32; //32 data words
U16BIT wMsgGapTime = 100; // 100 µsec gap time

nResult = aceBCMsgModifyBcstRTtoRT(DevNum, nMsgBlkID, nDataBlkID, 1, 1,
                                   2, 2, wMsgGapTime,
                                   ACE_BCCTRL_CHL_A,
                                   ACE_BC_MOD_DBLK1);

if(nResult)
{
      printf("Error in aceBCMsgModifyBcstRTtoRT() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCMsgCreateBcstRTtoRT()

aceBCMsgModifyBCtoRT

This function modifies an asynchronous BC to RT receive message.

PROTOTYPE

#include “BCOp.h”

```
S16BIT _DECL aceBCMsgModifyBCtoRT    (S16BIT DevNum,
                                      S16BIT nMsgBlkID,
                                      S16BIT nDataBlkID,
                                      U16BIT wRT,
                                      U16BIT wSA,
                                      U16BIT wWC,
                                      U16BIT wMsgGapTime,
                                      U32BIT dwMsgOptions,
                                      U16BIT wModFlags);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgBlkID	(input parameter) Unique ID number for the new message block Valid values: >0
nDataBlkID	(input parameter) Unique ID number for this block of data Valid values: >0
wRT	(input parameter) RTaddress

aceBCMsgModifyBCtoRT (continued)

wSA	(input parameter) RT subaddress
wWC	(input parameter) RT receive message data word count
wMsgGapTime	(input parameter) The time to next message in µseconds Valid values: >=0
dwOptions	(input parameter) The BC Control word of the message block Valid values: See aceBCMsgCreateBCtoRT() function
wModFlags	(input parameter) The specific parts of the message to modify Valid values: ACE_BC_MOD_DBLK1 ACE_BC_MOD_BCCTRL1 ACE_BC_MOD_CMDWRD1_1 ACE_BC_MOD_CMDWRD1_2 ACE_BC_MOD_GAPTIME1 ACE_BC_MOD_DBLK2 ACE_BC_MOD_BCCTRL2 ACE_BC_MOD_CMDWRD2_1 ACE_BC_MOD_CMDWRD2_2 ACE_BC_MOD_GAPTIME2

aceBCMsgModifyBCtoRT (continued)

DESCRIPTION

This function modifies an existing message block's parameters and options. The existing message specified by the message block id input to this function should have previously been created using the aceBCMsgCreate(), aceBCMsgCreateBCtoRT(), or aceBCAsyncMsgCreateBCtoRT() function calls. This message must be a BC to RT receive message.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	One or more of the input parameters is/are invalid
ACE_ERR_UNRES_MSGBLK	The message id specified in this function does not exist
ACE_ERR_UNRES_DATABLK	The data id specified in this function does not exist
ACE_ERR_MEMMGR_FAIL	The new message is an RT-to-RT message and the original was not so there is not enough memory to complete this operation
ACE_ERR_BC_DBLK_SIZE	The data block size is invalid
ACE_ERR_NOT_ASYNC_MODE	The software is not in asynchronous mode. This state must be selected in the aceBCConfigure() function.

aceBCMsgModifyBCtoRT (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nMsgBlkID = 10; // unique user message id
S16BIT nDataBlkID1 = MDBLK1; // handle for data block
U16BIT wRT = 1;
U16BIT wSA = 1; //RT SA 1
U16BIT wWC = 32; //32 data words
U16BIT wMsgGapTime = 100; // 100 µsec gap time

nResult = aceBCMsgModifyBCtoRT(DevNum, nMsgBlkID, nDataBlkID, wRT, wSA,
                                   wWC,wMsgGapTime,
                                   ACE_BCCTRL_CHL_A,
                                   ACE_BC_MOD_DBLK1);

if(nResult)
{
      printf("Error in aceBCMsgModifyBCtoRT() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCMsgCreateBCtoRT()

aceBCMsgModifyMode

This function modifies an asynchronous Mode code message.

PROTOTYPE

```
#include "BCOp.h"

S16BIT _DECL aceBCMsgModifyMode  (S16BIT DevNum,
                                  S16BIT nMsgBlkID,
                                  S16BIT nDataBlkID,
                                  U16BIT wRT,
                                  U16BIT wTR,
                                  U16BIT wModeCmd,
                                  U16BIT wMsgGapTime,
                                  U32BIT dwMsgOptions,
                                  U16BIT wModFlags);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgBlkID	(input parameter) Unique ID number for the new message block Valid values: >0
nDataBlkID	(input parameter) Unique ID number for this block of data Valid values: >0
wRT	(input parameter) Rtaddress

aceBCMsgModifyMode (continued)

wTR	(input parameter) The transmit/receive bit
wWC	(input parameter) RT message data word count
wMsgGapTime	(input parameter) The time to next message in μseconds Valid values: >=0
dwOptions	(input parameter) The BC Control word of the message block Valid values: See aceBCMsgCreateMode() function
wModFlags	(input parameter) The specific parts of the message to modify Valid values: ACE_BC_MOD_DBLK1 ACE_BC_MOD_BCCTRL1 ACE_BC_MOD_CMDWRD1_1 ACE_BC_MOD_CMDWRD1_2 ACE_BC_MOD_GAPTIME1 ACE_BC_MOD_DBLK2 ACE_BC_MOD_BCCTRL2 ACE_BC_MOD_CMDWRD2_1 ACE_BC_MOD_CMDWRD2_2 ACE_BC_MOD_GAPTIME2

aceBCMsgModifyMode (continued)

DESCRIPTION

This function modifies an existing message block's parameters and options. The existing message specified by the message block id input to this function should have previously been created using the aceBCMsgCreate(), aceBCMsgCreateMode(), or aceBCAsyncMsgCreateMode() function calls. This message must be a Mode code message.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	One or more of the input parameters is/are invalid
ACE_ERR_UNRES_MSGBLK	The message id specified in this function does not exist
ACE_ERR_UNRES_DATABLK	The data id specified in this function does not exist
ACE_ERR_MEMMGR_FAIL	The new message is an RT-to-RT message and the original was not so there is not enough memory to complete this operation
ACE_ERR_BC_DBLK_SIZE	The data block size is invalid
ACE_ERR_NOT_ASYNC_MODE	The software is not in asynchronous mode. This state must be selected in the aceBCConfigure() function.

aceBCMsgModifyMode (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nMsgBlkID = 10; // unique user message id
S16BIT nDataBlkID1 = MDBLK1; // handle for data block
U16BIT wRT = 1;
U16BIT wTR = 1;
U16BIT wWC = 32; //32 data words
U16BIT wMsgGapTime = 100; // 100 µsec gap time
U16BIT wModeCmd = 0x3;

nResult = aceBCMsgModifyMode(DevNum, nMsgBlkID, nDataBlkID, wRT, wTR,
                                   wModeCmd,wMsgGapTime,
                                   ACE_BCCTRL_CHL_A,
                                   ACE_BC_MOD_DBLK1);

if(nResult)
{
      printf(“Error in aceBCMsgModifyMode() function \n”);
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCMsgCreateMode()

aceBCMsgModifyRTtoBC

This function modifies an asynchronous RT to BC transmit message.

PROTOTYPE

```
#include "BCOp.h"

S16BIT _DECL aceBCMsgModifyRTtoBC  (S16BIT DevNum,
                                    S16BIT nMsgBlkID,
                                    S16BIT nDataBlkID,
                                    U16BIT wRT,
                                    U16BIT wSA,
                                    U16BIT wWC,
                                    U16BIT wMsgGapTime,
                                    U32BIT dwMsgOptions,
                                    U16BIT wModFlags);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgBlkID	(input parameter) Unique ID number for the new message block Valid values: >0
nDataBlkID	(input parameter) Unique ID number for this block of data Valid values: >0
wRT	(input parameter) Rtaddress

aceBCMsgModifyRTtoBC (continued)

wSA	(input parameter) The RT subaddress
wWC	(input parameter) RT receive message data word count
wMsgGapTime	(input parameter) The time to next message in µseconds Valid values: >=0
dwOptions	(input parameter) The BC Control word of the message block Valid values: See aceBCMsgCreateRTtoBC() function
wModFlags	(input parameter) The specific parts of the message to modify Valid values: ACE_BC_MOD_DBLK1 ACE_BC_MOD_BCCTRL1 ACE_BC_MOD_CMDWRD1_1 ACE_BC_MOD_CMDWRD1_2 ACE_BC_MOD_GAPTIME1 ACE_BC_MOD_DBLK2 ACE_BC_MOD_BCCTRL2 ACE_BC_MOD_CMDWRD2_1 ACE_BC_MOD_CMDWRD2_2 ACE_BC_MOD_GAPTIME2

aceBCMsgModifyRTtoBC (continued)

DESCRIPTION

This function modifies an existing message block's parameters and options. The existing message specified by the message block id input to this function should have previously been created using the aceBCMsgCreate(), aceBCMsgCreateRTtoBC(), or aceBCAsyncMsgCreateRTtoBC() function calls. This message must be a RT to BC transmit message.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	One or more of the input parameters is/are invalid
ACE_ERR_UNRES_MSGBLK	The message id specified in this function does not exist
ACE_ERR_UNRES_DATABLK	The data id specified in this function does not exist
ACE_ERR_MEMMGR_FAIL	The new message is an RT-to-RT message and the original was not so there is not enough memory to complete this operation
ACE_ERR_BC_DBLK_SIZE	The data block size is invalid
ACE_ERR_NOT_ASYNC_MODE	The software is not in asynchronous mode. This state must be selected in the aceBCConfigure() function.

aceBCMsgModifyRTtoBC (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nMsgBlkID = 10; // unique user message id
S16BIT nDataBlkID1 = MDBLK1; // handle for data block
U16BIT wRT = 1;
U16BIT wSA = 1;
U16BIT wWC = 32; //32 data words
U16BIT wMsgGapTime = 100; // 100 µsec gap time
U16BIT wModeCmd = 0x3;

nResult = aceBCMsgModifyRTtoBC(DevNum, nMsgBlkID, nDataBlkID, wRT, wSA,
                                   wWC,wMsgGapTime,
                                   ACE_BCCTRL_CHL_A,
                                   ACE_BC_MOD_DBLK1);

if(nResult)
{
      printf("Error in aceBCMsgModifyRTtoBC() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCMsgCreateRTtoBC()

aceBCMsgModifyRTtoRT

This function modifies an asynchronous RT to RT message.

PROTOTYPE

```
#include "BCOp.h"
S16BIT _DECL aceBCMsgModifyRTtoRT  (S16BIT DevNum,
                                    S16BIT nMsgBlkID,
                                    S16BIT nDataBlkID,
                                    U16BIT wRTRx,
                                    U16BIT wSARx,
                                    U16BIT wWC,
                                    U16BIT wRTTx,
                                    U16BIT wSATx,
                                    U16BIT wMsgGapTime,
                                    U32BIT dwMsgOptions,
                                    U16BIT wModFlags);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgBlkID	(input parameter) Unique ID number for the new message block Valid values: >0
nDataBlkID	(input parameter) Unique ID number for this block of data Valid values: >0
wRTRx	(input parameter) The receiving RT's address

aceBCMsgModifyRTtoRT (continued)

wSARx	(input parameter) The receiving RT's subaddress
wWC	(input parameter) RT receive message data word count
wRTTx	(input parameter) The transmitting RT's address
wSATx	(input parameter) The transmitting RT's subaddress
wMsgGapTime	(input parameter) The time to next message in μseconds Valid values: >=0
dwOptions	(input parameter) The BC Control word of the message block Valid values: See aceBCMsgCreateRTtoBC() function
wModFlags	(input parameter) The specific parts of the message to modify Valid values: ACE_BC_MOD_DBLK1 ACE_BC_MOD_BCCTRL1 ACE_BC_MOD_CMDWRD1_1 ACE_BC_MOD_CMDWRD1_2 ACE_BC_MOD_GAPTIME1 ACE_BC_MOD_DBLK2 ACE_BC_MOD_BCCTRL2 ACE_BC_MOD_CMDWRD2_1 ACE_BC_MOD_CMDWRD2_2 ACE_BC_MOD_GAPTIME2

aceBCMsgModifyRTtoRT (continued)

DESCRIPTION

This function modifies an existing message block's parameters and options. The existing message specified by the message block id input to this function should have previously been created using the aceBCMsgCreate(), aceBCMsgCreateRTtoRT(), or aceBCAsyncMsgCreateRTtoRT() function calls. This message must be a RT to RT transmit message.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	One or more of the input parameters is/are invalid
ACE_ERR_UNRES_MSGBLK	The message id specified in this function does not exist
ACE_ERR_UNRES_DATABLK	The data id specified in this function does not exist
ACE_ERR_MEMMGR_FAIL	The new message is an RT-to-RT message and the original was not so there is not enough memory to complete this operation
ACE_ERR_BC_DBLK_SIZE	The data block size is invalid
ACE_ERR_NOT_ASYNC_MODE	The software is not in asynchronous mode. This state must be selected in the aceBCConfigure() function.

aceBCMsgModifyRTtoRT (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nMsgBlkID = 10; // unique user message id
S16BIT nDataBlkID1 = MDBLK1; // handle for data block
U16BIT wRTRx = 1;
U16BIT wSARx = 1;
U16BIT wRTTx = 2;
U16BIT wSATx = 1;
U16BIT wWC = 32; //32 data words
U16BIT wMsgGapTime = 100; // 100 µsec gap time

/* RT 2 will transmit to RT 1 */
nResult = aceBCMsgModifyRTtoRT(DevNum, nMsgBlkID, nDataBlkID, wRTRx,
                               wSARx, wWC, wRTTx, wSATx, wMsgGapTime,
                               ACE_BCCTRL_CHL_A, ACE_BC_MOD_DBLK1);

if(nResult)
{
      printf("Error in aceBCMsgModifyRTtoRT() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCMsgCreateRTtoRT()

aceBCOpCodeCreate

This function creates an opcode to be used in the creation of a frame.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCOpCodeCreate(S16BIT DevNum,
                               S16BIT nOpCodeID,
                               U16BIT wOpCodeType,
                               U16BIT wCondition,
                               U16BIT wParameter1,
                               U16BIT wParameter2,
                               U32BIT dwReserved);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

nOpCodeID (input parameter)
Unique, user supplied ID number of Opcode

wOpCodeType the type of opcode to be created
Valid values: complete with description and params

ACE_S_OPCODE_AMSG
This parameter is a software opcode which will call a series of hardware opcodes to create an asynchronous message.

ACE_OPCODE_XEQ
This parameter is an execute message hardware opcode. This will execute the message at the previously defined Message Block ID specified by the wParameter1 input parameter if the condition in the wCondition input parameter tests true, otherwise execution will continue at the next opcode instruction. The wParameter2 input parameter is not used.

aceBCOpCodeCreate (continued)

ACE_OPCODE_JMP
This parameter is a jump hardware opcode. This parameter will cause a jump to the specified offset within a frame if the wCondition input parameter tests true. The wParameter1 input parameter specifies the number of opcodes to jump forward or back from your current frame position. An input of zero will cause a jump to the same jump instruction and should be avoided.

ACE_OPCODE_CAL
This parameter is a subroutine call hardware opcode. This parameter will cause a jump to the frame specified in the wParameter1 input parameter if the wCondition input parameter tests true. The wParameter 2 input parameter is not used.

ACE_OPCODE_RTN
This parameter is a return from subroutine call hardware opcode. This parameter will return from the subroutine if the wCondition parameter tests true. The wParameter1 and wParameter2 input parameters are not used.

ACE_OPCODE_IRQ
This parameter is an interrupt request hardware opcode. This parameter will generate an interrupt if the wCondition input parameter tests true. The wParameter1 input parameter specifies which of the Enhanced BC IRQ bits 5-2 will be set in Interrupt Status Register # 2 at memory location 0x1E on an interrupt condition. Only the four LSB of the wParameter1 input parameter are used. A wParameter1 input parameter that contains 0's for the 4 LSB will not generate an interrupt. The wParameter 2 input parameter is not used.

ACE_OPCODE_HLT
This parameter is a halt operation hardware opcode. This parameter will cause the device to stop execution of the Message Sequence Control Program if the wCondition input parameter tests true. The wParameter1 and wParameter2 input parameters are not used.

ACE_OPCODE_DLY
This parameter is a delay operation hardware opcode. This parameter will introduce a delay specified by the wParameter1 input parameter if the wCondition input parameter tests true. The resolution of the wParameter1 input parameter is 1μsecond/LSB.

aceBCOpCodeCreate (continued)

ACE_OPCODE_WFT
This parameter is a wait until frame timer is 0 hardware opcode. This parameter will cause the Message Sequence Control Program to wait until the frame timer is zero before continuing execution if the wCondition input parameter tests true.

ACE_OPCODE_CFT
This parameter is a compare to frame timer hardware opcode. The wCondition input parameter is not used. The wParameter1 input parameter specifies the value to compare to the frame timer. The resolution of this value is 100µseconds/LSB. The operation will set the value of the Less than flag bits 0 and the Equal to flag bit 1 in the BC Condition Code Register at memory location 0x1B accordingly.

ACE_OPCODE_FLG
This parameter is a set, clear, or toggle GP flag bits hardware opcode. This operation will set, clear, or toggle the value of the GP Flag bits in the BC Condition Code Register at memory location 0x1B. The wParameter1 input parameter specifies which bits to set, clear, and/or toggle. Bits 0 and 8 affect GP0, bits 1 and 9 affect GP1, bits 2 and 10 affect GP2, etc. If both bits for a GP register are 0, there is no change. If both bits for a GP register are 1, a toggle is performed. If the lower bit is 1 and the higher bit is 0 the GP register is set to 1. If the lower bit is 0 and the associated higher bit is 1 the GP register is cleared to 0.

ACE_OPCODE_LTT
This parameter is a load time tag counter hardware opcode. This operation will load the time tag counter with the time value specified in the wParameter1 input parameter if the wCondition input parameter tests true. The resolution of the time value in the wParameter input parameter is specified by bits 9-7 of Configuration Register # 2 at memory location 0x02. These bits can be set by calling the aceSetTimeTagRes() function. The wParameter2 input parameter is not used.

ACE_OPCODE_LFT
This parameter is a load frame timer hardware opcode. This operation will load the frame timer with the value specified by the wParameter1 input parameter if the wCondition input parameter tests true. The resolution of the wParameter1 input parameter is 100µseconds/LSB. The wParameter2 input parameter is not used.

aceBCOpCodeCreate (continued)

ACE_OPCODE_SFT
This parameter is a start frame timer hardware opcode. This operation will start the frame time counter with the time value in the frame register if the wCondition input parameter tests true. The wParameter1 and wParameter2 input parameters are not used.

ACE_OPCODE_PTT
This parameter is a push time tag register onto GP Queue hardware opcode. This operation will push the value of the Time Tag Register onto the General Purpose Queue if the wCondition input parameter tests true. The wParameter1 and wParameter2 input parameters are not used.

ACE_OPCODE_PBS
This parameter is a push block status word onto GP Queue hardware opcode. This operation will push the Block Status Word for the most recent message onto the General Purpose Queue if the wCondition input parameter tests true. The wParameter1 and wParameter2 input parameters are not used.

ACE_OPCODE_PSI
This parameter is a push immediate value onto GP Queue hardware opcode. This operation will push the immediate value represented by the wParameter1 input parameter onto the General Purpose Queue if the wCondition input parameter tests true. The wParameter2 input parameter is not used.

ACE_OPCODE_PSM
This parameter is a push data at specified location onto GP Queue hardware opcode. This operation will push the data stored at a specific memory location onto the General Purpose Queue if the wCondition input parameter tests true. The wParameter1 input parameter represents the memory location that contains the data to push onto the General Purpose Queue.

ACE_OPCODE_WTG
This parameter is a wait for external trigger hardware opcode. This operation will cause the device to wait until a logic 0 to logic 1 transition takes place on the EXT_TRIG input signal before executing the next opcode instruction if the wCondition input parameter tests true. The wParameter1 and wParameter2 input parameters are not used.

aceBCOpCodeCreate (continued)

ACE_OPCODE_XQF
This parameter is an execute and flip hardware opcode. The wParameter1 input parameter specifies the message ID to perform the operation on. After this message is processed, if the internal condition flag tests true, bit 4 of the Message Control/Status Block is flipped, and the new Message Block Address as the updated value of the parameter. As a result, the next time that this line in the instruction list is executed, the Message Control/Status Block at the updated address, rather than the old address, will be processed.

ACE_OPCODE_TRP
This parameter is a trap a bad sequence hardware opcode. The wParameter1 and wParameter2 input parameters are not used. The wCondition input parameter is not used. This operation will allow the device to halt if the BC has fetched an illegal opcode or if the frame timer has timed out. An illegal opcode is one that is not defined, fails it's parity check, and/or has an incorrect value for one or more of the defined constant bits in the opcode. Any one of these conditions will cause the BC to halt operation.

wCondition (input parameter)
The opcode will run when this condition is true. The following values will set the appropriate bits in the BC Condition Code Register at memory location 0x1B.
Valid values:

ACE_CNDTST_LT
This will cause the operation to execute if the value is less than the compared value

ACE_CNDTST_GT
This will cause the operation to execute if the value is greater than the compared value

ACE_CNDTST_EQ
This will cause the operation to execute if the value is equal to the compared value

ACE_CNDTST_NEQ
This will cause the operation to execute if the value is less than or greater than the compared value

ACE_CNDTST_GP1_1
This will cause the operation to execute if the value in GP1 is logic 1.

aceBCOpCodeCreate (continued)

ACE_CNDTST_GP1_0
This will cause the operation to execute if the value in GP1 is logic 0.

ACE_CNDTST_GP2_1
This will cause the operation to execute if the value in GP2 is logic 1.

ACE_CNDTST_GP2_0
This will cause the operation to execute if the value in GP2 is logic 0.

ACE_CNDTST_GP3_1
This will cause the operation to execute if the value in GP3 is logic 1.

ACE_CNDTST_GP3_0
This will cause the operation to execute if the value in GP3 is logic 0.

ACE_CNDTST_GP4_1
This will cause the operation to execute if the value in GP4 is logic 1.

ACE_CNDTST_GP4_0
This will cause the operation to execute if the value in GP4 is logic 0.

ACE_CNDTST_GP5_1
This will cause the operation to execute if the value in GP5 is logic 1.

ACE_CNDTST_GP5_0
This will cause the operation to execute if the value in GP5 is logic 0.

ACE_CNDTST_GP6_1
This will cause the operation to execute if the value in GP6 is logic 1.

ACE_CNDTST_GP6_0
This will cause the operation to execute if the value in GP6 is logic 0.

ACE_CNDTST_GP7_1
This will cause the operation to execute if the value in GP7 is logic 1.

aceBCOpCodeCreate (continued)

ACE_CNDTST_GP7_0
This will cause the operation to execute if the value in GP7 is logic 0.

ACE_CNDTST_NO_RES
This will cause the operation to execute if there is no response from an RT.

ACE_CNDTST_RES
This will cause the operation to execute if there is a response from an RT.

ACE_CNDTST_FMT_ERR
This will cause the operation to execute if the received portion of the most recent message contained one or more violations of the 1553 message validation criteria (sync, encoding, parity, bit count, word count, etc.), or the status word received from a responding RT contained an incorrect RT address field.

ACE_CNDTST_NO_FMT_ERR
This will cause the operation to execute if the received portion of the most recent message passes the 1553 message validation criteria.

ACE_CNDTST_GD_XFER
This will cause the operation to execute after the completion of a valid (error-free) RT-to-BC transfer, RT-to-RT transfer, or transmit mode code with data message.

ACE_CNDTST_BAD_XFER
This will cause the operation to execute if the RT-to-BC transfer, RT-to-RT transfer, or transmit mode code with data message contained an error.

ACE_CNDTST_MSK_STS_SET
This will cause the operation to execute if one or two of the following conditions have occurred on the most recent message: (1) If one (or more) of the Status Mask bits (14 through 9) in the BC Control Word is/are logic 0 and the corresponding bit(s) is/are set to logic 1 in the received RT Status Word. In the case of the RESERVED BITS MASK (bit 9) set to logic "0," any or all of the 3 Reserved status word bits being set will result in a MASKED STATUS SET condition; **and/or** (2) If BROADCAST MASK ENABLED/XOR* (bit 11 of Configuration Register # 4 at memory location 0x08 is logic 0 **and** the logic sense of the MASK BROADCAST bit of the message's BC Control Word and the BROADCAST COMMAND RECEIVED bit in

aceBCOpCodeCreate (continued)

the received RT Status Word are **opposite**; **or** (3) If BROADCAST MASK ENABLED/XOR* (bit 11 of Configuration Register # 4 is logic 1 and the MASK BROADCAST bit of the message's BC Control Word is logic 0 and the BROADCAST COMMAND RECEIVED bit in the received RT Status Word is logic 1.

ACE_CNDTST_MSK_STS_CLR
This will cause the operation to execute if the Mask Status Bit is set to a 0. (See above)

ACE_CNDTST_BAD_MSG
This will cause the operation to execute if either a format error, loop test fail, or no response error for the most recent message.

ACE_CNDTST_GOOD_MSG
This will cause the operation to execute if the most recent message was successful.

ACE_CNDTST_0RETRY
This will cause the operation to execute if zero retries occurred.

ACE_CNDTST_1RETRY
This will cause the operation to execute if one retry occurred.

ACE_CNDTST_2RETRY
This will cause the operation to execute if two retries occurred.

ACE_CNDTST_ALWAYS
This will cause the operation to always execute.

ACE_CNDTST_NEVER
This will cause the operation to never execute.

wParameter1	(input parameter) parameter info depends on opcode (see opcode)
wParameter2	(input parameter) parameter info depends on opcode (see opcode)
dwReserved	(input parameter) Reserved for future use

aceBCOpCodeCreate (continued)

DESCRIPTION

This function creates an opcode to be used in the creation of a frame. All messages and frame controls must be encapsulated in an opcode. Opcodes can be used to control the flow and operation of the frame such as jumps, and executes. If the ACE_S_OPCODE_AMSG software opcode is selected to run a series of hardware opcodes, the wCondition input parameter can only range from ACE_CNDTST_LT to ACE_CNDTST_GP7_0 to set the condition as one of the General Purpose Flag bits. If wCondition is outside of this range, ACE_ERR_PARAMETER will be returned by the function. All opcodes that deal with placing values on the General Purpose Queue must be done in pairs. The first value is a unique header identified by the user and the second value is the data itself. The user can use any 16-bit value for the header except for 0xFFFF and 0xFFF8 which is used internally by the library.

Note: *While the ACE_OPCODE_XEQ (Execute Message) instruction is conditional, not all condition codes listed as values for the wCondition input parameter may be used to enable its use. The ACE_CNDTST_ALWAYS and the ACE_CNDTST_NEVER condition codes may always be used with the ACE_OPCODE_XEQ (Execute Message) instruction. Five of the available eight general purpose flag bits, GP2 through GP6, may also be used. However, if GP1 to GP2 are used, it is imperative that the user's application does not modify the value of the specific general purpose flag bit that enabled a particular message while that message is being processed (EOM bit not set in BSW). If the application does modify the value of the flag bit before the message is completed then the message will be aborted and will not appear in the BC Host Buffer. General Purpose Flag bits 0, 1, and 7 (GP0, GP1, and GP7) cannot be used because they are either used internally by the library or the hardware device.*

Similarly, the ACE_CNDTST_LT, ACE_CNDTST_GT, ACE_CNDTST_EQ, and ACE_CNDTST_NEQ, which the BC only updates by means of the ACE_OPCODE_CFT instruction, may also be used. However, these flags are dual use. Therefore, if these are used, it is imperative that the user's application does not modify the value of the specific flag that enabled a particular message while that message is being processed. If the application does modify the value of the flag bit before the message is completed then the message will be aborted and will not appear in the BC Host Buffer.

The following conditions are not available for use with the ACE_OPCODE_XEQ instruction and should not be used to enable its execution:
ACE_CNDTST_NORES, ACE_CNDTST_FMT_ERR,
ACE_CNDTST_GD_XFER, ACE_CNDTST_BAD_XFER,
ACE_CNDTST_MSK_STS_SET, ACE_CNDTST_MSK_STS_CLR,
ACE_CND_TST_BAD_MSG, ACE_CNDTST_0RETRY,
ACE_CNDTST_1RETRY and ACE_CNDTST_2RETRY.

The conditions listed above can be used with other instructions like the ACE_OPCODE_JMP opcode instruction.

aceBCOpCodeCreate (continued)

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	The wCondition input parameter is out of range for the specified opcode or the wOpCodeType input parameter is invalid
ACE_ERR_BC_DBLK_EXISTS	The nOpCodeID input parameter already exists
ACE_ERR_BC_DBLK_ALLOC	The required memory needed to add this opcode could not be allocated
ACE_WRN_BC_OPCODE_INVALID	The selected BC Opcode Condition could cause performance problems. Please see the note under the description section of this function.

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT OP1 = 1;
U16BIT MSG1 = 2;
// Create XEQ opcode that will use msg block

nResult = aceBCOpCodeCreate(DevNum, OP1, ACE_OPCODE_XEQ,
                            ACE_CNDTST_ALWAYS, MSG1, 0, 0);

if(nResult)
{
      printf("Error in aceBCOpCodeCreate() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCOpCodeDelete()

aceBCOpCodeDelete

This function will delete a previously created opcode.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCOpCodeDelete(S16BIT DevNum,
                               S16BIT nOpCodeID);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nOpCodeID	(input parameter) Unique ID that was provided by user when opcode was created using the aceBCOpCodeCreate() function

DESCRIPTION

This function deletes a previously created opcode from a frame.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_NODE_NOT_FOUND	The opcode ID specified by the nOpCodeID input parameter could not be found

aceBCOpCodeDelete (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nOpCodeID = 42;
S16BIT nResult = 0;

nResult = aceBCOpCodeDelete(DevNum, nOpCodeID);

if(nResult)
{
      printf("Error in aceBCOpCodeDelete() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCOpCodeCreate()

aceBCSendAsyncMsgHP

This function sends a previously created message in high priority mode onto the 1553 bus.

PROTOTYPE

```
#include "BCOp.h"

S16BIT _DECL aceBCSendAsyncMsgHP(S16BIT DevNum,
                                 U16BIT nMsgID,
                                 U16BIT wTimeFactor)
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgID	(input parameter) Unique ID number for a previously created message Valid values: >0
wTimeFactor	(input parameter) The largest value of frame time used in any call to aceBCFrameCreate()

DESCRIPTION

This function sends a previously created asynchronous message onto the 1553 bus. The function sends the message in high priority mode which means that the message will be sent at the end of the current message. The message is specified by the nMsgID input parameter.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	An invalid input parameter was input by the user

aceBCSendAsyncMsgHP (continued)

ACE_ERR_BC_DBLK_EXISTS	The message block specified by the nMsgBlkID input parameter already exists
ACE_ERR_NODE_NOT_FOUND	The data block specified by the nDataBlkID can not be created. Make sure that this data block has not been previously created with the aceBCDataBlkCreate() function. Asynchronouse messages will call the aceBCDataBlkCreate() function internally and this function does not need to be called by the user.
ACE_ERR_BC_DBLK_SIZE	The data block size is not valid
ACE_ERR_BC_DBLK_ALLOC	The message block could not be allocated
ACE_ERR_MEMMGR_FAIL	The required memory for the message block could not be allocated

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult= aceBCSendAsyncMsgHP(DevNum,
                              333,    //msg. id number
                              1000);  //max. used frame time
if(nResult)
{
      printf("Error occurred in aceBCSendAsyncMsgHP function\n");
      PrintError(nResult);
      return;
}
```

SEE ALSO

aceBCSendAsyncMsgLP()
aceBCConfigure()

aceBCSendAsyncMsgLP

This function sends asynchronous messages on the low priority queue onto the 1553 data bus.

PROTOTYPE

```
#include "BCOp.h"

S16BIT _DECL aceBCSendAsyncMsgLP(S16BIT DevNum,
                                  U16BIT *pMsgLeft,
                                  U16BIT wTimeFactor)
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMsgID	(output parameter) The number of messages that are left on the asynchronous message queue
wTimeFactor	(input parameter) The largest value of frame time used in any call to aceBCFrameCreate()

DESCRIPTION

This function sends a previously created asynchronous message onto the 1553 bus. The function sends the message in low priority mode which means that the messages will be sent at the end of the current frame only if frame time permits.

RETURN VALUE

ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	An invalid input parameter was input by the user
ACE_ERR_BC_DBLK_EXISTS	The message block specified by the nMsgBlkID input parameter already exists

aceBCSendAsyncMsgLP (continued)

ACE_ERR_NODE_NOT_FOUND	The data block specified by the nDataBlkID can not be created. Make sure that this data block has not been previously created with the aceBCDataBlkCreate() function. Asynchronouse messages will call the aceBCDataBlkCreate() function internally and this function does not need to be called by the user.
ACE_ERR_BC_DBLK_SIZE	The data block size is not valid
ACE_ERR_BC_DBLK_ALLOC	The message block could not be allocated
ACE_ERR_MEMMGR_FAIL	The required memory for the message block could not be allocated
ACE_ERR_UNRES_ASYNC_OP	Asynchronous OpCode block not found
ACE_ERR_UNRES_MSGBLK	Message block not found

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT pMsgLeft;


nResult= aceBCSendAsyncMsgLP(DevNum,
                              &pMsgLeft,   //msg. id number
                              1000);       //max. used frame time
if(nResult)
{
      printf("Error occurred in aceBCSendAsyncMsgLP function\n");
      PrintError(nResult);
      return;
}
```

SEE ALSO

aceBCSendAsyncMsgHP()
aceBCConfigure()

aceBCSetGPFState

This function allows the user to modify a general purpose flag.

