Bus Management for Windows Programmer's Reference Guide

RadiSys® Corporation

15025 S.W. Koll Parkway

Beaverton, OR 97006

Phone: (503) 646-1800

FAX: (503) 646-1850

07-0222-01 December 1994

EPC and RadiSys are registered trademarks and EPConnect is a trademark of RadiSys Corporation.

Microsoft and MS-DOS are registered trademarks of Microsoft Corporation and Windows is a trademark of Microsoft Corporation.

IBM and PC/AT are trademarks of International Business Machines Corporation.

May 1994

Copyright © 1994 by RadiSys Corporation

All rights reserved.

Software License and Warranty

YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE OPENING THE DISKETTE OR DISK UNIT PACKAGE. BY OPENING THE PACKAGE, YOU INDICATE THAT YOU ACCEPT THESE TERMS AND CONDITIONS. IF YOU DO NOT AGREE WITH THESE TERMS AND CONDITIONS, YOU SHOULD PROMPTLY RETURN THE UNOPENED PACKAGE, AND YOU WILL BE REFUNDED.

LICENSE

You may:

1. Use the product on a single computer;
2. Copy the product into any machine-readable or printed form for backup or modification purposes in support of your use of the product on a single computer;
3. Modify the product or merge it into another program for your use on the single computer—any portion of this product merged into another program will continue to be subject to the terms and conditions of this agreement;
4. Transfer the product and license to another party if the other party agrees to accept the terms and conditions of this agreement—if you transfer the product, you must at the same time either transfer all copies whether in printed or machine-readable form to the same party or destroy any copy not transferred, including all modified versions and portions of the product contained in or merged into other programs.

You must reproduce and include the copyright notice on any copy, modification, or portion merged into another program.

YOU MAY NOT USE, COPY, MODIFY, OR TRANSFER THE PRODUCT OR ANY COPY, MODIFICATION, OR MERGED PORTION, IN WHOLE OR IN PART, EXCEPT AS EXPRESSLY PROVIDED FOR IN THIS LICENSE.

IF YOU TRANSFER POSSESSION OF ANY COPY, MODIFICATION, OR MERGED PORTION OF THE PRODUCT TO ANOTHER PARTY, YOUR LICENSE IS AUTOMATICALLY TERMINATED.

TERM

The license is effective until terminated. You may terminate it at any time by destroying the product and all copies, modifications, and merged portions in any form. The license will also terminate upon conditions set forth elsewhere in this agreement or if you fail to comply with any of the terms or conditions of this agreement. You agree upon such termination to destroy the product and all copies, modifications, and merged portions in any form.

LIMITED WARRANTY

RadiSys Corporation ("RadiSys") warrants that the product will perform in substantial compliance with the documentation provided. However, RadiSys does not warrant that the functions contained in the product will meet your requirements or that the operation of the product will be uninterrupted or error-free.

RadiSys warrants the diskette(s) on which the product is furnished to be free of defects in materials and workmanship under normal use for a period of ninety (90) days from the date of shipment to you.

LIMITATIONS OF REMEDIES

RadiSys' entire liability shall be the replacement of any diskette that does not meet RadiSys' limited warranty (above) and that is returned to RadiSys.

IN NO EVENT WILL RADISYS BE LIABLE FOR ANY DAMAGES, INCLUDING LOST PROFITS OR SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF OR INABILITY TO USE THE PRODUCT EVEN IF RADISYS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

GENERAL

You may not sublicense the product or assign or transfer the license, except as expressly provided for in this agreement. Any attempt to otherwise sublicense, assign, or transfer any of the rights, duties, or obligations hereunder is void.

This agreement will be governed by the laws of the state of Oregon.

If you have any questions regarding this agreement, please contact RadiSys by writing to RadiSys Corporation, 15025 SW Koll Parkway, Beaverton, Oregon 97006.

YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT, AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU FURTHER AGREE THAT IT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF THE AGREEMENT BETWEEN US WHICH SUPERSEDES ANY PROPOSAL OR PRIOR AGREEMENT, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATION BETWEEN US RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

NOTES

Table of Contents

NOTES

1. Introducing Bus Management for Windows

This manual is intended for programmers using the Bus Management for Windows programming interface to develop enhanced mode Windows applications that control I/O modules via the VXI expansion interface on an EPC or a VXLink card. You are expected to have read the *EPConnect/VXI for DOS & Windows User's Guide* for an understanding of what is in EPConnect/VXI, how to install it, and how to use the Start-Up Resource Manager (SURM). You are not expected to have in-depth knowledge of Windows.

Bus Management for Windows is designed to execute under enhanced mode Windows only. It will not execute properly under Windows standard mode. It is also designed to execute on EPC-7 hardware or better. It will not execute properly on an EPC-2.

The Bus Management for Windows API provides a powerful interface for interacting with the VXI expansion interface on an EPC or a VXLink card. The Bus Management API offers considerable flexibility by providing a C/C++ dynamic link library (DLL) interface that obeys the MS Pascal binding conventions. By observing the same conventions, you can use EPConnect with other languages, such as Visual Basic.

This chapter introduces you to the RadiSys® Bus Management for Windows product. In it you will find the following:

- What is in this manual and how to use it
- What is Bus Management for Windows?
- Programming, Compiling and Linking
- What to do next

1.1 How This Manual is Organized

This manual has five chapters:

Chapter 1, *Introduction*, introduces Bus Management for Windows and this manual.

Chapter 2, *Function Descriptions*, describes the major categories of functions and gives complete descriptions of each function. The function descriptions also contain supporting examples or references to an example that demonstrates use of the function. Function descriptions are alphabetic by function name.

Chapter 3, *Advanced Topics*, provides information about byte swapping, interrupts, and the backward-compatibility library.

Chapter 4, *Support and Service*, describes how to contact RadiSys Technical Support for support and service.

1.2 What is Bus Management for Windows?

Bus Management for Windows consists of those portions of the EPConnect software package that are required by C/C++ and Visual BASIC programmers developing VXI control applications that execute in enhanced mode Windows. Figure 1-1 is a diagram of the Bus Management for Windows software architecture that shows how the architecture relates to the VXIbus hardware.

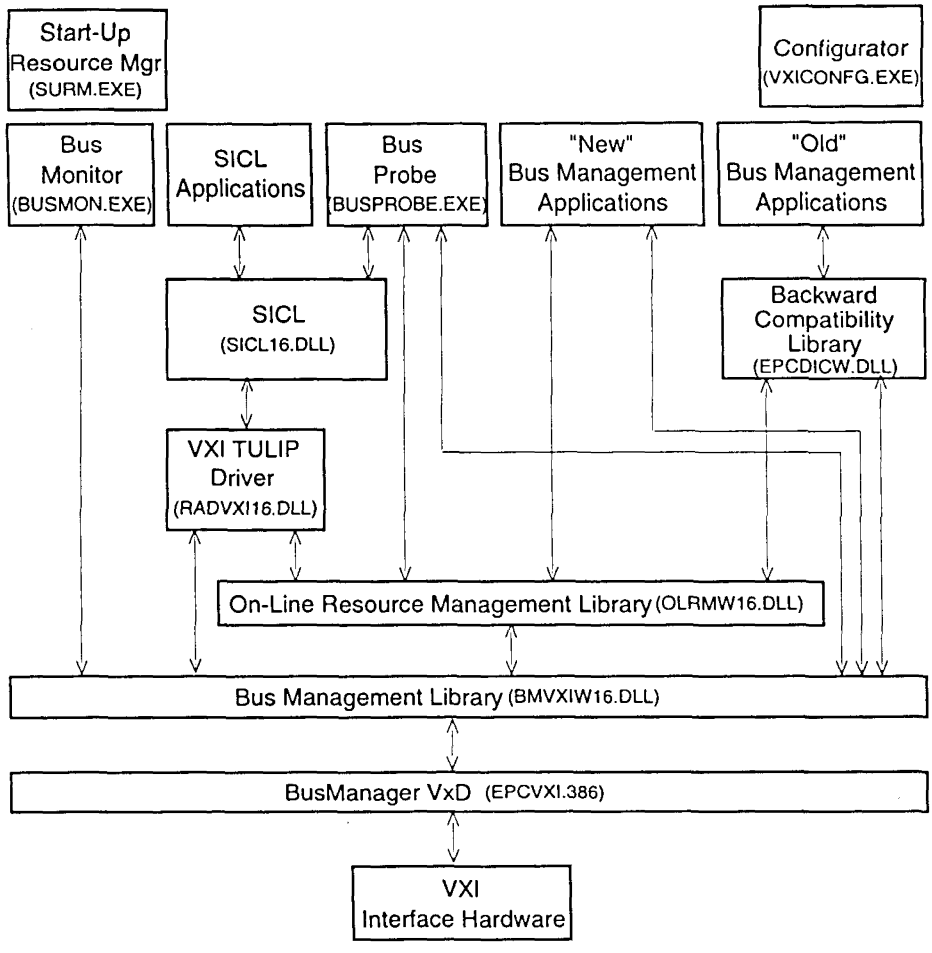

Figure 1-1. EPConnect/VXI for Windows Software Architecture

1.2.1 Bus Management Library and BusManager VxD

The Bus Management library and BusManager VxD are at the foundation of EPConnect. They provide the lowest level interface to the VXIbus hardware through their function libraries. These functions allow you to:

- Control VXIbus word serial registers.
- Send word serial commands of all sizes.
- Transfer blocks of data to and from VXIbus devices, with BERR detection.
- Control EPC Slave memory
- Query EPC driver, firmware, and hardware version or type.

The Bus Management DLL supports ANSI "C" compilers such as Microsoft C/C++ and Borland C/C++, as well as Visual Basic.

The Bus Management Library is fully re-entrant.

1.2.2 OLRM

The On-Line Resource Management library (**OLRMW16.DLL**) provides user applications with access to results of the resource management process, as well as retrieving status information from devices over the VXI bus. A C/C++ language interface is provided to access OLRM functions. OLRM accesses the VXIbus through the Bus Management library and the BusManager VxD.

1.2.3 Backward-Compatibility Library

The Backward-Compatibility Library (**EPCDICW.DLL**) maintains compatibility for applications that were developed using previous versions of EPConnect. It provides a C/C++ interface that is compatible with the Bus Management for DOS API.

1.2.4 SURM

The Start-Up Resource Manager (SURM) determines the physical content of the system and configures the devices. It is typically the first program to run after DOS boots. The SURM is the EPConnect implementation of the resource manager defined in the VXIbus specification. However, SURM extends the specification definition to include non-VXIbus devices, such as GPIB instruments. The SURM uses the **DEVICES** file to obtain device information not directly available from the devices. SURM accesses VXIbus devices in the system directly.

1.3 Programming, Compiling and Linking

This section contains information about programming with Bus Management for Windows. Included is a list of the header files provided, the programming interfaces, and compiling and linking hints.

1.3.1 Header Files

Bus Management for Windows provides the following header files:

BMVXI.BAS A Microsoft Visual Basic (Professional Edition) header file containing constant definitions and function declarations.

BUSMGR.H A "C" header file containing the constant definitions, macro definitions, and function prototypes required to compile EPConnect applications using any Microsoft or Borland "C" or C++ compiler.

BUSMGR.INC A copy of **BUSMGR.H** that's been converted so that it is suitable for inclusion into an assembly language source file.

EPC_OBM.H A "C" header file containing the constant definitions, macro definitions, structure definitions, and function prototypes required to compile Bus Management applications for DOS. This header file also provides backward compatibility for Bus Management for Windows applications written for releases preceding revision 4.0.

EPC_OBM.H should never be included in a source file directly. **BUSMGR.H** includes **EPC_OBM.H**.

EPC_OBM.INC A copy of **EPC_OBM.H** that has been converted so that it is suitable for inclusion into an assembly language source file.

EPC_OBM.INC should never be included in a source file directly. **BUSMGR.INC** includes **EPC_OBM.INC.**

EPCSTD.H A "C" header file containing macro definitions to standardize non-ANSI, compiler-dependent keywords. By using the macros defined here, an application can compile successfully using any revision of Microsoft or Borland "C" or C++ compiler without modifying the source file.

EPCSTD.H should never be included in a source file directly. **BUSMGR.H** includes **EPC_OBM.H.**

EPCSTD.INC A copy of **EPCSTD.H** that has been converted so that it is suitable for inclusion into an assembly language source file.

EPCSTD.INC should never be included in a source file directly. **BUSMGR.INC** includes **EPCSTD.INC.**

OLRM.H A "C" header file containing the constant definitions, macro definitions, and function prototypes required to compile OLRM applications using any Microsoft or Borland C/C++ compiler.

OLRM.INC A copy of **OLRM.H** that has been converted so that it is suitable for inclusion into an assembly language source file.

OBS_OLRM.H A "C" header file containing the constant definitions, macro definitions, and function prototypes required to compile OLRM applications for DOS. This header file also provides backward-compatibility for Bus Management for Windows applications written for releases preceding revision 4.0.

OBS_OLRM.H should never be included in a source file directly. **OLRM.H** includes **OBS_OLRM.H.**

OBS_OLRM.INC
A copy of **OBS_OLRM.H** that has been converted so that it is suitable for inclusion into an assembly language source file.

VMEREGS.H A "C" header file containing constant and macro definitions for accessing the EPC VMEbus control registers.

VMEREGS.INC
A copy of **VMEREGS.H** that has been converted so that it is suitable for inclusion into an assembly language source file.

All Bus Management for Windows header files contain an **#if/#endif** pair surrounding the contents of the header file so that the file can be included multiple times without causing compiler errors.

All Bus Management for Windows "C" header files also contain **extern "C"{}** bracketing for C++ compilers. Because **extern "C"** is strictly a C++ keyword, it is also bracketed and only visible when compiling under C++ and not standard "C."

1.3.2 Programming Interface

Bus Management for Windows functions are accessible through interfaces for "C" and Visual BASIC languages. The following table shows the interface libraries and header files for each of the language interfaces.

Language	*Library files*	*Header files*
MS "C"	**BMVXIW16.LIB** **OLRMW16.LIB**	**BUSMGR.H** **OLRM.H**
Borland "C"	**BMVXIW16.LIB** **OLRMW16.LIB**	**BUSMGR.H** **OLRM.H**
Visual Basic	**BMVXIW16.LIB** **OLRMW16.LIB**	**BMVXI.BAS** **OLRM.BAS**

The use of these files is discussed in the following sections.

Calling Bus Management for Windows from MS "C" and QuickC

The "C" language interface is designed to work with Version 5.1 and later versions of the Microsoft "C" compiler and libraries. The libraries are created for the large memory model (far code and far data). This is sufficient for linking programs of any model size, due to the prototyping of all library functions in the header files. The include files provide strong type checking and convert near code and data to far code and data for programs using the small (near code and near data), compact (near code and far data), or medium (far code and near data) memory models.

Calling Bus Management for Windows from Borland C

Bus Management for Windows was designed to work with "C" compilers adhering to the Microsoft "C" calling conventions. Both Microsoft and Borland "C" compilers work equally well.

Calling Bus Management for Windows from Visual Basic

Calling Bus Management for Windows functions from Visual Basic requires using the **BMVXI.BAS** and **OLRM.BAS** header files in your project.

To compile and link a program once your project is built, choose "Make .EXE File" from the File Menu.

For more information about calling "C" DLLs from Visual Basic, refer to the *Microsoft Visual Basic Programming System for Windows Programmer's Guide*.

1.3.3 Compiling and Linking Applications

NOTE: For specific compiler and/or linker options, refer to your vendor's documentation.

The following examples assume that EPConnect software has been installed in the **C:\EPCONNEC** directory.

When compiling applications, ensure that the Bus Management for Windows header files are in the compiler search path by doing one of the following:

1. Specify the entire header file pathname when including the header file in the source file.

2. Specify **C:\EPCONNEC\INCLUDE** as part of the header file search path parameter on the compiler invocation line.

3. Specify **C:\EPCONNEC\INCLUDE** as part of the header file search path environment variable.

Also, ensure that Bus Management for Windows libraries are in the linker search path by doing one of the following:

1. Specify the entire library pathname when linking object files.

2. Specify **C:\EPCONNEC\LIB** as part of the linker library search path.

1.4 What to do Next

To begin using Bus Management for Windows:

1. If Bus Management for Windows is not pre-installed on your system, install and configure it using the procedures in Chapter 2 of the *EPConnect/VXI for DOS & Windows User's Guide.*

2. Refer to the function descriptions in Chapter 2 of this manual for details about a function and/or its parameters to develop applications.

3. Refer to the sample code included with the Bus Management for Windows software under the **C:\EPCONNEC\SAMPLES\BUSMGR.W31** directory.