PROTOTYPE

```
#include "bcop.h"


S16BIT _DECL aceBCSetGPFState(S16BIT DevNum,
                              U16BIT wGPF,
                              U16BIT wStateChange);
```

STATE

Ready, Run

MODE

BC

PARAMETERS

DevNum	Logical Device Number Valid values: 0 – 31
wGPF	This parameter indicates which one of the general purpose flag bits to modify Valid values: ACE_ GPF0 (reserved for library) ACE _GPF1 ACE _GPF2 ACE _GPF3 ACE _GPF4 ACE _GPF5 ACE _GPF6 ACE _GPF7 (reserved for library)
wStateChange	The effect taken place on GPF Valid values: ACE_GPF_SET ACE_GPF_CLEAR ACE_GPF_TOGGLE ACE_GPF_LEAVE

aceBCSetGPFState (continued)

DESCRIPTION

This function allows the user to modify a general purpose flag from the host. The flag can either be set, cleared, toggled, or have no change made to it. The function will perform a write to the BC General Purpose Flag register at memory location 0x1B to perform the specified operation to one of the general purpose flag bits.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	The wGPF input parameter contains a value equal to zero or greater than six

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceBCSetGPFState(DevNum, ACE_CND_GP1, ACE_GPF_TOGGLE);

if(nResult)
{
      printf("Error in aceBCSetGPFState() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

None

aceBCSetMsgRetry

This function sets up how retry messages will be retried.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCSetMsgRetry(S16BIT DevNum,
                              U16BIT wNumOfRetries,
                              U16BIT wFirstRetryBus,
                              U16BIT wSecondRetryBus,
                              U16BIT wReserved);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

wNumOfRetries (input parameter)
This is the number of retries that will be attempted if a message fails, and retries are enabled.
Valid values:

ACE_RETRY_NONE
This parameter will cause no retries if a message fails

ACE_RETRY_ONCE
This parameter will cause one retry if a message fails

ACE_RETRY_TWICE
This parameter will cause two retries if a message fails

wFirstRetryBus (input parameter)
This is the 1553 bus that will be used when the fist retry attempt is processed.
Valid values:

ACE_RETRY_SAME
This parameter will cause the first retry to occur on the same bus

aceBCSetMsgRetry (continued)

ACE_RETRY_ALT
This parameter will cause the first retry to occur on the other bus

wSecondRetryBus — This is the 1553 bus that will be used when the second retry attempt is processed.
Valid values:

ACE_RETRY_SAME
This parameter will cause the second retry to occur on the same bus

ACE_RETRY_ALT
This parameter will cause the second retry to occur on the other bus

wReserved — Reserved for future use

DESCRIPTION

This function sets up how all messages with the retry bit set in the BC Control Word will be retried if a failure or status set condition occurs. The function writes the appropriate value to bits 3 and 4 of Configuration Register # 1 at memory location 0x01 to set the number of retries input by the user. The function will then set bits 8 and 7 of Configuration Register # 4 at memory location 0x08 to set the first and second retry buses specified by wFirstRetryBus and wSecondRetryBus respectively.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_PARAMETER	The wNumOfRetries input parameter contains a value greater than two and/or the wFirstRetryBus contains a value greater than one and/or the wSecondRetryBus contains a value greater than one

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceBCSetMsgRetry(DevNum, ACE_RETRY_TWICE,
                                          ACE_RETRY_SAME, ACE_RETRY_ALT,
                                          0);

if(nResult)
{
      printf("Error in aceBCSetMsgRetry() function \n");
      PrintOutError(nResult);
      return;
}
```

aceBCSetMsgRetry (continued)

SEE ALSO

None

aceBCSetWatchDogTimer

This function enables/disables the BC watchdog timer.

PROTOTYPE

```
#include "bcop.h"

S16BIT _DECL aceBCSetWatchDogTimer(S16BIT DevNum,
                                   U16BIT bEnable,
                                   U16BIT wTimeOut);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
bEnable	(input parameter) Enable/Disable the watchdog timer Valid values: TRUE Enables the timer FALSE Disables the timer
wTimeOut	(input parameter) The time for a minor frame to wait until TRAP (100µs resolution)

DESCRIPTION

This function enables or disables the BC watchdog timer by writing to bit 1 of Configuration Register # 7 at memory location 0x19. This timer will TRAP the BC if the time out value is exceeded. An interrupt may be generated when the BC traps if the user has previously called the aceSetIrqConditions() function with the dwIrqMask input parameter set to ACE_IMR2_BC_TRAP.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in BC mode

aceBCSetWatchDogTimer (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT wTimeOut = 800;

nResult = aceBCSetWatchDogTimer(DevNum, TRUE, wTimeOut);
{
      printf("Error in aceBCSetWatchDogTimer() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

None

aceBCStart

This function will start the BC.

PROTOTYPE

```
#include "bcop.h"


S16BIT _DECL aceBCStart(S16BIT DevNum,
                        S16BIT nMjrFrmID,
                        S32BIT lMjrFrmCount);
```

STATE

Ready

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nMjrFrmID	(input parameter) Unique frame ID provided by user when frame was created using the aceBCFrameCreate() function
lMjrFrmCount	(input parameter) Number of times that the frame should be executed Valid values: number of times frame is processed -1 = Forever

DESCRIPTION

This function configures the device for Enhanced BC mode and then starts the Bus Controller given a major frame. After this function is executed the device will be in a Run state. Upon calling this function all BC messages, OpCodes, and Frames will be resolved from host memory down into BC RAM. Due to this fact this function was not intended to be a real-time function as any call to aceBCStart() will require some time before data appears on the 1553 bus.

Note: If the user places a breakpoint in application code for debugging purposes, the BC will run forever because synchronization between the Enhanced Mini-ACE device and the runtime library will be lost and the number of frames sent out will never be decremented so that the HLT opcode can be placed after the set number of frames are sent out by the library.

aceBCStart (continued)

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_NODE_NOT_FOUND	The nMjrFrmID could not be found
ACE_ERR_FRAME_NOT_MAJOR	The nMjrFrmID frame is not a major frame
ACE_ERR_UNRES_DATABLK	The data block address could not be resolved
ACE_ERR_UNRES_FRAME	The opcode could not be resolved

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
S32BIT lMjrFrmCount = -1;
S16BIT nMjrFrmID = 42;
/* Initialize EMA */
/* create data blks, msgs, frames */
/* set watchdog, Create Binary Image File */

nResult = aceBCStart(DevNum, nMjrFrmID, lMjrFrmCount);

if(nResult)
{
      printf("Error in aceBCStart() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceBCStop()

aceBCStop

This function stops the BC.

PROTOTYPE

#include “bcop.h”

S16BIT _DECL aceBCStop(S16BIT DevNum);

STATE

Run

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function stops the Bus Controller for the designated Logical Device Number, either at the end of the current message or at the end of the current frame. The routine will immediately start attempting to halt the device. If the device does not halt within a predetermined time (based on the library), the device is put through a protocol reset. After this function is finished executing the device will be in a Ready state.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode

EXAMPLE

```
S16BIT DevNum = 0;

/* aceBCRun() */


nResult = aceBCStop(DevNum);

if(nResult)
{
      printf(“Error in aceBCStop() function \n”);
      PrintOutError(nResult);
      return;
}
```

aceBCStop (continued)

SEE ALSO

aceBCRun()

aceBCUninstallHBuf

This function removes the BC host buffer.

PROTOTYPE

#include "bcop.h"

S16BIT _DECL aceBCUninstallHBuf(S16BIT DevNum);

STATE

Ready

MODE

BC

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function removes the BC host buffer if present, and releases all memory resources back to the system.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Run state
ACE_ERR_INVALID_MODE	The device is not in BC mode
ACE_ERR_HBUF	The device has no host buffer associated with it

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

/* init Device DevNum, attach a host buffer */

nResult = aceBCUninstallHBuf(DevNum);

if(nResult)
{
      printf("Error in aceBCUninstallHBuf() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceInstallHBuf()

RT Functions

Table 6. RT Functions Listing

aceRTBITWrdConfig

This function configures the Built in Test word.

PROTOTYPE

```
#include "Rtop.h"


S16BIT _DECL aceRTBITWrdConfig(S16BIT DevNum,
                               U16BIT wBITLoc,
                               U16BIT wBITBusyInh);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

wBITLoc (input parameter)
Destination as to the type of BIT.
Valid values:
The following values will set/clear the External Bit Word Enable bit 15 of Configuration Register # 4 at memory location 0x08.

ACE_RT_BIT_INTERNAL
This value will clear bit 15 to a 0. The Enhanced Mini-ACE RT will respond to a Transmit BIT word mode command with the contents of the Enhanced Mini-ACE's internal BIT Word Register as the data word.

ACE_RT_BIT_EXTERNAL
This value will set bit 15 to a 1. The Enhanced Mini-ACE will access the BIT data word from a location in the shared RAM. In this instance, the BIT Word must be written to RAM by the host processor.

wBITBusyInh Inhibit RT Bit if Busy is active.
Valid values:
The following values will set/clear the Inhibit Bit Word Transmit If Busy bit 14 of Configuration Register # 4 at memory location 0x08.

aceRTBITWrdConfig (continued)

ACE_RT_BIT_NO_INHIBIT
This value will clear bit 14 to a 0. In this case, the Enhanced Mini-ACE will respond to a Transmit BIT Word mode command with its RT Status Word with the BUSY bit set, followed by its internal or external Built-in-Test (BIT) Word.

ACE_RT_BIT_INHIBIT
This value will set bit 14 to a 1. In this case, the Enhanced Mini-ACE will respond with its RT Status Word with the BUSY bit set, but no Data Word (BIT Word) will be transmitted.

DESCRIPTION

This function will set/clear bits 14 and 15 of Configuration Register # 4 at memory location 0x08 in order to configure the way in which the Built in Test word will be read and written and whether or not it will be inhibited if the RT is busy. External BIT word is read (written) from the memory location that the transmit Mode BIT word mode code data would be stored (EMA mode code memory offset + 0x13). The internal BIT word is read (written) from the internal EMA BIT register.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT wBITLoc = ACE_RT_BIT_INTERNAL;
U16BIT wBITBusyInh = ACE_RT_BIT_INHIBIT;

/* designate RT to read BIT word internally, and inhibit during Busy
Bit active */

nResult = aceRTBITWrdConfig(DevNum, wBITLoc, wBITBusyInh);

if(nResult)
{
      printf("Error in aceRTBITWrdConfig() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTBITWrdRead()

aceRTBITWrdWrite()

aceRTBITWrdRead

This function will read the current BIT word from the device's BIT register.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTBITWrdRead(S16BIT DevNum,
                             U16BIT wBITLoc,
                             U16BIT *pBITWrd);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

wBITLoc (input parameter)
Destination as to the type of BIT.
Valid values:
The following values will set/clear the External Bit Word Enable bit 15 of Configuration Register # 4 at memory location 0x08.

ACE_RT_BIT_INTERNAL
This value will clear bit 15 to a 0. The Enhanced Mini-ACE RT will respond to a Transmit BIT word mode command with the contents of the Enhanced Mini-ACE's internal BIT Word Register as the data word.

ACE_RT_BIT_EXTERNAL
This value will set bit 15 to a 1. The Enhanced Mini-ACE will access the BIT data word from a location in the shared RAM. In this instance, the BIT Word must be written to RAM by the host processor.

pBITWrd (output parameter)
Pointer to an unsigned 16-bit value that will be filled with the BIT word value. The individual bit descriptions are given below. A 1 will represent the condition shown for the bit.

aceRTBITWrdRead (continued)

Valid values:

Bit 15
Transmitter Timeout
Set if the Enhanced Mini-ACE's failsafe timer detected a fault condition. The transmitter timeout circuit will automatically shut down the CH. A or CH. B transmitter if it transmits for longer than 668 µs. In RT mode, the Enhanced Mini-ACE will terminate the processing of the current message as the result of a transmitter timeout, however, it **will respond** to the next message received.

Bit 14, 13
Loop Test Failure B, Loop Test Failure A
A loopback test is performed on the transmitted portion of every non-broadcast message. A validity check is performed on the received version of every word transmitted by the Enhanced Mini-ACE. In addition, a bit-by-bit comparison is performed on the last word transmitted by the RT for each message. If either the received version of any transmitted word is determined to be invalid (sync, encoding, bit count, or parity error) and/or the received version of the last transmitted word does not match the transmitted version, or a failsafe timeout occurs on the respective channel, the Loop Test Failure bit for the respective bus channel will be set.

Bit 12
Handshake Failure
If this bit is set, it indicates that the subsystem has failed to respond with the DMA handshake input DTGRT* asserted within the allotted time, in response to the Enhanced Mini-ACE asserting DTREQ*. Alternatively, a handshake failure will occur if the host PROCESSOR fails to clear STRBD* (high) within the allotted time, after the Enhanced Mini-ACE has asserted its READYD* output (low). The allotted time is 4 µs for a 16 MHz clock, or 3.5 µs for a 12 MHz clock. All of DDC's 1553 Enhanced Mini-ACE cards provide a 16 MHz clock.

Bit 11, Bit 10
Transmitter Shutdown B, Transmitter Shutdown A
Indicates that the transmitter on the respective bus channel has been shut down by a Transmitter shutdown mode code command received on the alternate channel. If an Override transmitter shutdown mode code command is received on the alternate channel, this bit will revert back to logic 0.

aceRTBITWrdRead (continued)

Bit 9
Terminal Flag Inhibited
Set to logic 1 if the Enhanced Mini-ACE's Terminal Flag RT Status bit has been disabled by an Inhibit terminal flag mode code command. Will revert to logic 0 if an Override inhibit terminal flag mode code command is received.

Bit 8
Bit Test Fail
Represents the result of the Enhanced Mini-ACE RT's most recent built-in protocol self-test. A value of logic 0 for bit 8 indicates that the test passed. The bit will return a value of logic 1 if the Enhanced Mini-ACE has failed its most recent protocol self-test. If a subsequent performing of the protocol self-test passes, bit 8 will clear to 0. Also, note that the RAM self-test has no effect on bit 8.

Bit 7
High Word Count
Set to logic 1 if the most recent message had a high word count error.

Bit 6
Low Word Count
Set to logic 1 if the most recent message had a low word count error.

Bit 5
Incorrect Sync Received
Set to a logic 1 if the Enhanced Mini-ACE detected a Command sync in a received Data Word.

Bit 4
Invalid Word Received
Indicates that the Enhanced Mini-ACE RT received a Data Word containing one or more of the following error types: sync field error, Manchester encoding error, parity error, and/or bit count error.

Bit 3
RT-RT Gap/Sync/Address Error
This bit is set if the Enhanced Mini-ACE RT is the receiving RT for an RT-to-RT transfer and one or more of the following occurs: (1) If the GAP CHECK ENABLED bit (bit 8) of Configuration Register # 5 at memory location 0x09 is set to logic 1 and the transmitting RT responds with a response time of less than 4 µs, per MIL-STD-1553B (mid-parity bit to mid-sync); i.e., less than 2 µs dead time; and/or

aceRTBITWrdRead (continued)

(2) There is an incorrect sync type or format error (encoding, bit count, and/or parity error) in the transmitting RT Status Word; and/or (3) The RT address field of the transmitting RT Status Word does not match the RT address in the transmit Command Word.

Bit 2
RT-RT No Response Error
If this bit is set to a logic 1, this indicates that for the previous message, the Enhanced Mini-ACE was the receiving RT for an RT-to-RT transfer and that the transmitting RT either did not respond or responded later than the Enhanced Mini-ACE's RT-to-RT Timeout time. The Enhanced Mini-ACE's RT-to-RT Response Timeout Time is defined as the time from the mid-bit crossing of the parity bit of the transmit Command Word to the mid-sync crossing of the transmitting RT Status Word. The value of the RT-to-RT Response Timeout is 18.5 µs by default, or programmable from among nominal values of 18.5, 22.5, 50.5, or 130 µs by calling the aceSetRespTimeOut() function.

Bit 1
RT-RT 2nd Command Word Error
If the Enhanced Mini-ACE is the receiving RT for an RT-to-RT transfer, this bit set to a logic 1 indicates one or more of the following error conditions in the transmit Command Word: (1) T/R bit = logic "0"; (2) subaddress = 00000 or 11111; (3) same RT address field as the receive Command Word.

Bit 0
Command Word Contents Error
Indicates a received command word is not defined in accordance with MIL-STD-1553B specifications. This includes the following undefined Command Words: (1) BROADCAST DISABLED, bit 7 of Configuration Register # 5 at memory location 0x09 is logic 0 **and** the Command Word is a non-mode code, broadcast, or transmit command; (2) The OVERRIDE MODE T/R* ERROR bit, bit 6 of Configuration Register # 3 at memory location 0x07 is logic 0 **and** a message with a T/R* bit of 0, a subaddress/mode field of 00000 or 11111 and a mode code field between 00000 and 01111; (3) BROADCAST DISABLED, bit 7 of Configuration Register # 5 is logic 0 **and** a mode code command that is not permitted to be broadcast (e.g., Transmit status) is sent to the broadcast address (11111).

aceRTBITWrdRead (continued)

DESCRIPTION

This function reads the current BIT word from the BIT register. External BIT word is read from the memory location that the transmit Mode BIT word mode code data would be stored (EMA mode code memory offset + 0x13). The internal BIT word is read from the internal RT BIT Word register.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The wBITLoc input parameter is greater than one and/or the pBITWrd is Null

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT pBITWrd;

/* Read Internal BIT word from the EMA RT BIT Word register. The BIT
word is returned in parameter pBITWrd */

nResult = aceRTBITWrdRead(DevNum, ACE_RT_BIT_INTERNAL, &pBITWrd);

if(nResult)
{
      printf("Error in aceRTBITWrdRead() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTBITWrdConfig()   aceRTBITWrdWrite()

aceRTBITWrdWrite

This function will write a BIT word to the external BIT word location.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTBITWrdWrite(S16BIT DevNum,
                              U16BIT *wBITWrd);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum

Logical Device Number
Valid values:
0 – 31

wBITWrd

(input parameter)
Pointer to an unsigned 16-bit value that will be written into the external BIT word location. The individual bit descriptions are given below. A 1 will represent the condition shown for the bit.
Valid values:

Bit 15
Transmitter Timeout
Set if the Enhanced Mini-ACE's failsafe timer detected a fault condition. The transmitter timeout circuit will automatically shut down the CH. A or CH. B transmitter if it transmits for longer than 668 µs. In RT mode, the Enhanced Mini-ACE will terminate the processing of the current message as the result of a transmitter timeout, however, it **will respond** to the next message received.

Bit 14, 13
Loop Test Failure B, Loop Test Failure A
A loopback test is performed on the transmitted portion of every non-broadcast message. A validity check is performed on the received version of every word transmitted by the Enhanced Mini-ACE. In addition, a bit-by-bit comparison is performed on the last word transmitted by the RT for each message. If either the received version of any transmitted word is determined to be invalid (sync, encoding, bit count, or parity error) and/or

aceRTBITWrdWrite (continued)

the received version of the last transmitted word does not match the transmitted version, or a failsafe timeout occurs on the respective channel, the Loop Test Failure bit for the respective bus channel will be set.

Bit 12
Handshake Failure
If this bit is set, it indicates that the subsystem has failed to respond with the DMA handshake input DTGRT* asserted within the allotted time, in response to the Enhanced Mini-ACE asserting DTREQ*. Alternatively, a handshake failure will occur if the host PROCESSOR fails to clear STRBD* (high) within the allotted time, after the Enhanced Mini-ACE has asserted its READYD* output (low). The allotted time is 4 µs for a 16 MHz clock, or 3.5 µs for a 12 MHz clock. All of DDC's 1553 Enhanced Mini-ACE cards provide a 16 MHz clock.

Bit 11, Bit 10
Transmitter Shutdown B, Transmitter Shutdown A
Indicates that the transmitter on the respective bus channel has been shut down by a Transmitter shutdown mode code command received on the alternate channel. If an Override transmitter shutdown mode code command is received on the alternate channel, this bit will revert back to logic 0.

Bit 9
Terminal Flag Inhibited
Set to logic 1 if the Enhanced Mini-ACE's Terminal Flag RT Status bit has been disabled by an Inhibit terminal flag mode code command. Will revert to logic 0 if an Override inhibit terminal flag mode code command is received.

Bit 8
Bit Test Fail
Represents the result of the Enhanced Mini-ACE RT's most recent built-in protocol self-test. A value of logic 0 for bit 8 indicates that the test passed. The bit will return a value of logic 1 if the Enhanced Mini-ACE has failed its most recent protocol self-test. If a subsequent performing of the protocol self-test passes, bit 8 will clear to 0. Also, note that the RAM self-test has no effect on bit 8.

aceRTBITWrdWrite (continued)

Bit 7
High Word Count
Set to logic 1 if the most recent message had a high word count error.

Bit 6
Low Word Count
Set to logic 1 if the most recent message had a low word count error.

Bit 5
Incorrect Sync Received
Set to a logic 1 if the Enhanced Mini-ACE detected a Command sync in a received Data Word.

Bit 4
Invalid Word Received
Indicates that the Enhanced Mini-ACE RT received a Data Word containing one or more of the following error types: sync field error, Manchester encoding error, parity error, and/or bit count error.

Bit 3
RT-RT Gap/Sync/Address Error
This bit is set if the Enhanced Mini-ACE RT is the receiving RT for an RT-to-RT transfer and one or more of the following occurs: (1) If the GAP CHECK ENABLED bit (bit 8) of Configuration Register # 5 at memory location 0x09 is set to logic 1 and the transmitting RT responds with a response time of less than 4 µs, per MIL-STD-1553B (mid-parity bit to mid-sync); i.e., less than 2 µs dead time; and/or (2) There is an incorrect sync type or format error (encoding, bit count, and/or parity error) in the transmitting RT Status Word; and/or (3) The RT address field of the transmitting RT Status Word does not match the RT address in the transmit Command Word.

Bit 2
RT-RT No Response Error
If this bit is set to a logic 1, this indicates that for the previous message, the Enhanced Mini-ACE was the receiving RT for an RT-to-RT transfer and that the transmitting RT either did not respond or responded later than the Enhanced Mini-ACE's RT-to-RT Timeout time. The Enhanced Mini-ACE's RT-to-RT Response Timeout Time is defined as the time from the mid-bit crossing of the

aceRTBITWrdWrite (continued)

parity bit of the transmit Command Word to the mid-sync crossing of the transmitting RT Status Word. The value of the RT-to-RT Response Timeout is 18.5 μs by default, or programmable from among nominal values of 18.5, 22.5, 50.5, or 130 μs by calling the aceSetRespTimeOut() function.

Bit 1
RT-RT 2nd Command Word Error
If the Enhanced Mini-ACE is the receiving RT for an RT-to-RT transfer, this bit set to a logic 1 indicates one or more of the following error conditions in the transmit Command Word: (1) T/R bit = logic "0"; (2) subaddress = 00000 or 11111; (3) same RT address field as the receive Command Word.

Bit 0
Command Word Contents Error
Indicates a received command word is not defined in accordance with MIL-STD-1553B specifications. This includes the following undefined Command Words: (1) BROADCAST DISABLED, bit 7 of Configuration Register # 5 at memory location 0x09 is logic 0 **and** the Command Word is a non-mode code, broadcast, or transmit command; (2) The OVERRIDE MODE T/R* ERROR bit, bit 6 of Configuration Register # 3 at memory location 0x07 is logic 0 **and** a message with a T/R* bit of 0, a subaddress/mode field of 00000 or 11111 and a mode code field between 00000 and 01111; (3) BROADCAST DISABLED, bit 7 of Configuration Register # 5 is logic 0 **and** a mode code command that is not permitted to be broadcast (e.g., Transmit status) is sent to the broadcast address (11111).

DESCRIPTION

This function writes the BIT word if set as external. External BIT word is written to the memory location that the transmit Mode BIT word mode code data would be stored (EMA mode code memory offset + 0x13). If the location is configured as internal, this function will return ACE_ERR_PARAMETER.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The BIT location has been configured as ACE_RT_BIT_INTERNAL

aceRTBITWrdWrite (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT wBITWrd = 0x5555;
// Write External BIT. The BIT word configuration must specify External

nResult = aceRTBITWrdWrite(DevNum, &wBITWrd);

if(nResult)
{
      printf("Error in aceRTBITWrdWrite() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTBITWrdConfig()   aceRTBITWrdRead()

aceRTBusyBitsTblClear

This function will disable certain subaddresses from returning the BUSY bit in their status word set to a 1.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTBusyBitsTblClear(S16BIT DevNum,
                                   U16BIT wOwnAddrOrBcst,
                                   U16BIT wTR,
                                   U32BIT dwSAMask);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
- 0 – 31

wOwnAddrOrBcst (input parameter)
This parameter specifies whether the RT address is own or broadcast
Valid values:
- ACE_RT_OWN_ADDRESS
- ACE_RT_BCST_ADDRSS
- ACE_RT_MODIFY_ALL

wTR (input parameter)
Specify the direction Transmit/Receive
Valid values:
- 1 = Transmit
- 0 = Receive
- ACE_RT_MODIFY_ALL

dwSAMask (input parameter)
An unsigned 32-bit packed value that represents the sub-addresses that should respond with the BUSY bit cleared to a value of 0 in the status word. A '1' indicates the BUSY BIT should be inactive. The value is an OR'ed combination of the following

aceRTBusyBitsTblClear (continued)

Valid value:

ACE_RT_SAXX
Specifies the subaddress where XX = 0 – 31

ACE_RT_SA_ALL
Selects all subaddresses

DESCRIPTION

This function will disable a selected subaddress from setting the BUSY bit in their status words. The table is set based on the type of message as defined by the following parameters:
Own Address/Bcst*
T/R*

Using the ACE_RT_MODIFY_ALL constant will clear the status word BUSY bit for all messages of a certain type. In addition to this you can use the ACE_RT_SA_ALL constant to disable the BUSY bit for all subaddresses.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The wTR and/or the wOwnAddrOrBcst input parameter(s) contain an incorrect value

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT wOwnAddrOrBcst = 1, wTR = 1;
U32BIT dwSAMask = (ACE_RT_SA0 | ACE_RT_SA22 | ACE_RT_SA25);

nResult = aceRTBusyBitsTblClear(DevNum, wOwnAddrOrBcst, wTR,
                                dwSAMask);

if(nResult)
{
      printf("Error in aceRTBusyBitsTblClear() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTBusyBitsTblSet() aceRTBusyBitsTblStatus()

aceRTBusyBitsTblSet

This function will enable certain subaddresses to return the BUSY bit in their status word set to a 1.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTBusyBitsTblSet(S16BIT DevNum,
                                 U16BIT wOwnAddrOrBcst,
                                 U16BIT wTR,
                                 U32BIT dwSAMask)
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wOwnAddrOrBcst	(input parameter) This parameter specifies whether the RT address is own or broadcast Valid values: ACE_RT_OWN_ADDRESS ACE_RT_BCST_ADDRSS ACE_RT_MODIFY_ALL
wTR	(input parameter) Specify the direction Transmit/Receive Valid values: 1 = Transmit 0 = Receive ACE_RT_MODIFY_ALL
dwSAMask	(input parameter) An unsigned 32-bit packed value that represents the sub-addresses that should respond with the BUSY bit set in the status word. A ‘1’ indicates the BUSY bit should be active. The value is an OR’ed combination of the following

aceRTBusyBitsTblSet (continued)

Valid value:

ACE_RT_SAXX
Specifies the subaddress where XX = 0 – 31

ACE_RT_SA_ALL
Selects all subaddresses

DESCRIPTION

This function will enable certain subaddresses to return the BUSY bit in their status words set. The table is set based on the type of message as defined by the following parameters:
Own Address/Bcst*
T/R*
Using the ACE_RT_MODIFY_ALL constant will set the status word BUSY BIT for all messages of a certain type. In conjunction with this you can use the ACE_RT_SA_ALL constant to make all subaddresses respond with the busy bit set.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The wTR and/or the wOwnAddrOrBcst input parameter(s) contain an incorrect value

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT wOwnAddrOrBcst = 1, wTR = 1;
U32BIT dwSAMask = ACE_RT_SA0 | ACE_RT_SA22 | ACE_RT_SA25;

nResult = aceRTBusyBitsTblSet(DevNum, wOwnAddrOrBcst, wTR,
                              dwSAMask);

if(nResult)
{
      printf("Error in aceRTBusyBitsTblSet() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTBusyBitsTblClear()   aceRTBusyBitsTblStatus()

aceRTBusyBitsTblStatus

This function reports the status of the BUSY bit.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTBusyBitsTblStatus(S16BIT DevNum,
                                    U16BIT wOwnAddrOrBcst,
                                    U16BIT wTR,
                                    U32BIT *pdwSABusyBits);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wOwnAddrOrBcst	(input parameter) This parameter specifies whether the RT address is own or broadcast Valid values: ACE_RT_OWN_ADDRESS ACE_RT_BCST_ADDRSS
wTR	(input parameter) Specify the direction Transmit/Receive Valid values: 1 = Transmit 0 = Receive
pdwSABusyBits	(output parameter) An unsigned 32-bit packed value that represents the sub-addresses that are presently setup to return the BUSY bit set in the status word to a value of 1. A 1 in this 32-bit packed value indicates the BUSY bit is active. The value returned may be masked with the following macros for decoding. Valid value: ACE_RT_SAXX Specifies the subaddress where XX = 0 – 31

aceRTBusyBitsTblStatus (continued)

ACE_RT_SA_ALL
All subaddresses

DESCRIPTION

This function reads the Busy Bit table and reports the status of each of the 32 subaddresses for a particular type of command.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The wTR and/or the wOwnAddrOrBcst input parameter(s) contain an incorrect value and/or the pdwSABusyBits is Null

EXAMPLE

```
S16BIT DevNum = 0;
U16BIT wOwnAddrOrBcst = ACE_RT_OWN_ADDRESS, wTR = 1;
U32BIT pdwSABusyBits;

/* the value returned in pdwSABusyBits can be decoded by masking with
the Sub-Address macros (ACE_RT_SAXX) */

nResult = aceRTBusyBitsTblStatus(DevNum, wOwnAddrOrBcst, wTR,
                                 &pdwSABusyBits)

if(nResult)
{
      printf("Error in aceRTBusyBitsTblStatus() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTBusyBitsTblClear()   aceRTBusyBitsTblSet()

aceRTConfigure

This function configures a RT.

PROTOTYPE

```
#include "Rtop.h"


S16BIT _DECL aceRTConfigure(S16BIT DevNum,
                            U16BIT wCmdStkSize,
                            U32BIT dwOptions);
```

STATE

Ready

MODE

RT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

wCmdStkSize (input parameter)
This is the size of the desired RT command stack size
Valid values:
ACE_RT_CMDSTK_256
256 words

ACE_RT_CMDSTK_512
512 words

ACE_RT_CMDSTK_1K
1024 words

ACE_RT_CMDSTK_2K
2048 words

dwOptions (input parameter)
The options designate the operation of the RT. The value is an U32BIT value that is an OR'ed combination of the following values.
Valid values:
The following parameters will set bits in Configuration Register # 2 at memory location 0x02:

aceRTConfigure (continued)

ACE_RT_OPT_CLR_SREQ
Sets the Clear Service Request bit 2 to a 1. This will clear a service request after a tx vector word.

ACE_RT_OPT_LOAD_TT
Sets the Load/Transmit Time Tag on Synchronize bit 5 to a 1. This will cause the reception of a Synchronize (with data) mode command which will cause the Data Word from the Synchronize message to be loaded into the Time Tag Register.

ACE_RT_OPT_CLEAR_TT
Sets the Clear Time Tag on Synchronize bit 6 to a 1. This will cause the reception of a Synchronize (without data) mode command which will cause the value of the internal Time Tag Register to clear to 0x0000.

ACE_RT_OPT_OVR_DATA
Sets the Overwrite Invalid Data bit 11 to a 1. This affects the operation of the RT subaddress circular buffer memory management mode. The Lookup Table address pointer will only be updated following a transmit message or following a valid receive or broadcast message to the respective Rx/Bcst subaddress. If the bit is logic 1, the Lookup Table pointer will not be updated following an invalid receive or broadcast message. In addition, if the bit is logic 1, an interrupt request for a circular buffer rollover condition (if enabled) will only occur following the end of a transmit message during which the last location in the circular buffer has been read or following the end of a valid receive or Broadcast message in which the last location in the circular buffer has been written to.

The following parameters will set bits in Configuration Register # 3 at memory location 0x07:

ACE_RT_OPT_OVR_MBIT
Sets Override Mode T/R* Error bit 6 to a 1. This will cause a mode code Command Word with a T/R* bit of 0 and an MSB of the mode code field of 0 will be considered a defined (reserved) mode Command Word. In this configuration, the Enhanced Mini-ACE will respond to such a command and the Message Error bit will not become set.

aceRTConfigure (continued)

ACE_RT_OPT_ALT_STS
Sets Alternate RT Status Word Enable bit 5 to a 1. This will cause all 11 RT Status Word bits to be under control of the host processor, by means of bits 11 through 1 of Configuration Register # 1.

ACE_RT_OPT_IL_RX_D
Sets Illegal Receive Transfer Disable bit 4 to a 1. This will cause the device to not store the received data words to the shared RAM if the ACE receives a receive command that has been illegalized,

ACE_RT_OPT_BSY_RX_D
Sets Busy Receive Transfer Disable bit 3 to a 1. If the host processor has programmed BUSY* to logic "0" or the particular Command Word (broadcast, T/R* bit, subaddress) has been programmed to be busy by means of the Busy lookup table and the Enhanced Mini-ACE RT receives a receive command, the Enhanced Mini-ACE will respond with its Status Word with the Busy bit set and will not store the received Data Words to the shared RAM.

ACE_RT_OPT_SET_RTFG
Sets RTFail*/RTFlag* Wrap Enable bit 2 to a 1. The Terminal flag status word bit will also become set if either a transmitter timeout (660.5 µs) condition had occurred or the ACE RT had failed its loopback test for the previous non-broadcast message. The loopback test is performed on all non-broadcast messages processed by the Enhanced Mini-ACE RT. The received version of all transmitted words is checked for validity (sync and data encoding, bit count, parity) and correct sync type. In addition, a 16-bit comparison is performed on the received version of the last word transmitted by the Enhanced Mini-ACE RT. If any of these checks or comparisons do not verify, the loopback test is considered to have failed.

ACE_RT_OPT_1553A_MC
Sets 1553A Mode Codes Enabled bit 1 to a 1. If this option is chosen, the Enhanced Mini-ACE RT or Message Monitor considers only subaddress 0 to be a mode code subaddress. Subaddress 31 is treated as a standard non-mode code subaddress. In this configuration, the Enhanced Mini-ACE will consider valid and respond only to mode code commands containing no data words. In this configuration, the Enhanced Mini-ACE RT will consider all mode commands followed by data

aceRTConfigure (continued)

words to be invalid and will not respond. In addition the Enhanced Mini-ACE will not decode for the MIL-STD-1553B "Transmit Status" and "Transmit Last Command" mode codes. As a result, the internal RT Status Word Register will be updated as a result of these commands.

The following parameters will set bits in Configuration Register # 4 at memory location 0x08:

ACE_RT_OPT_MC_O_BSY
Sets Mode Command Override Busy Bit 13 to a 1.
If BUSY* is programmed to logic "0" or if Busy Lookup Table (bit 13 of Configuration Register #2) is logic "1" and the respective bit(s) in the Busy Lookup Table (bit 0 of location 0242 and/or bit 15 of location 0243) is programmed to logic "1," the Enhanced Mini-ACE will transmit its Status Word with its BUSY bit set, followed by a single Data Word, in response to either a Transmit Vector Word mode command or a Reserved transmit mode command with data (transmit mode codes 10110 through 11111). The Busy Lookup Table functions are: aceRTBusyBitsTblSet(), aceRTBusyBitsTblClear(), and aceRTBusyBitsTblStatus().

The following parameters will set bits in Configuration Register # 5 at memory location 0x09:

ACE_RT_OPT_BCST_DIS
Sets Broadcast Disabled bit 7 to a 1. The Enhanced Mini-ACE will **not** recognize RT address 31 as the broadcast address. In this instance, RT address 31 may be used as a discrete RT address.

DESCRIPTION

This function configures the Remote Terminal configuration. This routine initializes the EMA for operation as a RT. The library configuration structures and EMA tables are initialized to default values, and the memory structures are created. All RT subaddresses are illegalized after this function has been called.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in RT mode
ACE_ERR_PARAMETER	The wCmdStkSize input parameter contains an invalid value

aceRTConfigure (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U32BIT dwOptions = ACE_RT_OPT_SET_RTFG |  ACE_RT_OPT_LOAD_TT;

// Initiate RT with a 512 word command stack
//Load time tag on sync mode code and set flag if loop back test fails
nResult = aceRTConfigure(DevNum, ACE_RT_CMDSTK_512, dwOptions);

if(nResult)
{
      printf("Error in aceRTConfigure() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTMTConfigure()   aceMTConfigure()

aceRTCreateImageFiles

This function will create image files.

PROTOTYPE

```
#include "Rtop.h"


S16BIT _DECL aceRTCreateImageFiles(S16BIT DevNum,
                                   char *pszIFile,
                                   char *pszHFile);
```

STATE

Ready

MODE

RT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
pszIFile	(output parameter) A character pointer to a string that contains the name of the file where the EMA image will be saved.
pszHFile	(output parameter) A character pointer to a string that contains the name of the file where the header information will be saved.

DESCRIPTION

This function creates two files. The first is a binary image of the Enhanced Mini-ACE's memory after it is initialized and setup by the appropriate functions. The second is a 'C' header file that contains all memory and structure offsets. The header file also contains interface functions that provide access to the EMA without the need for the entire RTL.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in RT mode
ACE_ERR_PARAMETER	The pszIFile parameter is Null and/or the pszHFile parameter is Null

aceRTCreateImageFiles (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

char pszIFile = “RTIMAGE.IMG”, pszHFile = “RTIMAGE.H”;

nResult = aceRTCreateImageFiles(DevNum, pszIFile, pszHFile);

if(nResult)
{
      printf(“Error in aceRTCreateImageFiles() function \n”);
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

None

aceRTDataBlkCircBufInfo

This function will return information about the circular buffer.

PROTOTYPE

```
#include "Rtop.h"


S16BIT _DECL aceRTDataBlkCircBufInfo(S16BIT DevNum,
                                     S16BIT nDataBlkID,
                                     U16BIT *pUserRWOffset,
                                     U16BIT *pAceRWOffset);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nDataBlkID	(input parameter) Unique data block ID that was supplied by the user during data block creation. Valid values: >=0
pUserRWOffset	(output parameter) Pointer to an unsigned 16-bit word to be filled with the last location read or written by the user.
pAceRWOffset	(output parameter) Pointer to an unsigned 16-bit word to be filled with the last location read/written by the hardware.

DESCRIPTION

This function returns information about a circular buffer. This info includes the last read or written location performed by the user and the last location read or written by the EMA itself.

Note: The addresses are offsets based on the starting address of the circular buffer in the EMA. If the User Offset = 32 and the Card Offset = 0, then 96 will be returned.

aceRTDataBlkCircBufInfo (continued)

RETURN VALUE

S16BIT nResult   Returns the difference in Offset locations, counting forward, between the User Offset and the Card Offset (for example, If the User Offset = 32 and the Card Offset = 0, then 96 will be returned)

ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The nDataBlkID input parameter is less than zero and/or the pUserRWOffset parameter is Null and/or the pAceRWOffset parameter is Null
ACE_ERR_NODE_NOT_FOUND	The data block specified by the nDataBlkID input parameter does not exist
ACE_ERR_RT_DBLK_NOT_CB	The data block specified by the nDataBlkID input parameter is not defined to be a circular buffer

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
S16BIT nDataBlkID = 42;
U16BIT pUserRWOffset, pAceRWOffset;

nResult = aceRTDataBlkCircBufInfo(DevNum, nDataBlkID, &pUserRWOffset,
                                  &pAceRWOffset);

if(nResult)
{
      printf("Error in aceRTDataBlkCircBufInfo() function \n");
      PrintOutError(nResult);
      return;
}

printf("Last   user   R/W   at   %x,   \nLast   EMA   R/W   at   %x\n",
      pUserRWOffset, pAceRWOffset);
```

SEE ALSO

None

aceRTDataBlkCreate

This function creates a RT data block to be used.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTDataBlkCreate(S16BIT DevNum,
                                S16BIT nDataBlkID,
                                U16BIT wDataBlkType,
                                U16BIT *pBuffer,
                                U16BIT wBufferSize);
```

STATE

Ready

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nDataBlkID	(input parameter) A unique user supplied ID that will identify this data block for future use and removal Valid values: >0
wDataBlkType	The type of data block to be allocated Valid Values: 1 - 32 This value defines the number of words for Single Buffer Mode ACE_RT_DBLK_DOUBLE Double buffered message datablock 64 words long ACE_RT_DBLK_C_128 Circular buffer 128 words long ACE_RT_DBLK_C_256 Circular buffer 256 words long ACE_RT_DBLK_C_512 Circular buffer 512 words long

aceRTDataBlkCreate (continued)

ACE_RT_DBLK_C_1K
Circular buffer 1024 words long

ACE_RT_DBLK_C_2K
Circular buffer 2048 words long

ACE_RT_DBLK_C_4K
Circular buffer 4096 words long

ACE_RT_DBLK_C_8K
Circular buffer 8192 words long

ACE_RT_DBLK_GBL_C_128
Global circular buffer 128 words long

ACE_RT_DBLK_GBL_C_256
Global circular buffer 256 words long

ACE_RT_DBLK_GBL_C_512
Global circular buffer 512 words long

ACE_RT_DBLK_GBL_C_1K
Global circ buffer 1024 words long

ACE_RT_DBLK_GBL_C_2K
Global circular buffer 2048 words long

ACE_RT_DBLK_GBL_C_4K
Global circular buffer 4096 words long

ACE_RT_DBLK_GBL_C_8K
Global circular buffer 8192 words long

pBuffer (input parameter)
Address of a U16BIT buffer containing info to be copied to the created data block

wBufferSize (input parameter)
Number of words in buffer to be copied into the created data block

DESCRIPTION

This function creates an RT data block identified by the nDataBlkID input parameter. After this data block has been created, it can be used by an RT at any subaddress.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode

aceRTDataBlkCreate (continued)

ACE_ERR_PARAMETER	The wDataBlkType and/or nDataBlkID input parameter(s) contain an incorrect value
ACE_ERR_MEMMGR_FAIL	Memory for the data block could not be allocated
ACE_ERR_RT_DBLK_ALLOC	A new data block could not be created

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
S16BIT nDataBlkID = 42;
U16BIT pBuffer[1024], wBufferSize = 1024;

/* Load pBuffer from external data file */

/* create a 1K data block as a circular buffer and initialize it to
data that was read from an external file */

nResult = aceRTDataBlkCreate(DevNum, nDataBlkID,
                             ACE_RT_DBLK_C_1K, pBuffer,
                             wBufferSize);

if(nResult)
{
      printf("Error in aceRTDataBlkCreate() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTDataBlkDelete()

aceRTDataBlkDelete

This function will delete a data block.

PROTOTYPE

```
#include "Rtop.h"

BIT _DECL aceRTDataBlkDelete(S16BIT DevNum,
                             S16BIT nDataBlkID);
```

STATE

Ready

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nDataBlkID	(input parameter) Unique user supplied ID that identifies the data block that was previously created using the aceRTDataBlkCreate() function. Valid values: >0

DESCRIPTION

This function removes a data block from memory and frees all resources associated with it.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The nDataBlkID input parameter specified by the user contains a value less than zero
ACE_ERR_NODE_NOT_FOUND	The data block specified by the nDataBlkID input parameter does not exist

aceRTDataBlkDelete (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
S16BIT nDataBlkID = 42;

nResult = aceRTDataBlkDelete(DevNum, nDataBlkID)

if(nResult)
{
      printf("Error in aceRTDataBlkDelete() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTDataBlkCreate()

aceRTDataBlkMapToSA

This function maps a data block to a subaddress.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTDataBlkMapToSA(S16BIT DevNum,
                                 S16BIT nDataBlkID,
                                 U16BIT wSA,
                                 U16BIT wMsgType,
                                 U16BIT wIrqOptions,
                                 U16BIT wLegalizeSA)
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

nDataBlkID (input parameter)
The unique user supplied ID of the previously created data block that will be mapped to an SA. The user provided this ID during creation of the data block with the aceRTDataBlkCreate() function.
Valid values:
>0

wSA (input parameter)
The subaddress to be mapped
Valid values:
0 – 31

wMsgType (input parameter)
Description of the message types that will be mapped by this command. This parameter is generated by OR'ing the following message types together.

aceRTDataBlkMapToSA (continued)

Valid values:
ACE_RT_MSGTYPE_RX
ACE_RT_MSGTYPE_TX
ACE_RT_MSGTYPE_BCST
ACE_RT_MSGTYPE_ALL

wIrqOptions (input parameter)
Interrupts will be generated based on the value of this parameter. The value for this parameter can be 0 or any of the following macros "OR'd" together.
Valid values:
0
No IRQ options

ACE_RT_DBLK_EOM_IRQ (end of message)
This will cause an interrupt at the end of the message to be set in the RT Subaddress Control Word. An interrupt will be created at the end of every message if the EOM bit is set in the Interrupt Mask Register by calling the aceSetIrqConditions() function.

ACE_RT_DBLK_CIRC_IRQ (circular buffer)
This will cause an interrupt when the circular buffer rolls over. An interrupt will be created if one of the CIRCBUF_ROVER bits are set in the Interrupt Mask Register by calling the aceSetIrqConditions() function.

wLegalizeSA If this value is set to TRUE, then the Sub-Address being mapped will also be legalized.
Valid values:
TRUE
FALSE

DESCRIPTION

This function maps a Data Block (defined using aceRTDataBlkCreate) with one of the 32 subaddresses of the RT. The parameters are the RT subaddress (0-31), the Data Block ID, the Type of messages that will use the Data Block (Tx, Rx, and/or Bcst), and the options for messages received that will access this data block. If the sub-address being mapped is not legal, the Legalize parameter may be set to TRUE in order to legalize it.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode

aceRTDataBlkMapToSA (continued)

ACE_ERR_PARAMETER	The nDataBlkID parameter contains a value less than zero, and/or the wSA input parameter is greater than 31, and/or the wMsgType input parameter is 0 or greater than 7
ACE_ERR_NODE_NOT_FOUND	The data block specified by the nDataBlkID input parameter does not exist
S16BIT nResult	The data block is already mapped to the subaddress specified by the nResult value

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0
S16BIT nDataBlkID = 42;
U16BIT wSA = 13, wMsgType = ACE_RT_MSGTYPE_RX;
U16BIT wIrqOptions = ACE_RT_DBLK_CIRC_IRQ;
U16BIT wLegalizeSA = TRUE;
/* Create data block.  Map to SA13 for Receive messages.  Options for
generating interrupt is for circular buffer rollover.  If this Sub-
address is not legal, then legalize it all done by the
aceRTDataBlkMapToSA() function */
nResult = aceRTDataBlkMapToSA(DevNum, nDataBlkID, wSA, wMsgType,
                              wIrqOptions, wLegalizeSA);
if(nResult)
{
      printf("Error in aceRTDataBlkMapToSA() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTDataBlkUnmapFromSA()

aceRTDataBlkRead

This function reads data from a data block.

PROTOTYPE

```
#include “Rtop.h”

S16BIT _DECL aceRTDataBlkRead(S16BIT DevNum,
                              S16BIT nDataBlkID,
                              U16BIT *pBuffer,
                              U16BIT wBufferSize,
                              U16BIT wOffset);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nDataBlkID	(input parameter) Unique user supplied ID that was established during data block creation with the aceRTDataBlkCreate() function. Valid values: >0
pBuffer	(output parameter) Pointer to an unsigned 16-bit user buffer that will receive the data from the data block.
wBufferSize	The size of the buffer in words Valid values: >0
wOffset	Offset into data buffer where read data will be written to Valid values: >=0

DESCRIPTION

This function transfers data from an EMA data block to a host buffer given the unique data block ID and device number.

aceRTDataBlkRead (continued)

RETURN VALUE

ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The nDataBlkID input parameter is less than zero and/or the pBuffer input parameter is Null and/or the wBufferSize input parameter is less than one
ACE_ERR_NODE_NOT_FOUND	The data block specified by the nDataBlkID input parameter does not exist
S16BIT nResult	The number of words read

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
S16BIT nDataBlkID = 42;
U16BIT pBuffer[1024], wBufferSize = 1024, wOffset = 64;
/* read data from nDataBlkID into pBuffer starting at
pBuffer offset (word number) 64 */

nResult = aceRTDataBlkRead(DevNum, nDataBlkID, pBuffer,
                           wBufferSize, wOffset);
if(nResult)
{
      printf("Error in aceRTDataBlkRead() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTDataBlkWrite()

aceRTDataBlkUnmapFromSA

This function unmaps a data block from a subaddress.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTDataBlkUnmapFromSA(S16BIT DevNum,
                                     S16BIT nDataBlkID,
                                     U16BIT wSA,
                                     U16BIT wMsgType);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

nDataBlkID (input parameter)
The unique user supplied ID of the previously created data block that will be unmapped from an SA. The user provided this ID during creation of the data block with the aceBCDataBlkCreate() function.
Valid values:
>0

wSA (input parameter)
The subaddress to be unmapped
Valid values:
1 - 32
(SA 0 = SA 32)

wMsgType (input parameter)
Description of the message types that will be unmapped by this command. This parameter is generated by OR'ing the following message types together.

aceRTDataBlkUnmapFromSA (continued)

Valid values:
ACE_RT_MSGTYPE_RX
ACE_RT_MSGTYPE_TX
ACE_RT_MSGTYPE_BCST
ACE_RT_MSGTYPE_ALL

DESCRIPTION

This function unmaps a data block from a subaddress. The parameters are the RT subaddress (1-32), the Data Block ID, and the Type of messages that will use the Data Block (Tx, Rx, and/or Bcst).

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The wSA input parameter is 0 or greater than 32, and/or the wMsgType input parameter is 0 or greater than 7
ACE_ERR_NODE_NOT_FOUND	The data block specified by the nDataBlkID input parameter does not exist

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
S16BIT nDataBlkID = 42;
U16BIT wSA = 13, wMsgType = ACE_RT_MSGTYPE_RX;

/* Unmap previously created and mapped data block from specified Sub-
address.  Unmap from SA13 for Receive messages. */

nResult = aceRTDataBlkUnmapFromSA(DevNum, nDataBlkID, wSA, wMsgType);

if(nResult)
{
      printf("Error in aceRTDataBlkUnmapFromSA() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTDataBlkmapToSA()

aceRTDataBlkWrite

This function writes to a data block.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTDataBlkWrite(S16BIT DevNum,
                               S16BIT nDataBlkID,
                               U16BIT *pBuffer,
                               U16BIT wBufferSize,
                               U16BIT wOffset);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
nDataBlkID	(input parameter) Unique user supplied ID that was established during data block creation with the aceRTDataBlkCreate() function Valid values: >0
pBuffer	Pointer to an unsigned 16-bit word user buffer that will supply the data to be written to the data block
wBufferSize	The number of unsigned 16-bit words to write to the data block from the buffer Valid values: >0
wOffset	Offset into data buffer where the function will start to write those unsigned 16-bit words to the data block.

DESCRIPTION

This function writes to a data block given a buffer and a data block ID.

aceRTDataBlkWrite (continued)

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The nDataBlkID input parameter is less than zero and/or the pBuffer input parameter is Null and/or the wBufferSize input parameter is less than one
ACE_ERR_NODE_NOT_FOUND	The data block specified by the nDataBlkID input parameter does not exist

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
S16BIT nDataBlkID = 42;
U16BIT pBuffer[1024], wBufferSize = 1024, wOffset = 64;

/* Load pBuffer with data from an external file */
/* write data to nDataBlkID from pBuffer starting at
offset (word number) 64 */
nResult = aceRTDataBlkWrite(DevNum, nDataBlkID, pBuffer,
                            wBufferSize, wOffset);

if(nResult)
{
      printf("Error in aceRTDataBlkWrite() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTDataBlkRead()

aceRTDecodeRawMsg

This function will decode a raw message into a formatted message structure.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTDecodeRawMsg(S16BIT DevNum,
                               U16BIT *pBuffer,
                               MSGSTRUCT *pMsg)
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

pBuffer (input parameter)
Pointer to an unsigned 16-bit buffer that is ACE_MSGSIZE_RT(36) words long containing the raw messages
Valid values:
Array length >= ACE_MSGSIZE_RT

pMsg (output parameter)
Pointer to a structure of type MSGSTRUCT that will contain the resultant decoded message. The table below lists all member variables that exist in the MSGSTRUCT structure along with their definition.

Member Variable Name	Definition
wBlkSts	Contains the block status word of the message
wTimeTag	Contains the time tag of the message
wCmdWrd1	Contains the command word
wCmdWrd2	Contains the second command word for RT to RT transfers
wCmdWrd1Flg	Indicates the validity of the first command word
wCmdWrd2Flg	Indicates the validity of the second command word
wStsWrd1	Contains first status word
wStsWrd2	Contains second status word

aceRTDecodeRawMsg (continued)

Member Variable Name	Definition
wStsWrd1Flg	Indicates the validity of the first status word
wStsWrd2Flg	Indicates the validity of the second status word
wWordCount	Contains the number of valid data words
adataWrds[32]	An array that will contain the data words
wBCCtrlWrd	Contains the BC Control Word for BC mode messages only
wBCGapTime	Contains the message gap time only for BC
wBCLoopBack1	Contains the first looped back word for BC mode messages only
wBCLoopBack2	Contains the second looped back word for BC mode messages only
wBCLoopBack1Flg	Indicates validity of first loop back word for BC mode message only
wBCLoopBack2Flg	Indicates validity of second loop back word for BC mode message only

DESCRIPTION

This function takes an unsigned 16-bit buffer and decodes the raw message it contains into a decoded structure of type MSGSTRUCT. The decoding process breaks down the raw message into a neat, well defined structure that can be used for data processing or message display.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_MSGSTRUCT	The pMsg parameter is Null
ACE_ERR_INVALID_BUF	The pBuffer input parameter is Null

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT pBuffer[36];
MSGSTRUCT pMsg[36];

/*  initialize RT, start, read raw messages using aceRTGetStkMsgsRaw.
Process the raw messages into decoded message using the following */

nResult = aceRTDecodeRawMsg(DevNum, pBuffer, pMsg)

if(nResult)
{
      printf("Error in aceRTDecodeRawMsg() function \n");
      PrintOutError(nResult);
      return;
}
```

aceRTDecodeRawMsg (continued)

SEE ALSO

aceRTGetStkMsgsRaw()	aceRTGetStkMsgsDecoded()
aceRTGetHBufMsgsRaw()	aceRTGetHBufMsgsDecoded()

aceRTGetAddress

This function returns the Remote Terminal address.

PROTOTYPE

#include “rtop.h”

S16BIT _DECL aceRTGetAddress(S16BIT DevNum,
U16BIT *pRTAddress);

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
pRTAddress	(output parameter) This is a pointer to an unsigned 16-bit word that will contain the RT address for the device number input by the user

DESCRIPTION

This function returns the Remote Terminal address for the logical device number input by the user in the DevNum input parameter.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The pRTAddress parameter is Null

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT *pRTAddress;;

nResult = aceRTGetAddress(DevNum, pRTAddress);

if(nResult)
{
      printf(“Error in aceRTGetAddress() function \n”);
      PrintOutError(nResult);
      return;
}
```

aceRTGetAddress (continued)

SEE ALSO

None

aceRTGetAddrSource

This function will get whether the RT address source is internal or external.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTBusyBitsTblSet(S16BIT DevNum,
                                  U16BIT *wRTSource);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wRTSource	(output parameter) This parameter will specify whether the RT address is internal or external after the function has been run Valid values: ACE_RT_INTERNAL_ADDR ACE_RT_EXTERNAL_ADDR

DESCRIPTION

This function will get whether the RT address source is internal or external.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode

aceRTGetAddrSource (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT *wRTSource

nResult = aceRTGetAddrSource(DevNum, wRTSource);

if(nResult)
{
      printf("Error in aceRTGetAddrSource() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTSetAddrSource()   aceSetAddress()

aceRTGetHBufMetric

This function returns performance information about the Host Buffer.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTGetHBufMetric(S16BIT DevNum,
                                HBUFMETRIC *pMetric,
                                U16BIT bReset);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

pMetric (output parameter)
Pointer to an HBUFMETRIC structure to be filled in with metrics. The HBUFMETRIC structure contains the following members: dwCount, dwLost, dwPctFull, and dwHighPct. The dwCount member parameter contains the total number of messages in the host buffer. The dwLost member parameter contains the total number of messages lost in the host buffer. The dwPctFull parameter contains the percentage of the host buffer used at one snapshot in time. The dwHighPct parameter contains the highest percentage of the host buffer used over an extended period of time.

Member Variable Name	**Definition**
dwCount	The number of messages in the host buffer
dwLost	The total number of messages lost since the host buffer was installed
dwPctFull	The current percentage of host buffer used
dwHighPct	The highest percentage of the host buffer used since the host buffer was installed or metrics were reset

bReset (input parameter)
This will specify if the highest percentage should be reset after this function returns.

aceRTGetHBufMetric (continued)

Valid values:

FALSE (0)
Do not reset the highest percentage value

TRUE (1)
Reset the highest percentage value

DESCRIPTION

This function returns performance information about the RT Command Stack that is also referred to as the RT Descriptor Stack. Built-in test metrics can report the number of messages in the host buffer, the total number of messages lost since the host buffer was installed, the current percentage of the host buffer that is used, and the highest percentage of the host buffer used since it was installed.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The pMetric pointer input by the user is NULL.
ACE_ERR_METRICS_NOT_ENA	Metrics are not enabled and should be set by calling the aceSetMetrics() function

EXAMPLE

```
S16BIT DevNum = 0;
HBUFMETRIC *pMetric;
U16BIT bReset = 1;
S16BIT nResult = 0;

nResult = aceRTGetHBufMetric(DevNum, pMetric, bReset)

if(nResult)
{
      printf("Error in aceRTGetHBufMetric () function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceSetMetrics()	aceRTGetStkMetric()
aceMTGetHBufMetric()	aceMTGetStkMetric()

aceRTGetHBufMsgCount

This function returns the number of messages in the host buffer.

PROTOTYPE

#include “Rtop.h”

S16BIT _DECL aceRTGetHBufMsgCount(S16BIT DevNum);

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function returns the number of messages in the host buffer. Only complete messages that were transferred to the host buffer will be included in the count. All messages may be retrieved from the host buffer with either the aceGetHBufMsgsRaw() function or the aceGetHBufMsgDecoded() function.

RETURN VALUE

S16BIT nResult	The number of messages in the host buffer
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceRTGetHBufMsgCount(DevNum);

if(nResult < 0)
{
      printf(“Error in aceRTGetHBufMsgCount() function \n”);
      PrintOutError(nResult);
      return;
}
```

aceRTGetHBufMsgCount (continued)

```
else
{
      printf("Number of message in Hbuf = %d\n, nResult);
}
```

SEE ALSO

aceRTStkToHBuf()
aceRTGetHBufMsgsRaw()
aceRTGetHBufMsgDecoded()

aceRTGetHBufMsgDecoded

This function reads a raw message from the host buffer and decodes it.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTGetHBufMsgDecoded(S16BIT DevNum,
                                    MSGSTRUCT *pMsg,
                                    U32BIT *pdwMsgCount,
                                    U32BIT *pdwMsgLostStk,
                                    U32BIT *pdwMsgLostHBuf,
                                    U16BIT wMsgLoc);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

pMsg (output parameter)
pointer to a structure of type MSGSTRUCT that will contain the entry read off of the host buffer. The table below lists all member variables that exist in the MSGSTRUCT structure along with their definition.

Member Variable Name	Definition
wBlkSts	Contains the block status word of the message
wTimeTag	Contains the time tag of the message
wCmdWrd1	Contains the command word
wCmdWrd2	Contains the second command word for RT to RT transfers
wCmdWrd1Flg	Indicates the validity of the first command word
wCmdWrd2Flg	Indicates the validity of the second command word
wStsWrd1	Contains first status word
wStsWrd2	Contains second status word

aceRTGetHBufMsgDecoded (continued)

Member Variable Name	Definition
wStsWrd1Flg	Indicates the validity of the first status word
wStsWrd2Flg	Indicates the validity of the second status word
wWordCount	Contains the number of valid data words
adataWrds[32]	An array that will contain the data words
wBCCtrlWrd	Contains the BC Control Word for BC mode messages only
wBCGapTime	Contains the message gap time only for BC
wBCLoopBack1	Contains the first looped back word for BC mode messages only
wBCLoopBack2	Contains the second looped back word for BC mode messages only
wBCLoopBack1Flg	Indicates validity of first loop back word for BC mode message only
wBCLoopBack2Flg	Indicates validity of second loop back word for BC mode message only

pdwMsgCount (output parameter)
pointer to an unsigned 16-bit parameter that will contain the number of messages decoded.
Valid values:
0 = No messages returned
1 = One message decoded and returned

pdwMsgLostStk (output parameter)
The approximate number of messages lost due to a stack full condition.

pdwMsgLostHBuf (output parameter)
Possible lost message count when messages were transferred from the device's stack to the host buffer. This would be due to stack overflow prior to transferring messages to the host buffer.

wMsgLoc (input parameter)
Parameter that describes which message should be read off of the RT Command Stack and whether it should be purged or left on the stack. Next indicates that the next 'Unread' message should be read. Latest will be used if the latest message received is to be read from the stack. This may leave messages unread between the last read message and the latest message received.
Valid values:

ACE_RT_MSGLOC_NEXT_PURGE
Reads next message and takes it off of the host buffer

ACE_RT_MSGLOC_NEXT_NPURGE
Reads next message and leaves it on the host buffer

aceRTGetHBufMsgDecoded (continued)

ACE_RT_MSGLOC_LATEST_PURGE
Reads current message and takes it off of the host buffer

ACE_RT_MSGLOC_LATEST_NPURGE
Reads current message and leaves it on the host buffer

DESCRIPTION

This function reads and decodes a message from the host buffer, if one is present, and places the decoded message into the formatted MSGSTRUCT structure. The function will get a raw message from the host buffer and then call the aceRTDecodeRawMsg() function to pass the formatted message to the pMsg parameter.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The pdwMsgLostHBuf parameter and/or the pdwMsgLostStk parameter is Null
ACE_ERR_MT_MSGLOC	The wMsgLoc input parameter is greater than three

EXAMPLE

```
S16BIT DevNum = 0;
U16BIT wMsgLoc = ACE_RT_MSGLOC_NEXT_PURGE;
U32BIT pdwMsgCount, pdwMsgLostStk, pdwMsgLostHBuf;
MSGSTRUCT pMsg;

/* Read the next message from the Hbuf, and purge when completed */

nResult = aceRTGetHBufMsgDecoded(S16BIT DevNum, &pMsg, &pdwMsgCount,
                                 &pdwMsgLostStk, &pdwMsgLostHBuf,
                                 wMsgLoc);
if(nResult)
{
      printf("Error in aceRTGetHBufMsgDecoded() function \n");
      PrintOutError(nResult);
      return;
}

else
{
printf("Messages Read = %d, messages lost (buf) = %d,       messages
lost (HBuf) = %d\n, pdwMsgCount,    pdwMsgLostStk,
      pdwMsgLostHBuf);
}
```

SEE ALSO

aceRTStkToHBuf()	aceRTGetHBufMsgCount()
aceRTGetHBufMsgsRaw()	

aceRTGetHBufMsgsRaw

This function will read raw messages from the host buffer.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTGetHBufMsgsRaw(S16BIT DevNum,
                                 U16BIT *pBuffer,
                                 U16BIT wBufferSize,
                                 U32BIT *pdwMsgCount,
                                 U32BIT *pdwMsgLostStk,
                                 U32BIT *pdwMsgLostHBuf);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum
(input parameter)
Logical Device Number
Valid values:
0 – 31

pBuffer
(output parameter)
Pointer to an unsigned 16-bit word buffer that will contain the raw messages read from the host buffer.

wBufferSize
(input parameter)
Size of buffer in 16-bit words
Valid value:
There is no requirement for size of the buffer as there is for the host buffer, but it is a more efficient use of resources if the buffer size is a multiple of ACE_MSGSIZE_RT. For example, if it is desired to copy 10 messages at a time from the host buffer to the user buffer, the user buffer size would be ACE_MSGSIZE_RT * 10.

pdwMsgCount
(output parameter)
Pointer to an unsigned 32-bit double word to contain the message count read from the host buffer.

aceRTGetHBufMsgsRaw (continued)

pdwMsgLostStk (output parameter)
Pointer to an unsigned 32-bit double word that contains the approximate number of messages lost due to a stack full condition.

pdwMsgLostHBuf (output parameter)
Pointer to an unsigned 32-bit double word that contains the possible lost message count when messages were transferred from the device's stack to the host buffer. This would be due to a stack overflow prior to transferring messages to the host buffer.

DESCRIPTION

This function reads as many messages as possible off of the host buffer. If no errors occur the number of messages read will be returned. The limiting factor when copying messages to the local buffer is the local buffer size and the number of messages available on the host buffer.

Note: Each message is a fixed length of ACE_MSGSIZE_RT words. This macro should be used when creating the host buffer as the size of this structure is subject to change based on the version of the library.