NOTES

2. Function Descriptions

2

This chapter lists the Bus Management for Windows library functions by category and by name. It is for the programmer who needs a particular fact, such as what function performs a specific task or what a function's arguments are.

The first section lists the functions categorically by the task each performs. It also gives you a brief description of what each function does. The second section lists the functions alphabetically and describes each function in detail.

2.1 Functions by Category

The categorical listing provides an overview of the operations performed by the EPConnect functions. Included with each category is a description of the operations performed, a listing of the functions in the category, and a brief description of each function.

The categories of the Bus Management for Windows library functions include:

- Environment
- Bus Sessions
- Locking
- Memory Mapping
- Byte Order
- Events
- EPC Configuration
- Bus Control Lines
- Watchdog Timer
- Commander Support
- Servant Support

2.1.1 Environment

Bus Management for Windows provides support that allows an application to query and verify the state of its environment.

The Bus Management for Windows library supplies two functions for environment support:

Function	Description
EpcGetErrorString	Queries a null-terminated string corresponding to an EPConnect return value.
EpcVerifyEnvironment	Verifies and queries the EPConnect environment.

2.1.2 Sessions

Bus Management for Windows provides support for multiple simultaneous *sessions.* A session encapsulates shared operating system and interface hardware state in an environment where multiple applications may be accessing the interface hardware simultaneously.

Each session contains a set of attributes that define how resources are managed. Session attributes include:

- A locking timeout. A locking timeout defines how long the session will wait for shared EPC hardware to become unlocked. See the **Locking** section for more information.

- A list of memory mappings. A memory mapping defines where in the EPC's address space an access takes place and how data is accessed. See the **Memory Mapping** section for more information.

- An enabled event mask attribute. The enabled event mask attribute defines the set of events that the session can receive when each of the events' corresponding interrupt or error occurs. See the **Events** section for more information.

- An event handler attribute array. The event handler attribute array defines the functions that are called when the session receives events. The session maintains one entry in the event handler attribute array for each possible event. See the **Events** section for more information.

The Bus Management for Windows library supplies the following functions in support of sessions:

Function	Description
EpcCloseSession	Destroys an open session.
EpcGetSessionData	Queries a session's application-specified data.
EpcOpenSession	Creates a session
EpcSetSessionData	Defines a session's application-specified data.

To use session functionality, an application must first call **EpcOpenSession** to create a session. Once a session exists, an application can access and manage the bus using any of the remaining Bus Management for Windows library functions. The application can define and query application-specific data using **EpcSetSessionData** and **EpcGetSessionData**. When the application is finished with a session, it should call **EpcCloseSession** to destroy the session. Failing to destroy an existing session before an application terminates may result in the loss of both virtual and physical resources.

2.1.3 Locking

Bus Management for Windows provides support for *locking*. Locking gives a session exclusive access to shared interface hardware. Locking is used in multithreaded environments to prevent simultaneous, potentially conflicting hardware manipulation.

Locks can be nested. Bus Management for Windows maintains a global lock counter. At most one session may "own" the lock counter. Initially, the lock counter is zero, indicating that no session has locked the shared interface hardware. Locking acquires and increments the lock counter for a session. Unlocking decrements the lock counter for the same session. A non-zero lock counter indicates that shared interface hardware is locked.

When an application calls a Bus Management for Windows library function that obeys the locking paradigm, the function checks for an existing lock. If no lock exists or the specified session "owns" the lock, the function proceeds. Otherwise, the function suspends execution until the lock is released or the specified session's locking timeout expires. If the existing lock is not released before the specified session's locking timeout expires, the function returns **EPC_LOCKED**.

The Bus Management for Windows library supplies the following functions in support of locking:

Function	Description
EpcGetLockingTimeout	Queries a session's locking timeout.
EpcLockSession	Locks shared interface hardware for a session.
EpcSetLockingTimeout	Defines a session's locking timeout.
EpcUnlockSession	Unlocks shared interface hardware for a session.

To use locking functionality, an application must first call **EpcOpenSession** to create a session. Once a session exists, an application can lock and unlock shared interface hardware for the session using **EpcLockSession** and **EpcUnlockSession**, respectively. When the application completes executing locked operations, it should unlock the session. Failing to unlock a session before an application terminates, either explicitly using **EpcUnlockSession** or implicitly using **EpcCloseSession**, may prevent other applications from accessing shared interface hardware.

An application can also query and define the session's locking timeout using **EpcGetLockingTimeout** and **EpcSetLockingTimeout**.

Only Bus Management for Windows library functions that may make conflicting hardware accesses obey the locking paradigm:

EpcAssertInterrupt	**EpcSetEpcLines**
EpcCmdReceiveWSBuffer	**EpcSetEpcMODID**
EpcCmdSendWSBuffer	**EpcSetEpcTriggers**
EpcCmdSendWSCommand	**EpcSetMiscAttributes**
EpcDeassertInterrupt	**EpcSetSlaveMapping**
EpcLockSession	**EpcSetULA**
EpcMapBusMemory	**EpcSrvEnableWsCommand**
EpcMapEpcTriggers	**EpcSrvReceiveWSCommand**
EpcMapSharedMemory	**EpcSrvSendProtocolEvent**
EpcPulseEpcLines	**EpcSrvSendWSProtocolError**
EpcPulseEpcTriggers	**EpcSrvSendWSResponse**
EpcSetBusAttributes	**EpcValidateBusMapping**

Note that the ability to directly map bus memory allows an application to circumvent the locking protections provided in Bus Management for Windows for VXIbus word serial, byte transfer, and event protocols. Each application is responsible for ensuring that it obeys all bus protocols when accessing bus memory directly.

Locking is not a substitute for a sound shared memory protocol. Locking does not protect against multiple processors making simultaneous accesses to the same memory.

2.1.4 Memory Mapping

EPConnect provides support for *memory mappings.* A memory mapping defines where in the interface's physical address space a mapped access takes place and how data is accessed. Each session contains a list of memory mappings.

A memory mapping can map either bus memory or shared memory. Bus memory is VMEbus memory accessed using the interface's VMEbus hardware. Shared memory is an area of local memory that has a fixed size and a fixed physical location and is accessible via the VMEbus, thereby making it suitable for implementing shared memory communication protocols in a multiple processor system.

Memory Mapping Attributes

Each memory mapping contains a set of attributes that define where and how memory is accessed. Memory mapping attributes include:

- An address modifier attribute. The address modifier attribute defines whether the memory mapping maps to bus memory or shared memory. If the memory mapping maps to bus memory, the address modifier attribute also defines the mapping's VMEbus address space and VMEbus access mode.

- A byte ordering attribute. The byte ordering attribute defines whether Motorola or Intel byte ordering is assumed when the memory mapping is used to access data in widths greater than 8 bits. For a bus memory mapping, the byte ordering attribute specifies either Motorola or Intel byte ordering. For a shared memory mapping, the byte ordering attribute always specifies Intel byte ordering.

- A base address attribute. The base address attribute defines where the memory mapping begins. For a bus memory mapping, the base address attribute is an address in one of the VMEbus address spaces. For a shared memory mapping, the base address attribute is an address in the local address space.

- A size attribute. The size attribute defines the extent of a memory mapping, in bytes.

- A type attribute. The type attribute defines whether a mapping is a shared memory mapping, a bus memory mapping that uses statically configured bus window hardware, or a bus memory mapping that uses dynamically configured bus window hardware.

  Statically configured bus window hardware corresponds to a fixed address modifier, byte ordering, and bus address range. A bus memory mapping that uses statically configured bus window hardware can access mapped bus memory at will.

  Dynamically configured bus window hardware corresponds to a variable address modifier, byte ordering, and bus address ramge. A bus memory mapping that uses dynamically configured bus window hardware must configure its bus window hardware before accessing mapped bus memory.

The EPConnect Bus Management Library supplies the following functions in support of memory mappings:

Function	Description
EpcCopyData	Copy a block of data from consecutive memory locations to consecutive memory locations.
EpcGetMappingAttributes	Query a memory mapping's attributes.
EpcMapBusMemory	Create a bus memory mapping using a statically configured bus window.
EpcMapBusMemoryExt	Create a bus memory mapping using a dynamically configured bus window.
EpcMapSharedMemory	Create a shared memory mapping.
EpcPopData	Pop a block of data from a single memory location to consecutive memory locations.
EpcPushData	Push a block of data from consecutive memory locations to a single memory location.
EpcUnmapBusMemory	Destroy a bus memory mapping.
EpcUnmapSharedMemory	Destroy a shared memory mapping.
EpcValidateBusMapping	Validate a bus memory mapping that uses a dynamically configured bus window.

2.1.4 Memory Mapping

To use memory mapping functionality, an application must first call **EpcOpenSession** to open a bus session and either **EpcMapBusMemory, EpcMapBusMemoryExt,** or **EpcMapSharedMemory** to create a memory mapping. Once a memory mapping exists, an application can access the mapped memory either directly or by using **EpcCopyData, EpcPopData,** or **EpcPushData.** When the application is finished with a memory mapping, it should call either **EpcUnmapBusMemory** or **EpcUnmapSharedMemory** to destroy the mapping. Failing to destroy an existing memory mapping before an application terminates, either explicitly using **EpcUnmapBusMemory** or **EpcUnmapSharedMemory** or implicitly using **EpcCloseSession,** may result in the loss of both virtual and physical resources.

Direct access of mapped memory provides the maximum possible data transfer performance, but it does not automatically detect and handle misaligned data, potential bus errors, or hardware restrictions. Direct access requires that the application guarantee data alignment and bus error avoidance. Direct access using a bus memory mapping created with **EpcMapBusMemoryExt** requires using **EpcValidateBusMapping** before an access to insure that the dynamically configured bus window references the desired bus memory. Finally, direct access using a bus memory mapping created with **EpcMapBusMemoryExt** in a preemptively scheduled environment requires using **EpcLockSession** before **EpcValidateBusMapping** and **EpcUnlockSession** after the direct access to ensure that the dynamically configured bus window is not reconfigured by another thread during the direct access. Note that **EpcCopyData, EpcPopData,** and **EpcPushData** copy blocks of data while taking hardware restrictions, data alignment, and potential bus error considerations into account.

An application can use **EpcGetMappingAttributes** to query an existing memory mapping's attributes.

2.1.5 Byte Order

The Bus Management for Windows library provides support for converting data between Intel and Motorola byte order through byte-swapping.

The Bus Management for Windows library supplies the following functions in support of byte order conversion:

Function	Description
EpcSwapBuffer	Byte-swaps a buffer of values.
EpcSwap16	Byte-swaps a 16-bit value.
EpcSwap32	Byte-swaps a 32-bit value.
EpcSwap48	Byte-swaps a 48-bit value.
EpcSwap64	Byte-swaps a 64-bit value.
EpcSwap80	Byte-swaps an 80-bit value.

2.1.6 Events

EPConnect provides support for *events.* An event is an interrupt or error that occurs asynchronously with respect to normal program execution.

Each session contains a set of event attributes that define how the session handles events. Event attributes include:

- An enabled event mask attribute. The enabled event mask attribute defines the set of events that the session can receive when each of the events' corresponding interrupt or error occurs.

- An array of event handlers. The event handler array defines the functions that are called when the session receives events. The session maintains one entry in the event handler array for each possible event.

2.1.6 Events

The EPConnect Bus Management Library supplies the following functions in support of events:

Function	Description
EpcGetEventEnableMask	Queries a session's enabled event mask attribute.
EpcGetEventHandler	Queries an entry in a session's event handler array.
EpcSetEventEnableMask	Defines a session's enabled event mask attribute.
EpcSetEventHandler	Defines an entry in a session's event handler array.
EpcWaitForEvent	Waits for an event to occur.

To use event functionality, an application must first call **EpcOpenSession** to create a session. Once a session exists, an application can either wait for the desired events to occur using **EpcWaitForEvent** or it can define handlers for the desired events using **EpcSetEventHandler**. In either case, the application must enable reception of the events using **EpcSetEventEnableMask** to receive them.

When the application is finished receiving events, it should disable reception of the events. Failing to disable reception of events before an application terminates, either explicitly using **EpcSetEventEnableMask** or implicitly using **EpcCloseSession**, may result in a system crash the next time an event occurs.

An application can use **EpcGetEventEnableMask** and **EpcGetEventHandler** to query a session's current event attributes.

The Bus Management for Windows library supports all possible VXIbus events. In practice, however, event support is limited by the underlying interface hardware.

The table below describes the events:

Event	Description
EPC_MSG_INT	Message interrupt (EPC-7 and EPC-8 only)
EPC_VME1_INT	VMEbus interrupt 1
...	
EPC_VME7_INT	VMEbus interrupt 7
EPC_SIGNAL_INT	VXIbus signal FIFO interrupt
EPC_TTL_TRIG0_INT	VXIbus TTL Trigger 0 interrupt (EPC-7 only)
...	
EPC_TTL_TRIG7_INT	VXIbus TTL Trigger 7 interrupt (EPC-7 only)
EPC_SYSRESET_ERR	VMEbus SYSRESET error
EPC_ACFAIL_ERR	VMEbus power failure error
EPC_BERR_ERR	VMEbus access error
EPC_SYSFAIL_ERR	VMEbus SYSFAIL error
EPC_WATCHDOG_ERR	Watchdog timer expiration error (EPC-7 and EPC-8 only)
EPC_EXT_TRIG0_INT	External Trigger 0 interrupt (VXLink only)
EPC_EXT_TRIG1_INT	External Trigger 1 interrupt (VXLink only)

2.1.7 EPC Configuration

Bus Management for Windows provides support for maintaining global interface configuration attributes. The values of global interface configuration attributes affect all the behavior of the interface hardware for all sessions.

The Bus Management for Windows library supplies the following functions in support of interface configuration:

Function	Description
EpcGetBusAttributes	Queries the interface's bus management attributes.
EpcGetMiscAttributes	Queries the interface's miscellaneous configuration attributes.
EpcGetSlaveMapping	Queries the interface's slave memory mapping.
EpcGetULA	Queries the interface's unique logical address.
EpcSetBusAttributes	Defines the interface's bus management attributes.
EpcSetMiscAttributes	Defines the interface's miscellaneous configuration attributes.
EpcSetSlaveMapping	Defines the interface's slave memory mapping.
EpcSetULA	Defines the interface's unique logical address.

To use interface configuration functionality, an application must first call **EpcOpenSession** to create a session. Once a session exists, an application can define the global interface configuration attributes using **EpcSetBusAttributes**, **EpcSetSlaveMapping**, and **EpcSetULA**. An application can query the global interface configuration attributes using **EpcGetBusAttributes**, **EpcGetSlaveMapping**, and **EpcGetULA**.

2.1.8 Bus Lines

Bus Management for Windows provides support for defining, querying, and pulsing the interface line state. It also provides support for monitoring actual bus line state.

In general, interface line state reflects the state of bits in the interface's line drive registers, while actual bus line state is an OR'd combination of the states of all devices on the bus. If the interface asserts a line, the actual bus line transitions from deasserted to asserted only if all other devices on the bus have previously deasserted the line. Likewise, if the interface deasserts a line, the actual bus line transitions from asserted to deasserted only if all devices on the bus have previously deasserted the line.

The Bus Management for Windows library supplies the following functions in support of bus lines:

Function	Description
EpcAssertInterrupt	Asserts a VME interrupt.
EpcDeassertInterrupt	Deasserts a VME interrupt.
EpcGetBusInterrupts	Queries actual bus VME interrupt line state.
EpcGetBusLines	Queries actual bus control line state.
EpcGetBusMODID	Queries actual bus MODID line state.
EpcGetBusTriggers	Queries actual bus trigger line state.
EpcGetEpcInterrupt	Queries interface VME interrupt assertion state.
EpcGetEpcLines	Queries interface control line state.
EpcGetEpcMODID	Queries interface MODID line state.
EpcGetEpcTriggers	Queries interface trigger line state.
EpcGetEpcTriggerMapping	Queries an interface trigger line mapping.
EpcMapEpcTriggers	Maps one interface trigger line to another.
EpcPulseEpcLines	Pulses interface control lines.
EpcPulseEpcTriggers	Pulses interface trigger lines.
EpcSetEpcLines	Defines the interface control line state.

EpcSetEpcMODID	Defines the interface MODID line state.
EpcSetEpcTriggers	Defines the interface trigger line state.