Note: This function will still return ACE_ERR_SUCCESS if there were no messages to read. If this is the case, the pdwMsgCount parameter will point to a value of zero.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The pdwMsgLostHBuf parameter and/or the pdwMsgLostStk parameter is Null
ACE_ERR_INVALID_BUF	The pBuffer parameter is Null and/or the wBufferSize parameter is less than ACE_MSGSIZE_RT

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT pBuffer[ACE_MSGSIZE_RT << 5];
U16BIT wBufferSize = ACE_MSGSIZE_RT << 5; /* 32 messages */
U32BIT pdwMsgCount, pdwMsgLostStk, pdwMsgLostHBuf;

nResult = aceRTGetHBufMsgsRaw(DevNum, &pBuffer, wBufferSize,
                              &pdwMsgCount, &pdwMsgLostStk,
                              &pdwMsgLostHBuf);

if(nResult)
```

aceRTGetHBufMsgsRaw (continued)

```
{
      printf("Error in aceRTGetHBufMsgsRaw() function \n");
      PrintOutError(nResult);
      return;
}

else
{
      printf("Messages Read = %d, messages lost (buf) = %d,
            messages lost (HBuf) = %d\n, pdwMsgCount, pdwMsgLostStk,
            pdwMsgLostHBuf);
}
```

SEE ALSO

aceRTStkToHBuf()
aceRTGetHBufMsgCount()
aceRTGetHBufMsgDecoded()

aceRTGetStkMetric

This function returns performance information about the RT Command Stack.

PROTOTYPE

```
#include "rtop.h"

S16BIT _DECL aceMTGetHBufMetric(S16BIT DevNum,
                                STKMETRIC *pMetric,
                                U16BIT bReset);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum
(input parameter)
Logical Device Number
Valid values:
0 – 31

pMetric
(output parameter)
Pointer to a STKMETRIC structure to be filled in with metrics. The STKMETRIC structure contains the following members: dwLost, dwPctFull, and dwHighPct. The dwLost member parameter contains the total number of messages lost on the hardware stack. The dwPctFull parameter contains the percentage of the stack used at one snapshot in time. The dwHighPct parameter contains the highest percentage of the stack used over an extended period of time.

bReset
(input parameter)
This will specify if the highest perecentage should be reset after this function returns
Valid values:
FALSE (0)
Do not reset the highest percentage value

TRUE (1)
Reset the highest percentage value

aceRTGetStkMetric (continued)

DESCRIPTION

This function returns performance information about the RT Command Stack that is also referred to as the RT Descriptor Stack. Built-in test metrics can report the number of messages in the host buffer, the total number of messages lost since the host buffer was installed, the current percentage of the host buffer that is used, and the highest percentage of the host buffer used since it was installed.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The pMetric pointer input by the user is NULL
ACE_ERR_METRICS_NOT_ENA	Metrics are not enabled and should be set by calling the aceSetMetrics() function

EXAMPLE

```
S16BIT DevNum = 0;
STKMETRIC *pMetric;
U16BIT bReset = 1;
S16BIT nResult = 0;

nResult = aceRTGetStkMetric(DevNum, pMetric, bReset)

if(nResult)
{
      printf("Error in aceRTGetStkMetric() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceSetMetrics()	aceMTGetHBufMetric()
aceRTGetHBufMetric()	aceMTGetStkMetric()

aceRTGetStkMsgDecoded

This function reads raw messages off of the RT command stack and decodes them into a message structure so that they can be easily read and viewed.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTGetStkMsgDecoded(S16BIT DevNum,
                                   MSGSTRUCT *pMsg,
                                   U16BIT wMsgLoc);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

pMsg (output parameter)
Pointer to a MSGSTRUCT parameter that will return the decoded message. The table below lists all member variables that exist in the MSGSTRUCT structure along with their definition.

Member Variable Name	Definition
wBlkSts	Contains the block status word of the message
wTimeTag	Contains the time tag of the message
wCmdWrd1	Contains the command word
wCmdWrd2	Contains the second command word for RT to RT transfers
wCmdWrd1Flg	Indicates the validity of the first command word
wCmdWrd2Flg	Indicates the validity of the second command word
wStsWrd1	Contains first status word
wStsWrd2	Contains second status word

aceRTGetStkMsgDecoded (continued)

Member Variable Name	Definition
wStsWrd1Flg	Indicates the validity of the first status word
wStsWrd2Flg	Indicates the validity of the second status word
wWordCount	Contains the number of valid data words
adataWrds[32]	An array that will contain the data words
wBCCtrlWrd	Contains the BC Control Word for BC mode messages only
wBCGapTime	Contains the message gap time only for BC
wBCLoopBack1	Contains the first looped back word for BC mode messages only
wBCLoopBack2	Contains the second looped back word for BC mode messages only
wBCLoopBack1Flg	Indicates validity of first loop back word for BC mode message only
wBCLoopBack2Flg	Indicates validity of second loop back word for BC mode message only

wMsgLoc (input parameter)
Parameter that describes which message should be read off of the RT Command Stack and whether it should be purged or left on the stack. Next indicates that the next 'Unread' message should be read. Latest will be used if the latest message received is to be read from the stack. This may leave messages unread between the last read message and the latest message received.
Valid values:

ACE_RT_MSGLOC_NEXT_PURGE
Reads next message and takes it off of the stack

ACE_RT_MSGLOC_NEXT_NPURGE
Reads next message and leaves it on the stack

ACE_RT_MSGLOC_LATEST_PURGE
Reads current message and takes it off of the stack

ACE_RT_MSGLOC_LATEST_NPURGE
Reads current message and leaves it on the stack

DESCRIPTION

This function reads either the next unread message or the latest message received on the RT Command Stack. It decodes the message by placing all the info into a MSGSTRUCT by calling the aceRTDecodeRawMsg() function. After reading the message, the user may have this routine purge the message from the RT stack or leave it in place.

aceRTGetStkMsgDecoded (continued)

RETURN VALUE

S16BIT nResult	1 if message read, 0 if no message read
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_MSGSTRUCT	The pMsg parameter is Null
ACE_ERR_PARAMETER	The wMsgLoc input parameter is greater than three

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT wMsgLoc;
MSGSTRUCT pMsg;

nResult = aceRTGetStkMsgDecoded(DevNum, &pMsg, wMsgLoc);

if(nResult < 0)
{
      printf("Error in aceRTGetStkMsgDecoded() function \n");
      PrintOutError(nResult);
      return;
}

else
{
      printf("Number of Messages returned = %d\n", nResult);
}
```

SEE ALSO

aceRTDecodeRawMsg()	aceRTGetStkMsgsRaw()
aceRTGetHBufMsgsRaw()	aceRTGetHBufMsgsDecoded()

aceRTGetStkMsgsRaw

This function reads raw messages off of the RT command stack.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTGetStkMsgsRaw(S16BIT DevNum,
                                U16BIT *pBuffer,
                                U16BIT wBufferSize);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
pBuffer	(output parameter) Pointer to an unsigned 16-bit buffer that will be used to return the raw RT Stack Messages. Each message is a fixed length of ACE_MSGSIZE_RT words in length.
wBufferSize	(input parameter) This is the size in words of the unsigned 16-bit word buffer that will contain the returned raw messages from the RT command stack. The most efficient size for this buffer is calculated as ACE_MSGSIZE_RT. This value represents the maximum number of words to be read.

DESCRIPTION

This function reads as many messages as possible off of the RT command stack. If no errors occur the amount of messages will be returned. The limiting factor when copying messages to the buffer is the buffer size and the number of messages available on the stack.

Note: Each message is a fixed length of ACE_MSGSIZE_RT words.

aceRTGetStkMsgsRaw (continued)

RETURN VALUE

S16BIT nResult	This represents the number of messages transferred to the buffer
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_INVALID_BUF	The pBuffer parameter is Null and/or the wBufferSize input parameter is less than ACE_MSGSIZE_RT

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT pBuffer[ACE_MSGSIZE_RT << 8]; /* 256 messages */
U16BIT wBufferSize = ACE_MSGSIZE_RT << 8;

nResult = aceRTGetStkMsgsRaw(DevNum, pBuffer, wBufferSize);

if(nResult < 0)
{
      printf("Error in aceRTGetStkMsgsRaw() function \n");
      PrintOutError(nResult);
      return;
}

else
{
      printf("Number of Raw Messages returned = %d\n", nResult);
}
```

SEE ALSO

aceRTDecodeRawMsg()	aceRTGetStkMsgsDecoded()
aceRTGetHBufMsgsRaw()	aceRTGetHBufMsgsDecoded()

aceRTInstallHBuf

This function will allocate a host buffer.

PROTOTYPE

```
#include "Rtop.h"


S16BIT _DECL aceRTInstallHBuf(S16BIT DevNum,
                              U32BIT dwHBufSize);
```

STATE

Ready

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
dwHBufSize	(input parameter) Size of the desired host buffer in 16-bit words. Refer to description for size restrictions. Valid values: [(RT command stack size/4)*ACE_MSGSIZE_RT*3] to 5,120,000 words

DESCRIPTION

This function allocates a host buffer based on the size parameter. For this function to succeed the size must be at least 3 times greater than the number of messages that can be stored in the command stacks multiplied by ACE_MSGSIZE_RT (fixed length RT msgs). The size of the host buffer cannot exceed 5000K.

Example: if the command stack is defined as 256 words then the dwHBufSize input parameter must be at least: (256/4) * ACE_MSGSIZE_RT * 3 = 6912 words.

256 = length of command stack
256/4 = number of messages that can be stored in the command stack
ACE_MSGSIZE_RT = 36 words

Note: The dwHBufSize parameter is in words. The macro ACE_MSGSIZE_RT should be used as it may change in future configurations.

If the host buffer has already been allocated, this function will call the aceRTUninstallHBuf() function to uninstall it and then install it again based on the dwHBufSize input parameter.

aceRTInstallHBuf (continued)

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_MT_HBUFSIZE	The dwHBufSize input parameter is too small
ACE_ERR_RT_HBUF	The proper memory for the host buffer could not be allocated
ACE_WRN_RT_CFG_INVALID	Operation of your device may be problematic because one or more of the following interrupts have been enabled: Time Tag Rollover, RT Address Parity Error, and/or Ram Parity Error along with the Interrupt Status Queue. See Appendix B for details.

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U32BIT dwHBufSize = ((256/4) * ACE_MSGSIZE_RT * 3);

// create a host buffer that can be used with a stack size of 256 words

nResult = aceRTInstallHBuf(DevNum, dwHBufSize);

if(nResult)
{
      printf("Error in aceRTInstallHBuf() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTUninstallHBuf()

aceRTModeCodeIrqDisable

This function will disable mode code interrupts.

PROTOTYPE

```
#include "Rtop.h"


S16BIT _DECL aceRTModeCodeIrqDisable(S16BIT DevNum,
                                      U16BIT wModeCodeType,
                                      U16BIT wModeCodeIrq);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wModeCodeType	(input parameter) An unsigned 16-bit parameter describing the mode code type. The type is a combination of Receive/Transmit, with/without data, and Broadcast. Please refer to the aceRTModeCodeIrqEnable() function for valid values.
wModeCodeIrq	An unsigned 16-bit parameter that indicates which mode codes to disable. This value is an OR'ed combination of the following values. Please refer to the aceRTModeCodeIrqEnable() function for valid values.

DESCRIPTION

This function will disable the hardware from interrupting the host based on the reception of certain mode codes. The mode codes are specified by their type and their command.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The wModeCodeType input parameter contains a value greater than seven

aceRTModeCodeIrqDisable (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

U16BIT wModeCodeIrq =
(ACE_RT_MCIRQ_TX_VECTOR_WRD | ACE_RT_MCIRQ_TX_BIT_WRD);

/* disable interrupt generation on transmit with data
mode codes, actual mode codes to generate interrupts are
trans vector word and transmit bit word */

nResult = aceRTModeCodeIrqDisable(DevNum, ACE_RT_MCTYPE_TX_DATA,
                                  wModeCodeIrq);

if(nResult)
{
      printf(“Error in aceRTModeCodeIrqDisable() function \n”);
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTModeCodeIrqEnable()   aceRTModeCodeIrqStatus()

aceRTModeCodeIrqEnable

This function will cause an interrupt on a received mode code.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTModeCodeIrqEnable(S16BIT DevNum,
                                    U16BIT wModeCodeType,
                                    U16BIT wModeCodeIrq);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wModeCodeType	(input parameter) An unsigned 16-bit parameter describing the mode code type. The type is a combination of Receive/Transmit, with/without data, and Broadcast. The type value may be any one of the following. Valid values: ACE_RT_MCTYPE_RX_NO_DATA Receive mode codes without data ACE_RT_MCTYPE_RX_DATA Receive mode codes with data ACE_RT_MCTYPE_TX_NO_DATA Transmit mode codes without data ACE_RT_MCTYPE_TX_DATA Transmit mode codes with data ACE_RT_MCTYPE_BCST_RX_NO_DATA Broadcast receive mode codes without data ACE_RT_MCTYPE_BCST_RX_DATA Broadcast receive mode codes with data

aceRTModeCodeIrqEnable (continued)

ACE_RT_MCTYPE_BCST_TX_NO_DATA
Broadcast transmit mode codes without data

ACE_RT_MCTYPE_BCST_TX_DATA
Broadcast transmit mode codes with data

wModeCodeIrq — An unsigned 16-bit parameter that indicates which mode codes will generate the interrupt. This value is an OR'ed combination of the following values. The qualifying types are listed in italics. Valid values:

ACE_RT_MCIRQ_DYN_BUS_CTRL
TX_NO_DATA
ACE_RT_MCIRQ_SYNCHRONIZE
(BCST_)(TX|RX)(_NO|)_DATA
ACE_RT_MCIRQ_TX_STATUS_WRD
TX_NO_DATA
ACE_RT_MCIRQ_INIT_SELF_TEST
(BCST_)TX_NO_DATA
ACE_RT_MCIRQ_TX_SHUTDOWN
(BCST_)TX_NO_DATA
ACE_RT_MCIRQ_OVRD_TX_SHUTDOWN
(BCST_)TX_NO_DATA
ACE_RT_MCIRQ_INH_TERM_FLAG
(BCST_)TX_NO_DATA
ACE_RT_MCIRQ_OVRRD_INH_TERM_FLG
(BCST_)TX_NO_DATA
ACE_RT_MCIRQ_RESET_REMOTE_TERM
(BCST_)TX_NO_DATA
ACE_RT_MCIRQ_TX_VECTOR_WRD
TX_DATA
ACE_RT_MCIRQ_TX_LAST_CMD
TX_DATA
ACE_RT_MCIRQ_TX_BIT_WRD
TX_DATA
ACE_RT_MCIRQ_SEL_TX_SHUTDOWN
(BCST_)RX_DATA
ACE_RT_MCIRQ_OVRD_SEL_TX_SHUTDWN
(BCST_)RX_DATA
ACE_RT_MCIRQ_RESERVED_BIT6
ACE_RT_MCIRQ_RESERVED_BIT7
ACE_RT_MCIRQ_RESERVED_BIT8
ACE_RT_MCIRQ_RESERVED_BIT9
ACE_RT_MCIRQ_RESERVED_BIT10
ACE_RT_MCIRQ_RESERVED_BIT11
ACE_RT_MCIRQ_RESERVED_BIT12
ACE_RT_MCIRQ_RESERVED_BIT13
ACE_RT_MCIRQ_RESERVED_BIT14
ACE_RT_MCIRQ_RESERVED_BIT15

aceRTModeCodeIrqEnable (continued)

DESCRIPTION

This function will set the hardware to interrupt the host processor based on reception of selected mode codes. The mode codes are specified by their type and their command.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The wModeCodeType input parameter contains a value greater than seven

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

U16BIT                          wModeCodeIrq                          =
(ACE_RT_MCIRQ_TX_VECTOR_WRD|ACE_RT_MCIRQ_TX_BIT_WRD);

/* generate interrupt on transmit with data mode code
actual mode code to generate interrupts are transmit vector word and
transmit bit word */

nResult = aceRTModeCodeIrqEnable(DevNum, ACE_RT_MCTYPE_TX_DATA,
                                 wModeCodeIrq);

if(nResult)
{
      printf("Error in aceRTModeCodeIrqEnable() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTModeCodeIrqDisable() aceRTModeCodeIrqStatus()

aceRTModeCodeIrqStatus

This function will return the status of a mode code generating an interrupt.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTModeCodeIrqStatus(S16BIT DevNum,
                                     U16BIT wModeCodeType,
                                     U16BIT *pwMCIrqStatus);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wModeCodeType	(input parameter) An unsigned 16-bit parameter describing the mode code type. The type is a combination of Receive/Transmit, with/without data, and Broadcast. The type value may be any one of the following. Please refer to the aceRTModeCodeIrqEnable() for valid values.
pwMCIrqStatus	(output parameter) Pointer to an unsigned 16-bit parameter which will receive the mode codes that will generate an interrupt. This is a bit packed value that represents the OR'ed combination of mode codes as specified in aceRTModeCodeIrqEnable().

DESCRIPTION

This function will return information regarding the status of a mode code generating an interrupt by reading one of the Mode Code Interrupt Lookup locations.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode

aceRTModeCodeIrqStatus (continued)

ACE_ERR_PARAMETER	The wModeCodeType input parameter contains a value greater than seven and/or the pwMCIrqStatus is Null

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT wModeCodeIrq;

/* Get the status of which mode codes will generate
interrupts given the mode code type to be. The value returned in
wModeCodeIrq can be decoded by applying the defined macros for the
different mode codes as defined in aceRTModeCodeIrqEnable() */

nResult = aceRTModeCodeIrqStatus(DevNum, ACE_RT_MCTYPE_TX_DATA,
                                  &wModeCodeIrq);

if(nResult)
{
      printf("Error in aceRTModeCodeIrqStatus() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTModeCodeIrqEnable()   aceRTModeCodeIrqDisable()

aceRTModeCodeReadData

This function will read data from the Enhanced Mode Code Data Locations table.

PROTOTYPE

```
#include "Rtop.h"


S16BIT _DECL aceRTModeCodeReadData(S16BIT DevNum,
                                   U16BIT wModeCode,
                                   U16BIT *pMCData);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum
(input parameter)
Logical Device Number
Valid values:
0 – 31

wModeCode
(input parameter)
This parameter specifies which mode code contains the data that should be read.
Valid values:
ACE_RT_MCDATA_RX_SYNCHRONIZE
ACE_RT_MCDATA_RX_SEL_T_SHUTDWN
ACE_RT_MCDATA_RX_OVR_SEL_T_SHUTDWN
ACE_RT_MCDATA_TX_TRNS_VECTOR
ACE_RT_MCDATA_TX_TRNS_LAST_CMD
ACE_RT_MCDATA_TX_TRNS_BIT
ACE_RT_MCDATA_BCST_SYNCHRONIZE
ACE_RT_MCDATA_BCST_SEL_T_SHUTDWN
ACE_RT_MCDATA_BCST_OVR_SEL_T_SHUTDWN

pMCData
(output parameter)
A single unsigned 16-bit piece of data returned from the Mode Code data table

DESCRIPTION

This function will read data from the Mode Code data table. The mode code for which data is to be read is specified by wModeCode. The data returned will be a single U16BIT word that is read from the Mode Code data table.

aceRTModeCodeReadData (continued)

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The wModeCode input parameter contains a value greater than 0x2F and/or the pMCData parameter is Null

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT pMCData;
/*Read the data that is associated with the
ACE_RT_MCDATA_TX_TRNS_BIT mode code. */

nResult = aceRTModeCodeReadData(DevNum,
                                ACE_RT_MCDATA_TX_TRNS_BIT,
                                &pMCData);

if(nResult)
{
      printf("Error in aceRTModeCodeReadData() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTModeCodeWriteData()

aceRTModeCodeWriteData

This function will write to the Enhanced Mode Code Data Locations table.

PROTOTYPE

```
#include "Rtop.h"


S16BIT _DECL aceRTModeCodeWriteData(S16BIT DevNum,
                                    U16BIT wModeCode,
                                    U16BIT wMCData);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wModeCode	(input parameter) This parameter specifies which mode code contains the data that should be read. Valid values: ACE_RT_MCDATA_RX_SYNCHRONIZE ACE_RT_MCDATA_RX_SEL_T_SHUTDWN ACE_RT_MCDATA_RX_OVR_SEL_T_SHUTDWN ACE_RT_MCDATA_TX_TRNS_VECTOR ACE_RT_MCDATA_TX_TRNS_LAST_CMD ACE_RT_MCDATA_TX_TRNS_BIT ACE_RT_MCDATA_BCST_SYNCHRONIZE ACE_RT_MCDATA_BCST_SEL_T_SHUTDWN ACE_RT_MCDATA_BCST_OVR_SEL_T_SHUTDWN
wMCData	(output parameter) A single unsigned 16-bit piece of data returned from the Mode Code data table

DESCRIPTION

This function will write data to the specified Mode Code data table. The mode code for which data is to be written is specified by wModeCode. The data written will be a single U16BIT word that is written to the Mode Code data table.

aceRTModeCodeWriteData (continued)

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The wModeCode parameter

EXAMPLE

```
S16BIT DevNum = 0;
U16BIT wMCData = 0x4DDC;
/*Read the data that is associated with the
ACE_RT_MCDATA_TX_TRNS_BIT mode code. */

nResult = aceRTModeCodeWriteData(DevNum,
                                 ACE_RT_MCDATA_TX_TRNS_BIT,
                                 wMCData);

if(nResult)
{
      printf("Error in aceRTModeCodeWriteData() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTModeCodeReadData()

aceRTMsgLegalityDisable

This function will illegalize a message for a subaddress

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTMsgLegalityDisable(S16BIT DevNum,
                                     U16BIT wOwnAddrOrBcst,
                                     U16BIT wTR,
                                     U16BIT wSA,
                                     U32BIT dwWC_MCMask);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum
(input parameter)
Logical Device Number
Valid values:
0 – 31

wOwnAddrOrBcst
(input parameter)
This parameter specifies whether the RT address is own or broadcast
Valid values:
ACE_RT_OWN_ADDRESS
ACE_RT_BCST_ADDRSS
ACE_RT_MODIFY_ALL

wTR
(input parameter)
Specify the direction Transmit/Receive
Valid values:
1 for transmit
0 for receive
ACE_RT_MODIFY_ALL

wSA
(input parameter)
Specify the sub-address to be illegalized
Valid value:
0 - 31
ACE_RT_MODIFY_ALL

aceRTMsgLegalityDisable (continued)

dwWC_MCMask (input parameter)
An unsigned 32-bit packed value that represents the 32 possible word counts for the selected sub-address.
Valid values:
0x00000000 – 0xFFFFFFFF
where the least significant bit = word count 0(32), and the most significant bit = word count 31. (i.e. if the selected sub-address should be illegalized for word counts 1, 10, 16, 28 and 30 the dwWC_MCMask would equal 0x50010402)

DESCRIPTION

This function will illegalize msgs received by the RT. The selection is based on the following properties of the message.

Broadcast/Own RT Address
Transmit/Receive
Sub-address
Word count/Mode Code

The ACE_RT_MODIFY_ALL can illegalize all messages of a certain type for all subaddresses.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The wOwnAddrOrBcst, wTR, and/or wSA parameter(s) contain an incorrect input value

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U32BIT dwWC_MCMask = 0x50044202;

/* illegalize all sub-addresses of RT address 5 for word
counts of 30, 28, 18, 14, 9 and 1 */

nResult = aceRTMsgLegalityDisable(DevNum, 1, 1, ACE_RT_MODIFY_ALL,
                                  dwWC_MCMask);

if(nResult)
{
      printf("Error in aceRTMsgLegalityDisable() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTMsgLegalityDisable() aceRTMsgLegalityStatus()

aceRTMsgLegalityEnable

This function will legalize a message for a subaddress.

PROTOTYPE

```
#include "Rtop.h"


S16BIT _DECL aceRTMsgLegalityEnable(S16BIT DevNum,
                                    U16BIT wOwnAddrOrBcst,
                                    U16BIT wTR,
                                    U16BIT wSA,
                                    U32BIT dwWC_MCMask);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wOwnAddrOrBcst	(input parameter) This parameter specifies whether the RT address is own or broadcast Valid values: ACE_RT_OWN_ADDRESS ACE_RT_BCST_ADDRSS ACE_RT_MODIFY_ALL
wTR	(input parameter) Specify the direction Transmit/Receive Valid values: 1 for transmit 0 for receive ACE_RT_MODIFY_ALL
wSA	(input parameter) Specify the sub-address to be legalized. OR'ed combination of the following values. Valid values: 0-31 ACE_RT_MODIFY_ALL

aceRTMsgLegalityEnable (continued)

dwWC_MCMask	(input parameter) U32BIT bit packed value that represents the 32 possible word counts for the selected sub-address. Valid values: 0x00000000 – 0xFFFFFFFF where the least significant bit = word count 0(32), and the most significant bit = word count 31. (i.e. if the selected sub-address should be legalized for word counts 1, 10, 16, 28 and 30 the dwWC_MCMask would equal 0x50010402)

DESCRIPTION

This function will legalize messages received by the RT. The legalization is based on whether the message is Broadcast or to the RTs own address. Additionally, legality of the message is based on transmit or receive, the specific subaddress, and the word count (mode code) of the msg. The ACE_RT_MODIFY_ALL can legalize all messages of a certain type on all subaddresses. The aceRTDataBlkMapToSA() function calls this function to legalize broadcast, transmit, and receive messages.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The wOwnAddrOrBcst, wTR, and/or wSA parameter(s) contain an incorrect input value

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT wOwnAddrOrBcst = 1, wTR = 1;
U32BIT dwWC_MCMask = 0x50044202;
/* legalize all sub-addresses of RT address 5 for word
counts of 30, 28, 18, 14, 9 and 1 */

nResult = aceRTMsgLegalityEnable(DevNum, wOwnAddrOrBcst, wTR,
                                 ACE_RT_MODIFY_ALL, dwWC_MCMask);

if(nResult)
{
      printf("Error in aceRTMsgLegalityEnable() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTMsgLegalityDisable()   aceRTMsgLegalityStatus()

aceRTMsgLegalityStatus

This function will report the status of a particular command's legality for a subaddress.

PROTOTYPE

```
#include "Rtop.h"


S16BIT _DECL aceRTMsgLegalityStatus(S16BIT DevNum,
                                    U16BIT wOwnAddrOrBcst,
                                    U16BIT wTR,
                                    U16BIT wSA,
                                    U32BIT *pdwWC_MCMask);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wOwnAddrOrBcst	(input parameter) This parameter specifies whether the RT address is own or broadcast Valid values: ACE_RT_OWN_ADDRESS ACE_RT_BCST_ADDRSS
wTR	(input parameter) Specify the direction Transmit/Receive Valid values: 1 for transmit 0 for receive
wSA	(input parameter) Specify the sub-address to be illegalized Valid value: – 31

aceRTMsgLegalityStatus (continued)

dwWC_MCMask (output parameter)
Pointer to an unsigned 32-bit packed value that represents the 32 possible word counts for the selected sub-address.
'1' = illegal
Valid values:
0x00000000 – 0xFFFFFFFF
where the least significant bit = word count 0(32), and the most significant bit = word count 31. (e.g. if the selected sub-address should be illegalized for word counts 1, 10, 16, 28 and 30 the dwWC_MCMask would equal 0x50010402)

DESCRIPTION

This function reads the Command illegalizing table and reports the status of a particular command's legality for a particular RT subaddress.

The selection is based on the following properties of the message.
Broadcast/Own RT Address
Transmit/Receive
Subaddress
Word count/Mode Code

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The wOwnAddrOrBcst, wTR, and/or wSA parameter(s) contain an incorrect input value

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

U16BIT wOwnAddrOrBcst = 1, wTR = 1;
U16BIT wSA = 12;
U32BIT dwWC_MCMask = 0x50044202;
// get the legalize status for sub-addresses 12 of RT address 5

nResult = aceRTMsgLegalityStatus(DevNum, wOwnAddrOrBcst,wTR, wSA,
                                 &dwWC_MCMask);

if(nResult)
{
      printf("Error in aceRTMsgLegalityStatus() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTMsgLegalityDisable() aceRTMsgLegalityEnable()

aceRTRelatchAddr

This function latches the RT address that is currently being input to the device.

PROTOTYPE

#include “Rtop.h”

S16BIT _DECL aceRTSetAddress(S16BIT DevNum);

STATE

Ready

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function latches the current RT address that is being input on the device. This function only applies to devices that have the capability to modify the RT address through external inputs. Some of the cards manufactured by DDC do not support this function. Please see your specific card manual to see if your card can set the RT address through external means and not just by software control.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode

aceRTRelatchAddr (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT wRTAddress = 10;

// set EMA device number to RT address 10
nResult = aceRTRelatchAddr(DevNum);

if(nResult)
{
      printf("Error in aceRTRelatchAddr() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTSetAddrSource() aceRTGetAddrSource()

aceRTSetAddress

This function configures the RT address.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTSetAddress(S16BIT DevNum,
                             U16BIT wRTAddress);
```

STATE

Ready

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wRTAddress	Remote Terminal address to be assigned to the device. Valid values: 0 – 31

DESCRIPTION

This function configures the Remote Terminal Address. The address will be assigned to the EMA that is designated by the DevNum input parameter.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The wRTAddress parameter is greater than 31

aceRTSetAddress (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT wRTAddress = 10;

NResult = aceRTSetAddrSource(

// set EMA device number to RT address 10
nResult = aceRTSetAddress(DevNum, wRTAddress);

if(nResult)
{
      printf("Error in aceRTSetAddress() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTSetAddrSource()   aceRTGetAddrSource()

aceRTSetAddrSource

This function will set whether the RT address source is internal or external.

PROTOTYPE

```
#include "Rtop.h"


S16BIT _DECL aceRTBusyBitsTblSet(S16BIT DevNum,
                                 U16BIT wRTSource);
```

STATE

Ready

MODE

RT, RTMT

PARAMETERS

DevNum
(input parameter)
Logical Device Number
Valid values:
0 – 31

wRTSource
(input parameter)
This parameter specifies whether the RT address is internal or external
Valid values:
ACE_RT_INTERNAL_ADDR
The RT address is set on the device's configuration register by calling the aceRTSetAddress() function.

ACE_RT_EXTERNAL_ADDR
The RT address is set externally on the device's pin.

DESCRIPTION

This function will set whether the RT address source is internal or external. The function does not need to be called by the user because it is automatically called by the Enhanced Mini-ACE runtime library when the user sets up the device for RT, or RTMT operation. The function is called internally by the runtime library with the wRTSource input parameter set to ACE_RT_INTERNAL_ADDR.

NOTE: *Some DDC card products do not have the capability of using an external address.*

aceRTSetAddrSource (continued)

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_PARAMETER	The wRTSource input parameter does not contain one of the following valid inputs: ACE_RT_INTERNAL_ADDR, or ACE_RT_EXTERNAL_ADDR

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceRTSetAddrSource(DevNum, ACE_RT_EXTERNAL_ADDR);

if(nResult)
{
      printf("Error in aceRTSetAddrSource() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTGetAddrSource()   aceSetAddress()

aceRTStart

This function starts the RT.

PROTOTYPE

#include “Rtop.h”

S16BIT _DECL aceRTStart(S16BIT DevNum);

STATE

Ready

MODE

RT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function sets up all required registers, sets up enhanced mode code handling, and then starts the Remote Terminal responding to messages on the 1553 bus. The device will transition from a Ready state to a Run state after this function has been called.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in RT mode
ACE_ERR_INVALID_ACCESS	Access is not defined as ACE_ACCESS_CARD or ACE_ACCESS_USR for this device

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

/* initialize the EMA, create the Data Blocks, Map the Data Blocks, and
setup legalization, then call aceRTStart() */

nResult = aceRTStart(DevNum);

if(nResult)
{
      printf(“Error in aceRTStart() function \n”);
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTStop()

aceRTStatusBitsClear

This function deactivates the status bits for all RT responses.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTStatusBitsClear(S16BIT DevNum,
                                  U16BIT wStatusBits);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

wStatusBits (input parameter)
An unsigned 16-bit value that represents the bits in the RT Status Word that should be cleared to a value of 0. This is an OR'ed combination of the following values.
Valid values:

All of the following values will either set or clear bits in Configuration Register # 1 at memory location 0x01.

The following options will be deactivated if alternate Status is set. These bits are really set to a 1 value internally since the hardware has defined these register bits as active low. This will cause the operation specified by the bit to become inactive in alternate status mode. If no alternate status is set, then the option will write a 1 to the bit location causing it to be active.

ACE_RT_STSBIT_DBCA
This deactivates the Dynamic Bus Controller Acceptance bit 11 by setting it (active low).

ACE_RT_STSBIT_BUSY
This activates the Busy bit 10 by setting it (active low).

aceRTStatusBitsClear (continued)

ACE_RT_STSBIT_SREQ
This activates the Service Request bit 9 by setting it (active low).

ACE_RT_STSBIT_SSFLAG
This activates the Subsystem Flag bit 8 by setting it (active low).

ACE_RT_STSBIT_RTFLAG
This activates the RT Flag bit 7 by setting it (active low).

The following bits may be deactivated if in alternate Status Word mode. These bits are cleared internally to a 0 value since the hardware has defined these register bits as active high. The "Alternate" status word mode: With this option, **all 11** RT Status Word bits are programmable by the host processor, by means of bits 11 through 1 of Configuration Register #1 at memory location 0x01. This mode may be used to support MIL-STD-1553A, McAir, G.D. F16, or other "non-1553B" applications.

ACE_RT_STSBIT_S10
Clears Status bit 11.

ACE_RT_STSBIT_S09
Clears Status bit 10.

ACE_RT_STSBIT_S08
Clears Status bit 9.

ACE_RT_STSBIT_S07
Clears Status bit 8.

ACE_RT_STSBIT_S06
Clears Status bit 7.

ACE_RT_STSBIT_S05
Clears Status bit 6.

ACE_RT_STSBIT_S04
Clears Status bit 5.

ACE_RT_STSBIT_S03
Clears Status bit 4.

aceRTStatusBitsClear (continued)

ACE_RT_STSBIT_S02
Clears Status bit 3.

ACE_RT_STSBIT_S01
Clears Status bit 2.

ACE_RT_STSBIT_S00
Clears Status bit 1.

DESCRIPTION

This function deactivates the status bits for all RT responses. Some of the status bits may only be set when in 'Alternate Status Word' mode. These are designated in the parameter descriptions.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT wStatusBits = (ACE_RT_STSBIT_BUSY | ACE_RT_STSBIT_SREQ);

/* command Device Number ‘DevNum’ to respond with the BUSY BIT and the
SERVICE REQUEST bit cleared in the RT Status word */

nResult = aceRTStatusBitsClear(DevNum, wStatusBits);

if(nResult)
{
      printf(“Error in aceRTStatusBitsClear() function \n”);
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTStatusBitsSet()   aceRTStatusBitsStatus()

aceRTStatusBitsSet

This function activates the status bits for all RT responses.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTStatusBitsSet(S16BIT DevNum,
                                U16BIT wStatusBits);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wStatusBits	(input parameter) An unsigned 16-bit value that represents the bits in the RT Status Word that should be set to a value of 1. This is an OR'ed combination of the following values. Valid values: All of the following values will either set or clear bits in Configuration Register # 1 at memory location 0x01. The following options will be activated if no alternate Status is set. These bits are really cleared to a 0 value internally since the hardware has defined these register bits as active low. If alternate status is set, the following option will write a 1 to the bit to deactivate the option. ACE_RT_STSBIT_DBCA This activates the Dynamic Bus Controller Acceptance bit 11 by clearing it (active low). ACE_RT_STSBIT_BUSY This activates the Busy bit 10 by clearing it (active low). ACE_RT_STSBIT_SREQ This activates the Service Request bit 9 by clearing it (active low).

aceRTStatusBitsSet (continued)

ACE_RT_STSBIT_SSFLAG
This activates the Subsystem Flag bit 8 by clearing it (active low).

ACE_RT_STSBIT_RTFLAG
This activates the RT Flag bit 7 by clearing it (active low).

The following bits may be activated if in alternate Status Word mode. These bits are set internally to a 1 value since the hardware has defined these register bits as active high. The "Alternate" status word mode: With this option, **all 11** RT Status Word bits are programmable by the host processor, by means of bits 11 through 1 of Configuration Register #1 at memory location 0x01. This mode may be used to support MIL-STD-1553A, McAir, G.D. F16, or other "non-1553B" applications.

ACE_RT_STSBIT_S10
Sets Status bit 11.

ACE_RT_STSBIT_S09
Sets Status bit 10.

ACE_RT_STSBIT_S08
Sets Status bit 9.

ACE_RT_STSBIT_S07
Sets Status bit 8.

ACE_RT_STSBIT_S06
Sets Status bit 7.

ACE_RT_STSBIT_S05
Sets Status bit 6.

ACE_RT_STSBIT_S04
Sets Status bit 5.

ACE_RT_STSBIT_S03
Sets Status bit 4.

ACE_RT_STSBIT_S02
Sets Status bit 3.

ACE_RT_STSBIT_S01
Sets Status bit 2.

ACE_RT_STSBIT_S00
Sets Status bit 1.

aceRTStatusBitsSet (continued)

DESCRIPTION

This function sets the status bits for all RT responses. Some of the status bits may only be set when in 'Alternate Status Word' mode. These are designated in the parameter descriptions. If the wStatusBits input parameter contains one of the Alternate Status Word inputs and the RT is not set up to support an Alternate Status Word, only bits 7-11 of Configuration Register # 1 will be activated since the rest of the bits will be internally masked. The Alternate Status can be activated from the aceRTConfigure() function.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT wStatusBits = (ACE_RT_STSBIT_BUSY | ACE_RT_STSBIT_SREQ);

/* command Device Number 'DevNum' to respond with the BUSY BIT and the
SERVICE REQUEST bit set in the RT Status word */

nResult = aceRTStatusBitsSet(DevNum, wStatusBits);

if(nResult)
{
      printf("Error in aceRTStatusBitsSet() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTStatusBitsClear() aceRTStatusBitsStatus()

aceRTStatusBitsStatus

This function will retrieve the status bits for all RT responses.

PROTOTYPE

```
#include "Rtop.h"


S16BIT _DECL aceRTStatusBitsStatus(S16BIT DevNum,
                                   U16BIT *wStatusBits);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
wStatusBits	(output parameter) Pointer to an unsigned 16-bit value that represents the bits in the RT Status Word. The following mask values can be used to represent the bit values. Valid values: ACE_RT_STSBIT_DBCA ACE_RT_STSBIT_BUSY ACE_RT_STSBIT_SREQ ACE_RT_STSBIT_SSFLAG ACE_RT_STSBIT_RTFLAG The following mask values may be used if in alternate Status mode. ACE_RT_STSBIT_S10 ACE_RT_STSBIT_S09 ACE_RT_STSBIT_S08 ACE_RT_STSBIT_S07 ACE_RT_STSBIT_S06 ACE_RT_STSBIT_S05 ACE_RT_STSBIT_S04 ACE_RT_STSBIT_S03 ACE_RT_STSBIT_S02 ACE_RT_STSBIT_S01 ACE_RT_STSBIT_S00

aceRTStatusBitsStatus (continued)

DESCRIPTION

This function retrieves the status bits for all RT responses. Some of the status bits will only be available when in 'Alternate Status Word' mode. These are designated in the parameter descriptions. The returned status may be decoded by masking with the Status Bit macros as defined in aceRTStatusBitsSet() function.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT wStatusBits;

/* Acquire the RT Status word response status.  The returned value may
be decoded using the wStatusBits macros defined in aceRTStatusBitsSet
*/

nResult = aceRTStatusBitsStatus(DevNum, &wStatusBits);

if(nResult)
{
      printf("Error in aceRTStatusBitsStatus() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTStatusBitsSet() aceRTStatusBitsClear()

aceRTStkToHBuf

This function will transfer data from the device's RT hardware stack to the host buffer.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTStkToHBuf(S16BIT DevNum);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function copies all messages to the host buffer. This is a very quick way to transfer data from the EMA RT hardware stack to the host buffer. Once in the host buffer the function aceGetHBufMsgDecoded can be used to convert the raw stack information into user friendly messages by placing all raw data into the formatted MSGSTRUCT structure.

The Enhanced Mini-ACE runtime library contains an internal Interrupt Service Routine that will get triggered if a time tag rollover has occurred, a circular buffer 50% rollover occurred, a circular buffer 100% rollover occurred, a command stack 50% rollover occurred and/or a command stack 100% rollover occurred in remote terminal mode of operation.

The internal Interrupt Service Routine (ISR) will call this function. The library will do this to reliably transfer messages and data from the hardware stacks so that the user does not need to ever call this function. This function is provided in the Enhanced Mini-ACE library as an advanced mode function that can be used to transfer messages and data to your host buffer if your operating system doesn't support the use of interrupts. In Windows operating systems that support interrupt generation, this function should **not** be called by the user.

Note: *The host buffer is guaranteed to be able to hold the messages from the hardware stack, but if the host buffer message processing is not performed regularly, then the new message data may overwrite existing un-read messages in the host buffer.*

aceRTStkToHBuf (continued)

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_RT_HBUF	No Host Buffer exists for this RT

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

/* previously initialized RT with host buffer created.
During operation, the RT will generate an interrupt and the
Definition of the program specifies that all unread
Messages will be transferred to the host buffer.
*/

nResult = aceRTStkToHBuf(DevNum);
if(nResult)
{
      printf("Error in aceRTStkToHBuf() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTGetHBufMsgCount()
aceRTGetHBufMsgsRaw()
aceRTGetHBufMsgDecoded()

aceRTStkToHBuf32

This function will transfer data from the device's RT hardware stack to the host buffer.

PROTOTYPE

```
#include "Rtop.h"

S16BIT _DECL aceRTStkToHBuf32(S16BIT DevNum);
```

STATE

Ready, Run

MODE

RT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function copies all messages to the host buffer. This is a very quick way to transfer data from the EMA RT hardware stack to the host buffer. Once in the host buffer the function aceGetHBufMsgDecoded can be used to convert the raw stack information into user friendly messages by placing all raw data into the formatted MSGSTRUCT structure.

The Enhanced Mini-ACE runtime library contains an internal Interrupt Service Routine that will get triggered if a time tag rollover has occurred, a circular buffer 50% rollover occurred, a circular buffer 100% rollover occurred, a command stack 50% rollover occurred and/or a command stack 100% rollover occurred in remote terminal mode of operation.

The internal Interrupt Service Routine (ISR) will call this function. The library will do this to reliably transfer messages and data from the hardware stacks so that the user does not need to ever call this function. This function is provided in the Enhanced Mini-ACE library as an advanced mode function that can be used to transfer messages and data to your host buffer if your operating system doesn't support the use of interrupts. In Windows operating systems that support interrupt generation, this function should **not** be called by the user.

This functions is used on all cards except for the BU-65567/68 and the BU-65553 cards because these cards are ISA devices that use the 16-bit memory accesses in the aceMTStkToHBuf() function call.

Note: *The host buffer is guaranteed to be able to hold the messages from the hardware stack, but if the host buffer message processing is not performed regularly, then the new message data may overwrite existing un-read messages in the host buffer.*

aceRTStkToHBuf32 (continued)

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RT or RTMT mode
ACE_ERR_RT_HBUF	No Host Buffer exists for this RT

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

/* previously initialized RT with host buffer created.
During operation, the RT will generate an interrupt and the
Definition of the program specifies that all unread
Messages will be transferred to the host buffer.
*/

nResult = aceRTStkToHBuf32(DevNum);
if(nResult)
{
      printf("Error in aceRTStkToHBuf32() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTGetHBufMsgCount()
aceRTGetHBufMsgDecoded()
aceRTGetHBufMsgsRaw()
aceRTStkToHBuf()

aceRTStop

This function stops the RT.

PROTOTYPE

#include “Rtop.h”

S16BIT _DECL aceRTStop(S16BIT DevNum);

STATE

Run

MODE

RT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function stops the Remote Terminal from responding to messages on the 1553 bus. The device will transition from a Run state to a Ready state after this function has been called.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in RT mode
ACE_ERR_INVALID_ACCESS	Access is not defined as ACE_ACCESS_CARD or ACE_ACCESS_USR for this device

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

/* initialize the EMA, create the Data Blocks, Map the Data Blocks, and
setup legalization.  Start the EMA and perform appropriate data
processing then call aceRTStop() */

nResult = aceRTStop(DevNum);

if(nResult)
{
      printf(“Error in aceRTStop() function \n”);
      PrintOutError(nResult);
      return;
}
```

aceRTStop (continued)

SEE ALSO

aceRTStart()

aceRTUninstallHBuf

This function will deallocate a host buffer.

PROTOTYPE

#include “Rtop.h”

S16BIT _DECL aceRTUninstallHBuf(S16BIT DevNum);

STATE

Ready

MODE

RT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function deallocates the RT host buffer if present and clears any internal interrupt mask register bits that were previously set by a call to the aceRTInstallHBuf() function. A RT will be assigned only one host buffer (HBuff), therefore there is no need for an HBuff ID.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in RT mode
ACE_ERR_RT_HBUF	A host buffer does not exist

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

/* RT is previously initialized with an HBuf */

nResult = aceRTUninstallHBuf(DevNum);

if(nResult)
{
        printf(“Error in aceRTUninstallHBuf() function \n”);
 PrintOutError(nResult);
 return;
}
```

aceRTUninstallHBuf (continued)

SEE ALSO

aceRTInstallHBuf()

RTMT Functions

Table 7. RTMT Functions Listing

aceRTMTConfigure

This function will configure the device for combined RT and MT operation.

PROTOTYPE

```
#include "RtMtop.h"

S16BIT _DECL aceRTMTConfigure(S16BIT DevNum,
                              U16BIT wRTCmdStkSize,
                              U16BIT wMTStkType,
                              U16BIT wMTCmdStkSize,
                              U16BIT wMTDataStkSize,
                              U32BIT dwOptions);
```

STATE

Ready

MODE

RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

wRTCmdStkSize (input parameter)
This is the size of the desired RT command stack size
Valid values:

ACE_RT_CMDSTK_256
256 words

ACE_RT_CMDSTK_512
512 words

ACE_RT_CMDSTK_1K
1024 words

ACE_RT_CMDSTK_2K
2048 words

wMTStkType (input parameter)
Specify the monitor stack type.
Valid values:

ACE_MT_SINGLESTK
Uses a single stack

aceRTMTConfigure (continued)

ACE_MT_DOUBLESTK
Uses Stack A and B
Note: not applicable for E^2MA devices

wMTCmdStkSize (input parameter)
This specifies the desired MT command stack size.
Valid values:

ACE_MT_CMDSTK_256
256 words

ACE_MT_CMDSTK_1K
1024 words

ACE_MT_CMDSTK_4K
4096 words

ACE_MT_CMDSTK_16K
16384 words

wMTDataStkSize (input parameter)
Specify the size for the monitor data stack size. This size is defined as the number of words to be allocated for the monitor data stack. There is no standard calculation for the number of words per message.

Valid values:

ACE_MT_DATASTK_512
512 words

ACE_MT_DATASTK_1K
1024 words

ACE_MT_DATASTK_2K
2048 words

ACE_MT_DATASTK_4K
4096 words

ACE_MT_DATASTK_8K
8192 words

ACE_MT_DATASTK_16K
16384 words

ACE_MT_DATASTK_32K
32768 words

aceRTMTConfigure (continued)

dwOptions (input parameter)
The options designate the operation of the RT and MT. The value is an unsigned 32-bit value that is an OR'ed combination of the following values.
Valid values:

The following parameters will set bits in Configuration Register # 2 at memory location 0x02:

ACE_RT_OPT_CLR_SREQ
Sets the Clear Service Request bit 2 to a 1. This will clear a service request after a tx vector word.

ACE_RT_OPT_LOAD_TT
Sets the Load/Transmit Time Tag on Synchronize bit 5 to a 1. This will cause the reception of a Synchronize (with data) mode command which will cause the Data Word from the Synchronize message to be loaded into the Time Tag Register.

ACE_RT_OPT_CLEAR_TT
Sets the Clear Time Tag on Synchronize bit 6 to a 1. This will cause the reception of a Synchronize (without data) mode command which will cause the value of the internal Time Tag Register to clear to 0x0000.

ACE_RT_OPT_OVR_DATA
Sets the Overwrite Invalid Data bit 11 to a 1. This affects the operation of the RT subaddress circular buffer memory management mode. The Lookup Table address pointer will only be updated following a transmit message or following a valid receive or broadcast message to the respective Rx/Bcst subaddress. If the bit is logic 1, the Lookup Table pointer will not be updated following an invalid receive or broadcast message. In addition, if the bit is logic 1, an interrupt request for a circular buffer rollover condition (if enabled) will only occur following the end of a transmit message during which the last location in the circular buffer has been read or following the end of a valid receive or Broadcast message in which the last location in the circular buffer has been written to.

The following parameters will set bits in Configuration Register # 3 at memory location 0x07:

aceRTMTConfigure (continued)

ACE_RT_OPT_OVR_MBIT
Sets Override Mode T/R* Error bit 6 to a 1. This will cause a mode code Command Word with a T/R* bit of 0 and an MSB of the mode code field of 0 will be considered a defined (reserved) mode Command Word. In this configuration, the Enhanced Mini-ACE will respond to such a command and the Message Error bit will not become set.

ACE_RT_OPT_ALT_STS
Sets Alternate RT Status Word Enable bit 5 to a 1. This will cause all 11 RT Status Word bits to be under control of the host processor, by means of bits 11 through 1 of Configuration Register # 1.

ACE_RT_OPT_IL_RX_D
Sets Illegal Receive Transfer Disable bit 4 to a 1. This will cause the device to not store the received data words to the shared RAM if the ACE receives a receive command that has been illegalized.

ACE_RT_OPT_BSY_RX_D
Sets Busy Receive Transfer Disable bit 3 to a 1. If the host processor has programmed BUSY* to logic "0" or the particular Command Word (broadcast, T/R* bit, subaddress) has been programmed to be busy by means of the Busy lookup table and the Enhanced Mini-ACE RT receives a receive command, the Enhanced Mini-ACE will respond with its Status Word with the Busy bit set and will not store the received Data Words to the shared RAM.

ACE_RT_OPT_SET_RTFG
Sets RTFail*/RTFlag* Wrap Enable bit 2 to a 1. The Terminal flag status word bit will also become set if either a transmitter timeout (660.5 μs) condition had occurred or the ACE RT had failed its loopback test for the previous non-broadcast message. The loopback test is performed on all non-broadcast messages processed by the Enhanced Mini-ACE RT. The received version of all transmitted words is checked for validity (sync and data encoding, bit count, parity) and correct sync type. In addition, a 16-bit comparison is performed on the received version of the last word transmitted by the Enhanced Mini-ACE RT. If any of these checks or comparisons do not verify, the loopback test is considered to have failed.

aceRTMTConfigure (continued)

ACE_RT_OPT_1553A_MC
ACE_MT_OPT_1553A_MC
Sets 1553A Mode Codes Enabled bit 1 to a 1. If this option is chosen, the Enhanced Mini-ACE RT or Message Monitor considers only subaddress 0 to be a mode code subaddress. Subaddress 31 is treated as a standard non-mode code subaddress. In this configuration, the Enhanced Mini-ACE will consider valid and respond only to mode code commands containing no data words. In this configuration, the Enhanced Mini-ACE RT will consider all mode commands followed by data words to be invalid and will not respond. In addition the Enhanced Mini-ACE will not decode for the MIL-STD-1553B "Transmit Status" and "Transmit Last Command" mode codes. As a result, the internal RT Status Word Register will be updated as a result of these commands.

The following parameters will set bits in Configuration Register # 4 at memory location 0x08:

ACE_RT_OPT_MC_O_BSY
Sets Mode Command Override Busy Bit 13 to a 1. If BUSY* is programmed to logic "0" or if Busy Lookup Table (bit 13 of Configuration Register #2) is logic "1" and the respective bit(s) in the Busy Lookup Table (bit 0 of location 0242 and/or bit 15 of location 0243) is programmed to logic "1," the Enhanced Mini-ACE will transmit its Status Word with its BUSY bit set, followed by a single Data Word, in response to either a Transmit Vector Word mode command or a Reserved transmit mode command with data (transmit mode codes 10110 through 11111). The Busy Lookup Table functions are: aceRTBusyBitsTblSet(), aceRTBusyBitsTblClear(), and aceRTBusyBitsTblStatus().

The following parameters will set bits in Configuration Register # 5 at memory location 0x09:

ACE_RT_OPT_BCST_DIS
Sets Broadcast Disabled bit 7 to a 1. The Enhanced Mini-ACE will **not** recognize RT address 31 as the broadcast address. In this instance, RT address 31 may be used as a discrete RT address.

DESCRIPTION

This function configures combined Remote Terminal and Monitor configuration. This routine initializes the EMA for operation as an RT and MT. The library configuration structures and EMA tables are initialized to default values, and the memory structures are created. All RT subaddresses are illegalized after this function has been called. Remember that the Enhanced Mini-ACE runtime library creates 50% and 100% stack rollover interrupts to reliably transfer data. This ensures reliability even if only a single stack is used.

aceRTMTConfigure (continued)

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in RTMT mode
ACE_ERR_MT_BUFTYPE	The wMTStkType input parameter is not 0 (ACE_MT_SINGLESTK). If the device is in Legacy mode then this parameter must be ACE_MT_DOUBLESTK. Only single stacks are supported in DOS. E²MA Devices only support ACE_MT_SINGLESTK
ACE_ERR_PARAMETER	The wRTCmdStkSize input parameter contains an incorrect value
ACE_ERR_MT_CMDSTK	The wMTCmdStkSize input parameter contains an incorrect value
ACE_ERR_MT_DATASTK	The wMTDataStkSize input parameter contains an incorrect value
ACE_ERR_INVALID_MALLOC	The proper amount of memory for an RT and/or MT structure could not be allocated

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT wRTCmdStkSize, wMTStkType, wMTCmdStkSize;
U16BIT wMTDataStkSize,
U32BIT dwOptions;

/* configure EMA to run as RT and MT.
RT Command stack size = 512 words = 512/4 commands
MT stack type = Double
MT Command stack size = 1024 words
MT data stack size = 2048 words
dwOptions = Use alternate status word and Clear Time Tag
*/
wRTCmdStkSize = ACE_RT_CMDSTK_512;
wMTStkType = ACE_MT_DOUBLESTK;
wMTCmdStkSize = ACE_MT_CMDSTK_1K;
wMTDataStkSize = ACE_MT_DATASTK_2K;
dwOptions = (ACE_RT_OPT_ALT_STS | ACE_RT_OPT_CLEAR_TT);
/* setup alternate status word. Use aceRTStatusBitsSet() */

nResult = aceRTMTConfigure(DevNum, wRTCmdStkSize, wMTStkType,
                              wMTCmdStkSize, wMTDataStkSize, dwOptions);

if(nResult)
{
       printf("Error in aceRTMTConfigure() function \n");
       PrintOutError(nResult);
       return;
}
```

SEE ALSO

aceRTConfigure() aceMTConfigure()

aceRTMTGetHBufMetric

This function returns performance information about the combined RT/MT host buffer.

PROTOTYPE

```
#include "RtMtOp.h"

S16BIT _DECL aceRTMTGetHBufMetric (S16BIT DevNum,
                                   HBUFMETRIC *pMetric,
                                   U16BIT bReset);
```

STATE

Ready, Run

MODE

RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

pMetric (output parameter)
Pointer to an HBUFMETRIC structure to be filled in with metrics

Member Variable Name	Definition
dwCount	The number of messages in the host buffer
dwLost	The total number of messages lost since the host buffer was installed
dwPctFull	The current percentage of host buffer used
dwHighPct	The highest percentage of the host buffer used since the host buffer was installed or metrics were reset

bReset (input parameter)
This will specify if the highest percentage should be reset after this function returns.
Valid values:
False (0)
Do not reset the highest percentage value

True (1)
Reset the highest percentage value

aceRTMTGetHBufMetric (continued)

DESCRIPTION

This function returns performance information about the host buffer. Built-in test metrics can report the number of messages in the host buffer, the total number of messages lost since the host buffer was installed, the current percentage of the host buffer that is used, and the highest percentage of the host buffer used since it was installed. These metrics are a useful tool in investigating errors.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_MODE	The device is not in RTMT mode
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_METRICS_NOT_ENA	Metrics are not enabled and should be set by calling the aceSetMetrics() function

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
HBUFMETRIC pMetric;

nResult = aceRTMTGetHBufMetric(DevNum,
                               &pMetric,
                               TRUE);

if(nResult)
{
      printf("Error in aceRTMTGetHBufMetric() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTMTGetHBufMsgCount () aceRTMTGetHBufMsgDecoded ()
aceRTMTGetHBufMsgsRaw ()

aceRTMTGetHBufMsgCount

This function returns the number of messages in the host buffer.

PROTOTYPE

#include “RtMtOp.h”

S16BIT _DECL aceRTMTGetHBufMsgCount (S16BIT DevNum);

STATE

Ready, Run

MODE

RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function returns total number of messages that are currently in the host buffer.

RETURN VALUE

S16BIT	Returns the number of messages in the host buffer if the function was successful
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_MODE	The device is not in RTMT mode
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_RTMT_COMBO_HBUF	An RTMT combination host buffer has not been installed and must be installed with the aceRTMTInstallHBuf() function

aceRTMTGetHBufMsgCount (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceRTMTGetHBufMsgCount(DevNum);

if(nResult)
{
      printf("Error in aceRTMTGetHBufMsgCount() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTMTGetHBufMetric() aceRTMTGetHBufMsgDecoded()
aceRTMTGetHBufMsgsRaw()

aceRTMTGetHBufMsgDecoded

This function reads a single decoded message from the host buffer if it is present.

PROTOTYPE

```
#include "RtMtOp.h"

S16BIT _DECL aceRTMTGetHBufMsgDecoded (S16BIT DevNum
                                        MSGSTRUCT *pMsg,
                                        U32BIT *pdwMsgCount,
                                        U32BIT *pdwRTMsgLostStk,
                                        U32BIT *pdwMTMsgLostStk,
                                        U32BIT *pdwMsgLostHBuf,
                                        U16BIT wMsgLoc);
```

STATE

Ready, Run

MODE

RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

pMsg (output parameter)
Pointer to a parameter of type MSGSTRUCT that will be used to return the decoded message from the host buffer.

Member Variable Name	Definition
wBlkSts	Contains the block status word of the message
wTimeTag	Contains the time tag of the message
wCmdWrd1	Contains the command word
wCmdWrd2	Contains the second command word for RT to RT transfers
wCmdWrd1Flg	Indicates the validity of the first command word
wCmdWrd2Flg	Indicates the validity of the second command word
wStsWrd1	Contains first status word
wStsWrd2	Contains second status word
wStsWrd1Flg	Indicates the validity of the first status word
wStsWrd2Flg	Indicates the validity of the second status word
wWordCount	Contains the number of valid data words

aceRTMTGetHBufMsgDecoded (continued)

Member Variable Name	Definition
adataWrds[32]	An array that will contain the data words
wBCCtrlWrd	Contains the BC Control Word for BC mode messages only
wBCGapTime	Contains the message gap time only for BC
wBCLoopBack1	Contains the first looped back word for BC mode messages only
wBCLoopBack2	Contains the second looped back word for BC mode messages only
wBCLoopBack1Flg	Indicates validity of first loop back word for BC mode message only
wBCLoopBack2Flg	Indicates validity of second loop back word for BC mode message only

pdwMsgCount (output parameter)
Pointer to an unsigned 32-bit parameter to be filled with message count.
Valid return values:

0
No message returned in pMsg

1
One message returned in pMsg

pdwRTMsgLostStk (output parameter)
Pointer to an unsigned 32-bit word that will contain the approximate number of messages lost due to overwrite prior to transferring the messages from the Enhanced Mini-ACE hardware RT stack to the software host buffer.

pdwMTMsgLostStk (output parameter)
Pointer to an unsigned 32-bit word that will contain the approximate number of messages lost due to overwrite prior to transferring the messages from the Enhanced Mini-ACE hardware MT stack to the software host buffer.

pdwMsgLostHBuf (output parameter)
Pointer to an unsigned 32-bit word that will contain the approximate number of messages lost while transferring the messages from the host buffer to the user buffer.

wMsgLoc (input parameter)
Specify what message should be read off the Host buffer. The choice is to read the next unread message or the last received message. Additionally, this parameter indicates whether to purge or not purge the message read.
Valid values:

ACE_RTMT_MSGLOC_NEXT_PURGE
Reads next message and takes it off of the host buffer

aceRTMTGetHBufMsgDecoded (continued)

ACE_RTMT_MSGLOC_NEXT_NPURGE
Reads next message and leaves it on the host buffer

ACE_RTMT_MSGLOC_LATEST_PURGE
Reads current message and takes it off of the host buffer

ACE_RTMT_MSGLOC_LATEST_NPURGE
Reads current message and leaves it on the host buffer

DESCRIPTION

This function reads a single decoded message from the host buffer if it is present. The function will use the aceMTDecodeRawMsg() and the aceRTDecodeRawMsg() functions to decode the raw message into the MSGSTRUCT structure.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_MODE	The device is not in RTMT mode
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_RTMT_COMBO_HBUF	An RTMT combination host buffer has not been installed and must be installed with the aceRTMTInstallHBuf() function
ACE_ERR_PARAMETER	One or all of the following parameters are NULL: pdwMsgLostHBuf pdwRTMsgLostStk pdwMTMsgLostStk
ACE_ERR_RTMT_MSGLOC	The wMsgLoc input parameter contains an invalid input

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U32BIT pdwMsgCount;
U32BIT pdwRTMsgLostStk;
U32BIT pdwMTMsgLostStk;
U32BIT pdwMsgLostHBuf;
MSGSTRUCT pMsg;
```

aceRTMTGetHBufMsgDecoded (continued)

```
nResult = aceRTMTGetHBufMsgDecoded(DevNum,
                                   &pMsg,
                                   &pdwMsgCount,
                                   &pdwRTMsgLostStk,
                                   &pdwMTMsgLostStk,
                                   &pdwMsgLostHBuf,
                                   ACE_MT_MSGLOC_NEXT_PURGE);

if(nResult)
{
      printf("Error in aceRTMTGetHBufMsgDecoded() function \n");
      PrintOutError(nResult);
      return;
}

if(!pdwMsgCount)
{
      printf("No message returned from Host buffer\n");
}
```

SEE ALSO

aceRTMTGetHBufMetric() aceRTMTGetHBufMsgCount()
aceRTMTGetHBufMsgsRaw()

aceRTMTGetHBufMsgsRaw

This function reads as many messages as possible off of the host buffer into a user buffer.

PROTOTYPE

```
#include "RtMtOp.h"

S16BIT _DECL aceRTMTGetHBufMsgsRaw(S16BIT DevNum,
                                   U16BIT *pBuffer,
                                   U16BIT wBufferSize,
                                   U32BIT *pdwMsgCount,
                                   U32BIT *pdwRTMsgLostStk,
                                   U32BIT *pdwMTMsgLostStk,
                                   U32BIT *pdwMsgLostHBuf);
```

STATE

Ready, Run

MODE

RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

pBuffer (output parameter)
Pointer to an unsigned 16-bit word buffer that will receive the raw messages from the host buffer.

wBufferSize (input parameter)
Specify the size of the pBuffer buffer that will receive the raw messages from the host buffer.
Valid values:
There is no restriction on the size of this buffer, but for efficiency of resource use, the size of this buffer should be calculated as the following formula
(Number of messages * ACE_MSGSIZE_MT) +
(Number of messages * ACE_MSGSIZE_RT).

pdwMsgCount (output parameter)
If no errors occurred, this is the return value of the number of messages transferred to the pBuffer buffer.

aceRTMTGetHBufMsgsRaw (continued)

pdwRTMsgLostStk	(output parameter) Estimates the number of messages lost due to overwrite prior to transferring the messages from the Enhanced Mini-ACE hardware RT stack to the software host buffer.
pdwMTMsgLostStk	(output parameter) Estimates the number of messages lost due to overwrite prior to transferring the messages from the Enhanced Mini-ACE hardware MT stack to the software host buffer.
pdwMsgLostHBuf	(output parameter) Estimates the number of messages lost due to overwrite when transferring the messages from the host buffer to the user buffer.

DESCRIPTION

This function reads as many messages as possible off of the host buffer into a user buffer without any decoding. The limiting factor when copying messages to the local buffer is the local buffer size and the number of messages available on the host buffer.

Note: This combined RT/MT Host buffer will contain messages from the RT stack and the MT stack on the hardware. The MT stack in the hardware does not include RT information for the RT address that the device is configured for. However, the RT/MT host buffer contains all of this information because post processing is performed to combine the RT Host buffer and the MT Host buffer into this RT/MT Host buffer. Each message is a fixed length of ACE_MSGSIZE_MT words or ACE_MSGSIZE_RT words. This macro must be used in size calculations as the size of the structure is subject to change.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in MT or RTMT mode
ACE_ERR_RTMT_COMBO_HBUF	An RTMT combination host buffer has not been installed and must be installed with the aceRTMTInstallHBuf() function
ACE_ERR_INVALID_BUF	The pBuffer or wBufferSize input parameters are not within a valid range
ACE_ERR_PARAMETER	The pdwMsgLostHBuf input parameter is Null and/or the pdwRTMsgLostHBuf input parameter is Null and/or the pdwMTMsgLostHBuf input parameter is Null

aceRTMTGetHBufMsgsRaw (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT pBuffer[(ACE_MSGSIZE_MT * 256)+ (ACE_MSGSIZE_RT * 256)];
U16BIT wBufferSize = ((ACE_MSGSIZE_MT * 256)+ (ACE_MSGSIZE_RT * 256));

U32BIT pdwMsgCount;
U32BIT pdwRTMsgLostStk;
U32BIT pdwMTMsgLostStk;
U32BIT pdwMsgLostHBuf;

nResult = aceRTMTGetHBufMsgsRaw(DevNum,
                                pBuffer,
                                wBufferSize,
                                &pdwMsgCount,
                                &pdwMsgLostStk,
                                &pdwMsgLostHBuf);

if(nResult)
{
      printf("Error in aceRTMTGetHBufMsgsRaw() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTMTGetHBufMetric()
aceRTMTGetHBufMsgCount()
aceRTMTGetHBufMsgDecoded()

aceRTMTInstallHBuf

This function installs a host buffer.

PROTOTYPE

```
#include "RtMtOp.h"

S16BIT _DECL aceRTMTInstallHBuf(S16BIT DevNum,
                                U32BIT dwRTMTHBufSize);
```

STATE

Ready

MODE

RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
dwRTMTHBufSize	(input parameter) Specify the size of the host buffer to be created for use by the combined remote terminal and monitor operation. Valid values: [(MT command stack size/4) *ACE_MSGSIZE_RTMT * 3] + [(RT command stack size/4) * ACE_MSGSIZE_RTMT * 3] to 5,120,000 words

DESCRIPTION

This function allocates a host buffer based on the size parameter for RT/MT mode. The Enhanced Mini-ACE device provides a combined RT/MT mode of operation. This allows you to run the device as an RT and a MT simultaneously on the 1553 data bus. This is an advanced feature to provide extended capabilities to the end user.

In RT/MT mode of operation the monitor will monitor the entire 1553 data bus except for it's own RT address. The MT stack on the hardware will contain all contents of the data bus except anything received or sent by the Enhanced Mini-ACE's own RT address. This is a function of the device and cannot be changed. When using this Enhanced Mini-ACE runtime library some post processing is performed to combine the MT stack and the RT stack into this one RT/MT host buffer that will contain all monitored messages and data on the 1553 data bus.

If using this library in RT/MT mode without a host buffer installed the MT stack will contain all monitored data on the 1553 data bus except for the RT defined to be the Enhanced Mini-ACE device.

aceRTMTInstallHBuf (continued)

For an RT-RT transfer command where the Enhanced Mini-ACE device is set up in RT/MT mode and is the receiving RT in the data transfer the Monitor will not pick up any data because the device is busy servicing the receive command. The RT stack will have the RT-RT Transfer bit set in the Block Status Word of the receiving RT to indicate that this received command is part of an RT-RT transfer command initiated by the BC.

If the device is set up in RT/MT mode and is the transmitting RT in an RT-RT transfer the monitor will see the following: (1) The monitor stack contains a command with the block status word set to 0x4000 which indicates an SOM. The RT-RT Transfer bit will **not** be set in the Block Status Word for this command. The monitor will also see this as a transmit command in the command word part of the stack entry. (2) A second entry will be placed in the monitor command stack for this one RT-RT command. This entry will have the following bits set in the Block Status Word: EOM, Error Flag, Format Error, Command Word Contents Error. Once again the RT-RT Transfer bit will **not** be set in the Block Status Word for this command.

When using a host buffer and an RT-RT transfer command is performed then the host buffer will not pick up any of the monitor data that was described above.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in RTMT mode
ACE_ERR_MT_HBUFSIZE	The dwHBufSize input parameter is too small
ACE_ERR_MT_HBUF	Memory for the host buffer could not be allocated

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U32BIT dwHBufSize;

/*
create a host buffer that is to be used for combined RT/MT operation
*/

dwRTMTHBufSize = 4095;

nResult = aceRTMTInstallHBuf(DevNum, dwRTMTHBufSize);
```

aceRTMTInstallHBuf (continued)

```
if(nResult)
{
      printf("Error in aceMTInstallHBuf() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTMTUninstallHBuf()

aceRTMTStart

This function starts the RT and MT.

PROTOTYPE

#include “RtMtop.h”

S16BIT _DECL aceRTMTStart(S16BIT DevNum);

STATE

Ready

MODE

RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function initializes all command and data stack pointers, monitor structures, remote terminal structures, remote terminal registers, and monitor registers necessary to run the device in both remote terminal mode and monitor mode. This function enables enhanced mode. After this function has been called the device is left in a Run state.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in RTMT mode
ACE_ERR_INVALID_ACCESS	The device is not set up for ACE_ACCESS_CARD or ACE_ACCESS_USR

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

/* configure the RTMT operation using aceRTMTConfigure() */

nResult = aceRTMTStart (DevNum);

if(nResult)
{
      printf(“Error in aceRTMTStart() function \n”);
      PrintOutError(nResult);
      return;
}
```

aceRTMTStart (continued)

SEE ALSO

aceRTMTStop()

aceRTMTStkToHBuf

This function swaps the active and inactive stacks and then copies all messages from the inactive stack to the host buffer.