To use bus control line functionality, an application must first call **EpcOpenSession** to create a session. Once a session exists, an application can define interface line state using **EpcAssertInterrupt**, **EpcDeassertInterrupt**, **EpcSetEpcLines**, **EpcSetEpcMODID**, or **EpcSetEpcTriggers**. An application can pulse interface lines using **EpcPulseEpcLines** or **EpcPulseEpcTriggers**. To query the interface line state, an application can use **EpcGetEpcInterrupt**, **EpcGetEpcLines**, **EpcGetEpcMODID**, **EpcGetEpcTriggers**, or **EpcGetEpcTriggerMapping**. To query actual bus line state, the application can use **EpcGetBusInterrupts**, **EpcGetBusLines**, **EpcGetBusMODID**, or **EpcGetBusTriggers**.

2.1.9 Watchdog Timer

Bus Management for Windows provides watchdog timer services that allow an application to prevent interface lock-up under extraordinary circumstances.

If an EPC's watchdog timer is not reset within the current watchdog timer period, either a system reset occurs or a watchdog timer error event occurs. In the latter case, an application can enable the event and install an event handler to gracefully handle the error.

The EPC's watchdog timer is typically reset in sections of code that execute frequently and/or execute at regular time intervals.

The Bus Management for Windows library supplies a single function in support of the watchdog timer:

Function	Description
EpcWatchdogTimer	Modifies EPC watchdog timer configuration.

To use watchdog timer functionality, an application must first call **EpcOpenSession** to create a session. Once a session exists, an application can configure the EPC's watchdog timer using **EpcWatchdogTimer**.

An EPC's watchdog timer is a shared EPC hardware resource. However, Bus Management for Windows provides no functionality for controlling shared access to the watchdog timer. Multiple applications may simultaneously use watchdog timer functionality. However, EPConnect software cannot guarantee the result.

The purpose of the watchdog timer hardware is to allow an application to prevent EPC lock-up under extraordinary circumstances. Placing additional layers of software between an application and the watchdog timer hardware to control sharing of the resource would necessarily restrict an application's access to the watchdog timer, thereby violating its original purpose.

By default, an EPC is configured to use a long watchdog timer period and to generate a watchdog error event upon expiration. Assuming that no application attempts to modify these default watchdog timer settings, any number of applications may use the watchdog timer simultaneously.

VXLink does not contain a watchdog timer. This function is valid only on an EPC-7 and EPC-8.

2.1.10 Commander Support

EPConnect provides support for using an EPC or VXlink card as a commander device in a VXIbus system.

The Bus Management for Windows library supplies the following functions in support of using an EPC as a commander device:

Function	Description
EpcCmdReceiveWSBuffer	Receives a buffer of data from a servant device.
EpcCmdSendWSBuffer	Sends a buffer of data to a servant device.
EpcCmdSendWSCommand	Sends a word serial command to a servant device.

To use commander functionality, an application must first call **EpcOpenSession** to create a session. Once a session exists, an application can send 16-bit, 32-bit, or 48-bit word serial commands and receive responses using **EpcCmdSendWSCommand**. To quickly send multiple data bytes to a servant device, an application should use **EpcCmdSendWSBuffer**. To quickly receive multiple data bytes from a servant device, an application should use **EpcCmdReceiveWSBuffer**.

2.1.11 Servant Support

Bus Management for Windows provides support for using an EPC as a servant device in a VMEbus system.

The Bus Management for Windows library supplies the following functions in support of using an EPC as a servant device:

Function	Description
EpcSrvEnableWSCommand	Enables word serial command reception.
EpcSrvReceiveWSCommand	Receives a word serial command from the commander device.
EpcSrvSendProtocolEvent	Sends a protocol event to the commander device.
EpcSrvSendWSProtocolError	Sends a word serial protocol error to the commander device.
EpcSrvSendWSResponse	Sends a word serial command response to the commander device.

To use servant functionality, an application must first call **EpcOpenSession** to create a session. Once a session exists, an application can receive 16-bit or 32-bit word serial commands using **EpcSrvEnableWSCommand** and **EpcSrvReceiveWSCommand** and send responses to received word serial commands using **EpcSrvSendWSResponse**. An application can use **EpcSrvSendProtocolEvent** to send events and/or responses to a commander device via the commander device's signal register.

Servant functionality is supported on an EPC-7 and EPC-8 only. A VXLink interface does not support servant functionality.

2.2 Functions By Name

This section contains an alphabetical listing of the Bus Management for Windows library functions. Each listing describes the function, gives its invocation sequence and arguments, discusses its operation, and lists its returned values. Where usage of the function may not be clear, an example with comments is given.

EpcAssertInterrupt

Description Asserts a VME interrupt.

C Synopsis

#include "busmgr.h"

short FAR PASCAL
EpcAssertInterrupt(unsigned long *Session_ID*, unsigned long *Event_Mask*);

Session_ID	*Session_ID* specifies a session.
Event_Mask	*Event_Mask* specifies a VME interrupt.

Visual Basic Synopsis

Declare Function
EpcAssertInterrupt% Lib "bmvxiw16.dll" (ByVal *Session_ID&*, ByVal *Event_Mask&*)

Remarks **EpcAssertInterrupt** causes the interface to assert a VME interrupt.

Event_Mask specifies the VME interrupt to assert. Valid values are:

Event_Mask	Description
EPC_VME1_INT	VMEbus interrupt 1.
...	
EPC_VME7_INT	VMEbus interrupt 7.

An interface acts as both a D08(O) and a D16 interrupter. For D08 interrupt acknowledge cycles, the interface uses its unique logical address as the status/ID value. For D16 interrupt acknowledge cycles, the interface uses the upper 8 bits of its response register for the upper 8 bits of the status/ID value and its unique logical address as the lower 8 bits of the status/ID value.

Return Value The function returns a Bus Management return value:

EPC_INV_ASSERT	The interface is already asserting a VMEbus interrupt.
EPC_INV_MASK	The parameter *Event_Mask* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present or there is a revision mismatch between the Bus Management Library and the BusManager VxD.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_SUCCESS	The function completed successfully.

See Also **EpcDeassertInterrupt, EpcGetEpcInterrupt, EpcSrvSendProtocolEvent.**

Example

```
/*
 * Copyright 1994 by RadiSys Corporation.  All rights reserved.
 */

/*
 * buslines.c -- Bus Management Library interface bus line functions sample code.
 */

#include "busmgr.h"

/*
 * FUNCTION PROTOTYPES...
 */

short FAR
BusLinesSample(void);

int FAR
WinPrintf(char FAR *Format_Ptr, ...);

/*
 * CODE...
 */

short FAR
BusLinesSample(void)
{
    char                err_string[ERROR_STRING_SZ];
    short               err_num;
    unsigned long       bus_int_mask;
    unsigned long       bus_line_mask;
    unsigned long       epc_int_mask;
```

```
    unsigned long         epc_line_mask;
    unsigned long         session_id;
    struct EpcEnvironment environment;

    /*
     * Verify the interfaceonnect environment.
     */

    if ((err_num = EpcVerifyEnvironment(&environment)) != EPC_SUCCESS)
    {
        EpcGetErrorString(err_num, err_string);
        WinPrintf("FAILURE:  EpcVerifyEnvironment() error == %s (%d).\n",
                  err_string,
                  err_num);
        return (err_num);
    }

    /*
     * Open a session.
     */

    if ((err_num = EpcOpenSession(&session_id)) != EPC_SUCCESS)
    {
        EpcGetErrorString(err_num, err_string);
        WinPrintf("FAILURE:  EpcOpenSession() error == %s (%d).\n",
                  err_string,
                  err_num);
        return (err_num);
    }

    /*
     * Assert VMEbus interrupt #1, query bus and interface VMEbus interrupt
       * assertions, deassert VMEbus interrupt #1, and query bus and interface
VMEbus
     * interrupt assertions again.
     *
     */

    EpcAssertInterrupt(session_id, EPC_VME1_INT);
    EpcGetBusInterrupts(session_id, &bus_int_mask);
    EpcGetEpcInterrupt(session_id, &epc_int_mask);
    WinPrintf("VMEbus interrupt 1 asserted.\n");
        WinPrintf("Bus  interrupt  mask  =  0x%08lX,  interface  interrupt  mask  =
0x%08lX\n",
                bus_int_mask,
                epc_int_mask);
    EpcDeassertInterrupt(session_id);
    EpcGetBusInterrupts(session_id, &bus_int_mask);
    EpcGetEpcInterrupt(session_id, &epc_int_mask);
    WinPrintf("VMEbus interrupt 1 deasserted.\n");
        WinPrintf("Bus  interrupt  mask  =  0x%08lX,  interface  interrupt  mask  =
0x%08lX\n",
                bus_int_mask,
                epc_int_mask);

    /*
     * Assert SYSFAIL, query bus and interface line assertions, pulse SYSFAIL, and
     * query bus and interface line assertions again.
     */

    EpcSetEpcLines(session_id, EPC_SYSFAIL);
    EpcGetBusLines(session_id, &bus_line_mask);
    EpcGetEpcLines(session_id, &epc_line_mask);
    WinPrintf("SYSFAIL asserted.\n");
    WinPrintf("Bus line mask = 0x%08lX, interface line mask = 0x%08lX\n",
```

```
            bus_line_mask,
            epc_line_mask);
    EpcPulseEpcLines(session_id, EPC_SYSFAIL);
    WinPrintf("SYSFAIL pulsed.\n");
    WinPrintf("Bus line mask = 0x%08lX, interface line mask = 0x%08lX\n",
            bus_line_mask,
            epc_line_mask);

    /*
     * Close the session and return.
     */

    EpcCloseSession(session_id);
    WinPrintf("SUCCESS:  BusLinesSample() complete.\n");
    return (EPC_SUCCESS);
}
```

EpcCloseSession

Description Destroys an open session.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcCloseSession(unsigned long Session_ID);
```

Session_ID — *Session_ID* specifies an open session.

Visual Basic Synopsis

```
Declare Function
EpcCloseSession% Lib "bmvxiw16.dll" (ByVal Session_ID&)
```

Remarks **EpcCloseSession** closes an open session.

If the specified session has locked shared interface hardware, the hardware is unlocked before the function destroys the session.

If the specified session has one or more enabled events, the events are disabled before the function destroys the session. Also, all of the session's defined event handlers are removed.

If the specified session contains one or more memory mappings, the mappings are destroyed before the function destroys the session.

Return Value The function returns a Bus Management return value:

EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present or there is a revision mismatch between the Bus Management Library and the BusManager VxD.
EPC_INV_USAGE	The session specified by *Session_ID* is currently in use by another thread.
EPC_SUCCESS	The function completed successfully.

See Also **EpcOpenSession, EpcSetEventEnableMask, EpcUnlockSession, EpcUnmapBusMemory, EpcUnmapSharedMemory.**

Example

```
/*
 * Copyright 1994 by RadiSys Corporation.  All rights reserved.
 */

/*
 * sessions.c -- Bus Management Library session functions sample code.
 */

#include "busmgr.h"

/*
 * FUNCTION PROTOTYPES...
 */

short FAR
SessionsSample(void);

int FAR
WinPrintf(char FAR *Format_Ptr, ...);

/*
 * CODE...
 */

short FAR
SessionsSample(void)
{
    char                    err_string[ERROR_STRING_SZ];
    short                   err_num;
    unsigned long           session_data;
    unsigned long           session_id;
    struct EpcEnvironment environment;

    /*
     * Verify the EPConnect environment.
     */

    if ((err_num = EpcVerifyEnvironment(&environment)) != EPC_SUCCESS)
```

```
    {
        EpcGetErrorString(err_num, err_string);
        WinPrintf("FAILURE:  EpcVerifyEnvironment() error == %s (%d).\n",
                  err_string,
                  err_num);
        return (err_num);
    }

    /**
     **Open a session.
     */

    if ((err_num = EpcOpenSession(&session_id)) != EPC_SUCCESS)
    {
        EpcGetErrorString(err_num, err_string);
        WinPrintf("FAILURE:  EpcOpenSession() error == %s (%d).\n",
                  err_string,
                  err_num);
        return (err_num);
    }

    /*
     **Define the session's application-specific data.
     *?

    EpcSetSessionData(session_id, session_id);

    /**
     **Query the session's application-specific data.
     */

    EpcGetSessionData(session_id, &session_data);

    /*
     **Close the session and return.
     *?

    EpcCloseSession(session_id);
    WinPrintf("SUCCESS:  SessionsSample() complete.\n");
    return (EPC_SUCCESS);
}
```

EpcCmdReceiveWSBuffer

Description Receives a buffer of data from a servant device.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcCmdReceiveWSBuffer
        (unsigned long          Session_ID,
        unsigned short          ULA,
        unsigned char FAR *     Buffer_Ptr,
        unsigned long           Buffer_Size,
        short                   Term_Character,
        unsigned long           Timeout,
        unsigned long FAR *     Term_Reason_Ptr,
        unsigned long FAR *     Receive_Size_Ptr);
```

Session_ID	*Session_ID* specifies a session.
ULA	*ULA* specifies a servant device's unique logical address.
Buffer_Ptr	*Buffer_Ptr* specifies the location of a buffer where the received data will be placed.
Buffer_Size	*Buffer_Size* specifies the size of the buffer where the received data will be placed.
Term_Character	*Term_Character* specifies a termination character for the receive operation.
Timeout	*Timeout* specifies the number of milliseconds to wait while receiving a buffer of data.

Term_Reason_Ptr	*Term_Reason_Ptr* specifies a location where a bit mask defining the reason(s) for terminating the receive operation will be placed.
Receive_Size_Ptr	*Receive_Size_Ptr* specifies a location where the actual number of bytes received will be placed.

Visual Basic Synopsis

Declare Function
EpcCmdReceiveWSBuffer% Lib "bmvxiw16.dll"

(ByVal *Session_ID&*,
ByVal *ULA%*,
ByVal *Buffer_Ptr$*,
ByVal *Buffer_Size&*,
ByVal *Term_Character%*,
ByVal *Timeout&*,
Term_Reason_Ptr&,
Receive_Size_Ptr&)

Remarks

EpcCmdReceiveWSBuffer receives up to *Buffer_Size* bytes of data from the servant device specified by *ULA* and places them in the buffer pointed to by *Buffer_Ptr*.

Term_Character specifies an optional termination character for the receive operation. Valid termination character values are -1 and 0 through 255. A termination character value of -1 specifies that no termination character is defined. A termination character value of 0 through 255 specifies a termination character. If the function detects a termination character while it's receiving data, it places the termination character in the buffer and returns **EPC_SUCCESS**.

If *Term_Reason_Ptr* is non-null and the function returns **EPC_SUCCESS**, the location pointed by *Term_Reason_Ptr* contains a bit mask defining the reason(s) for terminating the receive operation. The bit mask is an OR'd combination of the following constants:

Constant	Description
EPC_TERM_CHAR	The function detected the specified termination character.
EPC_TERM_EOI	The function received a data byte with the EOI indicator set.
EPC_TERM_FULL	The specified buffer is full.

The value of the location pointed to by *Term_Reason_Ptr* is undefined when the function does not return **EPC_SUCCESS.**

If *Receive_Size_Ptr* is non-null, the location it points to always contains the number of data bytes actually received.

If the function detects a word serial protocol error while receiving data, it returns **EPC_WS_PROTOCOL.** To determine the protocol error, use **EpcCmdSendWSCommand** to send a READ PROTOCOL ERROR word serial command to the servant device and receive its response.

EpcCmdReceiveWSBuffer is intended for use by a commander device to quickly receive multiple data bytes from one of its servants via word serial commands. A servant device should use **EpcSrvReceiveWSCommand** to receive a word serial command from its commander and **EpcSrvSendWSResponse** to send a word serial command response to its commander.

Return Value

The function returns a Bus Management return value:

EPC_INV_PTR	The parameter *Buffer_Ptr* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present or there is a revision mismatch between the Bus Management Library and the BusManager VxD.
EPC_INV_TERMCHR	The parameter *Term_Character* is invalid.

EPC_INV_ULA	The parameter *ULA* is invalid.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_RECV_BERR	A bus error occurred receiving a word serial command response.
EPC_RECV_TIMEOUT	A timeout occurred receiving a word serial command response.
EPC_SEND_BERR	A bus error occurred sending a word serial command.
EPC_SEND_TIMEOUT	A timeout occurred sending a word serial command.
EPC_SUCCESS	The function completed successfully.
EPC_WS_PROTOCOL	A word serial protocol error occurred.