PROTOTYPE

#include “RtMtOp.h”

S16BIT _DECL aceRTMTStkToHBuf(S16BIT DevNum);

STATE

Ready, Run

MODE

Advanced plus one of the following: RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31	

DESCRIPTION

This function swaps the active and inactive stacks and then copies all messages from the inactive stack to the host buffer. Once the messages have been moved to the host buffer, they can be processed by the application using one of the following routines:

aceMTGetHBufMsgsRaw()
aceMTGetHBufMsgsDecoded()

The Enhanced Mini-ACE runtime library calls this function inside of the internal interrupt service routine that is processed by the library on any of the following conditions:
ACE_IMR1_MT_DATASTK_ROVER
100% Monitor Hardware Data Stack Rollover

ACE_IMR2_MT_DSTK_50P_ROVER
50% Monitor Hardware Data Stack Rollover

ACE_IMR1_MT_CMDSTK_ROVER
100% Monitor Hardware Command Stack Rollover

ACE_IMR2_MT_CSTK_50P_ROVER
50% Monitor Hardware Command Stack Rollover

ACE_IMR1_TT_ROVER
Time Tag Rollover

aceRTMTStkToHBuf (continued)

ACE_IMR1_BCRT_CMDSTK_ROVER
100% Remote Terminal Hardware Command Stack Rollover

ACE_IMR2_RT_CSTK_50P_ROVER
50% Remote Terminal Hardware Command Stack Rollover

The library reliably transfers messages and data from the hardware stacks so that the user does not need to ever call this function. This function is provided in the Enhanced Mini-ACE library as an advanced mode function that can be used to transfer messages and data to your host buffer if your operating system doesn't support the use of interrupts. In Windows operating system this function should not be called by the user.

Note: The host buffer is guaranteed to be able to hold the messages from the hardware stack, but if the host buffer message processing is not performed regularly, then the new message data may overwrite existing un-read messages in the host buffer.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RTMT mode
ACE_ERR_RT_HBUF	The host buffer does not exist
ACE_ERR_RTMT_COMBO_HBUF	The RTMT host buffer is not used

EXAMPLE

```
/*
Please note that this function should not normally be called by the end
user and can cause errors
*/

S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceRTMTStkToHBuf(DevNum);

if(nResult)
{
      printf("Error in aceRTMTStkToHBuf() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTMTGetHBufMsgsRaw()   aceRTMTGetHBufMsgDecoded()

aceRTMTStkToHBuf32

This function swaps the active and inactive stacks and then copies all messages from the inactive stack to the host buffer.

PROTOTYPE

```
#include "RtMtOp.h"

S16BIT _DECL aceRTMTStkToHBuf32(S16BIT DevNum);
```

STATE

Ready, Run

MODE

Advanced plus one of the following: RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function swaps the active and inactive stacks and then copies all messages from the inactive stack to the host buffer. Once the messages have been moved to the host buffer, they can be processed by the application using one of the following routines:

aceMTGetHBufMsgsRaw()
aceMTGetHBufMsgsDecoded()

The Enhanced Mini-ACE runtime library calls this function inside of the internal interrupt service routine that is processed by the library on any of the following conditions:
ACE_IMR1_MT_DATASTK_ROVER
100% Monitor Hardware Data Stack Rollover

ACE_IMR2_MT_DSTK_50P_ROVER
50% Monitor Hardware Data Stack Rollover

ACE_IMR1_MT_CMDSTK_ROVER
100% Monitor Hardware Command Stack Rollover

ACE_IMR2_MT_CSTK_50P_ROVER
50% Monitor Hardware Command Stack Rollover

ACE_IMR1_TT_ROVER
Time Tag Rollover

aceRTMTStkToHBuf32 (continued)

ACE_IMR1_BCRT_CMDSTK_ROVER
100% Remote Terminal Hardware Command Stack Rollover

ACE_IMR2_RT_CSTK_50P_ROVER
50% Remote Terminal Hardware Command Stack Rollover

The library reliably transfers messages and data from the hardware stacks so that the user does not need to ever call this function. This function is provided in the Enhanced Mini-ACE library as an advanced mode function that can be used to transfer messages and data to your host buffer if your operating system doesn't support the use of interrupts. In Windows operating system this function should not be called by the user.

This function is used on all cards except for the BU-65567/68 and the BU-65553 cards because these cards are ISA devices that use the 16-bit memory accesses in the aceMTStkToHBuf() function call.

Note: The host buffer is guaranteed to be able to hold the messages from the hardware stack, but if the host buffer message processing is not performed regularly, then the new message data may overwrite existing un-read messages in the host buffer .

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in RTMT mode
ACE_ERR_RT_HBUF	The host buffer does not exist
ACE_ERR_RTMT_COMBO_HBUF	The RTMT host buffer is not used

EXAMPLE

```
/*
Please note that this function should not normally be called by the end
user and can cause errors
*/

S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceRTMTStkToHBuf(DevNum);

if(nResult)
{
      printf("Error in aceRTMTStkToHBuf() function \n");
      PrintOutError(nResult);
      return;
}
```

aceRTMTStkToHBuf32 (continued)

SEE ALSO

aceRTMTGetHBufMsgsRaw()
aceRTMTGetHBufMsgDecoded()

aceRTMTStop

This function stops the RT and MT from running.

PROTOTYPE

#include “RtMtop.h”

S16BIT _DECL aceRTMTStop(S16BIT DevNum);

STATE

Run

MODE

RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function stops the RT/MT from running in both the remote terminal and monitor modes. The device will be in a Ready state after this function has been called.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Run state
ACE_ERR_INVALID_MODE	The device is not in RTMT mode
ACE_ERR_INVALID_ACCESS	The device is not set up for ACE_ACCESS_CARD or ACE_ACCESS_USR

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

/* the RTMT mode is previously setup and running */

nResult = aceRTMTStop (DevNum);

if(nResult)
{
      printf(“Error in aceRTMTStop() function \n”);
      PrintOutError(nResult);
      return;
}
```

aceRTMTStop (continued)

SEE ALSO

aceRTMTStart()

aceRTMTUninstallHBuf

The function will uninstall the host buffer.

PROTOTYPE

#include “RtMtOp.h”

S16BIT _DECL aceRTMTUninstallHBuf(S16BIT DevNum);

STATE

Ready

MODE

RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function deallocates the RT/MT host buffer if present. There can be only one host buffer per mode, so there is no requirement to specify a host buffer handle.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in RTMT mode
ACE_ERR_RTMT_HBUF	The host buffer does not exist

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

/*
Remove the RT/MT mode host buffer if it is installed.
If it doesn’t exist, the error ‘ACE_ERR_RTMT_HBUF’ will be returned.
*/
```

aceRTMTUninstallHBuf (continued)

```
nResult = aceRTMTUninstallHBuf(DevNum);

if(nResult)
{
      printf("Error in aceMTUninstallHBuf() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTMTInstallHBuf()

MT Functions

Table 8. MT Functions Listing

aceMTClearHBufTrigger

This function turns HBuf capture messages trigger operations off.

PROTOTYPE

#include “Mtop.h”

S16BIT _DECL aceMTClearHBufTrigger(S16BIT DevNum);

STATE

Ready

MODE

MT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31	

DESCRIPTION

This function stops the host buffer from capturing messages only after a message is read that matches the trigger structure.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in MT or RTMT mode

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceMTClearHBufTrigger(DevNum);

if(nResult){
      PrintOutError(nResult);
      return;}
```

SEE ALSO

aceMTSetHBufTrigger()

aceMTConfigure

This function configures the device as a monitor.

PROTOTYPE

```
#include "Mtop.h"


S16BIT _DECL aceMTConfigure(S16BIT DevNum,
                            U16BIT wMTStkType,
                            U16BIT wCmdStkSize,
                            U16BIT wDataStkSize,
                            U32BIT dwOptions);
```

STATE

Ready

MODE

MT

PARAMETERS

DevNum
(input parameter)
Logical Device Number
Valid values:
0 – 31

wMTStkType
(input parameter)
This parameter specifies the type of stack that will be used in the device. There are two choices, a double buffered stack or a single buffered stack.
Valid values:
ACE_MT_SINGLESTK
Uses a single stack

ACE_MT_DOUBLESTK
Uses Stack A and B
Note: not applicable for E^2MA devices

wCmdStkSize
(input parameter)
This specifies the desired MT command stack size.
Valid values:
ACE_MT_CMDSTK_256
256 words

ACE_MT_CMDSTK_1K
1024 words

ACE_MT_CMDSTK_4K
4096 words

aceMTConfigure (continued)

ACE_MT_CMDSTK_16K
16384 words

wDataStkSize

Specify the size for the monitor data stack size. This size is defined as the number of words to be allocated for the monitor data stack. There is no standard calculation for the number of words per message.
Valid values:

ACE_MT_DATASTK_512
512 words

ACE_MT_DATASTK_1K
1024 words

ACE_MT_DATASTK_2K
2048 words

ACE_MT_DATASTK_4K
4096 words

ACE_MT_DATASTK_8K
8192 words

ACE_MT_DATASTK_16K
16384 words

ACE_MT_DATASTK_32K
32768 words

dwOptions

The options are an OR'ed combination of the following codes.
Valid values:

The following parameters will set bits in Configuration Register # 3 at memory location 0x07:

ACE_MT_OPT_1553A_MC
Sets 1553A Mode Codes Enabled bit 1 to a 1. If this option is chosen, the Enhanced Mini-ACE RT or Message Monitor considers only subaddress 0 to be a mode code subaddress. Subaddress 31 is treated as a standard non-mode code subaddress. In this configuration, the Enhanced Mini-ACE will consider valid and respond only to mode code commands containing no data words. In this configuration, the Enhanced Mini-ACE RT will consider all mode commands followed by data words to be invalid and will not respond. In addition the Enhanced Mini-ACE will not decode for the MIL-STD-1553B "Transmit Status" and "Transmit Last Command" mode codes. As a result, the internal RT Status Word Register will be updated as a result of these commands.

aceMTConfigure (continued)

DESCRIPTION

This function configures the device as a monitor on the 1553 bus. The library configuration structures and EMACE table are initialized to default values, and the memory structures are created. Remember that the Enhanced Mini-ACE runtime library generates 50% and 100% command stack and data stack rollover interrupts to reliably transfer data. This ensures reliability even if only a single stack is used.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in MT mode
ACE_ERR_MT_BUFTYPE	The wMTStkType input parameter is greater than one. If the device is in Legacy mode then this parameter must be ACE_MT_DOUBLESTK E²MA devices only support ACE_MT_SINGLESTK
ACE_ERR_MT_CMDSTK	The wMTCmdStkSize input parameter contains an incorrect value
ACE_ERR_MT_DATASTK	The wMTDataStkSize input parameter contains an incorrect value
ACE_ERR_INVALID_MALLOC	The proper amount of memory for an MT structure could not be allocated
ACE_ERR_MEMMGR_FAIL	Memory allocation failed

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT wMTStkType, wCmdStkSize, wDataStkSize;
U32BIT dwOptions;
wMTStkType = ACE_MT_SINGLESTK;
wCmdStkSize = ACE_MT_CMDSTK_1K;
wDataStkSize = ACE_MT_DATASTK_1K;
dwOptions = ACE_MT_OPT_1553A_MC;

nResult = aceMTConfigure(S16BIT DevNum, wMTStkType,
                         wCmdStkSize, wDataStkSize, U32BIT dwOptions);

if(nResult)
{
      printf("Error in aceMTConfigure() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceRTConfigure() aceRTMTConfigure()

aceMTContinue

This function resumes the Monitor capturing of messages.

PROTOTYPE

#include “Mtop.h”

S16BIT _DECL aceMTContinue(S16BIT DevNum);

STATE

Run

MODE

MT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function resumes the Monitor capturing of messages. The Monitor will begin capturing messages using the same internal state as when it was paused using the aceMTPause() function. The internal state is not modified by the use of this function.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Run state
ACE_ERR_INVALID_MODE	The device is not in MT mode
ACE_ERR_INVALID_ACCESS	The device is not in ACE_ACCESS_CARD access mode or ACE_ACCESS_USR access mode

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

/* Initialize Monitor mode, setup filter table, create hBuf, and start
the monitor.
At some point the monitor is PAUSED
 */

nResult = aceMTContinue (DevNum);

if(nResult)
{
```

aceMTContinue (continued)

```
        printf("Error in aceMTContinue() function \n");
        PrintOutError(nResult);
        return;
}
```

SEE ALSO

aceMTStart()
aceMTStop()
aceMTPause()

aceMTCreateImageFiles

This function will create image files.

PROTOTYPE

```
#include "Mtop.h"

S16BIT _DECL aceMTCreateImageFiles(S16BIT DevNum,
                                   char *pszIFile,
                                   char *pszHFile)
```

STATE

Ready

MODE

MT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
pszIFile	(output parameter) Pointer to a character string that specifies the name of the binary image file that should be written.
pszHFile	(output parameter) Pointer to a character string that specifies the name of the header file that should be created.

DESCRIPTION

This function outputs 2 files. The first is a binary image of the Enhanced Mini-ACE's memory. The second is a 'C' header file that contains all offset and sample functions that allow memory to be accessed easily in an embedded system.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in MT mode
ACE_ERR_PARAMETER	The pIFile parameter is Null and/or the pHFile parameter is Null

aceMTCreateImageFiles (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

char *pszIFile = “mtimage.img”;
char *pszHFile = “mtimage.h”;

nResult = aceMTCreateImageFiles(DevNum, pszIFile, pszHFile);

if(nResult)
{
      printf(“Error in aceMTCreateImageFiles() function \n”);
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

None

aceMTDecodeRawMsg

This function decodes a raw word into a decoded MSGSTRUCT structure.

PROTOTYPE

```
#include "Mtop.h"


S16BIT _DECL aceMTDecodeRawMsg(S16BIT DevNum,
                               U16BIT *pBuffer,
                               MSGSTRUCT *pMsg);
```

STATE

Ready, Run

MODE

MT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

pBuffer (input parameter)
Pointer to an unsigned 16-bit word buffer that will be used to return the monitor message information. This buffer must be at least ACE_MSGSIZE_MT words long, and must contain a valid raw message.

pMsg (output parameter)
Pointer to a single buffer of type MSGSTRUCT that will contain the decoded monitor message. The table below lists all member variables that exist in the MSGSTRUCT structure along with their definition.

Member Variable Name	Definition
wBlkSts	Contains the block status word of the message
wTimeTag	Contains the time tag of the message
wCmdWrd1	Contains the command word
wCmdWrd2	Contains the second command word for RT to RT transfers
wCmdWrd1Flg	Indicates the validity of the first command word
wCmdWrd2Flg	Indicates the validity of the second command word
wStsWrd1	Contains first status word
wStsWrd2	Contains second status word

aceMTDecodeRawMsg (continued)

Member Variable Name	Definition
wStsWrd1Flg	Indicates the validity of the first status word
wStsWrd2Flg	Indicates the validity of the second status word
wWordCount	Contains the number of valid data words
adataWrds[32]	An array that will contain the data words
wBCCtrlWrd	Contains the BC Control Word for BC mode messages only
wBCGapTime	Contains the message gap time only for BC
wBCLoopBack1	Contains the first looped back word for BC mode messages only
wBCLoopBack2	Contains the second looped back word for BC mode messages only
wBCLoopBack1Flg	Indicates validity of first loop back word for BC mode message only
wBCLoopBack2Flg	Indicates validity of second loop back word for BC mode message only

DESCRIPTION

This function takes an ACE_MSGSIZE_MT word buffer and decodes the raw message it contains into a decoded MSGSTRUCT structure.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Run or Ready state
ACE_ERR_INVALID_MODE	The device is not in MT or RTMT mode
ACE_ERR_MSGSTRUCT	The pMsg parameter is Null
ACE_ERR_INVALID_BUF	The pBuffer input parameter is Null

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT pBuffer[ACE_MSGSIZE_MT];
MSGSTRUCT pMsg;

/* initialize EMA for monitor mode, run the monitor.
Read a message into pBuffer (see aceMTGetStkMsgsRaw or
aceMTGetHBufMsgsRaw)
*/

nResult = aceMTDecodeRawMsg(DevNum,pBuffer, &pMsg);

if(nResult)
{
      printf("Error in aceMTDecodeRawMsg() function \n");
      PrintOutError(nResult);
      return;
}
```

aceMTDecodeRawMsg (continued)

SEE ALSO

None

aceMTDisableRTFilter

This function allows the user to **NOT** monitor selective data.

PROTOTYPE

```
#include "Mtop.h"

S16BIT _DECL aceMTDisableRTFilter(S16BIT DevNum,
                                  U16BIT wRT,
                                  U16BIT wTR,
                                  U32BIT dwSAMask);
```

STATE

Ready, Run

MODE

MT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

wRT (input parameter)
Specify the RT address of the message to stop from being monitored based on the other parameters.
Valid values:
0 – 31
Selects one RT subaddress value

ACE_MT_FILTER_ALL
Specifies all RT subaddresses

wTR (input parameter)
Specify the Transmit/Receive bit of the message to stop from being filtered.
Valid values:
ACE_MT_FILTER_TX
Only Transmit

ACE_MT_FILTER_RX
Only Receive

ACE_MT_FILTER_ALL
Transmit and Receive

aceMTDisableRTFilter (continued)

dwSAMask (input parameter)
Specify the subaddresses of the message to stop from being filtered.
OR'ed combination of the following valid values.
Valid values:

ACE_MT_FILTER_SA_ALL
All subaddresses to stop from being filtered

ACE_MT_FILTER_SAXX
Specific subaddress to stop from being filtered, where XX ranges from 0 -31

DESCRIPTION

The Enhanced Mini-ACE provides a flexible interface that allows selective monitoring of 1553 messages based on the RT address, T/R bit, and subaddress with very little host processor intervention. The Message Monitor mode of the Enhanced Mini-ACE recreates all command/response messages on the 1553 bus on channels A and B, and stores them into the shared RAM based on a user programmable filter (RT address, T/R bit, and subaddress). This monitor can be used as a monitor alone or in a combined RT/Monitor mode. This function writes to the MT Selective Monitor Lookup table. It sets the appropriate bits associated with a RT address, T/R bit, and subaddress. Any commands with those subaddresses that have their bits cleared **WILL NOT** be monitored.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in MT or RTMT mode
ACE_ERR_MT_FILTER_RT	The wRT input parameter contains a value greater than 31 or it is not equal to ACE_MT_FILTER_ALL
ACE_ERR_MT_FILTER_TR	The wTR input parameter does not contain one of the following valid values: ACE_MT_FILTER_TX ACE_MT_FILTER_RX ACE_MT_FILTER_ALL

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT wRT = 10, wTR = ACE_MT_FILTER_TX;
U32BIT dwSAMask = ACE_MT_FILTER_SA5 + ACE_MT_FILTER_SA30;

/* set Monitor operation to NOT monitor TRANSMIT messages to RT 10 for
sub-addresses 5 and 30.
*/

nResult = aceMTDisableRTFilter(DevNum, wRT, wTR, dwSAMask);

if(nResult)
```

aceMTDisableRTFilter (continued)

```
{
      printf("Error in aceMTDisableRTFilter() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceMTEnableRTFilter()   aceMTGetRTFilter()

aceMTEnableRTFilter

This function allows the user to only monitor selective data.

PROTOTYPE

```
#include "Mtop.h"


S16BIT _DECL aceMTEnableRTFilter(S16BIT DevNum,
                                 U16BIT wRT,
                                 U16BIT wTR,
                                 U32BIT dwSAMask)
```

STATE

Ready, Run

MODE

MT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
  0 – 31

wRT (input parameter)
Specify the RT address of the message to be monitored based on the other parameters.
Valid values:
  0 – 31
  Selects one RT subaddress value

  ACE_MT_FILTER_ALL
  Specifies all RT subaddresses

wTR (input parameter)
Specify the Transmit/Receive bit of the message to be filtered.
Valid values:
  ACE_MT_FILTER_TX
  Only Transmit

  ACE_MT_FILTER_RX
  Only Receive

  ACE_MT_FILTER_ALL
  Transmit and Receive

aceMTEnableRTFilter (continued)

dwSAMask (input parameter)
Specify the subaddresses of the message to be filtered.
OR'ed combination of the following valid values.
Valid values:

ACE_MT_FILTER_SA_ALL
All subaddresses to be filtered

ACE_MT_FILTER_SAXX
Specific subaddress to be filtered, where XX ranges from 0 -31

DESCRIPTION

The Enhanced Mini-ACE provides a flexible interface that allows selective monitoring of 1553 messages based on the RT address, T/R bit, and subaddress with very little host processor intervention. The Message Monitor mode of the Enhanced Mini-ACE recreates all command/response messages on the 1553 bus on channels A and B, and stores them into the shared RAM based on a user programmable filter (RT address, T/R bit, and subaddress). This monitor can be used as a monitor alone or in a combined RT/Monitor mode. This function writes to the MT Selective Monitor Lookup table. It sets the appropriate bits associated with a RT address, T/R bit, and subaddress. Any commands with those subaddresses that have their bits set **WILL** be monitored.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in MT or RTMT mode
ACE_ERR_MT_FILTER_RT	The wRT input parameter contains a value greater than 31 or it is not equal to ACE_MT_FILTER_ALL
ACE_ERR_MT_FILTER_TR	The wTR input parameter does not contain one of the following valid values: ACE_MT_FILTER_TX ACE_MT_FILTER_RX ACE_MT_FILTER_ALL

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT wRT = 10, wTR = ACE_MT_FILTER_TX;
U32BIT dwSAMask = ACE_MT_FILTER_SA5 + ACE_MT_FILTER_SA30;

/* set Monitor operation to only monitor TRANSMIT messages to RT 10 for
sub-addresses 5 and 30.
All other messages will be ignored.
*/

nResult = aceMTEnableRTFilter(DevNum, wRT, wTR, dwSAMask);

if(nResult)
{
```

aceMTEnableRTFilter (continued)

```
        printf("Error in aceMTEnableRTFilter() function \n");
        PrintOutError(nResult);
        return;
    }
```

SEE ALSO

aceMTDisableRTFilter()   aceMTGetRTFilter()

aceMTGetHBufMetric

This function returns performance information about the Host Buffer.

PROTOTYPE

```
#include "bcop.h"


S16BIT _DECL aceMTGetHBufMetric(S16BIT DevNum,
                                HBUFMETRIC *pMetric,
                                U16BIT bReset);
```

STATE

Ready, Run

MODE

MT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

pMetric (output parameter)
Pointer to an HBUFMETRIC structure to be filled in with metrics

Member Variable Name	Definition
dwCount	The number of messages in the host buffer
dwLost	The total number of messages lost since the host buffer was installed
dwPctFull	The current percentage of host buffer used
dwHighPct	The highest percentage of the host buffer used since the host buffer was installed or metrics were reset

bReset (input parameter)
This will specify if the highest percentage should be reset after this function returns.
Valid values:
FALSE (0)
Do not reset the highest percentage value

TRUE (1)
Reset the highest percentage value

aceMTGetHBufMetric (continued)

DESCRIPTION

This function returns performance information about the Host Buffer. Built-in test metrics can report the number of messages in the host buffer, the total number of messages lost since the host buffer was installed, the current percentage of the host buffer that is used, and the highest percentage of the host buffer used since it was installed.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in MT or RTMT mode
ACE_ERR_PARAMETER	The pMetric pointer input by the user is NULL
ACE_ERR_RTMT_COMBO_HBUF	An MT host buffer is not used but an RTMT host buffer is being used
ACE_ERR_METRICS_NOT_ENA	Metrics are not enabled and should be set by calling the aceSetMetrics() function

EXAMPLE

```
S16BIT DevNum = 0;
HBUFMETRIC *pMetric;
U16BIT bReset = 1;
S16BIT nResult = 0;

nResult = aceMTGetHBufMetric(DevNum, pMetric, bReset)

if(nResult)
{
      printf("Error in aceMTGetHBufMetric() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceSetMetrics()
aceMTGetStkMetric()
aceRTGetHBufMetric()
aceRTGetStkMetric()

aceMTGetHBufMsgCount

This function returns the number of messages in the host buffer.

PROTOTYPE

#include “Mtop.h”

S16BIT _DECL aceMTGetHBufMsgCount(S16BIT DevNum);

STATE

Ready, Run

MODE

MT, RTMT

PARAMETERS

DevNum	Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function returns the number of messages that are currently in the host buffer.

RETURN VALUE

S16BIT nResult	The number of messages in the host buffer
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in MT or RTMT mode

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
nResult = aceMTGetHBufMsgCount(DevNum);
if(nResult < 0)
{
      PrintOutError(nResult);
      return;
}
printf(“Number of messages in Host Buffer = %d\n”, nResult);
```

SEE ALSO

None

aceMTGetHBufMsgDecoded

This function reads a single decoded message from the host buffer if it is present.

PROTOTYPE

```
#include "Mtop.h"

S16BIT _DECL aceMTGetHBufMsgDecoded(S16BIT DevNum,
                                    MSGSTRUCT *pMsg,
                                    U32BIT *pdwMsgCount,
                                    U32BIT *pdwMsgLostStk,
                                    U32BIT *pdwMsgLostHBuf,
                                    U16BIT wMsgLoc);
```

STATE

Ready, Run

MODE

MT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

pMsg (output parameter)
Pointer to a parameter of type MSGSTRUCT that will be used to return the decoded message from the host buffer. The table below lists all member variables that exist in the MSGSTRUCT structure along with their definition.

Member Variable Name	Definition
wBlkSts	Contains the block status word of the message
wTimeTag	Contains the time tag of the message
wCmdWrd1	Contains the command word
wCmdWrd2	Contains the second command word for RT to RT transfers
wCmdWrd1Flg	Indicates the validity of the first command word
wCmdWrd2Flg	Indicates the validity of the second command word
wStsWrd1	Contains first status word
wStsWrd2	Contains second status word

aceMTGetHBufMsgDecoded (continued)

Member Variable Name	Definition
wStsWrd1Flg	Indicates the validity of the first status word
wStsWrd2Flg	Indicates the validity of the second status word
wWordCount	Contains the number of valid data words
adataWrds[32]	An array that will contain the data words
wBCCtrlWrd	Contains the BC Control Word for BC mode messages only
wBCGapTime	Contains the message gap timeonly for BC
wBCLoopBack1	Contains the first looped back word for BC mode messages only
wBCLoopBack2	Contains the second looped back word for BC mode messages only
wBCLoopBack1Flg	Indicates validity of first loop back word for BC mode message only
wBCLoopBack2Flg	Indicates validity of second loop back word for BC mode message only

pdwMsgCount (output parameter)
Pointer to an unsigned 32-bit parameter to be filled with message count.
Valid return values:

0
No message returned in pMsg

1
One message returned in pMsg

pdwMsgLostStk (output parameter)
Pointer to an unsigned 32-bit word that will contain the approximate number of messages lost due to overwrite prior to transferring the messages from the EMA to the host buffer.

pdwMsgLostHBuf (output parameter)
Pointer to an unsigned 32-bit word that will contain the approximate number of messages lost while transferring the messages from the host buffer to the user buffer.

wMsgLoc (input parameter)
Specify what message should be read off the Host Buffer. The choice is to read the next unread message or the last received message. Additionally, this parameter indicates whether to purge or not purge the message read.
Valid values:

ACE_MT_MSGLOC_NEXT_PURGE
Reads next message and takes it off of the host buffer

ACE_MT_MSGLOC_NEXT_NPURGE
Reads next message and leaves it on the host buffer

aceMTGetHBufMsgDecoded (continued)

ACE_MT_MSGLOC_LATEST_PURGE
Reads current message and takes it off of the host buffer

ACE_MT_MSGLOC_LATEST_NPURGE
Reads current message and leaves it on the host buffer

DESCRIPTION

This function reads a single decoded message from the host buffer if it is present. The function will use the aceMTDecodeRawMsg() function to decode the raw message into the MSGSTRUCT structure.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in MT or RTMT mode
ACE_ERR_PARAMETER	The pdwMsgLostStk input parameter is Null and/or the pdwMsgLostHBuf input parameter is Null
ACE_ERR_MT_MSGLOC	The wMsgLoc input parameter is not one of the following valid values: ACE_MT_MSGLOC_NEXT_PURGE ACE_MT_MSGLOC_NEXT_NPURGE ACE_MT_MSGLOC_LATEST_PURGE ACE_MT_MSGLOC_LATEST_NPURG

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U32BIT pdwMsgCount, pdwMsgLostStk, pdwMsgLostHBuf;
MSGSTRUCT *pMsg;
U16BIT wMsgLoc = 2;

nResult = aceMTGetHBufMsgDecoded(DevNum, &pMsg, &pdwMsgCount,
                                 &pdwMsgLostStk, &pdwMsgLostHBuf,
                                 wMsgLoc);
if(nResult)
{
      printf("Error in aceMTGetHBufMsgDecoded() function \n");
      PrintOutError(nResult);
      return;
}

if(!pdwMsgCount)
      printf("No message returned from Host Buffer\n");
```

SEE ALSO

aceMTStkToHBuf()   aceMTGetHBufMsgsRaw()

aceMTGetHBufMsgsRaw

This function reads as many messages as possible off of the host buffer into a user buffer.

PROTOTYPE

```
#include "Mtop.h"

S16BIT _DECL aceMTGetHBufMsgsRaw(S16BIT DevNum,
                                 U16BIT *pBuffer,
                                 U16BIT wBufferSize,
                                 U32BIT *pdwMsgCount,
                                 U32BIT *pdwMsgLostStk,
                                 U32BIT *pdwMsgLostHBuf);
```

STATE

Ready, Run

MODE

MT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

pBuffer (output parameter)
Pointer to an unsigned 16-bit word buffer that will receive the raw messages from the host buffer.

wBufferSize (input parameter)
Specify the size of the pBuffer buffer that will receive the raw messages from the host buffer.
Valid values:
There is no restriction on the size of this buffer, but for efficiency of resource use, the size of this buffer should be calculated as Number of messages * ACE_MSGSIZE_MT.

pdwMsgCount (output parameter)
If no errors occurred, this is the return value of the number of messages transferred to the pBuffer buffer.

pdwMsgLostStk (output parameter)
Estimates the number of messages lost due to overwrite prior to transferring the messages from the EMA to the host buffer.

aceMTGetHBufMsgsRaw (continued)

pdwMsgLostHBuf	(output parameter) Estimates the number of messages lost, due to overwrite when transferring the messages from the host buffer to the user buffer.

DESCRIPTION

This function reads as many messages as possible off of the host buffer into a user buffer without any decoding. The limiting factor when copying messages to the local buffer is the local buffer size and the number of messages available on the host buffer.

Note: Each message is a fixed length of ACE_MSGSIZE_MT words. This macro must be used in size calculations as the size of the structure is subject to change.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in MT or RTMT mode
ACE_ERR_PARAMETER	The pdwMsgLostStk input parameter is Null and/or the pdwMsgLostHBuf input parameter is Null

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT pBuffer[ACE_MSGSIZE_MT * 256];
U16BIT wBufferSize = ACE_MSGSIZE_MT * 256;
U32BIT pdwMsgCount, pdwMsgLostStk, pdwMsgLostHBuf;

nResult = aceMTGetHBufMsgsRaw(DevNum, pBuffer, wBufferSize,
                              &pdwMsgCount, &pdwMsgLostStk,
                              &pdwMsgLostHBuf);

if(nResult)
{
      printf("Error in aceMTGetHBufMsgsRaw() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceMTStkToHBuf()　　　aceMTGetHBufMsgDecoded()

aceMTGetInfo

This function will return the current configuration of the Monitor.

PROTOTYPE

#include “Mtop.h”

S16BIT _DECL aceMTGetInfo(S16BIT DevNum,
MTINFO *pInfo);

STATE

Ready, Run

MODE

MT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
pInfo	(output parameter) Pointer to a structure of type MTINFO that will be used to return the monitor mode information.

DESCRIPTION

This function takes in the address of an MTINFO structure and fills it in with the current configuration of the monitor.
The returned structure will contain the following information:
Command stack size
Data stack size
WStkMode (Double or Single)
1553 A Mode code use
Host buffer size

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in MT or RTMT mode
ACE_ERR_MT_HBUF	The host buffer does not exist

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
MTINFO *pInfo;

/* After setup of the monitor, the following statement can be used to
identify some of the monitor information.
*/
```

aceMTGetInfo (continued)

```
nResult = aceMTGetInfo(DevNum, pInfo);

if(nResult)
{
      printf(“Error in aceMTGetInfo() function \n”);

      PrintOutError(nResult);
      return;
}

printf(“wStkMode   = %s\nCommand  stack  size  =  %d\nData  stack  size  =
%d\n1553 A Mode code usage %s\nHost buffer size = %d\n”,
pInfo->wStkMode== ACE_MT_DOUBLESTK?”Double”:”Single”,
pInfo->wCmdStkSize, pInfo->wDataStkSize ,
pInfo->b1553aMCodes == TRUE?”Yes”:”No”, pInfo->dwHBufSize);
```

SEE ALSO

None

aceMTGetRTFilter

This function will read the selective Monitor Lookup table.