See Also **EpcCmdSendWSBuffer, EpcCmdSendWSCommand, EpcOpenSession, EpcSrvReceiveWSCommand, EpcSrvSendWSResponse.**

EpcCmdSendWSBuffer

Description Sends a buffer of data to a servant device.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcCmdSendWSBuffer( unsigned long          Session_ID,
                    unsigned short         ULA,
                    unsigned char  FAR *   Buffer_Ptr,
                    unsigned long          Buffer_Size,
                    unsigned short         EOI_Flag,
                    unsigned long          Timeout,
                    unsigned long  FAR *   Send_Size_Ptr);
```

Session_ID	*Session_ID* specifies a session.
ULA	*ULA* specifies a servant device's unique logical address.
Buffer_Ptr	*Buffer_Ptr* specifies the location of a buffer containing the data to be sent.
Buffer_Size	*Buffer_Size* specifies the number of bytes to be sent.
EOI_Flag	*EOI_Flag* specifies whether the EOI indicator should be set when the function sends the last byte from the specified buffer.
Timeout	*Timeout* specifies the number of milliseconds to wait while sending the buffer of data.
Send_Size_Ptr	*Send_Size_Ptr* specifies a location where the actual number of bytes sent will be placed.

Visual Basic Synopsis

```
Declare Function
EpcCmdSendWSBuffer% Lib "bmvxiw16.dll" (  ByVal
Session_ID&,
                                        ByVal ULA%,
                                        ByVal Buffer_Ptr$,
                                        ByVal Buffer_Size&,
                                        ByVal EOI_Flag%,
                                        ByVal Timeout&,
                                        Send_Size_Ptr&)
```

Remarks

EpcCmdSendWSBuffer sends up to *Buffer_Size* bytes of data from the buffer pointed to by *Buffer_Ptr* to the servant device specified by *ULA*.

EOI_Flag specifies whether the EOI indicator should be set when the function sends the last byte from the specified buffer. A non-zero *EOI_Flag* value causes the function to set the EOI indicator when it sends the last byte from the buffer. A zero *EOI_Flag* value causes the function to not set the EOI indicator when it sends the last byte from the specified buffer.

If *Send_Size_Ptr* is non-null, the location it points to always contains the number of data bytes actually sent.

If the function detects a word serial protocol error while sending data, it returns **EPC_WS_PROTOCOL**. To determine the protocol error, use **EpcCmdSendWSCommand** to send a READ PROTOCOL ERROR word serial command to the servant device and receive its response.

EpcCmdSendWSBuffer is intended for use by a commander device to quickly send multiple data bytes to one of its servants via word serial commands. A servant device should use **EpcSrvReceiveWSCommand** to receive a word serial command from its commander and **EpcSrvSendWSResponse** to send a word serial command response to its commander.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	The parameter *Buffer_Ptr* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present or there is a revision mismatch between the Bus Management Library and the BusManager VxD.
EPC_INV_ULA	The parameter *ULA* is invalid.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_SEND_BERR	A bus error occurred sending a word serial command.
EPC_SEND_TIMEOUT	A timeout occurred sending a word serial command.
EPC_SUCCESS	The function completed successfully.
EPC_WS_PROTOCOL	A word serial protocol error occurred.

See Also **EpcCmdReceiveWSBuffer, EpcCmdSendWSCommand, EpcOpenSession, EpcSrvReceiveWSCommand, EpcSrvSendWSResponse.**

EpcCmdSendWSCommand

Description Sends a word serial command to a servant device.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcCmdSendWSCommand( unsigned long  Session_ID,
                     unsigned short ULA,
                     void  FAR *    Command_Ptr,
                     void  FAR *    Response_Ptr,
                     unsigned short Size,
                     unsigned long  Timeout);
```

Session_ID	*Session_ID* specifies a session.
ULA	*ULA* specifies a servant device's unique logical address.
Command_Ptr	*Command_Ptr* specifies the location of a word serial command.
Response_Ptr	*Response_Ptr* specifies a location where the response to the word serial command will be placed.
Size	*Size* specifies the size of both the word serial command and the optional word serial command response.
Timeout	*Timeout* specifies the number of milliseconds to wait while sending the word serial command and receiving the word serial command response.

Visual Basic Synopsis

```
Declare Function
EpcCmdSendWSCommand% Lib "bmvxiw16.dll" (ByVal
Session_ID&,
                                    ByVal ULA%,
                                    Command_Ptr As Any,
                                    Response_Ptr As Any,
                                    ByVal Size%,
                                    ByVal Timeout&)
```

Remarks

EpcCmdSendWSCommand optionally sends a word serial command, then optionally receives a word serial command response. If *Command_Ptr* is not null, the function sends the word serial command at the location pointed to by *Command_Ptr* to the servant device specified by *ULA*. Otherwise, the function skips sending a command. If *Response_Ptr* is not null, the function then receives a word serial command response from the servant device specified by *ULA* and places it in the location pointed to by *Response_Ptr*. Otherwise, the function returns without attempting to receive a response.

Size specifies the size of both the word serial command and the word serial command response:

Size	Description
EPC_16_BIT	Send a 16-bit word serial command and receive a 16-bit word serial command response.
EPC_32_BIT	Send a 32-bit long word serial command and receive a 32-bit long word serial command response.
EPC_48_BIT	Send a 48-bit extended long word serial command and receive a 32-bit long word serial command response.

If the function detects a word serial protocol error while sending a command or receiving a response, it returns EPC_WS_PROTOCOL. To determine the protocol error, use **EpcCmdSendWSCommand** to send a READ PROTOCOL ERROR word serial command to the servant device and receive its response.

EpcCmdSendWSCommand is intended for use by a commander device to send a word serial command to one of its servants and/or to receive a word serial command response from one of its servants. A servant device should use **EpcSrvReceiveWSCommand** to receive a word serial command from its commander and **EpcSrvSendWSResponse** to send a word serial command response to its commander.

Return Value The function returns a Bus Management return value:

EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SIZE	The parameter *Size* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_INV_ULA	The parameter *ULA* is invalid.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_RECV_BERR	A bus error occurred receiving the word serial command response.
EPC_RECV_TIMEOUT	A timeout occurred receiving the word serial command response.
EPC_SEND_BERR	A bus error occurred sending the word serial command.
EPC_SEND_TIMEOUT	A timeout occurred sending the word serial command.
EPC_SUCCESS	The function completed successfully.
EPC_WS_PROTOCOL	A word serial protocol error occurred.

See Also EpcOpenSession, EpcSrvReceiveWSCommand, EpcSrvSendWSResponse.

EpcCopyData

Description Copies a block of data.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcCopyData( unsigned long        Session_ID,
             void HUGE *          Source_Ptr,
             void HUGE *          Dest_Ptr,
             unsigned long        Size,
             unsigned short       Data_Width,
             unsigned long FAR *  Actual_Size_Ptr);
```

Session_ID	*Session_ID* specifies a bus session.
Source_Ptr	*Source_Ptr* specifies the address of a data buffer from which data will be copied.
Dest_Ptr	*Dest_Ptr* specifies the address of a data buffer into which data will be copied.
Size	*Size* specifies the number of data bytes to copy.
Data_Width	*Data_Width* specifies the number of data bits to copy per bus access.
Actual_Size_Ptr	*Actual_Size_Ptr* specifies a location where the actual number of bytes copied will be placed.

Visual Basic Synopses

```
Declare Function
BasicCopyEpcToVME% Lib "bmvxiw16.dll" ( ByVal
Session_ID&,
                    Source_Ptr As Any,
                    ByVal Dest_Ptr As Any,
                    ByVal Size&,
                    ByVal Data_Width%,
                    Actual_Size_Ptr&)

Declare Function
BasicCopyVMEToEpc% Lib "bmvxiw16.dll" ( ByVal
Session_ID&,
                    ByVal Source_Ptr As Any,
                    Dest_Ptr As Any,
                    ByVal Size&,
                    ByVal Data_Width%,
                    Actual_Size_Ptr&)

Declare Function
BasicCopyVMEToVME% Lib "bmvxiw16.dll"

                    (ByVal Session_ID&,
                    ByVal Source_Ptr As Any,
                    ByVal Dest_Ptr As Any,
                    ByVal Size&,
                    ByVal Data_Width%,
                    Actual_Size_Ptr&)
```

Remarks

EpcCopyData efficiently copies blocks of data from consecutive memory locations to consecutive memory locations using the attributes of pointers *Source_Ptr* and *Dest_Ptr*. The intended use of the function is copying large blocks of data to or from consecutive bus locations.

The *Size* parameter should always express the number of bytes to be copied, regardless of the specified *Data_Width* parameter. Passing a zero *Size* parameter results in no data being copied.

The following constants define valid values for the *Data_Width* parameter:

Constant	Description
EPC_8_BIT	8-bit data width
EPC_8_BIT_ODD	8-bit data width, odd bytes only
EPC_16_BIT	16-bit data width
EPC_32_BIT	32-bit data width
EPC_FASTCOPY	To increase copy performance, don't check for intermediate bus errors. This constant cannot be used alone; it must be OR'd with one of the preceding constants.

The function returns the actual number of bytes copied in the location pointed to by *Actual_Size_Ptr*.

The function operates correctly using both unmapped pointers and memory mapped pointers for *Source_Ptr* and *Dest_Ptr*. EPC-to-EPC, EPC-to-VME, VME-to-EPC, and VME-to-VME copies all execute properly.

For a 16-bit or 32-bit copy to complete, no individual data element may span a segment boundary. Otherwise, the function returns an **EPC_INV_ALIGN** error. For example, if *Data_Width* is **EPC_16_BIT** and *Size* is greater than 64 Kbytes, both *Source_Ptr* and *Dest_Ptr* must be aligned on a 16-bit boundary for the copy operation to complete successfully

For a VME-to-VME copy to complete, both *Source_Ptr* and *Dest_Ptr* must correspond to VMEbus addresses aligned on an address boundary equivalent to the specified *Data_Width*. Otherwise, the function returns an **EPC_INV_ALIGN** error. For example, if both *Source_Ptr* and *Dest_Ptr* correspond to VMEbus memory and *Data_Width* is **EPC_16_BIT**, then both *Source_Ptr* and *Dest_Ptr* must correspond to VMEbus addresses aligned on a 16-bit boundary for the copy to complete successfully.

For EPC-to-VME, VME-to-EPC, and VME-to-VME copies to complete when hardware byte-swapping occurs, *Size* must be a multiple of the specified *Data_Width* and all VMEbus addresses must be aligned on an address boundary equivalent to the specified *Data_Width*. Otherwise, the function returns an **EPC_INV_SWAP** error. For example, if *Source_Ptr* corresponds to EPC memory, *Dest_Ptr* corresponds to VMEbus memory, and *Data_Width* is **EPC_16_BIT**, *Size* must be a multiple of two and *Dest_Ptr* must correspond to a VMEbus address aligned on a 16-bit boundary for the copy to complete successfully.

To ensure that all accesses are the specified *Data_Width*, the function handles non-aligned leading and trailing bytes as a special case. When transferring data from a non-aligned address, the function reads the nearest aligned chunk and extracts the non-aligned bytes. When transferring data to a non-aligned address, the function reads the nearest aligned chunk, copies the non-aligned bytes into the chunk, and replaces the chunk. Note that, for VMEbus transfers, this read-modify-write algorithm is executed in software -- it is not a read-modify-write bus cycle.

Return Value The function returns a Bus Management return value:

EPC_BERR	A bus error occurred during the copy.
EPC_INV_ALIGN	A 16-bit or 32-bit data element spans a segment boundary or both *Source_Ptr* and *Dest_Ptr* are mapped to VMEbus addresses and they are not aligned on equivalent VMEbus address boundaries.
EPC_INV_PTR	One or more of *Source_Ptr*, *Dest_Ptr*, or *Actual_Size_Ptr* is invalid.
EPC_INV_RANGE	The address range defined by *Source_Ptr* and *Size* and/or the address range defined by *Dest_Ptr* and *Size* contains bus addresses that are not currently mapped.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_INV_SWAP	*Source_Ptr* and/or *Dest_Ptr* are mapped to the VMEbus so that hardware byte-swapping will occur, but *Size* is not a multiple of *Data_Width* and/or a VMEbus address is misaligned.
EPC_INV_WIDTH	The *Data_Width* parameter is invalid.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_SUCCESS	The function completed successfully.

See Also **EpcGetMappingAttributes, EpcMapBusMemory, EpcMapSharedMemory, EpcOpenSession, EpcUnmapBusMemory, EpcUnmapSharedMemory.**

Example

```
/*
 * Copyright 1994 by RadiSys Corporation.  All rights reserved.
 */

/*
 * mapping.c -- Bus Management Library mapping functions sample code.
 */

#include "busmgr.h"

/*
 * FUNCTION PROTOTYPES...
 */

short FAR
MappingSample(void);

int FAR
WinPrintf(char FAR *Format_Ptr, ...);

/*
 * CODE...
 */

short FAR
MappingSample(void)
{
    char                  err_string[ERROR_STRING_SZ];
    short                 err_num;
    unsigned short        bus_add_mod;
    unsigned short        bus_byte_order;
    unsigned short        ula;
    unsigned long         actual_size;
    unsigned long         bus_base;
    unsigned long         bus_size;
    unsigned long         session_id;
    unsigned long         shared_base;
    unsigned long         shared_size;
    volatile void   HUGE *bus_ptr;
    volatile void   HUGE *shared_ptr;
    struct EpcEnvironment environment;

    /*
     * Verify the EPConnect environment.
     */

    if ((err_num = EpcVerifyEnvironment(&environment)) != EPC_SUCCESS)
    {
        EpcGetErrorString(err_num, err_string);
        WinPrintf("FAILURE:  EpcVerifyEnvironment() error == %s (%d).\n",
                  err_string,
                  err_num);
        return (err_num);
    }

    /*
     * Open a session.
     */

    if ((err_num = EpcOpenSession(&session_id)) != EPC_SUCCESS)
    {
        EpcGetErrorString(err_num, err_string);
        WinPrintf("FAILURE:  EpcOpenSession() error == %s (%d).\n",
```

```
                err_string,
                err_num);
    return (err_num);
}

/*
 * Map all of A16 space using Motorola byte ordering.
 */

if ((err_num = EpcMapBusMemory(session_id,
                               EPC_A16S,
                               EPC_MBO,
                               0x00000000,
                               0x00010000,
                               &bus_ptr)) != EPC_SUCCESS)
{
    EpcCloseSession(session_id);
    EpcGetErrorString(err_num, err_string);
    WinPrintf("FAILURE:  EpcMapBusMemory() error == %s (%d).\n",
                err_string,
                err_num);
    return (err_num);
}

/*
 * Query the bus mapping's attributes.
 */

EpcGetMappingAttributes(session_id,
                        bus_ptr,
                        &bus_add_mod,
                        &bus_byte_order,
                        &bus_base,
                        &bus_size);

/*
 * Map the EPC's shared memory buffer.
 */

if ((err_num = EpcMapSharedMemory(session_id,
                                  &shared_base,
                                  &shared_size,
                                  &shared_ptr)) != EPC_SUCCESS)
{
    EpcCloseSession(session_id);
    EpcGetErrorString(err_num, err_string);
    WinPrintf("FAILURE:  EpcMapSharedMemory() error == %s (%d).\n",
                err_string,
                err_num);
    return (err_num);
}

/*
 * Copy the EPC's A16 registers to the shared memory buffer in Motorola
 * byte order.
 */

EpcGetULA(session_id, &ula);
EpcCopyData(session_id,
            (void HUGE *) ((char HUGE *) bus_ptr + 0xC000 + (ula << 6)),
            (void HUGE *) shared_ptr,
            0x00000040,
            EPC_16_BIT,
            &actual_size);
```

```
    /*
     * Unmap A16 space.
     */

    EpcUnmapBusMemory(session_id, bus_ptr);

    /*
     * Unmap the shared memory buffer.
     */

    EpcUnmapSharedMemory(session_id, shared_ptr);

    /*
     * Close the session and return.
     */

    EpcCloseSession(session_id);
    WinPrintf("SUCCESS:  MappingSample() complete.\n");
    return (EPC_SUCCESS);
}
```

EpcDeassertInterrupt

Description Deasserts a VME interrupt.

C Synopsis

#include "busmgr.h"

short FAR PASCAL
EpcDeassertInterrupt(unsigned long *Session_ID*);

Session_ID *Session_ID* specifies a session.

Visual Basic Synopsis

Declare Function
EpcDeassertInterrupt% Lib "bmvxiw16.dll" (ByVal *Session_ID&*)

Remarks **EpcDeassertInterrupt** deasserts a currently asserted VME interrupt. If the interface is not currently asserting a VME interrupt, the function has no effect.

When an asserted VME interrupt is acknowledged by a device on the bus, it is automatically deasserted. No call to **EpcDeassertInterrupt** is necessary to deassert the VME interrupt.

Warning:

Deasserting a VME interrupt without waiting for interrupt acknowledgment may cause certain hardware configurations to "lock up." Deasserting a VME interrupt should be executed with extreme care.

Return Value The function returns a Bus Management return value:

EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_SUCCESS	The function completed successfully.

See Also **EpcAssertInterrupt, EpcGetEpcInterrupt.**

Example See **EpcAssertInterrupt.**

EpcGetBusAttributes

Description Queries the interface's bus management attributes.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcGetBusAttributes
        (unsigned long          Session_ID,
         unsigned short FAR *   Bus_Enable_Ptr,
         unsigned short FAR *   Bus_Arb_Mode_Ptr,
         unsigned short FAR *   Bus_Arb_Priority_Ptr,
         unsigned short FAR *   Bus_Release_Ptr);
```

Session_ID	*Session_ID* specifies a session.
Bus_Enable_Ptr	*Bus_Enable_Ptr* specifies a location where the interface's bus enable attribute will be placed.
Bus_Arb_Mode_Ptr	*Bus_Arb_Mode_Ptr* specifies a location where the interface's bus arbitration mode attribute will be placed.
Bus_Arb_Priority_Ptr	*Bus_Arb_Priority_Ptr* specifies a location where the interface's bus arbitration priority attribute will be placed.
Bus_Release_Ptr	*Bus_Release_Ptr* specifies a location where the interface's bus release mode attribute will be placed.