PROTOTYPE

```
#include "Mtop.h"


S16BIT _DECL aceMTDisableRTFilter(S16BIT DevNum,
                                  U16BIT wRT,
                                  U16BIT wTR,
                                  U32BIT * pSAMask);
```

STATE

Ready, Run

MODE

MT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

wRT (input parameter)
Specify the RT address of the message to get the filtering status.
Valid values:
0 – 31
Selects one RT subaddress value

wTR Specify the Transmit/Receive bit of the message to get the filtering status.
Valid values:
ACE_MT_FILTER_TX
Only Transmit

ACE_MT_FILTER_RX
Only Receive

pSAMask (output parameter)
Pointer to an unsigned 32-bit value that will be used to return a 32-bit packed value that represents the sub-addresses that will be monitored for the given Remote Terminal and T/R bit combination. The result can be compared to the following values.
Valid values:
ACE_MT_FILTER_SA_ALL
All subaddresses

aceMTGetRTFilter (continued)

ACE_MT_FILTER_SAXX
Specific subaddress where XX ranges from 0 -31

DESCRIPTION

The Enhanced Mini-ACE provides a flexible interface that allows selective monitoring of 1553 messages based on RT Address, T/R, and Subaddress with very little host processor intervention. The Message Monitor mode of the Enhanced Mini-ACE recreates all command/response messages on the 1553 bus on channels A and B, and stores them into the shared RAM based on a user programmable filter (RT address, T/R bit, and subaddress). This monitor can be used as a monitor alone or in a combined RT/Monitor mode.

This function reads the Filter status for the selected Remote Terminal based upon the RT address and the Transmit/Receive bit. The status returned will be a bit packed U32BIT value that represents the RT's subaddresses that will be monitored. The least significant bit represents SA0 while the most significant bit represents SA31. If the corresponding bit of a subaddress is set, then all messages to the RT that match the T/R will be monitored, otherwise they will not be.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in MT or RTMT mode
ACE_ERR_MT_FILTER_RT	The wRT input parameter contains a value greater than 31
ACE_ERR_MT_FILTER_TR	The wTR input parameter contains a value that is not one of the following valid values: ACE_MT_FILTER_TX ACE_MT_FILTER_RX
ACE_ERR_MT_FILTER_SA	The pSAMask parameter is Null

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT wRT = 10, wTR = ACE_MT_FILTER_TX;
U32BIT pwSAMask;

/* Read the monitor filter table for Remote Terminal 10 transmit
messages.
The return value “pwSAMask” may be decoded with the following macros:
ACE_MT_FILTER_SA_ALL = All sub-addresses
ACE_MT_FILTER_SAXX = Specific sub-address
*/

nResult = aceMTGetRTFilter(DevNum, wRT, wTR, pwSAMask);
```

aceMTGetRTFilter (continued)

```
if(nResult)
{
      printf("Error in aceMTGetRTFilter() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceMTDisableRTFilter() aceMTEnableRTFilter()

aceMTGetStkMetric

This function returns performance information about the MT Command Stack.

PROTOTYPE

```
#include "mtop.h"

S16BIT _DECL aceMTGetHBufMetric(S16BIT DevNum,
                                STKMETRIC *pMetric,
                                U16BIT wStk,
                                U16BIT bReset);
```

STATE

Ready, Run

MODE

MT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

pMetric (output parameter)
Pointer to a STKMETRIC structure to be filled in with metrics. The STKMETRIC structure contains the following members: dwLost, dwPctFull, and dwHighPct. The dwLost member parameter contains the total number of messages lost on the hardware stack. The dwPctFull parameter contains the percentage of the stack used at one snapshot in time. The dwHighPct parameter contains the highest percentage of the stack used over an extended period of time.

wStk (input parameter)
Specifies the stack to get metrics for
Valid values:

ACE_MT_STKA
Will get metrics for stack A

ACE_MT_STKB
Will get metrics for stack B

ACE_MT_STK_CMB
Will get the average metrics for stack A and B

aceMTGetStkMetric (continued)

bReset (input parameter)
This will specify if the highest percentage should be reset after this function returns.
Valid values:
FALSE (0)
Do not reset the highest percentage value

TRUE (1)
Reset the highest percentage value

DESCRIPTION

This function returns performance information about the MT Command Stack that is also referred to as the MT Descriptor Stack. Built-in test metrics can report the number of messages in the host buffer, the total number of messages lost since the host buffer was installed, the current percentage of the host buffer that is used, and the highest percentage of the host buffer used since it was installed.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in MT or RTMT mode
ACE_ERR_PARAMETER	The pMetric pointer input by the user is NULL and/or the wStk input parameter doesn't contain one of the following valid values: ACE_MT_DOUBLESTK, ACE_MT_STKA, ACE_MT_STKB, ACE_MT_STK_CMB
ACE_ERR_METRICS_NOT_ENA	Metrics are not enabled and should be set by calling the aceSetMetrics() function

EXAMPLE

```
S16BIT DevNum = 0;
STKMETRIC *pMetric;
U16BIT wStk = ACE_MT_STKA
U16BIT bReset = 1;
S16BIT nResult = 0;

nResult = aceMTGetStkMetric(DevNum, pMetric, wStk, bReset)

if(nResult)
{
      printf("Error in aceMTGetStkMetric() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceSetMetrics() aceMTGetHBufMetric()
aceRTGetHBufMetric() aceRTGetStkMetric()

aceMTGetStkMsgDecoded

This function reads a message on the Monitor stack and decodes it.

PROTOTYPE

```
#include "Mtop.h"

S16BIT _DECL aceMTGetStkMsgDecoded(S16BIT DevNum,
                                   MSGSTRUCT *pMsg,
                                   U16BIT wMsgLoc,
                                   U16BIT wStkLoc);
```

STATE

Ready, Run

MODE

MT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

pMsg (output parameter)
Pointer to a buffer of type MSGSTRUCT that will be used to return the decoded monitor message from the stack. The table below lists all member variables that exist in the MSGSTRUCT structure along with their definition.

Member Variable Name	Definition
wBlkSts	Contains the block status word of the message
wTimeTag	Contains the time tag of the message
wCmdWrd1	Contains the command word
wCmdWrd2	Contains the second command word for RT to RT transfers
wCmdWrd1Flg	Indicates the validity of the first command word
wCmdWrd2Flg	Indicates the validity of the second command word
wStsWrd1	Contains first status word
wStsWrd2	Contains second status word

aceMTGetStkMsgDecoded (continued)

Member Variable Name	Definition
wStsWrd1Flg	Indicates the validity of the first status word
wStsWrd2Flg	Indicates the validity of the second status word
wWordCount	Contains the number of valid data words
adataWrds[32]	An array that will contain the data words
wBCCtrlWrd	Contains the BC Control Word for BC mode messages only
wBCGapTime	Contains the message gap time only for BC
wBCLoopBack1	Contains the first looped back word for BC mode messages only
wBCLoopBack2	Contains the second looped back word for BC mode messages only
wBCLoopBack1Flg	Indicates validity of first loop back word for BC mode message only
wBCLoopBack2Flg	Indicates validity of second loop back word for BC mode message only

wMsgLoc

(input parameter)
Specify where the message is located on the selected stack. This parameter also indicates whether or not the selected message should be purged from the stack.
Valid values:

ACE_MT_MSGLOC_NEXT_PURGE
Retrieves the next message and takes it off of the stack

ACE_MT_MSGLOC_NEXT_NPURGE
Retrieves the next message and leaves it on the stack

ACE_MT_MSGLOC_LATEST_PURGE
Retrieves the current message and takes it off of the stack

ACE_MT_MSGLOC_LATEST_NPURGE
Retrieves the current message and leaves it on the stack

wStkLoc

(input parameter)
Defines which monitor stack that should be read. The monitor has two stacks that may be accessed as Stack A or Stack B. These two stacks may also be identified as the Active stack (the EMA is currently writing the active stack) or the Inactive stack (the EMA does not have access to the inactive stack until the stacks are swapped).
Valid values:

ACE_MT_STKLOC_ACTIVE
Reads the active stack

ACE_MT_STKLOC_INACTIVE
Reads the inactive stack

aceMTGetStkMsgDecoded (continued)

ACE_MT_STKLOC_STKA
Reads stack A

ACE_MT_STKLOC_STKB
Reads stack B

DESCRIPTION

This function reads either the next unread message or the latest message the monitor placed on the stack.

The function decodes the message by placing all of the message information into a MSGSTRUCT. This function decodes the raw message into a MSGSTRUCT structure by calling the aceMTDecodeRawMsg() function.

RETURN VALUE

S16BIT nResult	1 = message read 0 = message not read
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Run or Ready state
ACE_ERR_INVALID_MODE	The device is not in MT or RTMT mode
ACE_ERR_MSGSTRUCT	The pMsg parameter is Null
ACE_ERR_MT_STKLOC	The wStkLoc input parameter contains a value that is not one of the following valid types: ACE_MT_STKLOC_ACTIVE ACE_MT_STKLOC_INACTIVE ACE_MT_STKLOC_STKA ACE_MT_STKLOC_STKB
ACE_ERR_MT_MSGLOC	The wMsgLoc input parameter contains a value that is not one of the following valid types: ACE_MT_MSGLOC_NEXT_PURGE Reads next message and takes it off of the stack ACE_MT_MSGLOC_NEXT_NPURGE Reads next message and leaves it on the stack ACE_MT_MSGLOC_LATEST_PURGE Reads current message and takes it off of the stack ACE_MT_MSGLOC_LATEST_NPURGE Reads current message and leaves it on the stack

aceMTGetStkMsgDecoded (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
MSGSTRUCT pMsg;
U16BIT wStkLoc = ACE_MT_STKLOC_INACTIVE;
U16BIT wMsgLoc = ACE_MT_MSGLOC_LATEST_PURGE;

/* initialize Monitor, and start running.
Read the latest message from the inactive stack and purge the message
when it is read.
*/

nResult = aceMTGetStkMsgDecoded(DevNum, &pMsg, wMsgLoc,
                                wStkLoc);
if(nResult < 0)
{
      printf(“Error in aceMTGetStkMsgDecoded() function \n”);
      PrintOutError(nResult);
      return;
}

printf(“message read = %s\n”, nResult?”TRUE”:”FALSE”);
```

SEE ALSO

aceMTGetStkMsgsRaw()

aceMTGetStkMsgsRaw

This function reads as many messages as possible off of a given stack.

PROTOTYPE

```
#include "Mtop.h"

S16BIT _DECL aceMTGetStkMsgsRaw(S16BIT DevNum,
                                U16BIT *pBuffer,
                                U16BIT wBufferSize,
                                U16BIT wStkLoc);
```

STATE

Ready, Run

MODE

MT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

pBuffer (output parameter)
Pointer to an unsigned 16-bit word buffer that will be used to return the monitor message information.

wBufferSize (input parameter)
Size of buffer in words.
Valid values:
There is no restriction for the size of the buffer, but the most efficient use of resources is to create a buffer that is ACE_MSGSIZE_MT * the number of messages.

wStkLoc (input parameter)
Defines which monitor stack that should be read. The monitor has two stacks that may be accessed as Stack A or Stack B. These two stacks may also be identified as the Active stack (the EMA is currently writing the active stack) or the Inactive stack (the EMA does not have access to the inactive stack until the stacks are swapped).
Valid values:
ACE_MT_STKLOC_ACTIVE
Reads the active stack

aceMTGetStkMsgsRaw (continued)

ACE_MT_STKLOC_INACTIVE
Reads the inactive stack

ACE_MT_STKLOC_STKA
Reads stack A

ACE_MT_STKLOC_STKB
Reads stack B

DESCRIPTION

This function reads as many messages as possible off of a given stack. If no errors occur the number of messages will be returned. The limiting factor when copying messages to the buffer is the buffer size and the number of messages available on the stack.

Note: Each monitor message is a fixed length of ACE_MSGSIZE_MT words. This macro should be used for size calculation, as the size of the structure is subject to change.

RETURN VALUE

S16BIT nResult	Number of messages returned
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Run or Ready state
ACE_ERR_INVALID_MODE	The device is not in MT or RTMT mode
ACE_ERR_MT_STKLOC	The wStkLoc input parameter contains a value that is not one of the following valid types: ACE_MT_STKLOC_ACTIVE ACE_MT_STKLOC_INACTIVE ACE_MT_STKLOC_STKA ACE_MT_STKLOC_STKB
ACE_ERR_INVALID_BUF	The wBufferSize input parameter contains a value less than 40 and/or the pBuffer parameter is Null

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U16BIT pBuffer[128 * ACE_MSGSIZE_MT];
U16BIT wBufferSize = 128 * ACE_MSGSIZE_MT;
U16BIT wStkLoc = ACE_MT_STKLOC_INACTIVE;

/* initialize Monitor, and start running.
Read the 128 messages from the Monitor stack into the pBuffer parameter
*/

nResult = aceMTGetStkMsgsRaw(DevNum, pBuffer, wBufferSize,
                             wStkLoc);

if(nResult)
{
      printf("Error in aceMTGetStkMsgsRaw() function \n");
      PrintOutError(nResult);
      return;
}
```

aceMTGetStkMsgsRaw (continued)

SEE ALSO

aceMTGetStkMsgDecoded()

aceMTInstallHBuf

This function installs a host buffer.

PROTOTYPE

```
#include "Mtop.h"


S16BIT _DECL aceMTInstallHBuf(S16BIT DevNum,
                              U32BIT dwHBufSize);
```

STATE

Ready

MODE

MT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31
dwHBufSize	(input parameter) Specify the size of the host buffer to be created for use by the monitor operation. Valid values: (MT command stack size/4)*120 to 5,120,000 words

DESCRIPTION

This function allocates a host buffer based on the size parameter. For this function to succeed the size must be at least three times greater than the number of messages that can be stored in the command stacks multiplied by ACE_MSGSIZE_MT (this is the macro that describes the length in words of the monitor message).

For example, if the command stack is 256 words then the HBuf size must be at least:
(256/4) * ACE_MSGSIZE_MT

Note: The dwHBufSize parameter is in words. The macro ACE_MSGSIZE_MT should be used for buffer size calculations as the size of the message structure is subject to change.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in MT or RTMT mode
ACE_ERR_MT_HBUFSIZE	The dwHBufSize input parameter is too small
ACE_ERR_MT_HBUF	Memory for the host buffer could not be allocated

aceMTInstallHBuf (continued)

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
U32BIT dwHBufSize;

/* create a host buffer that is to be used for monitor operation.  The
monitor has a 512 word command stack.   This provides 512/4 =128
messages.  The host buffer must be (512/4)* ACE_MSGSIZE_MT words in
length
*/

dwHBufSize = (512/4)* ACE_MSGSIZE_MT;

nResult = aceMTInstallHBuf(DevNum, dwHBufSize);

if(nResult)
{
      printf(“Error in aceMTInstallHBuf() function \n”);
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceMTUninstallHBuf()

aceMTPause

This function temporarily stops the Monitor from capturing messages.

PROTOTYPE

#include “Mtop.h”

S16BIT _DECL aceMTPause(S16BIT DevNum);

STATE

Run

MODE

MT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function temporarily stops the Monitor from capturing messages. The Monitor can be resumed using its current state with the aceMTContinue() function. This function does not change the state of operation.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Run state
ACE_ERR_INVALID_MODE	The device is not in MT mode
ACE_ERR_INVALID_ACCESS	The device is not in ACE_ACCESS_CARD access mode or ACE_ACCESS_USR access mode

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceMTPause(DevNum);

if(nResult)
{
      printf(“Error in aceMTPause() function \n”);
      PrintOutError(nResult);
      return;
}
```

aceMTPause (continued)

SEE ALSO

aceMTStart()
aceMTStop()
aceMTContinue()

aceMTSetHBufTrigger

This function sets the host buffer to capture messages only after a message is read that matches the trigger structure.

PROTOTYPE

```
#include "Mtop.h"

S16BIT _DECL aceMTSetHBufTrigger(S16BIT DevNum,
                                 U16BIT wHBufPercent,
                                 MTTRIGGER *pTrg);
```

STATE

Ready

MODE

MT, RTMT

PARAMETERS

DevNum (input parameter)
Logical Device Number
Valid values:
0 – 31

wHBufPercent (input parameter)
Specify the percentage of the host buffer size that should be used to store messages prior to the trigger point.
Valid values:

ACE_MT_TRIG_HBUF_75P
Fill buffer to 75% prior to trigger

ACE_MT_TRIG_HBUF_50P
Fill buffer to 50% prior to trigger

ACE_MT_TRIG_HBUF_25P
Fill buffer to 25% prior to trigger

ACE_MT_TRIG_HBUF_0P
Trigger right away

aceMTSetHBufTrigger (continued)

pTrg

(input parameter)
Specify the trigger condition. This is a pointer to an MTTRIGGER structure that can be any combination of the message elements including a specific data word. Each message element also has a mask that can be applied to the actual message. A mask for the following parameters is set to 1, then that bit becomes DON'T CARE, if the bit is 0, then the value of the parameter must match the actual value of the message.
Valid values:

Pointer to an MTTRIGGER that contains the following elements:

Elements of the MTTRIGGER structure are:
wCmdWrd1
Command word 1

wCmdMsk1
Mask for command word 1

wCmdWrd2
Command word 2

wCmdMsk2
Mask for command word 2

wStsWrd1
Status word 1

wStsMsk1
Mask for Status word 1

wStsWrd2
Status word 2

wStsMsk2
Mask for Status word 2

wDataWrd
Selected data word

wDataMsk
Mask for selected data word

wDataPos
Position of selected data word (1 – 31)
wErrWrd
Block Status word errors

aceMTSetHBufTrigger (continued)

wErrFlg
Trigger based on all errors or just one error
Valid values: TRUE, FALSE
wTrigFlags
Number of triggers needed to produce a real trigger

wNextFlags
Used for complex triggering

wCount
Message word count

DESCRIPTION

This function sets the HBuf to capture messages only after a message is read (via StkToHBuf function) that matches the trigger structure. The wHBufPercent parameter gives information on how many messages will be stored in the host buffer prior to trigger. This allows for pre and post triggering.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in MT or RTMT mode
ACE_ERR_PARAMETER	The wHBufPercent input parameter contains an incorrect value and/or the pTrg parameter is Null

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;
MTTRIGGER pTrg;

pTrig.wCmdWrd1 = 0x2822; /* RT=5, Rec, SA=2, WC=2 */
pTrig.wCmdMsk1 = 0x0000; /* don’t mask any bits */
pTrig.wDataWrd = 0x0034; /* look for data 1234 */
pTrig.wDataMsk = 0xFF00; /* mask out the upper 8 bits */

nResult = aceMTSetHBufTrigger(DevNum, ACE_MT_TRIG_HBUF_50P,
                              &pTrg);

if(nResult)
{
      printf(“Error in aceMTSetHBufTrigger() function \n”);
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceMTClearHBufTrigger()

aceMTStart

This function starts the Monitor capturing messages.

PROTOTYPE

#include “Mtop.h”

S16BIT _DECL aceMTStart(S16BIT DevNum);

STATE

Ready

MODE

MT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function initializes all command and data stack pointers, monitor structures, and monitor registers necessary to run the device in monitor mode. This function enables enhanced mode. After this function has been called the device is left in a Run state.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in MT mode
ACE_ERR_INVALID_ACCESS	The device is not in ACE_ACCESS_CARD access mode or ACE_ACCESS_USR access mode

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

/* Initialize Monitor mode, setup filter table, create hBuf */

nResult = aceMTStart(DevNum);

if(nResult)
{
      printf(“Error in aceMTStart() function \n”);
      PrintOutError(nResult);
      return;
}
```

aceMTStart (continued)

SEE ALSO

aceMTStop()
aceMTPause()
aceMTContinue()

aceMTStkToHBuf

This function swaps the active and inactive stacks and then copies all messages from the inactive stack to the host buffer.

PROTOTYPE

```
#include "Mtop.h"

S16BIT _DECL aceMTStkToHBuf(S16BIT DevNum);
```

STATE

Ready, Run

MODE

Advanced plus one of the following: MT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function swaps the active and inactive stacks and then copies all messages from the inactive stack to the host buffer. Once the messages have been moved to the host buffer, they can be processed by the application using one of the following routines:

aceMTGetHBufMsgsRaw()
aceMTGetHBufMsgsDecoded()

The Enhanced Mini-ACE runtime library calls this function inside of the internal interrupt service routine that is processed by the library on any of the following conditions:

ACE_IMR1_MT_DATASTK_ROVER
100% Data Stack rollover point

ACE_IMR2_MT_DSTK_50P_ROVER
50% Data Stack rollover point

ACE_IMR1_MT_CMDSTK_ROVER
100% Command Stack rollover point

ACE_IMR2_MT_CSTK_50P_ROVER
50% Command Stack rollover point

ACE_IMR1_TT_ROVER
Time Tag rollover

The library will do this to reliably transfer messages and data from the hardware stacks so that the user does not need to ever call this function. This function is provided in the

aceMTStkToHBuf (continued)

Enhanced Mini-ACE library as an advanced mode function that can be used to transfer messages and data to your host buffer if your operating system doesn't support the use of interrupts. In Windows operating systems that support interrupt generation, this function should **not** be called by the user.

Note: *The host buffer is guaranteed to be able to hold the messages from the hardware stack, but if the host buffer message processing is not performed regularly, then the new message data may overwrite existing un-read messages in the host buffer.*

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in MT or RTMT mode

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceMTStkToHBuf(DevNum);

if(nResult)
{
      printf("Error in aceMTStkToHBuf() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceMTGetHBufMsgsRaw()   aceMTGetHBufMsgDecoded()

aceMTStkToHBuf32

This function swaps the active and inactive stacks and then copies all messages from the inactive stack to the host buffer by performing 32-bit memory accesses.

PROTOTYPE

```
#include "Mtop.h"


S16BIT _DECL aceMTStkToHBuf32(S16BIT DevNum);
```

STATE

Ready, Run

MODE

Advanced plus one of the following: MT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31	

DESCRIPTION

This function swaps the active and inactive stacks and then copies all messages from the inactive stack to the host buffer. Once the messages have been moved to the host buffer, they can be processed by the application using one of the following routines:
aceMTGetHBufMsgsRaw()
aceMTGetHBufMsgsDecoded()

The Enhanced Mini-ACE runtime library calls this function inside of the internal interrupt service routine that is processed by the library on any of the following conditions:
ACE_IMR1_MT_DATASTK_ROVER
100% Data Stack rollover point

ACE_IMR2_MT_DSTK_50P_ROVER
50% Data Stack rollover point

ACE_IMR1_MT_CMDSTK_ROVER
100% Command Stack rollover point

ACE_IMR2_MT_CSTK_50P_ROVER
50% Command Stack rollover point

ACE_IMR1_TT_ROVER
Time Tag rollover
The library will do this to reliably transfer messages and data from the hardware stacks so that the user does not need to ever call this function. This function is provided in the

aceMTStkToHBuf32 (continued)

Enhanced Mini-ACE library as an advanced mode function that can be used to transfer messages and data to your host buffer if your operating system doesn't support the use of interrupts. In Windows operating systems that support interrupt generation, this function should **not** be called by the user.

This function is used on all cards except for the BU-65567/68 and the BU-65553 cards because these cards are ISA devices that use the 16-bit memory accesses in the aceMTStkToHBuf() function call.

Note: *The host buffer is guaranteed to be able to hold the messages from the hardware stack, but if the host buffer message processing is not performed regularly, then the new message data may overwrite existing un-read messages in the host buffer.*

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready or Run state
ACE_ERR_INVALID_MODE	The device is not in MT or RTMT mode

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceMTStkToHBuf(DevNum);

if(nResult)
{
      printf("Error in aceMTStkToHBuf() function \n");
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceMTGetHBufMsgsRaw() aceMTGetHBufMsgDecoded()

aceMTStop

This function stops the Monitor from capturing messages.

PROTOTYPE

#include “Mtop.h”

S16BIT _DECL aceMTStop(S16BIT DevNum);

STATE

Run

MODE

MT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function stops the Monitor from capturing messages and puts the device into the Ready state.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Run state
ACE_ERR_INVALID_MODE	The device is not in MT mode
ACE_ERR_INVALID_ACCESS	The device is not in ACE_ACCESS_CARD access mode or ACE_ACCESS_USR access mode

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceMTStop(DevNum);

if(nResult)
{
      printf(“Error in aceMTStop() function \n”);
      PrintOutError(nResult);
      return;
}
```

SEE ALSO

aceMTStart() aceMTPause()
aceMTContinue()

aceMTSwapStks

This function swaps the active and inactive stacks.

PROTOTYPE

#include “Mtop.h”

S16BIT _DECL aceMTSwapStks(S16BIT DevNum);

STATE

Run

MODE

MT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function swaps the active and inactive stacks.