Visual Basic Synopsis

```
Declare Function
EpcGetBusAttributes% Lib "bmvxiw16.dll"
        (ByVal Session_ID&,
         Bus_Enable_Ptr%,
         Bus_Arb_Mode_Ptr%,
         Bus_Arb_Priority_Ptr%,
         Bus_Release_Ptr%)
```

Remarks

EpcGetBusAttributes queries the interface's bus management attributes and places them in the locations pointed to by *Bus_Enable_Ptr*, *Bus_Arb_Mode_Ptr*, *Bus_Arb_Priority_Ptr*, and *Bus_Release_Ptr*.

The interface's bus enable attribute defines whether accesses made by the interface reach the bus. Possible values placed at *Bus_Enable_Ptr* are:

**Bus_Enable_Ptr*	Description
EPC_DISABLE_BUS	Disable bus accesses for the interface. (EPC-7 and EPC-8 only)
EPC_ENABLE_BUS	Enable bus accesses for the interface.

The interface's bus arbitration mode defines how the interface arbitrates bus collisions. The value placed at *Bus_Arb_Mode_Ptr* only has meaning if the interface has been designated the VXIbus slot-0 controller. Possible values placed at *Bus_Arb_Mode_Ptr* are:

**Bus_Arb_Mode_Ptr*	Description
EPC_PRIORITY	Priority bus arbitration.
EPC_ROUND_ROBIN	Round-robin bus arbitration.

The interface's bus arbitration priority defines the priority level at which the interface arbitrates for the bus. Possible values placed at *Bus_Arb_Priority_Ptr* are:

**Bus_Arb_Priority_Ptr*	Description
EPC_PRIORITY0	Bus arbitration priority 0.
EPC_PRIORITY1	Bus arbitration priority 1.
EPC_PRIORITY2	Bus arbitration priority 2.
EPC_PRIORITY3	Bus arbitration priority 3.

The interface's bus release mode determines when the interface requests and/or releases the bus. Possible values placed at *Bus_Release_Ptr* are:

**Bus_Release_Ptr*	Description
EPC_ROR	"Release On Request" bus release mode.
EPC_RONR	"Release On No Request" bus release mode.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	One or more of the parameters *Bus_Enable_Ptr*, *Bus_Arb_Mode_Ptr*, *Bus_Arb_Priority_Ptr*, and *Bus_Release_Ptr* is invalid.
EPC_INV_SESSION	The parameter *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_SUCCESS	The function completed successfully.

See Also **EpcOpenSession, EpcSetBusAttributes.**

Example

```
/*
 * Copyright 1994 by RadiSys Corporation.  All rights reserved.
 */

/*
 * epccfg.c -- Bus Management Library interface configuration functions sample
code.
 */

#include "busmgr.h"

/*
 * FUNCTION PROTOTYPES...
 */

short FAR
EpcCfgSample(void);

int FAR
WinPrintf(char FAR *Format_Ptr, ...);

/*
 * CODE...
```

```
 */

short FAR
EpcCfgSample(void)
{
    char                    err_string[ERROR_STRING_SZ];
    short                   err_num;
    unsigned short          bus_enable;
    unsigned short          bus_arb_mode;
    unsigned short          bus_arb_priority;
    unsigned short          bus_release;
    unsigned short          slave_space;
    unsigned short          ula;
    unsigned long           misc_mask;
    unsigned long           session_id;
    unsigned long           slave_base;
    struct EpcEnvironment   environment;

    /*
     * Verify the EPConnect environment.
     */

    if ((err_num = EpcVerifyEnvironment(&environment)) != EPC_SUCCESS)
    {
        EpcGetErrorString(err_num, err_string);
        WinPrintf("FAILURE:  EpcVerifyEnvironment() error == %s (%d).\n",
                  err_string,
                  err_num);
        return (err_num);
    }

    /*
     * Open a session.
     */

    if ((err_num = EpcOpenSession(&session_id)) != EPC_SUCCESS)
    {
        EpcGetErrorString(err_num, err_string);
        WinPrintf("FAILURE:  EpcOpenSession() error == %s (%d).\n",
                  err_string,
                  err_num);
        return (err_num);
    }

    /*
     * Query the interface's current configuration settings.
     */

    EpcGetBusAttributes(session_id,
                        &bus_enable,
                        &bus_arb_mode,
                        &bus_arb_priority,
                        &bus_release);
    EpcGetSlaveMapping(session_id, &slave_space, &slave_base);
    EpcGetULA(session_id, &ula);
    EpcGetMiscAttributes(session_id, &misc_mask);

    /*
     * Define the interface's configuration settings.
     */

    EpcSetBusAttributes(session_id,
                        EPC_ENABLE_BUS,
                        EPC_PRIORITY,
                        EPC_PRIORITY0,
```

```
                        EPC_ROR);
    EpcSetSlaveMapping(session_id, EPC_A24, 0x00400000);
    EpcSetULA(session_id, 0xF8);
    EpcSetMiscAttributes(session_id, EPC_PASS | EPC_READY);

    /*
     * Restore the interface's original configuration settings.
     */

    EpcSetBusAttributes(session_id,
                        bus_enable,
                        bus_arb_mode,
                        bus_arb_priority,
                        bus_release);
    EpcSetSlaveMapping(session_id, slave_space, slave_base);
    EpcSetULA(session_id, ula);
    EpcSetMiscAttributes(session_id, misc_mask);

    /*
     * Close the session and return.
     */

    EpcCloseSession(session_id);
    WinPrintf("SUCCESS:  EpcCfgSample() complete.\n");
    return (EPC_SUCCESS);
}
```

EpcGetBusInterrupts

Description Queries actual bus VME interrupt line state.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcGetBusInterrupts( unsigned long          Session_ID,
                     unsigned long FAR *    Event_Mask_Ptr);
```

Session_ID	*Session_ID* specifies a session.
Event_Mask_Ptr	*Event_Mask_Ptr* specifies a location where the actual bus VME interrupt line state will be placed.

Visual Basic Synopsis

```
Declare Function
EpcGetBusAttributes% Lib "bmvxiw16.dll"
        (ByVal Session_ID&,
        Event_Mask_Ptr&)
```

Remarks **EpcGetBusInterrupts** queries the actual bus VME interrupt line state and places it in the location pointed to by *Event_Mask_Ptr*.

The value pointed to by *Event_Mask_Ptr* is either zero or an OR'd mask of the following constants. A set bit indicates that the corresponding actual bus VME interrupt line is asserted. A clear bit indicates that the corresponding actual bus VME interrupt line is deasserted:

Constant	Description
EPC_VME1_INT	VME interrupt 1.
...	
EPC_VME7_INT	VME interrupt 7.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	The parameter *Event_Mask_Ptr* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_SUCCESS	The function completed successfully.

See Also **EpcAssertInterrupt, EpcDeassertInterrupt, EpcGetEpcInterrupt, EpcOpenSession.**

Example See **EpcAssertInterrupt.**

EpcGetBusLines

Description Queries actual bus control line state.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcGetBusLines( unsigned long          Session_ID,
                unsigned long FAR *    Line_Mask_Ptr);
```

Session_ID	*Session_ID* specifies a session.
Line_Mask_Ptr	*Line_Mask_Ptr* specifies a location where the actual bus control line state will be placed.

Visual Basic Synopsis

```
Declare Function
EpcGetBusLines% Lib "bmvxiw16.dll"
                (ByVal Session_ID&, Line_Mask_Ptr&)
```

Remarks **EpcGetBusLines** queries the actual bus control line state and places it in the location pointed to by *Line_Mask_Ptr*.

The value pointed to by *Line_Mask_Ptr* is either zero or an OR'd mask of the following constants. A set bit indicates that the corresponding actual bus control line is asserted. A clear bit indicates that the corresponding actual bus control line is deasserted:

Constant	Description
EPC_ACFAIL	ACFAIL.
EPC_SYSFAIL	SYSFAIL.
EPC_SYSRESET	SYSRESET. Supported on EPC-7 and VXLink only.

The value pointed to by *Line_Mask_Ptr* reflects the actual bus control line state, not the interface control line state. Use **EpcGetEpcLines** to query the interface control line state.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	The parameter *Line_Mask_Ptr* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_SUCCESS	The function completed successfully.

See Also **EpcGetEpcLines, EpcOpenSession, EpcPulseEpcLines, EpcSetEpcLines.**

Example See **EpcAssertInterrupt.**

EpcGetBusMODID

Description Queries the actual bus MODID line state.

C Synopsis

```
#include "busmgr.h"

short
EpcGetBusMODID( unsigned long          Session_ID,
                unsigned long FAR *    MODID_Mask_Ptr);
```

Session_ID	*Session_ID* specifies a session.
MODID_Mask_Ptr	*MODID_Mask_Ptr* specifies a location where the actual bus MODID line state will be placed.

Visual Basic Synopsis

```
Declare Function
EpcGetBusMODID%Lib"bmvxiw16.dll" (ByVal
Session_ID&,MODID_Mask_Ptr&)
```

Remarks **EpcGetBusMODID** queries the actual bus MODID line state and places it in the location pointed to by *MODID_Mask_Ptr*.

The value pointed to by *MODID_Mask_Ptr* is either zero or an OR'd mask of the following constants. A set bit indicates that the corresponding actual bus MODID line is asserted.

A clear bit indicates that the corresponding actual bus MODID line is deasserted:

Constant	Description
EPC_SLOT_MODID	MODID line for the interface's bus slot.

The value pointed to by *MODID_Mask_Ptr* reflects the actual bus MODID line state, not the interface MODID line state. Use **EpcGetEpcMODID** to query the interface MODID line state.

A device can always query the state of the MODID bus control line corresponding to its bus slot. The **MODID_Mask_Ptr* state bit **EPC_SLOT_MODID** always contains valid data.

Return Value The function returns a EPConnect return value:

EPC_INV_PTR	The parameter *MODID_Mask_Ptr* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The Bus Manager device driver is not present.
EPC_SUCCESS	The function completed successfully.

See Also **EpcGetEpcMODID, EpcOpenSession, EpcSetEpcMODID.**

EpcGetBusTriggers

Description Queries the actual bus trigger line state.

C Synopsis

```
#include "busmgr.h"

short
EpcGetBusTriggers( unsigned long          Session_ID,
                   unsigned long FAR *    Trigger_Mask_Ptr);
```

Session_ID	*Session_ID* specifies a session.
Trigger_Mask_Ptr	*Trigger_Mask_Ptr* specifies a location where the actual bus trigger line state will be placed.

Visual Basic Synopsis

```
Declare Function
EpcGetBusTriggers% Lib "bmvxiw16.dll" (ByVal
Session_ID&,Trigger_Mask_Ptr&)
```

Remarks **EpcGetBusTriggers** queries the actual bus trigger line state and places it in the location pointed to by *Trigger_Mask_Ptr*.

The value pointed to by *Trigger_Mask_Ptr* is either zero or an OR'd mask of the following constants. A set bit indicates that the corresponding actual bus trigger line is asserted.

A clear bit indicates that the corresponding actual bus trigger line is deasserted:

Constant	Description
EPC_ECL_TRIG0	ECL trigger 0 (EPC-7 only).
EPC_ECL_TRIG1	ECL trigger 1 (EPC-7 only).
EPC_TTL_TRIG0	TTL trigger 0 (EPC-7 and VXLink only).
...	
EPC_TTL_TRIG7	TTL trigger 7 (EPC-7 and VXLink only).

The value pointed to by *Trigger_Mask_Ptr* reflects the actual bus trigger line state, not the interface trigger line state. Use **EpcGetEpcTriggers** to query the interface trigger line state.

Return Value The function returns a EPConnect return value:

EPC_INV_PTR	The parameter *Trigger_Mask_Ptr* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The Bus Manager device driver is not present.
EPC_SUCCESS	The function completed successfully.

See Also **EpcGetEpcTriggers, EpcOpenSession, EpcPulseEpcTriggers, EpcSetEpcTriggers.**

EpcGetEpcInterrupt

Description Queries the interface VME interrupt assertion state.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcGetEpcInterrupt( unsigned long        Session_ID,
                    unsigned long FAR *  Event_Mask_Ptr);
```

Session_ID	*Session_ID* specifies a session.
Event_Mask_Ptr	*Event_Mask_Ptr* specifies a location where the currently asserted VME interrupt will be placed.

Visual Basic Synopsis

```
Declare Function
EpcGetEpcInterrupt% Lib "bmvxiw16.dll" (ByVal Session_ID&,
Event_Mask_Ptr&)
```

Remarks **EpcGetEpcInterrupt** queries the VME interrupt currently asserted by the interface and places a it in the location pointed to by *Event_Mask_Ptr.*

The function places a constant at *Event_Mask_Ptr* specifying the VME interrupt currently asserted by the interface. Possible values are:

Constant	Description
EPC_NO_INT	The interface is not currently asserting a VME interrupt.
EPC_VME1_INT	The interface is currently asserting VME interrupt 1.
...	

EPC_VME7_INT	The interface is currently asserting VME interrupt 7.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	The parameter *Event_Mask_Ptr* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_SUCCESS	The function completed successfully.

See Also **EpcAssertInterrupt, EpcDeassertInterrupt, EpcGetBusInterrupts.**

Example See **EpcAssertInterrupt**.

EpcGetEpcLines

Description Queries the interface control line state.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcGetEpcLines(unsigned long          Session_ID,
               unsigned long FAR *    Line_Mask_Ptr);
```

Session_ID	*Session_ID* specifies a session.
Line_Mask_Ptr	*Line_Mask_Ptr* specifies a location where the interface control line state will be placed.

Visual Basic Synopsis

```
Declare Function
EpcGetEpcLines% Lib "bmvxiw16.dll" (ByVal Session_ID&,
Line_Mask_Ptr&)
```

Remarks **EpcGetEpcLines** queries the interface control line state and places it in the location pointed to by *Line_Mask_Ptr.*

The value pointed to by *Line_Mask_Ptr* is either zero or an OR'd mask of the following constants. A set bit indicates that the corresponding interface control line is asserted. A clear bit indicates that the corresponding interface control line is deasserted:

Constant	Description
EPC_SYSFAIL	SYSFAIL.
EPC_SYSRESET	SYSRESET.

The value pointed to by *Line_Mask_Ptr* reflects the interface control line state, not the actual bus control line state. Use **EpcGetBusLines** to query the actual bus control line state.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	The parameter *Line_Mask_Ptr* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_SUCCESS	The function completed successfully.

See Also **EpcGetBusLines, EpcOpenSession, EpcPulseEpcLines, EpcSetEpcLines.**

Example See **EpcAssertInterrupt.**

EpcGetEpcMODID

Description Queries the interface MODID line state.

C Synopsis

```
#include "busmgr.h"

short
EpcGetEpcMODID( unsigned long          Session_ID,
                unsigned long FAR *    MODID_Mask_Ptr);
```

Session_ID	*Session_ID* specifies a session.
MODID_Mask_Ptr	*MODID_Mask_Ptr* specifies a location where the interface MODID line state will be placed.

Visual Basic Synopsis

```
Declare Function
EpcGetEpcMODID% Lib "bmvxiw16.dll" (ByVal
Session_ID&,MODID_Mask_Ptr&)
```

Remarks **EpcGetEpcMODID** queries the interface MODID line state and places it in the location pointed to by *MODID_Mask_Ptr*.

The value pointed to by *MODID_Mask_Ptr* is either zero or an OR'd mask of the following constants. A set bit indicates that the corresponding interface MODID line is asserted. A clear bit indicates that the corresponding interface MODID line is deasserted:

Constant	Description
EPC_MODID0	MODID line 0 (EPC-7 and VXLink only).
...	
EPC_MODID12	MODID line 12 (EPC-7 and VXLink only).

The value pointed to by *MODID_Mask_Ptr* reflects the interface MODID line state, not the actual bus MODID line state. Use **EpcGetBusMODID** to query the actual bus MODID line state.

Return Value The function returns a EPConnect return value:

EPC_INV_PTR	The parameter *MODID_Mask_Ptr* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The Bus Manager device driver is not present.
EPC_SUCCESS	The function completed successfully.

See Also **EpcGetBusMODID, EpcOpenSession, EpcSetEpcMODID.**

EpcGetEpcTriggerMapping

Description Queries an interface trigger line mapping.

C Synopsis

```
#include "busmgr.h"

short
EpcGetEpcTriggerMapping
                (unsigned long Session_ID,
                unsigned long In_Trigger_Mask,
                unsigned long FAR * Out_Trigger_Mask_Ptr);
```

Session_ID	*Session_ID* specifies a session.
In_Trigger_Mask	*In_Trigger_Mask* specifies an interface trigger line.
Out_Trigger_Mask_Ptr	*Out_Trigger_Mask_Ptr* specifies a location where a mask of interface trigger lines will be placed.

Visual Basic Synopsis

```
Declare Function
EpcGetEpcTriggerMapping% Lib "bmvxiw16.dll" (ByVal
                Session_ID&, ByVal In_Trigger_Mask&,
                Out_Trigger_Mask_Ptr&)
```

Remarks **EpcGetEpcTriggerMapping** queries the interface trigger lines mapped to the specified *In_Trigger_Mask* and places a mask identifying them in the location pointed to by *Out_Trigger_Mask_Ptr*.

The parameter *In_Trigger_Mask* is a constant specifying a single interface trigger line. The value placed at the location pointed to by *Out_Trigger_Mask_Ptr* is an OR'd mask of constants identifying the interface trigger lines mapped to the specified input interface trigger line. The table below enumerates possible trigger mapping combinations for an EPC-7 interface:

In_Trigger_Mask	***Out_Trigger_Mask_Ptr**	**Description**
EPC_EXT_TRIG0	EPC_TTL_TRIG0 ... EPC_TTL_TRIG7	External trigger 0 mapped as input to a single TTL trigger line.
EPC_TTL_TRIG0 ... EPC_TTL_TRIG7	EPC_EXT_TRIG0	A single TTL trigger line mapped as input to external trigger 0.

The table below enumerates possible trigger mapping combinations for a VXLink interface:

In_Trigger_Mask	***Out_Trigger_Mask_Ptr**	**Description**
EPC_EXT_TRIG0	0x00000000	External trigger 0 unmapped.
EPC_EXT_TRIG0	EPC_TTL_TRIG0 ... EPC_TTL_TRIG7	External trigger 0 mapped as input to a single TTL trigger line.
EPC_TTL_TRIG0 ... EPC_TTL_TRIG7	EPC_EXT_TRIG1	A single TTL trigger line mapped as input to external trigger 1.

When an external trigger line is mapped as input to one or more interface trigger lines, asserting the external trigger line asserts all of the mapped interface trigger lines. Likewise, deasserting the external trigger line deasserts all of the mapped interface trigger lines.

When one or more interface trigger lines are mapped as input to an external trigger line, asserting one of the interface trigger lines asserts the mapped external trigger line. Likewise, deasserting one of the interface trigger lines deasserts the mapped external trigger line.

An EPC-7 interface provides a single bi-directional external trigger. The external trigger is always mapped; it cannot be unmapped. Specifying a mapping for external trigger 0 overrides the previous mapping. By default, TTL trigger 1 is mapped as an output to external trigger 0.

A VXLink interface provides two unidirectional external triggers. External trigger 0 is an input-only trigger and external trigger 1 is an output-only trigger. The external triggers can be independently mapped or unmapped. By default, both external triggers are unmapped.

Return Value The function returns a EPConnect return value:

EPC_INV_MASK	The parameter *In_Trigger_Mask* is invalid.
EPC_INV_PTR	The parameter *Out_Trigger_Mask_Ptr* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The Bus Manager device driver is not present.
EPC_SUCCESS	The function completed successfully.

See Also **EpcMapEpcTriggers, EpcOpenSession.**

EpcGetEpcTriggers

Description Query the interface trigger line state.

C Synopsis

```
#include "busmgr.h"

short
EpcGetEpcTriggers   (unsigned long      Session_ID,
                     unsigned long FAR * Trigger_Mask_Ptr);
```

Session_ID	*Session_ID* specifies a session.
Trigger_Mask_Ptr	*Trigger_Mask_Ptr* specifies a location where the EPC trigger line state will be placed.

Visual Basic Synopsis

```
Declare Function
EpcGetEpcTriggers% Lib "bmvxiw16.dll" (ByVal
Session_ID&,Trigger_Mask_Ptr&)
```

Remarks **EpcGetEpcTriggers** queries the interface trigger line state and places it in the location pointed to by *Trigger_Mask_Ptr.*

The value pointed to by *Trigger_Mask_Ptr* is either zero or an OR'd mask of the following constants. A set bit indicates that the corresponding interface trigger line is asserted.

A clear bit indicates that the corresponding interface trigger line is deasserted:

Constant	Description
EPC_ECL_TRIG0	ECL trigger 0 (EPC-7 only)
EPC_ECL_TRIG1	ECL trigger 1 (EPC-7 only)
EPC_TTL_TRIG0	TTL trigger 0 (EPC-7 and VXLink only)
...	
EPC_TTL_TRIG7	TTL trigger 7 (EPC-7 and VXLink only)

The value pointed to by *Trigger_Mask_Ptr* reflects the interface trigger line state, not the actual bus trigger line state. Use **EpcGetBusTriggers** to query the actual bus trigger line state.

Return Value The function returns a EPConnect return value:

EPC_INV_PTR	The parameter *Trigger_Mask_Ptr* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The Bus Manager device driver is not present.
EPC_SUCCESS	The function completed successfully.

See Also **EpcGetBusTriggers, EpcOpenSession, EpcPulseEpcTriggers, EpcSetEpcTriggers.**

EpcGetErrorString

Description Queries a null-terminated string corresponding to a Bus Management return value.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcGetErrorString( short      Return_Value
                   char FAR*  Buffer_Ptr);
```

Return_Value — *Return_Value* specifies a Bus Management return value.

Buffer_Ptr — *Buffer_Ptr* specifies the location of a buffer where the null-terminated string will be placed.

Visual Basic Synopsis

```
Declare Function
EpcGetErrorString% Lib "bmvxiw16.dll" (ByVal
Return_Value&, Buffer_Ptr&)
```

Remarks **EpcGetErrorString** places a null-terminated ASCII character string describing a Bus Management return value in the buffer pointed to by *Buffer_Ptr*.

Return_Value specifies a Bus Management return value. Specifying an invalid value results in the function returning a pointer to the string "Unknown EPConnect Return Value".

The buffer pointed to by *Buffer_Ptr* must be at least **ERROR_STRING_SZ** bytes long.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	The parameter *Buffer_Ptr* is invalid.
EPC_SUCCESS	The function completed successfully.

Example

```
/*
 * Copyright 1994 by RadiSys Corporation.  All rights reserved.
 */

/*
 * environ.c -- Bus Management Library environment functions sample code.
 */

#include "busmgr.h"

/*
 * FUNCTION PROTOTYPES...
 */

short FAR
EnvironmentSample(void);

int FAR
WinPrintf(char FAR *Format_Ptr, ...);

/*
 * CODE...
 */

short FAR
EnvironmentSample(void)
{
    char                    err_string[ERROR_STRING_SZ];
    short                   err_num;
    struct EpcEnvironment   environment;

    /*
     * Verify the EPConnect environment.
     */

    if ((err_num = EpcVerifyEnvironment(&environment)) != EPC_SUCCESS)
    {
        EpcGetErrorString(err_num, err_string);
        WinPrintf("FAILURE:  EpcVerifyEnvironment() error == %s (%d).\n",
                  err_string,
                  err_num);
        return (err_num);
    }
    WinPrintf("SUCCESS:  EnvironmentSample() complete.\n");
    return (EPC_SUCCESS);
}
```

EpcGetEventEnableMask

Description Queries a session's enabled event mask attribute.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcGetEventEnableMask
        (unsigned long          Session_ID,
        unsigned long FAR *  Event_Mask_Ptr);
```

Session_ID	*Session_ID* specifies a session.
Event_Mask_Ptr	*Event_Mask_Ptr* specifies a location where the enabled event mask attribute of the specified session will be placed.

Visual Basic Synopsis

```
Declare Function
EpcGetEventEnableMask% Lib "bmvxiw16.dll" (ByVal
Session_ID&, Event_Mask_Ptr&)
```

Remarks **EpcGetEventEnableMask** places the specified session's enabled event mask attribute in the location pointed to by *Event_Mask_Ptr.*

An enabled event mask attribute is a bit mask where each bit corresponds to an event. A zero in a bit position specifies that the corresponding event's reception is disabled. A one in a bit position specifies that the corresponding event's reception is enabled.

The mask is either zero or an OR'd combination of the following constants:

Event	Description
EPC_MSG_INT	Message interrupt (EPC-7 and EPC-8 only)
EPC_VME1_INT	VMEbus interrupt 1
...	
EPC_VME7_INT	VMEbus interrupt 7
EPC_SIGNAL_INT	VXIbus signal FIFO interrupt
EPC_TTL_TRIG0_INT	VXIbus TTL Trigger 0 interrupt (EPC-7 only)
...	
EPC_TTL_TRIG7_INT	VXIbus TTL Trigger 7 interrupt (EPC-7 only)
EPC_SYSRESET_ERR	VMEbus SYSRESET error
EPC_ACFAIL_ERR	VMEbus power failure error
EPC_BERR_ERR	VMEbus access error
EPC_SYSFAIL_ERR	VMEbus SYSFAIL error
EPC_WATCHDOG_ERR	Watchdog timer expiration error (EPC-7 and EPC-8 only)
EPC_EXT_TRIG0_INT	External trigger 0 interrupt (VXLink only)
EPC_EXT_TRIG1_INT	External trigger 1 interrupt (VXLink only)

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	The parameter *Event_Mask_Ptr* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_SUCCESS	The function completed successfully.
EPC_INV_SW	The BusManager device driver is not present.

See Also **EpcSetEventEnableMask, EpcOpenSession.**

Example

```
/*
 * Copyright 1994 by RadiSys Corporation.  All rights reserved.
 */

/*
 * events.c -- Bus Management Library events functions sample code.
 */

#include "busmgr.h"

/*
 * CONSTANTS...
 */

#define STACK_SIZE                4096

/*
 * FUNCTION PROTOTYPES...
 */

short FAR
EventsSample(void);

void FAR LOADDS
EventHandler(unsigned long Session_ID,
             unsigned long Event_Mask,
             unsigned long Event_Data);

int FAR
WinPrintf(char FAR *Format_Ptr, ...);

/*
 * GLOBAL DATA...
 */

unsigned char EventStack[STACK_SIZE] = { 0 };

/*
 * CODE...
 */

short FAR
EventsSample(void)
{
    char                  err_string[ERROR_STRING_SZ];
    short                 err_num;
    unsigned long         event_data;
    unsigned long         event_mask;
    unsigned long         session_id;
    void             FAR *event_stack;
    void            (FAR *event_handler)(unsigned long,
                                         unsigned long,
                                         unsigned long);
    struct EpcEnvironment environment;

    /*
     * Verify the EPConnect environment.
     */

    if ((err_num = EpcVerifyEnvironment(&environment)) != EPC_SUCCESS)
    {
```

```
        EpcGetErrorString(err_num, err_string);
        WinPrintf("FAILURE:  EpcVerifyEnvironment() error == %s (%d).\n",
                  err_string,
                  err_num);
        return (err_num);
    }

    /*
     * Open a session.
     */

    if ((err_num = EpcOpenSession(&session_id)) != EPC_SUCCESS)
    {
        EpcGetErrorString(err_num, err_string);
        WinPrintf("FAILURE:  EpcOpenSession() error == %s (%d).\n",
                  err_string,
                  err_num);
        return (err_num);
    }

    /*
     * Define the session's event handler for VMEbus interrupt 1.
     */

    EpcSetEventHandler(session_id,
                       EPC_VME1_INT,
                       (void (FAR *)(unsigned long,
                                     unsigned long,
                                     unsigned long)) EventHandler,
                       (void FAR *) &EventStack[STACK_SIZE]);

    /*
     * Define the session's event enable mask to enable VMEbus interrupt 1.
     */

    EpcSetEventEnableMask(session_id, EPC_VME1_INT);

    /*
     * Query the session's event handler for VMEbus interrupt 1.
     */

    EpcGetEventHandler(session_id, EPC_VME1_INT, &event_handler, &event_stack);

    /*
     * Query the session's event enable mask.
     */

    EpcGetEventEnableMask(session_id, &event_mask);

    /*
     * Wait up to one second (1000 ms) for VMEbus interrupt 1 to occur.
     */

    EpcWaitForEvent(session_id, 1000, EPC_VME1_INT, &event_mask, &event_data);

    /*
     * Close the session and return.
     */

    EpcCloseSession(session_id);
    WinPrintf("SUCCESS:  EventsSample() complete.\n");
    return (EPC_SUCCESS);
}

void FAR LOADDS
```

```
EventHandler(unsigned long Session_ID,
             unsigned long Event_Mask,
             unsigned long Event_Data)
{
    /*
     * Avoid compiler warnings.
     */

    Session_ID = Session_ID;
    Event_Mask = Event_Mask;
    Event_Data = Event_Data;
}
```

EpcGetEventHandler

Description Queries an entry in a session's event handler array.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcGetEventHandler

(unsigned long      Session_ID,
unsigned long       Event_Mask,
void (FAR * FAR *   Event_Handler_Ptr)(unsigned long,
                      unsigned long, unsigned long),
void FAR * FAR *    Stack_Ptr_Ptr);
```

Session_ID — *Session_ID* specifies a session.

Event_Mask — *Event_Mask* specifies an event.

Event_Handler_Ptr — *Event_Handler_Ptr* specifies a location where the specified session's specified event handler will be placed.

Stack_Ptr_Ptr — *Stack_Ptr_Ptr* specifies a location where the specified session's specified event handler stack will be placed.

Visual Basic Synopsis

```
Declare Function
EpcGetEventHandler% Lib "bmvxiw16.dll"
                    (ByVal Session_ID&,
                    ByVal Event_Mask&,
                    Event_Handler_Ptr As Any,
                    Stack_Ptr_Ptr As Any)
```

Remarks **EpcGetEventHandler** places the specified session's specified event handler address and event handler stack pointer in the locations pointed to by *Event_Handler_Ptr* and *Stack_Ptr_Ptr*.

The *Event_Mask* parameter is a bit mask where each bit corresponds to an event. The *Event_Mask* parameter should be one of the following constants:

Event	Description
EPC_MSG_INT	Message interrupt (EPC-7 and EPC-8 only)
EPC_VME1_INT	VMEbus interrupt 1
...	
EPC_VME7_INT	VMEbus interrupt 7
EPC_SIGNAL_INT	VXIbus signal FIFO interrupt
EPC_TTL_TRIG0_INT	VXIbus TTL Trigger 0 interrupt (EPC-7 only)
...	
EPC_TTL_TRIG7_INT	VXIbus TTL Trigger 7 interrupt (EPC-7 only)
EPC_SYSRESET_ERR	VMEbus SYSRESET error
EPC_ACFAIL_ERR	VMEbus power failure error
EPC_BERR_ERR	VMEbus access error
EPC_SYSFAIL_ERR	VMEbus SYSFAIL error
EPC_WATCHDOG_ERR	Watchdog timer expiration error (EPC-7 and EPC-8 only)
EPC_EXT_TRIG0_INT	External trigger 0 interrupt (VXLink only)
EPC_EXT_TRIG1_INT	External trigger 1 interrupt (VXLink only)

If the session has no event handler defined for the specified event, the function places NULL in the locations pointed to by *Event_Handler_Ptr* and *Stack_Ptr_Ptr*.

Return Value The function returns a Bus Management return value:

EPC_INV_MASK	*Event_Mask* contains more than one event or contains an event that is not valid for this EPC.
EPC_INV_PTR	One or both of the parameters *Event_Handler_Ptr* and *Stack_Ptr_Ptr* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_SUCCESS	The function completed successfully.

See Also **EpcSetEventHandler, EpcOpenSession.**

Example See **EpcGetEventEnableMask.**

EpcGetLockingTimeout

Description Queries a session's locking timeout.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcGetLockingTimeout( unsigned long          Session_ID,
                      unsigned long FAR *    Timeout_Ptr);
```

Session_ID	*Session_ID* specifies a session.
Timeout_Ptr	*Timeout_Ptr* specifies a location where the specified session's locking timeout will be placed.