RETURN VALUE

ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Run state
ACE_ERR_INVALID_MODE	The device is not in MT mode, or is not configured for ACE_MT_DOUBLESTK
‘0’	Stack A is now active
‘1’	Stack B is now active

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

nResult = aceMTSwapStks (DevNum);

if(nResult)
{
printf(“Error       in       aceMTSwapStks()       function       \n”);
PrintOutError(nResult);
      return;
}
```

SEE ALSO

None

aceMTUninstallHBuf

The function will uninstall the host buffer.

PROTOTYPE

#include “Mtop.h”

S16BIT _DECL aceMTUninstallHBuf(S16BIT DevNum);

STATE

Ready

MODE

MT, RTMT

PARAMETERS

DevNum	(input parameter) Logical Device Number Valid values: 0 – 31

DESCRIPTION

This function deallocates the MT host buffer if present. There can be only one host buffer per mode, so there is no requirement to specify a host buffer handle.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	An invalid device number was input by the user
ACE_ERR_INVALID_STATE	The device is not in a Ready state
ACE_ERR_INVALID_MODE	The device is not in MT or RTMT mode
ACE_ERR_MT_HBUF	The host buffer does not exist

EXAMPLE

```
S16BIT DevNum = 0;
S16BIT nResult = 0;

/* remove the Monitor mode host buffer if it is installed.
If it doesn’t exist, the error ‘ACE_ERR_MT_HBUF’ will be returned.
*/

nResult = aceMTUninstallHBuf(DevNum);

if(nResult)
{
      printf(“Error in aceMTUninstallHBuf() function \n”);
      PrintOutError(nResult);
      return;
}
```

aceMTUninstallHBuf (continued)

SEE ALSO

aceMTInstallHBuf()

VXWORKS FUNCTION DEFINITIONS

This section contains all of the VxWorks specific functions. These functions are used to set up your card to be ready for use in a VxWorks operating system. All of the PMC card specific functions are listed first and the PC/104 card functions follow.

VXWORKS Functions

Table 9. VXWORKS Functions Listing

aceVxCreateDevs

This function is used to setup PMC Cards located on a VME carrier card.

PROTOTYPE

```
S16BIT _DECL aceVxCreateDevs(U32BIT dwCarrierBase,
                             U16BIT wIrqLevel);
```

STATE

Initialization

MODE

N/A

PARAMETERS

dwCarrierBase	(input parameter) An unsigned 32-bit word value that specifies the base address of the VME carrier card that the EMA card is plugged onto. This address would normally be set on the card by adjusting jumpers or switches on the VME carrier card.
wIrqLevel	An unsigned 16-bit word value that specifies the interrupt vector that should be used by the device driver for all EMA cards that are installed on the VME card. Adjusting jumpers on the card to the desired interrupt vector value would normally set this address.

DESCRIPTION

This function is used to create accessible VxWorks device names (one per channel on a card) for all PMC Cards (BU-65565) located on a VME carrier card. The carrier is specified by its base address input by the user in the dwCarrierBase input parameter. This parameter should be zero if enumerating the local PCI bus. This function may be called once for each VME carrier card in the system and once for the local PCI bus.

RETURN VALUE

S16BIT nResult	The number of devices that have been created
ACE_ERR_TOO_MANY_DEVS	There are more than 32 devices defined
ACE_ERR_INVALID_OS	This is an invalid operating system and/or the device is not a PMC card

EXAMPLE

None

SEE ALSO

None

aceVxCreateEBRDevs

This function is used to setup the Enhanced Bit Rate 1553 PC/104 Cards located on the ISA bus.

PROTOTYPE

```
S16BIT _DECL aceVxCreateEBRDevs (U32BIT dwMemBaseAddr,
                                 U16BIT wIrqLevel,
                                 U32BIT dwRegBaseAddr);
```

STATE

Initialization

MODE

N/A

PARAMETERS

dwMemBaseAddr	(input parameter) An unsigned 32-bit word value that specifies the base address of the VME carrier card that the device is plugged onto. This address would normally be set on the card by adjusting jumpers or switches on the VME carrier card.
wIrqLevel	(input parameter) An unsigned 16-bit word value that specifies the interrupt vector that should be used by the device driver for all EMA cards that are installed on the VME card. Adjusting jumpers on the card to the desired interrupt vector value would normally set this address.
dwRegBaseAddr	(input parameter) An unsigned 32-bit word value that specifies the base register address.

DESCRIPTION

This function is used to setup the Enhanced Bit Rate 1553 PC/104 Cards located on the ISA bus. This function is only applicable to the Enhanced Bit Rate PC/104 card.

RETURN VALUE

S16BIT nResult	The number of devices that have been created
ACE_ERR_MAPMEM_ACC	The card did not map to memory properly
ACE_ERR_INVALID_OS	This is an invalid operating system and/or the device is not a PC/104 card

EXAMPLE

None

SEE ALSO

None

aceVxCreateISADevs

This function is used to setup the PC/104 cards located on the ISA bus.

PROTOTYPE

```
S16BIT _DECL aceVxCreateISADevs(U32BIT dwMemBaseAddr,
                                U16BIT wIrqLevel,
                                U32BIT dwRegBaseAddr);
```

STATE

Initialization

MODE

N/A

PARAMETERS

dwMemBaseAddr	(input parameter) An unsigned 32-bit word value that specifies the base memory address.
wIrqLevel	(input parameter) An unsigned 16-bit word value that specifies the interrupt vector that should be used by the device driver for all EMACE cards that are installed on the ISA bus.
dwRegBaseAddr	(input parameter) An unsigned 32-bit word value that specifies the register base memory address.

DESCRIPTION

This function is used to setup the PC/104 Cards located on the ISA Bus. The function configures the I/O registers and sets up the base memory address input by the user and the base register address input by the user.

RETURN VALUE

S16BIT nResult	The number of devices that have been created
ACE_ERR_INVALID_OS	This is an invalid operating system and/or the device is not a PC/104 card

EXAMPLE

None

SEE ALSO

None

aceVxEnableSBMode

This function allows the user to enable / disable the South Bridge mode feature of BU-65568 and BU-6558x PC/104 cards .

PROTOTYPE

```
S16BIT _DECL aceVxEnableSBMode (U16BIT wCardType,
                                U16BIT bEnable);
```

STATE

Initialization

MODE

N/A

PARAMETERS

wCardType	(input parameter) Card Type Selection Valid values: 0 for BU-65568, 1 for BU-6558x
bEnable	(input parameter) Switch to enable / disable South Bridge Mode Valid values: FALSE This will disable interrupts TRUE This will enable interrupts

DESCRIPTION

This will enable / disable the South Bridge feature of the BU-65568 and BU-6558x PC/104 devices. This feature, when enabled, turns on an enhanced address decoder built into later generation DDC PC/104 cards. The enhancements include:

1) Support for address decoding of 128K memory space allowing for 16-bit and 8-bit memory devices to be located within the same 128K paragraph of system memory.
2) Enhancement to decoding of Latched Address (LA) bus that provides for increased reliability of address decoding. This feature takes advantage of systems that have both the SA and LA address bus signals at the same time (i.e. systems that have the ISA address bus are based on a South Bridge PCI interface).

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully

EXAMPLE

```
//Enable South Bridge Mode for a BU-65568 assigned as device #0
  aceVxEnableSBMode (0, TRUE);
```

aceVxEnableSBMode (continued)

SEE ALSO

None

aceVxGetDevInfo

This function is used to retrieve physical information associated with a mapped logical device number.

PROTOTYPE

```
S16BIT _DECL aceVxGetDevInfo(S16BIT DevNum,
                             U32BIT *pCarrierBase,
                             S32BIT *pBusNo,
                             S32BIT *pDevNo,
                             S32BIT *pFuncNo,
                             U16BIT *pChannel);
```

STATE

Initialization

MODE

N/A

PARAMETERS

DevNum	(input parameter) A signed 16-bit word value that specifies which device the requested information pertains to. This device number may have been received from a previous call to aceVxGetDevNum and is based on the location in the computer.
pCarrierBase	(output parameter) A pointer to an unsigned 32-bit word that will contain the base address of the VME carrier card that the specified device is plugged into. This address would normally be set on the card by adjusting jumpers or switches on the VME card.
pBusNo	(output parameter) A pointer to a signed 32-bit word that will contain the number of the PCI bus where the specified device resides.
pDevNo	A pointer to a signed 32-bit word that will contain the PCI Device (or slot) number where the card resides.
pFuncNo	A pointer to a signed 32-bit word that will return the PCI function number. Valid values: 0 BU-65565 PMC card Other values are reserved for future use.

aceVxGetDevInfo (continued)

pChannel	A pointer to a signed 32-bit word that will contain the channel number of the device on the card.

DESCRIPTION

This function is used to retrieve physical information associated with a mapped logical device number. The channel is fully specified by its address (0 if on local PCI bus), the channel on the card, its PCI Bus number, its PCI Device number, its PCI function number, and the channel # on the card.

RETURN VALUE

S16BIT nResult	0 if device is found -1 if device is not found
ACE_ERR_INVALID_OS	This is not VxWorks and/or this is not a PMC card

EXAMPLE

None

SEE ALSO

None

aceVxGetDevNum

This function is used to retrieve a logical device number associated with a particular channel on the PMC card.

PROTOTYPE

```
S16BIT _DECL aceVxGetDevNum(S16BIT *pDevNum,
                            U32BIT dwCarrierBaseAddr,
                            S32BIT nBusNo,
                            S32BIT nDevNo,
                            S32BIT nFuncNo,
                            U16BIT wChannel)
```

STATE

Initialization

MODE

N/A

PARAMETERS

pDevNum	(output parameter) A signed 16-bit word pointer to a signed short variable that will be filled with the device number by function. This device number will be used when accessing all functions for the runtime library.
dwCarrierBaseAddr	(input parameter) An unsigned 32-bit word value that specifies the base memory address of the VME carrier card on which the EMA card is plugged into. This address would normally be set on the card by adjusting jumpers or switches on the VME card. This parameter should be set to 0 if the EMA card is located on the processor card in order to specify the local PCI bus.
nBusNo	(input parameter) Specify the number of the PCI bus where EMA card resides.
nDevNo	(input parameter) Specify the PCI Device (or slot) number where card resides.

aceVxGetDevNum (continued)

nFuncNo	(input parameter) Specify the PCI function number. Valid values: 0 Use this value for the BU-65565 PMC card Other values reserved for future use
wChannel	Specify the channel number on the card. If the EMA card contains more than one 1553 channel, this parameter will specify which one is being accessed. Valid values: 0 First installed channel 1 Second installed channel >1 and < maximum channels on card Other installed channels

DESCRIPTION

This function is used to retrieve a logical device number associated with a particular channel. The channel is fully specified by its address (0 if on local PCI bus), the channel on the card, its PCI Bus number, its PCI Device number, its PCI function number, and the channel number of the device on the card. This function can not be used with the PC/104 card.

RETURN VALUE

S16BIT nResult	0 if device is found -1 if device is not found
ACE_ERR_INVALID_OS	This is not VxWorks and/or this is not a PMC card

EXAMPLE

None

SEE ALSO

None

aceVxGetISADevInfo

This function is used to retrieve physical information associated with a mapped logical device number for a PC/104 card.

PROTOTYPE

```
S16BIT _DECL aceVxGetISADevInfo (U16BIT DevNum,
                                 U16BIT *pChannel,
                                 U32BIT *pBaseMem,
                                 U32BIT *pBaseReg);
```

STATE

Initialization

MODE

N/A

PARAMETERS

DevNum	(input parameter) An unsigned 16-bit word value that specifies which device the requested information pertains to.
pChannel	(output parameter) A pointer to an unsigned 16-bit word that will contain the channel number of the device on the card.
pBaseMem	(output parameter) A pointer to an unsigned 32-bit word that will contain the base memory address of the card.
pBaseReg	(output parameter) A pointer to an unsigned 32-bit word that will contain the base register address of the card.

DESCRIPTION

This function is used to retrieve physical information associated with a mapped logical device number.

RETURN VALUE

S16BIT nResult	-1: Error
ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_OS	This is an invalid operating system and/or the device is not a PC/104 card

EXAMPLE

None

SEE ALSO

None

aceVxSetIOPort

This function allows the user to set the base IO address of the PC/104 card.

PROTOTYPE

```
VOID _DECL aceVxSetIOPort (U16BIT wPort);
```

STATE

Initialization

MODE

N/A

PARAMETERS

wPort	(input parameter) The user specified port number.

DESCRIPTION

This function allows the user to set the base IO address of the card. The default IO base address is 0x360.

Note: This function is effective only if the aceVxCreateISADevs() function has not been called since the target hardware has been powered up.

RETURN VALUE

None

EXAMPLE

None

SEE ALSO

None

aceVxSetPCIAddressInfo

This function allows the user to specify where a PMC card residing on the local bus will be placed in memory.

PROTOTYPE

```
S16BIT _DECL aceVxSetPCIAddressInfo(U32BIT dwPCIBaseAddress,
                                    U32BIT dwPCIWindowSize);
```

STATE

Initialization

MODE

N/A

PARAMETERS

dwPCIBaseAddress	(input parameter) An unsigned 32-bit word value that specifies the start location into which cards memory can be configured.
dwPCIWindowSize	(input parameter) An unsigned 32-bit word value that specifies the total size of the memory window where cards will reside.

DESCRIPTION

This function allows the user to specify where a PMC card residing on the local bus will be placed in memory. The user can specify the base address and size of memory window that the PMC card(s) are allowed to map to.
Example: The default base address is 0xFD000000, with a size of 0x01000000 (16MB).

Note: Since all PMC cards that reside on the local PCI bus will be configured with one call to aceVxCreateDevs, this function need only be called once before the call to aceVxCreateDevs.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_OS	This is not VxWorks and/or this is not a PMC card

EXAMPLE

None

SEE ALSO

None

aceVxSetTaskPriority

This function allows the user to change the priority of the DPC ISR task thread.

PROTOTYPE

```
S16BIT _DECL aceVxSetTaskPriority (S16BIT DevNum,
                                   int nPriority);
```

STATE

N/A

MODE

N/A

PARAMETERS

DevNum	(input parameter) A signed 16-bit word value that specifies the device.
nPriority	(input parameter) An integer value that specifies the interrupt priority in the system. Valid Values: 0 to 255

DESCRIPTION

When a hardware interrupt fires, the device driver Hardware Interrupt Procedure is called (e.g. vxHwIntProc). The Hardware Interrupt Procedure is responsible for clearing any hardware interrupts generated by the applicable DDC hardware product. Once the hardware interrupt is cleared, a DPC (Deferred Process Call) is made to the previously spawned Interrupt Service Routine (ISR). The ISR is responsible for servicing any RTL internal processes as well as any assigned external ISR assigned by the user. The aceVxSetTaskPriority function allows the user to set the priority of the DPC task.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	The device number input by the user is invalid
ACE_ERR_PARAMETER	The priority input is invalid or the ISR does not exist
ACE_ERR_TASK_FAIL	The priority failed to be set properly

EXAMPLE

```
//set device 0's DPC task thread priority to 100
nResult = aceVxSetTaskPriority(DevNum, 100);
```

SEE ALSO

None

DOS FUNCTION DEFINITIONS

This section contains all of the DOS specific functions. These functions are used to set up your card to be ready for use in a DOS operating system.

DOS Functions

Table 10. DOS Functions Listing

aceDOSCreateDevice

This function configures the device for use in a DOS operating system.

PROTOTYPE

```
S16BIT _DECL aceDOSCreateDevice (char *pFileName);
```

STATE

Initialization

MODE

N/A

PARAMETERS

pFileName	(input parameter) A pointer to a configuration file that contains the card information.

DESCRIPTION

This function configures the device for use in a DOS operating system. The function maps the card's memory into the system, and gives driver access to the card to start initializing its structure.

RETURN VALUE

S16BIT nResult	0 An error occurred >0 The number of devices that have been created
ACE_ERR_INVALID_OS	This is not a DOS operating system

EXAMPLE

None

SEE ALSO

None

aceDOSEnableSBMode

This function allows the user to enable / disable the South Bridge mode feature of BU-65568 and BU-6558x PC/104 cards.

PROTOTYPE

```
S16BIT _DECL aceDOSEnableSBMode (U16BIT wCardType,
                                 U16BIT bEnable);
```

STATE

Initialization

MODE

N/A

PARAMETERS

wCardType	(input parameter) Card Type Selection Valid values: 0 for BU-65568, 1 for BU-6558x
bEnable	(input parameter) Switch to enable / disable South Bridge Mode Valid values: FALSE This will disable interrupts TRUE This will enable interrupts

DESCRIPTION

This will enable / disable the South Bridge feature of the BU-65568 and BU-6558x PC/104 devices. This feature, when enabled, turns on an enhanced address decoder built into later generation DDC PC/104 cards. The enhancements include:

1) Support for address decoding of 128K memory space allowing for 16-bit and 8-bit memory devices to be located within the same 128K paragraph of system memory.
2) Enhancement to decoding of Latched Address (LA) bus that provides for increased reliability of address decoding. This feature takes advantage of systems that have both the SA and LA address bus signals at the same time (i.e. systems that have the ISA address bus are based on a South Bridge PCI interface).

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully

EXAMPLE

```
//Enable South Bridge Mode for a BU-65568 assigned as device #0
  aceDOSEnableSBMode (0, TRUE);
```

aceDOSEnableSBMode (continued)

SEE ALSO

None

INTEGRITY FUNCTION DEFINITIONS

This section contains all of the Integrity specific functions. These functions are used to set up your card to be ready for use in the Green Hills Integrity operating system.

INTEGRITY Functions

Table 11. INTEGRITY Functions Listing

aceSetIntegrityIntPriority

This function allows the user to change the priority of the DPC ISR task thread.

PROTOTYPE

#include integrityop.h

S16BIT aceSetIntegrityIntPriority (S16BIT DevNum, S16BIT wPriority);

STATE

N/A

MODE

N/A

PARAMETERS

DevNum	(input parameter) A signed 16-bit word value that specifies the device.
wPriority	(input parameter) An integer value that specifies the interrupt priority in the system. Valid Values: 0 to 255

DESCRIPTION

When a hardware interrupt fires, the device driver Hardware Interrupt Procedure is called. The Hardware Interrupt Procedure is responsible for clearing any hardware interrupts generated by the applicable DDC hardware product. Once the hardware interrupt is cleared, a DPC (Deferred Process Call) is made to the previously spawned Interrupt Service Routine (ISR). The ISR is responsible for servicing any RTL internal processes as well as any assigned external ISR assigned by the user. The aceSetIntegrityIntPriority function allows the user to set the priority of the DPC task.

RETURN VALUE

ACE_ERR_SUCCESS	The function completed successfully
ACE_ERR_INVALID_DEVNUM	The device number input by the user is invalid
ACE_ERR_PARAMETER	The priority input is invalid or the ISR does not exist
ACE_ERR_TASK_FAIL	The priority failed to be set properly

EXAMPLE

```
//set device 0's DPC task thread priority to 100
nResult = aceSetIntegrityIntPriority (0, 100);
```

SEE ALSO

None

APPENDIX A

Error Messages

The following section contains a list of the error macros that are returned by the functions of the Enhanced Mini-ACE library and the text that is associated with the error. The error macro can be converted with the aceErrorStr() function to the actual error text.

ACE_ERR_SUCCESS

Error Number: 0

Text: No error occurred

Description: The function completed successfully.

ACE_ERR_INVALID_DEVNUM

Error Number: -50

Text: Invalid device number

Description: The device number parameter of the function call is not registered with the system as being a valid device. Device numbers range from 0 – 31.

Solution: Select a device number that corresponds to an actual EMA device in the system.

ACE_ERR_INVALID_ACCESS

Error Number: -51

Text: Invalid access type

Description: The EMA was initialized with an access type (Simulate, Card Access, or User memory) that is not consistent with the operation requested.

Solution: Close the device with the aceFree() function and re-initialize the device to the proper access usage using the aceInitialize() function.

ACE_ERR_INVALID_MODE

Error Number: -52

Text: Invalid mode of operation

Description: The device was initialized to a mode of operation (BC, RT, MT or RTMT) that is not consistent with the operation requested.

Solution: Select a correct operation or modify the mode of the device by closing it with the aceFree() function and reinitialization with the aceInitialize() function.

ACE_ERR_INVALID_STATE

Error Number: -53

Text: Invalid device state

Description: The device is in an operation state (Ready or Run) that is not constant with the operation requested.

Solution: Modify the state of the device or select another operation that will work with the current state.

ACE_ERR_INVALID_MEMSIZE

Error Number: -54

Text: Bad memory word size

Description: The specified memory size is not valid.

Solution: Verify that the memory size you are trying to pass in is valid and call the function again.

ACE_ERR_INVALID_ADDRESS

Error Number: -55

Text: Invalid device address

Description: The application is attempting to access an address that is not associated with either memory or registers for the selected device.

Solution: Verify that the address being accessed is truly valid for the device.

ACE_ERR_INVALID_OS

Error Number: -56

Text: Invalid Operating System

Description: The application is attempting to use functionality that is specific to a different operating system. Some of the functions in the library can only be performed in a Win32 operating system.

Solution: Verify that the function and the operating system are compatible.

ACE_ERR_INVALID_MALLOC

Error Number: -57

Text: Memory allocation failed

Description: The library attempted to create a data structure that requires dynamic allocation of memory and the operation failed. This will usually be found in systems that have limited memory resources.

Solution: Suggest a smaller data structure if possible or add more memory to the system.

ACE_ERR_INVALID_BUF

Error Number: -58

Text: Invalid buffer

Description: This error is returned when a library function requires a user supplied buffer. If a buffer pointer is supplied in the parameter list of the function, but the buffer was never created, the 'Invalid buffer' error will be returned.

Solution: Declare and correctly create a buffer of the appropriate type and size for the operation being performed.

ACE_ERR_INVALID_ADMODE

Error Number: -59

Text: Bad addressing mode

Description: The way in which hardware registers and memory are being addressed is incorrect. If you are running a DDC card product this value should never be changed in the library.

Solution: The address mode can be changed with the aceSetAddressMode() function as long as you are in Advanced mode.

ACE_ERR_SIMWRITEREG

Error Number: -60

Text: Invalid simulated write register

Description: The library was not able to write to one of the simulated hardware registers.

Solution: Reinstall EMA or complete installation.

ACE_ERR_TIMETAG_RES

Error Number: -61

Text: Invalid time tag resolution

Description: An invalid time tag resolution has been selected.

Solution: Select a valid time tag resolution.

ACE_ERR_RESPTIME

Error Number: -62

Text: Invalid response timeout value

Description: A timeout response has occurred on the hardware.

Solution: Try the function again or select a longer response time with the aceSetRespTimeOut() function.

ACE_ERR_CLOCKIN

Error Number: -63

Text: Invalid clock input value

Description: An invalid clock frequency has been selected or is being used.

Solution: Try the function again or select a different clock frequency with the aceSetClockFreq() function.

ACE_ERR_MSGSTRUCT

Error Number: -64

Text: Invalid message structure

Description: An invalid message structure has been specified.

Solution: Try the function again and check to make sure the value of the message structure is not Null.

ACE_ERR_MSGSTRUCT

Error Number: -64

Text: Invalid message structure

Description: An invalid message structure has been specified.

Solution: Try the function again and check to make sure the value of the message structure is not NULL.

ACE_ERR_PARAMETER

Error Number: -65

Text: Invalid parameter

Description: An invalid parameter has been input by the user.

Solution: Try the function again and make sure that the parameters are all within the defined boundaries.

ACE_ERR_INVALID_MODE_OP

Error Number: -66

Text: Invalid mode/options combination

Description: An invalid mode has been selected by the user.

Solution: Try the function again and make sure that you select one of the defined modes of operation.

ACE_ERR_METRICS_NOT_ENA

Error Number: -67

Text: Performance Metrics not enabled

Description: The user has attempted to get metrics but metrics have not been enabled.

Solution: Call aceSetMetrics() to enable metrics and then try again.

ACE_ERR_REG_ACCESS

Error Number: -80

Text: Unable to open Win32 registry key

Description: The library and Enhanced Mini-ACE driver were not able to access a required key in the Win32 registry. This may be due to an incorrect or incomplete installation of the Enhanced Mini-ACE.

Solution: Reinstall Enhanced Mini-ACE or complete installation.

ACE_ERR_INVALID_CARD

Error Number: -81

Text: Not a valid Mini-ACE Card

Description: The device selected is not a valid device.

Solution: Correct the device number allocation or select a different DevNum.

ACE_ERR_DRIVER_OPEN

Error Number: -82

Text: Unable to open device driver

Description: This error is usually due to inappropriate resource allocation for the device.

Solution: Reinstall or correct device memory and interrupt allocation for system.

ACE_ERR_MAPMEN_ACC

Error Number: -83

Text: Unable to access mapped memory

Description: This error is usually due to inappropriate mapped resource in DOS or VxWorks operating systems.

Solution: Reinstall or correct mapped memory allocation for system.

ACE_ERR_NODE_NOT_FOUND

Error Number: -100

Text: Element not found

Description: The application requested that a data block be used that cannot be located.

Solution: Verify that the requested data block was previously created.

ACE_ERR_NODE_MEMBLOCK

Error Number: -101

Text: Element not a memory block

Description: The application requested that a data block be used that is not in a valid memory block.

Solution: Verify that the requested data block was previously created. If it was, destroy it and create it again.

ACE_ERR_NODE_EXISTS

Error Number: -102

Text: Element is already defined

Description: The function requested that a data block be created that has already been previously defined.

Solution: Verify that the requested data block was previously created. If it was, destroy it and create it again or change the ID number on the new data block.

ACE_ERR_MEMMGR_FAIL

Error Number: -150

Text: Not enough memory on device

Description: An error has occurred in the creation or use of a block of memory. This might be due to incorrect parameters of the function list (e.g. size or boundary conditions).

Solution: Verify that the parameters for the function are correct.

ACE_ERR_TEST_BADSTRUCT

Error Number: -200

Text: Invalid test result structure

Description: An invalid test result structure has been specified as an input to a function.

Solution: Verify that the parameters for the function are correct. Reinstall Enhanced Mini-ACE software if the problem persists.

ACE_ERR_TEST_FILE

Error Number: -201

Text: Invalid file

Description: An invalid or Null test file has been specified.

Solution: Verify that the parameters for the function are correct. Reinstall Enhanced Mini-ACE software if the problem persists.

ACE_ERR_MT_BUFTYPE

Error Number: -300

Text: Not a valid MT buffering mode

Description: This error is returned if the application requests the use of a Monitor buffer type that is inconsistent with the mode of operation or if the type does not exist.

Solution: Verify the monitor buffer type being requested is valid for the mode of operation.

ACE_ERR_MT_CMDSTK

Error Number: -301

Text: Not a valid MT command stack size

Description: The requested monitor command stack has an incorrect size.

Solution: Verify the requested stack size is correct.

ACE_ERR_MT_DATASTK

Error Number: -302

Text: Not a valid MT data stack size

Description: The requested monitor data stack is being created with an incorrect size.

Solution: Verify that the requested stack size is correct.

ACE_ERR_MT_FILTER_RT

Error Number: -303

Text: Not a valid RT address

Description: The specified RT address is outside of the valid range.

Solution: Verify that the RT address is correct.

ACE_ERR_MT_FILTER_TR

Error Number: -304

Text: Not a valid T/R bit

Description: The specified T/R bit is invalid.

Solution: Verify that the T/R bit is correct.

ACE_ERR_MT_FILTER_SA

Error Number: -305

Text: Not a valid subaddress buffer

Description: The subaddress output parameter pointer is NULL.

Solution: Recreate the pointer and make sure it is valid.

ACE_ERR_MT_STKLOC

Error Number: -306

Text: Invalid MT stack location

Description: The stack location is not valid.

Solution: Please make sure the location is one of the following values: ACE_MT_STKLOC_ACTIVE

ACE_MT_STKLOC_INACTIVE

ACE_MT_STKLOC_STKA

ACE_MT_STKLOC_STKB.

ACE_ERR_MT_MSGLOC

Error Number: -307

Text: Invalid MT message location

Description: The message location is not valid.

Solution: Please make sure the location is one of the following values: ACE_RT_MSGLOC_NEXT_PURGE

ACE_RT_MSGLOC_NEXT_NPURGE

ACE_RT_MSGLOC_LATEST_PURGE

ACE_RT_MSGLOC_LATEST_NPURGE.

ACE_ERR_MT_HBUFSIZE

Error Number: -308

Text: Not a valid host buffer size

Description: The host buffer must be created with the correct size. (see the aceMTInstallHBuf function definition).

Solution: Create a host buffer with the correct size.

ACE_ERR_MT_HBUF

Error Number: -309

Text: Host buffer is not allocated

Description: The host buffer is not created.

Solution: Create a host buffer.

ACE_ERR_RT_DBLK_EXISTS

Error Number: -400

Text: RT data block already defined

Description: The defined data block ID already exists.

Solution: Create the data block with a different ID.

ACE_ERR_RT_DBLK_ALLOC

Error Number: -401

Text: RT data block already defined

Description: The defined data block ID already exists.

Solution: Create the data block with a different ID.

ACE_ERR_RT_DBLK_MAPPED

Error Number: -402

Text: RT data block is currently linked to a subaddress

Description: The defined data block ID is already mapped to a subaddress.

Solution: Unmap the data block and try again or choose another data block.

ACE_ERR_RT_DBLK_NOT_CB

Error Number: -403

Text: RT data block is not a circular buffer

Description: The specified data block is not a circular buffer.

Solution: Choose another data block.

ACE_ERR_RT_HBUF

Error Number: -410

Text: RT host buffer is not allocated

Description: The host buffer has not been allocated.

Solution: Install the host buffer and try the operation again.

ACE_ERR_BC_DBLK_EXISTS

Error Number: -500

Text: BC data block already defined

Description: The defined data block ID already exists.

Solution: Create the data block with a different ID.

ACE_ERR_BC_DBLK_ALLOC

Error Number: -501

Text: BC data block already defined

Description: The defined data block ID already exists.

Solution: Create the data block with a different ID.

ACE_ERR_BC_DBLK_SIZE

Error Number: -502

Text: Invalid BC data block size

Description: The defined data block size is not valid.

Solution: Create the data block with a proper size.

ACE_ERR_ UNRES_DATABLK

Error Number: -503

Text: BC data block not defined

Description: The specified data block has not been previously defined.

Solution: Define the data block.

ACE_ERR_ UNRES_MSGBLK

Error Number: -504

Text: BC message block not defined

Description: The specified message block has not been previously defined.

Solution: Define the message block.

ACE_ERR_ UNRES_FRAME

Error Number: -505

Text: BC frame block not defined

Description: The specified frame block has not been previously defined.

Solution: Define the frame block.

ACE_ERR_ UNRES_OPCODE

Error Number: -506

Text: BC opcode block not defined

Description: The specified opcode has not been previously defined.

Solution: Define the opcode.

ACE_ERR_ UNRES_JUMP

Error Number: -507

Text: Jump (JMP) address is out of frame range

Description: The jump operation you are trying to perform is not valid.

Solution: Verify that the jump parameter is correct and try again.

ACE_ERR_ FRAME_NOT_MAJOR

Error Number: -508

Text: Selected frame is not a major frame

Description: This is not a valid minor frame.

Solution: Verify that the frame is correct or define major frame and try this operation again.

ACE_ERR_ HBUFSIZE

Error Number: -600

Text: Host buffer size is incompatible

Description: The size of the host buffer is not valid.

Solution: Please specify a valid size for the host buffer and try this operation again.

ACE_ERR_ HBUF

Error Number: -601

Text: Host buffer is not allocated

Description: The host buffer has not been installed.

Solution: Please install the host buffer and try this operation again.

ACE_ERR_TOO_MANY_DEVS

Error Number: -700

Text: Too many devices allocated

Description: This error occurs in VxWorks if too many devices are allocated for use.

Solution: Please reset the system and allocate less.

All unknown errors

Unknown error number

An error value was returned that has no defined meaning.

Contact DDC if this error occurs. Please have as much information regarding the error and the circumstances that led to the error.

APPENDIX B

The three errors described in this appendix affect only RT and RT/Monitor modes of operation. The BC and Monitor (only) modes are not affected. Occurrences of these problems are very rare.

One error affects the first data word transmitted by the RT, while the other two errors affect the RT's internal operations following receipt of specific mode code messages. These errors will only occur if ***all*** of the following configurations and conditions are in effect:

- The 1553 terminal is operating in RT mode.

AND

- One or more of certain specific non-message interrupts are enabled by the respective Interrupt Mask Register bit. The library functions that can set these interrupt conditions are aceSetIrqConditions(), aceRTInstallHBuf(), and aceRTMTInstallHBuf(). Please note that the use of a host buffer enables the time tag interrupt to periodically transfer messages and data from the card's memory to your host buffer. These functions will return the ACE_WRN_RT_CFG_INVALID warning back to the user and will continue. The Enhanced Mini-ACE's non-message interrupts that are applicable are TIME TAG ROLLOVER, RT ADDRESS PARITY ERROR, and RAM PARITY ERROR.

AND

- The interrupt status queue is enabled (i.e., bit 6 of Configuration Register #6 is logic "1" by calling aceISQEnable(TRUE)). In RT mode or RTMT mode the interrupt status queue is disabled by default.

AND, either:

- {The RT receives a non-mode code transmit command, or mode code transmit command involving a memory read operation for the transmitted data word. Receive commands to the RT are **not** affected.

  AND

- The internal write transfer to the interrupt status queue resulting from a non-message interrupt begins on a particular *single clock cycle* relative to the first data word read cycle; i.e., the read transfer from shared RAM and ensuing write to the internal Manchester encoder register.}

OR…

- {For the case of a Transmit vector word mode command, CLEAR SERVICE REQUEST, bit 2 of Configuration Register #2, is programmed to logic "1". For the case of a Synchronize (without data) mode command, CLEAR TIME TAG ON SYNCHRONIZE, bit 6 of Configuration Register #2, is logic “1”.

AND

- The RT receives a Transmit vector word mode command (with CLEAR SERVICE REQUEST = logic “1”) or Synchronize (without data) mode command (with CLEAR TIME TAG ON SYNCHRONIZE = logic “1”. All other mode code commands are **not** affected.}

AND

- The internal write transfer to the interrupt status queue resulting from a non-message interrupt begins on a particular *six clock cycle* window just prior to the start of the PCI Enhanced Mini-ACE’s RT EOM (end of message) sequence.

If **all** of the above conditions/events occur simultaneously, then it is possible for one of the following three errors to occur:

(1) The value of the first data word transmitted by the RT may be equal to the RT status word, rather than the word read from memory. The second and subsequent words data transmitted by the RT will all be the *correct* values, as read from RAM.
(2) The RT’s Service request status word bit may fail to automatically clear following receipt of a Transmit vector word mode command.
(3) The RT’s time tag register may fail to automatically clear following receipt of a Synchronize (without data) mode command. Note however that for the case of a TIME TAG ROLLOVER interrupt and a Synchronize (without data) mode command, the issuance of the interrupt is indicative that the time tag counter has correctly reset *regardless of whether or not the error has occurred*.

Note that given the conditions above, occurrences of these error conditions will be probabilistic in nature, with a mean time between occurrences that’s a function of clock input frequency, time tag counter resolution, and the rate of transmit messages processed. For the case of an error in the first transmitted data word, the mean time between errors is given by the following equation:

$$T_{ERR} = \frac{65{,}536 \bullet f_{CLK} \bullet R}{N}$$

For the case of the Service request status word bit or the Time Tag counter failing to clear, the mean time between errors is given by the following equation:

$$T_{ERR} = \frac{13107.2 \bullet f_{CLK} \bullet R}{N}$$

where (for either equation):

T_{ERR} = average time between errors, in seconds
f_{CLK} = input clock frequency, in Megahertz (16 Megahertz for DDC cards)
R = time tag resolution, in µS/LSB (default of 2 µS/LSB)
N = number of *transmit* messages per second processed by the RT

For example, for an input clock frequency of 16 MHz, and a programmed time tag resolution of 64 µS/LSB, an error in the first transmitted word will occur (on average) once in every 2^{26} = 67,108,864 transmit messages. For an RT responding to 100 transmit messages per second, this equates to (on average) one error every 671,089 seconds, or one error every 186 hours (7.8 days). Similarly, for an RT responding to 10 Transmit vector word or 10 Synchronize (without data) mode code messages per second, this equates to (on average) one error every 1,342,177 seconds, or one error every 373 hours (15.5 days). Note that in either case, the error is *highly likely* to be corrected following the RT's next reception of a Transmit vector word or Synchronize (without data) mode command.

APPENDIX C

Integrity Release Information

The following section contains information specific to the Enhanced Mini-ACE library release for the Integrity O/S (BU-69090S5). For additional information please view the “README” file included with the release.

For information regarding revision history please view the “Version.txt” file included with the release.

Interrupt Processing

Interrupt servicing is accomplished by spawning a secondary task from the main user application task that will block until an interrupt has been issued. Upon blocking, a library level interrupt service routine will be called and an optional user specified routine may be called.

Package / Controller Adaptation

The distribution is organized such that a separate source module exists for each specific controller board type for the purpose of maintaining a generic card interface layer. All controller specific calls will exist in these modules. The appropriate driver module must be compiled into the kernel.

RTL Installation and Usage

To install the BU-69090S5 on your system perform the following steps:

1) Unzip the emace_int.zip file to your system.

2) Create a new Kernel build for your board type.

3) In the kernel build include the EmacePciDriver.o file from your installation directory. i.e. C:\69090S5\Drivers\<YOUR BSP>\EmacePciDriver.o

4) Rebuild your kernel and either flash it and ROM Boot or Network boot. (Reference Green Hills Integrity documentation on how to boot your new kernel).

5) Open the default.bld file located in your installation directory.

6) Rebuild the samples using the provided build specifications.

The sample programs are now ready to be downloaded to the target for execution.

For your own applications be sure to include/link in the emace_lib.a file located in your installation directory in the emace_lib path.

INDEX