Visual Basic Synopsis

```
Declare Function
EpcGetLockingTimeout% Lib "bmvxiw16.dll" (ByVal
Session_ID&, Timeout_Ptr&)
```

Remarks **EpcGetLockingTimeout** queries the specified session's locking timeout and places it in the location pointed to by *Timeout_Ptr*.

Upon successful function completion, *Timeout_Ptr* contains the session's locking timeout, in milliseconds.

By default, a session has a locking timeout of zero milliseconds. When the session encounters a locking conflict, an **EPC_LOCKED** error is returned immediately.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	The parameter *Timeout_Ptr* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_SUCCESS	The function completed successfully.

See Also **EpcLockSession, EpcOpenSession, EpcSetLockingTimeout.**

Example

```
/*
 * Copyright 1994 by RadiSys Corporation.  All rights reserved.
 */

/*
 * locking.c -- Bus Management Library locking functions sample code.
 */

#include "busmgr.h"

/*
 * FUNCTION PROTOTYPES...
 */

short FAR
LockingSample(void);

int FAR
WinPrintf(char FAR *Format_Ptr, ...);

/*
 * CODE...
 */

short FAR
LockingSample(void)
{
    char                    err_string[ERROR_STRING_SZ];
    short                   err_num;
    unsigned long           session_id1;
    unsigned long           session_id2;
    unsigned long           timeout;
    struct EpcEnvironment   environment;

    /*
     * Verify the EPConnect environment.
     */

    if ((err_num = EpcVerifyEnvironment(&environment)) != EPC_SUCCESS)
    {
        EpcGetErrorString(err_num, err_string);
        WinPrintf("FAILURE:  EpcVerifyEnvironment() error == %s (%d).\n",
                  err_string,
                  err_num);
        return (err_num);
    }

    /*
     * Open two sessions.
     */

    if ((err_num = EpcOpenSession(&session_id1)) != EPC_SUCCESS ||
        (err_num = EpcOpenSession(&session_id2)) != EPC_SUCCESS  )
    {
        EpcCloseSession(session_id1);
        EpcGetErrorString(err_num, err_string);
        WinPrintf("FAILURE:  EpcOpenSession() error == %s (%d).\n",
                  err_string,
                  err_num);
        return (err_num);
    }

    /*
```

```
     * Define the second session's locking timeout to be one second (1000 ms).
     */

    EpcSetLockingTimeout(session_id2, 1000);

    /*
     * Query the second session's locking timeout.
     */

    EpcGetLockingTimeout(session_id2, &timeout);

    /*
     * Lock shared interface hardware.
     *
     * NOTES:
     *      1.  The EpcLockSession() call for the second session fails after a
     *          one second (1000 ms) timeout, since shared interface hardware is
     *          already locked by the first session.
     */

    EpcLockSession(session_id1);
    EpcLockSession(session_id2);

    /*
     * Unlock shared interface hardware with both sessions.
     */

    EpcUnlockSession(session_id1);

    /*
     * Close the sessions and return.
     */

    EpcCloseSession(session_id1);
    EpcCloseSession(session_id2);
    WinPrintf("SUCCESS:  LockingSample() complete.\n");
    return (EPC_SUCCESS);
}
```

EpcGetMappingAttributes

Description Queries a memory mapping's attributes.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcGetMappingAttributes
    (unsigned long              Session_ID,
    volatile void      HUGE *   Mapped_Ptr,
    unsigned short     FAR *    Address_Mod_Ptr,
    unsigned short     FAR *    Byte_Ordering_Ptr,
    unsigned long      FAR *    Base_Address_Ptr,
    unsigned long      FAR *    Size_Ptr);
```

Session_ID — *Session_ID* specifies a bus session.

Mapped_Ptr — *Mapped_Ptr* specifies a pointer to the base of a memory mapping.

Address_Mod_Ptr — *Address_Mod_Ptr* specifies a location where the address modifier attribute of the specified memory mapping will be placed.

Byte_Ordering_Ptr — *Byte_Ordering_Ptr* specifies a location where the byte ordering attribute of the specified memory mapping will be placed.

Base_Address_Ptr — *Base_Address_Ptr* specifies a location where the base address attribute of the specified memory mapping will be placed.

Size_Ptr — *Size_Ptr* specifies a location where the size attribute of the specified memory mapping, in bytes, will be placed.

Visual Basic Synopsis

```
Declare Function
EpcGetMappingAttributes% Lib "bmvxiw16.dll"
                         (ByVal Session_ID&,
                         ByVal Mapped_Ptr As Any,
                         Address_Mod_Ptr%,
                         Byte_Ordering_Ptr%,
                         Base_Address_Ptr&,
                         Size_Ptr&)
```

Remarks **EpcGetMappingAttributes** places the specified memory mapping's attributes in the locations pointed to by *Address_Mod_Ptr*, *Byte_Ordering_Ptr*, *Base_Address_Ptr*, and *Size_Ptr*, respectively.

The location pointed to by *Address_Mod_Ptr* can contain the following values:

Constant	Description
EPC_A16N	VMEbus A16 non-supervisory address modifier.
EPC_A16S	VMEbus A16 supervisory address modifier.
EPC_A24ND	VMEbus A24 non-supervisory data address modifier.
EPC_A24SD	VMEbus A24 supervisory data address modifier.
EPC_A24NP	VMEbus A24 non-supervisory program address modifier.
EPC_A24SP	VMEbus A24 supervisory program address modifier.
EPC_A32ND	VMEbus A32 non-supervisory data address modifier.
EPC_A32SD	VMEbus A32 supervisory data address modifier.
EPC_A32NP	VMEbus A32 non-supervisory program address modifier.
EPC_A32SP	VMEbus A32 supervisory program address modifier.
EPC_SHARED	Shared memory address modifier.

The location pointed to by *Byte_Ordering_Ptr* can have the following values:

Constant	Description
EPC_IBO	Intel (80X86) byte ordering.
EPC_MBO	Motorola (68XXX) byte ordering.

For shared memory mappings, the value in the location pointed to by *Byte_Ordering_Ptr* is always **EPC_IBO**.

The values in the locations pointed to by *Base_Address_Ptr* and *Size_Ptr* define a range of addresses α, where:

**Base_Address_Ptr* <= α <= **Base_Address_Ptr* + **Size_Ptr* - 1;

For bus memory mappings, the value in the location pointed to by *Base_Address_Ptr* specifies a physical VMEbus address.

For shared memory mappings, the value in the location pointed to by *Base_Address_Ptr* specifies a physical PC address. To determine the corresponding physical VMEbus address, the value should be added to the base address of the interface's slave memory. Use **EpcGetSlaveMapping** to determine the base address of the interface's slave memory.

Return Value The function returns a Bus Management return value:

EPC_INV_MAP	The specified *Mapped_Ptr* is invalid.
EPC_INV_PTR	One or more of the parameters *Address_Mod_Ptr*, *Byte_Ordering_Ptr*, *Base_Address_Ptr*, or *Size_Ptr* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_SUCCESS	The function completed successfully.

See Also **EpcGetSlaveMapping**, **EpcMapBusMemory**, **EpcMapSharedMemory**, **EpcOpenSession**.

Example See **EpcCopyData**.

EpcGetMiscAttributes

Description Queries the interface's miscellaneous configuration attributes.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcGetMiscAttributes( unsigned long        Session_ID,
                      unsigned long FAR *  Misc_Mask_Ptr);
```

Session_ID	*Session_ID* specifies a session.
Misc_Mask_Ptr	*Misc_Mask_Ptr* specifies a location where the interface's miscellaneous configuration attributes will be placed.

Visual Basic Synopsis

```
Declare Function
EpcGetMiscAttributes% Lib "bmvxiw16.dll" (ByVal
Session_ID&, Misc_Mask_Ptr&)
```

Remarks **EpcGetMiscAttributes** queries the interface's miscellaneous configuration attributes and places them in the location pointed to by *Misc_Mask_Ptr*.

The location pointed to by *Misc_Mask_Ptr* contains either a zero or an OR'd bit mask of the following constants, where a set bit indicates that the corresponding miscellaneous interface attribute is asserted.

A clear bit indicates that the corresponding miscellaneous interface attribute is deasserted:

Constant	Description
EPC_DIR	Word serial byte transfer protocol DIR bit. Asserting the bit indicates that the interface is ready to receive data from its commander device. Supported on EPC-7 and EPC-8 only.

EPC_DOR	Word serial byte transfer protocol DOR bit. Asserting the bit indicates that the interface is ready to send data to its commander device. Supported on EPC-7 and EPC-8 only.
EPC_ERR	Word serial protocol ERR* bit. Asserting the bit indicates to the commander device that the interface has detected a word serial protocol error. Supported on EPC-7 and EPC-8 only.
EPC_LOCK	VXIbus message-based device LOCKED* bit. Asserting the bit indicates that the commander has locked access to the interface from local sources (IEEE-488 local lockout). Supported on EPC-7 and EPC-8 only.
EPC_MULTIPLE_ LOCK	Word serial protocol extension multiple commander lock bit. When asserted, the first commander to read the asserted bit from interface's Response register can safely send a word serial command. Supported on EPC-7 and EPC-8 only.
EPC_PASS	Device initialization PASSED bit. Asserting the bit indicates that the interface has passed self-test.
EPC_PIPELINE_BUSY	Bus hardware pipeline busy bit. When asserted, the bit indicates that the interface is executing a pipelined write to the bus.
EPC_READY	Device initialization READY bit. Asserting the bit indicates that the interface is ready to begin normal operation.

EPC_RRDY	Word serial protocol Read Ready bit. Asserting the bit indicates to a commander device that the interface has a word serial response in its message register. Supported on EPC-7 and EPC-8 only.
EPC_RSRC_MGR	Resource manager execution bit. Asserting the bit indicates that resource manager execution is complete.
EPC_STICKY_BERR	"Sticky" bus error bit. When asserted, the bit indicates that a bus error has occurred since the bit was last deasserted.
EPC_SYSFAIL_OUT	SYSFAIL output enable bit. When asserted, the interface can assert SYSFAIL. When deasserted, the interface cannot assert SYSFAIL.
EPC_SYSRESET_IN	SYSRESET input enable bit. When asserted, asserting SYSRESET resets the interface. When deasserted, asserting SYSRESET does not reset the interface.
EPC_TTL_LATCH0 ... EPC_TTL_LATCH7	TTL trigger latch bits. When asserted, a bit indicates that the interface has latched the corresponding TTL trigger interrupt. Supported on EPC-7 only.
EPC_WATCHDOG	Watchdog timer expiration bit. When asserted, the bit indicates that a watchdog timeout error has occurred since the watchdog timer was last reset. Supported on EPC-7 and EPC-8 only.
EPC_WRDY	Word serial protocol Write Ready bit. Asserting the bit indicates to a commander device that the interface is ready to receive a word serial command. Supported on EPC-7 and EPC-8 only.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	The parameter *Misc_Mask_Ptr* is invalid.
EPC_INV_SESSION	The parameter *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_SUCCESS	The function completed successfully.

See Also **EpcOpenSession, EpcSetMiscAttributes.**

Example See **EpcGetBusAttributes.**

EpcGetSessionData

Description Queries a session's application-specified data.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcGetSessionData( unsigned long          Session_ID,
                   unsigned long FAR *    Session_Data_Ptr);
```

Session_ID	*Session_ID* specifies an open session.
Session_Data_Ptr	*Session_Data_Ptr* specifies a location where the session's application-specified data will be placed.

Visual Basic Synopsis

```
Declare Function
EpcGetSessionData% Lib "bmvxiw16.dll" (ByVal Session_ID&,
Session_Data_Ptr&)
```

Remarks **EpcGetSessionData** queries the specified session's application-specified data and places it in the location pointed to by *Session_Data_Ptr*.

The application-specified data is a 4-byte quantity.

Typically, an application uses **EpcSetSessionData** to store a pointer to one of its data structures. Later, the application uses **EpcGetSessionData** to quickly retrieve the pointer during performance-critical operations (like event handling).

Return Value The function returns a Bus Management return value:

EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_PTR	The *Session_Data_Ptr* parameter is invalid.
EPC_SUCCESS	The function completed successfully.

See Also **EpcOpenSession, EpcSetSessionData.**

Example See **EpcCloseSession.**

EpcGetSlaveMapping

Description Queries the interface's slave memory mapping.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcGetSlaveMapping
        (unsigned long          Session_ID,
        unsigned short FAR *    Addess_Space_Ptr,
        unsigned long  FAR *    Base_Addess_Ptr);
```

Session_ID	*Session_ID* specifies a session.
Addess_Space_Ptr	*Addess_Space_Ptr* specifies a location where the interface's slave memory address space will be placed.
Base_Addess_Ptr	*Base_Addess_Ptr* specifies a location where the interface's slave memory base address will be placed.

Visual Basic Synopsis

```
Declare Function
EpcGetSlaveMapping% Lib "bmvxiw16.dll"
                        ( ByVal Session_ID&,
                        Address_Space_Ptr%,
                        Base_Address_Ptr&)
```

Remarks **EpcGetSlaveMapping** queries the mapping of the interface's slave memory and places the result in the locations pointed to by *Addess_Space_Ptr* and *Base_Addess_Ptr*.

Possible values at *Addess_Space_Ptr* and *Base_Addess_Ptr* are dependent on the interface type:

Interface Type	*Addess_Space_Ptr	*Base_Addess_Ptr
EPC-4	**EPC_DISABLED**	N/A
	EPC_A24	0x00000000, 0x00400000, ..., 0x00C00000
	EPC_A32	0x18000000, 0x19000000, ..., 0x1F000000
EPC-5	**EPC_DISABLED**	N/A
	EPC_A24	0x00000000, 0x00400000, ..., 0x00C00000
	EPC_A32	0x18000000, 0x19000000, ..., 0x1F000000
EPC-7	**EPC_DISABLED**	N/A
	EPC_A24	0x00000000, 0x00400000, ..., 0x00C00000
	EPC_A32	0x00000000, 0x01000000, ..., 0xFF000000
EPC-8	**EPC_DISABLED**	N/A
VXLink	**EPC_DISABLED**	N/A

A24 base addresses are aligned on a 4 Mbyte boundary, and only the first 4 Mbytes of the interface's slave memory is mapped to the bus. A32 base addresses are aligned on a 16 Mbyte boundary, and only the first 16 Mbytes of the interface's slave memory is mapped to the bus.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	One or more of the parameters *Addess_Space_Ptr* and *Base_Addess_Ptr* is invalid.
EPC_INV_SESSION	The parameter *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_SUCCESS	The function completed successfully.

See Also **EpcOpenSession, EpcSetSlaveMapping.**

Example See **EpcGetBusAttributes.**

EpcGetULA

Description Queries the interface's unique logical address.

C Synopsis

#include "busmgr.h"

short FAR PASCAL
EpcGetULA(unsigned long *Session_ID*, unsigned short **FAR** **ULA_Ptr*);

Session_ID	*Session_ID* specifies a session.
ULA_Ptr	*ULA_Ptr* specifies a location where the interface's unique logical address will be placed.

Visual Basic Synopsis

Declare Function
EpcGetULA% Lib "bmvxiw16.dll" (ByVal *Session_ID&*, *ULA_Ptr%*)

Remarks **EpcGetULA** queries the interface's unique logical address and places the result in the locations pointed to by *ULA_Ptr*. Possible unique logical addresses are 0x00 through 0xFF.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	The parameters *ULA_Ptr* is invalid.
EPC_INV_SESSION	The parameter *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_SUCCESS	The function completed successfully.

See Also **EpcOpenSession, EpcSetULA.**

Example See **EpcGetBusAttributes.**

EpcLockSession

Description Locks shared interface hardware for a session.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcLockSession(unsigned long Session_ID);
```

Session_ID *Session_ID* specifies a session.

Visual Basic Synopsis

```
Declare Function
EpcLockSession% Lib "bmvxiw16.dll" (ByVal Session_ID&)
```

Remarks **EpcLockSession** locks shared interface hardware for the specified *Session_ID*.

Locking gives a session exclusive access to shared interface hardware. Locking is used in multithreaded environments to prevent simultaneous, potentially conflicting hardware accesses.

Locks can be nested. EPConnect maintains a global lock counter. The global lock counter can be "owned" by at most one session. Initially, the lock counter is zero, indicating that no session has locked shared interface hardware. **EpcLockSession** acquires and increments the lock counter for a session. **EpcUnlockSession** decrements the lock counter for the same session. A non-zero lock counter indicates that shared interface hardware is locked.

When an application calls a Bus Management Bus Management Library function that obeys the locking paradigm, the function checks for an existing lock. If no lock exists or the specified session "owns" the lock, the function proceeds. Otherwise, the function suspends execution until the lock is released or the specified session's locking timeout expires. If the existing lock is not released before the specified session's locking timeout expires, the function returns **EPC_LOCKED**.

Use **EpcGetLockingTimeout** and **EpcSetLockingTimeout** to query and define a session's locking timeout.

The following EPConnect Bus Management Library functions obey locks:

EpcAssertInterrupt	**EpcSetEpcLines**
EpcCmdReceiveWSBuffer	**EpcSetEpcMODID**
EpcCmdSendWSBuffer	**EpcSetEpcTriggers**
EpcCmdSendWSCommand	**EpcSetMiscAttributes**
EpcDeassertInterrupt	**EpcSetSlaveMapping**
EpcLockSession	**EpcSetULA**
EpcMapBusMemory	**EpcSrvEnableWsCommand**
EpcMapEpcTriggers	**EpcSrvReceiveWSCommand**
EpcMapSharedMemory	**EpcSrvSendProtocolEvent**
EpcPulseEpcLines	**EpcSrvSendWSProtocolError**
EpcPulseEpcTriggers	**EpcSrvSendWSResponse**
EpcSetBusAttributes	**EpcValidateBusMapping**

Return Value

The function returns a Bus Management return value:

EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_SUCCESS	The function completed successfully.

See Also

EpcGetLockingTimeout, EpcOpenSession, EpcSetLockingTimeout, EpcUnlockSession.

Example

See **EpcGetLockingTimeout.**

EpcMapBusMemory

2

Description Creates a bus memory mapping using statically configured bus window hardware.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcMapBusMemory( unsigned long       Session_ID,
                 unsigned short      Address_Mod,
                 unsigned short      Byte_Ordering,
                 unsigned long       Base_Address,
                 unsigned long       Size,
                 void HUGE * FAR *   Mapped_Ptr_Ptr);
```

Session_ID	*Session_ID* specifies a bus session.
Address_Mod	*Address_Mod* specifies the address modifier attribute of the desired memory mapping.
Byte_Ordering	*Byte_Ordering* specifies the byte ordering attribute of the desired memory mapping.
Base_Address	*Base_Address* specifies base address attribute of the desired memory mapping.
Size	*Size* specifies the size attribute of the desired memory mapping, in bytes.
Mapped_Ptr_Ptr	*Mapped_Ptr_Ptr* points to a location where a pointer to the base of the desired memory will be placed.

Visual Basic Synopsis

```
Declare Function
EpcMapBusMemory% Lib "bmvxiw16.dll"
                 (ByVal Session_ID&,
                 ByVal Address_Mod%,
                 ByVal Byte_Ordering%,
                 ByVal Base_Address&,
                 ByVal Size&,
                 Mapped_Ptr_Ptr As Any)
```

Remarks

EpcMapBusMemory creates a memory mapping with the specified attributes using statically configured bus window hardware and places a pointer to the base of the memory in the location pointed to by *Mapped_Ptr_Ptr*.

The following constants define valid values for the *Address_Mod* parameter:

Constant	Description
EPC_A16N	VMEbus A16 non-supervisory address modifier.
EPC_A16S	VMEbus A16 supervisory address modifier.
EPC_A24ND	VMEbus A24 non-supervisory data address modifier.
EPC_A24SD	VMEbus A24 supervisory data address modifier.
EPC_A24NP	VMEbus A24 non-supervisory program address modifier.
EPC_A24SP	VMEbus A24 supervisory program address modifier.
EPC_A32ND	VMEbus A32 non-supervisory data address modifier.
EPC_A32SD	VMEbus A32 supervisory data address modifier.
EPC_A32NP	VMEbus A32 non-supervisory program address modifier.
EPC_A32SP	VMEbus A32 supervisory program address modifier.

The following constants define valid values for the *Byte_Ordering* parameter:

Constant	Description
EPC_IBO	Intel (80X86) byte ordering.
EPC_MBO	Motorola (68000) byte ordering.

EPC hardware provides a number of statically configured bus windows. The table below enumerates the bus memory mapping attributes supported by an EPC's statically configured bus window hardware:

Address_Mod	*Byte_Odrering*	*Base_Address* Range	*Size* Range
EPC_A16S	EPC_MBO EPC_IBO	0x00 to 0x0000FFFF	0x00010000 to 0x01
EPC_A24SD	EPC_MBO EPC_IBO	0x00 to 0x00FFFFFF	0x01000000 to 0x01
EPC_A32SD	EPC_MBO EPC_IBO	0x00 to 0x3FFFFFFF	0x40000000 to 0x01

The *Base_Address* and *Size* parameters define a range of addresses α, where:

$$Base\_Address <= \alpha <= Base\_Address + Size - 1;$$

The function rounds the specified *Base_Address* down to the nearest 4-byte boundary. The function also limits the size of the mapping according to the specified *Base_Address* and the bus window's maximum accessible bus address.

EPC and VXLink hardware provides one or more dynamically configured bus memory windows. Use **EpcMapBusMemoryExt** to map bus memory using dynamically configured bus memory window hardware.

Return Value The function returns a Bus Management return value:

EPC_INV_ADDMOD	The specified *Address_Mod* parameter is invalid.
EPC_INV_BORDER	The specified *Byte_Ordering* parameter is invalid.
EPC_INV_PTR	The specified *Mapped_Ptr_Ptr* parameter is invalid.
EPC_INV_RANGE	The specified *Base_Address* and *Size* Parameters define a bus address range that contains invalid addresses for the specified *Address_Mod* parameter and/or this interface.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_OS_ERROR	An operating system error occurred.
EPC_OUT_OF_RSRCS	The underlying operating system currently contains insufficient resources to create the specified mapping.
EPC_SUCCESS	The function completed successfully.

See Also **EpcCopyData, EpcGetMappingAttributes, EpcMapBusMemoryExt, EpcOpenSession, EpcPopData, EpcPushData, EpcUnmapBusMemory.**

Example See **EpcCopyData.**

EpcMapBusMemoryExt

Description Creates a bus memory mapping using dynamically configured bus window hardware.

C Synopsis

short **FAR PASCAL**
EpcMapBusMemoryExt(unsigned long *Session_ID*,
unsigned short *Address_Mod*,
unsigned short *Byte_Ordering*,
unsigned long *Base_Address*,
volatile void **HUGE * FAR ***
Mapped_Ptr_Ptr);

Session_ID	*Session_ID* specifies a bus session.
Address_Mod	*Address_Mod* specifies the address modifier attribute of the desired memory mapping.
Byte_Ordering	*Byte_Ordering* specifies the byte ordering attribute of the desired memory mapping.
Base_Address	*Base_Address* specifies base address attribute of the desired memory mapping.
Mapped_Ptr_Ptr	*Mapped_Ptr_Ptr* points to a location where a pointer to the base of the desired memory will be placed.

Visual Basic Synopsis

Declare Function
EpcMapBusMemoryExt% Lib "bmvxiw16.dll"
(ByVal *Session_ID&*,
ByVal *Address_Mod%*,
ByVal *Byte_Ordering%*,
ByVal *Base_Address&*,
ByVal *Size&*,
Mapped_Ptr_Ptr As Any)

Remarks

EpcMapBusMemoryExt creates a memory mapping with the specified attributes using statically configured bus window hardware and places a pointer to the base of the memory in the location pointed to by *Mapped_Ptr_Ptr*.

The following constants define valid values for the *Address_Mod* parameter:

Constant	Description
EPC_A16N	VMEbus A16 non-supervisory address modifier.
EPC_A16S	VMEbus A16 supervisory address modifier.
EPC_A24ND	VMEbus A24 non-supervisory data address modifier.
EPC_A24SD	VMEbus A24 supervisory data address modifier.
EPC_A24NP	VMEbus A24 non-supervisory program address modifier.
EPC_A24SP	VMEbus A24 supervisory program address modifier.
EPC_A32ND	VMEbus A32 non-supervisory data address modifier.
EPC_A32SD	VMEbus A32 supervisory data address modifier.
EPC_A32NP	VMEbus A32 non-supervisory program address modifier.
EPC_A32SP	VMEbus A32 supervisory program address modifier.

The following constants define valid values for the *Byte_Ordering* parameter:

Constant	Description
EPC_IBO	Intel (80X86) byte ordering.
EPC_MBO	Motorola (68000) byte ordering.

EPC and VXLink hardware provide one or more 64-Kbyte dynamically configured bus windows. The table below enumerates the bus memory mapping attributes supported by the dynamically configured bus window hardware:

Address_Mod	*Byte_Ordering*	*Base_Address* Range
EPC_A16N **EPC_A16S**	**EPC_MBO** **EPC_IBO**	0x000000000
EPC_A24ND **EPC_A24NP** **EPC_A24SD** **EPC_A24SP**	**EPC_MBO** **EPC_IBO**	0x000000000, 0x00010000, ..., 0x00FF0000
EPC_A32ND **EPC_A32NP** **EPC_A32SD** **EPC_A32SP**	**EPC_MBO** **EPC_IBO**	0x000000000, 0x00010000, ..., 0xFFFF0000

The function rounds the specified *Base_Address* down to the nearest bus window size boundary (64 Kbytes) and sets the size of the mapping to the size of the bus window (64 Kbytes). Mapping an address range larger than the bus window size requires multiple mappings. Also, mapping an address range that spans a bus window size boundary requires multiple mappings.

EPC hardware also provides a number of statically configured bus memory windows. Use **EpcMapBusMemory** to map bus memory using statically configured bus memory window hardware.

Return Value The function returns a Bus Management return value:

EPC_INV_ADDMOD	The specified *Address_Mod* parameter is invalid.
EPC_INV_BORDER	The specified *Byte_Ordering* parameter is invalid.
EPC_INV_PTR	The specified *Mapped_Ptr_Ptr* parameter is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_OS_ERROR	An operating system error occurred.
EPC_OUT_OF_RSRCS	The underlying operating system currently contains insufficient resources to create the specified mapping.
EPC_SUCCESS	The function completed successfully.

See Also **EpcCopyData, EpcGetMappingAttributes, EpcMapBusMemory, EpcOpenSession, EpcPopData, EpcPushData, EpcUnmapBusMemory.**

Example See **EpcCopyData.**

EpcMapEpcTriggers

Description Maps one interface trigger line to another.

C Synopsis

```
#include "busmgr.h"

short
EpcMapEpcTriggers( unsigned long  Session_ID,
                   unsigned long  In_Trigger_Mask,
                   unsigned long  Out_Trigger_Mask);
```

Session_ID	*Session_ID* specifies a session.
In_Trigger_Mask	*In_Trigger_Mask* specifies an input interface trigger line.
Out_Trigger_Mask	*Out_Trigger_Mask* specifies output interface trigger lines.

Visual Basic Synopsis

```
Declare Function
EpcMapEpcTriggers% Lib "bmvxiw16.dll"
            (ByVal Session_ID&,
            ByVal In_Trigger_Mask&,
            ByVal Out_Trigger_Mask&)
```

Remarks **EpcMapEpcTriggers** maps the interface trigger lines specified by *Out_Trigger_Mask* as outputs of the interface trigger line specified by *In_Trigger_Mask*.

The parameters *In_Trigger_Mask* is a constant specifying an input interface trigger line. The parameter *Out_Trigger_Mask* is an OR'd mask of constants specifying output interface trigger lines.

The table below enumerates valid trigger mapping combinations for an EPC-7 interface:

In_Trigger_Mask	*Out_Trigger_Mask*	Description
EPC_EXT_TRIG0	**EPC_TTL_TRIG0** ... **EPC_TTL_TRIG7**	Maps external trigger 0 as input to a single TTL trigger line.
EPC_TTL_TRIG0 ... **EPC_TTL_TRIG7**	**EPC_EXT_TRIG0**	Maps a single TTL trigger line as input to external trigger 0.

The table below enumerates valid trigger mapping combinations for a VXLink interface:

In_Trigger_Mask	*Out_Trigger_Mask*	Description
EPC_EXT_TRIG0	0x00000000	Unmaps external trigger 0.
EPC_EXT_TRIG0	**EPC_TTL_TRIG0** ... **EPC_TTL_TRIG7**	Maps external trigger 0 as input to a single TTL trigger line.
0x00000000	**EPC_EXT_TRIG1**	Unmaps external trigger 1.
EPC_TTL_TRIG0 ... **EPC_TTL_TRIG7**	**EPC_EXT_TRIG1**	Maps a single TTL trigger line as input to external trigger 0.

When an external trigger line is mapped as input to one or more interface trigger lines, asserting the external trigger line asserts all of the mapped interface trigger lines. Likewise, deasserting the external trigger line deasserts all of the mapped interface trigger lines.

When one or more interface trigger lines are mapped as input to an external trigger line, asserting one of the interface trigger lines asserts the mapped external trigger line. Likewise, deasserting one of the interface trigger lines deasserts the mapped external trigger line.

An EPC-7 interface provides a single bi-directional external trigger. The external trigger is always mapped; it cannot be unmapped. Specifying a mapping for external trigger 0 overrides the previous mapping. By default, TTL trigger 1 is mapped as an output to external trigger 0.

A VXLink interface provides two unidirectional external triggers. External trigger 0 is an input-only trigger and external trigger 1 is an output-only trigger. The external triggers can be independently mapped or unmapped. By default, both external triggers are unmapped.

Return Value The function returns a EPConnect return value:

EPC_INV_MASK	Either *In_Trigger_Mask* or *Out_Trigger_Mask* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The Bus Manager device driver is not present.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_SUCCESS	The function completed successfully.

See Also **EpcGetEpcTriggerMapping, EpcOpenSession.**

EpcMapSharedMemory

Description Creates a shared memory mapping.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcMapSharedMemory
        (unsigned long          Session_ID,
        unsigned long  FAR *    Base_Address_Ptr,
        unsigned long  FAR *    Size_Ptr,
        void HUGE * FAR *       Mapped_Ptr_Ptr);
```

Session_ID	*Session_ID* specifies a bus session.
Base_Address_Ptr	*Base_Address_Ptr* points to a location where the base address attribute of the shared memory mapping will be placed.
Size_Ptr	*Size_Ptr* points to a location where the size attribute of the shared memory mapping, in bytes, will be placed.
Mapped_Ptr_Ptr	*Mapped_Ptr_Ptr* points to a location where a pointer to the base of the desired memory will be placed.

Visual Basic Synopsis

```
Declare Function
EpcMapSharedMemory% Lib "bmvxiw16.dll"
                        (ByVal Session_ID&,
                        Base_Address_Ptr&,
                        Size_Ptr&,
                        Mapped_Ptr_Ptr As Any)
```

Remarks **EpcMapSharedMemory** creates a shared memory mapping and places the base address attribute of the memory mapping, the size attribute of the memory mapping, and a pointer to the base of the memory in the locations pointed to by *Base_Address_Ptr*, *Size_Ptr*, and *Mapped_Ptr_Ptr*, respectively.

The values in the locations pointed to by *Base_Address_Ptr* and *Size_Ptr* define a range of addresses α, where:

**Base_Address_Ptr* <= α <= **Base_Address_Ptr* + **Size_Ptr* - 1;

The value in the location pointed to by *Base_Address_Ptr* specifies a physical local address. To determine the corresponding physical VMEbus address, the value should be added to the base address of the slave memory. Use **EpcGetSlaveMapping** to determine the base address of the slave memory.

A shared memory area is a global resource. **EpcMapSharedMemory** and **EpcUnmapSharedMemory** map and unmap the entire shared memory area. Once a session maps the shared memory area, it cannot be mapped again until the original session unmaps it.

An interface must contain dual-ported slave memory to support a shared memory area. Only the EPC-7 supports shared memory area functionality.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	One or more of the *Base_Address_Ptr*, *Size_Ptr*, and *Mapped_Ptr_Ptr* parameters is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_LOCKED	Shared interface hardware is locked by another bus session.
EPC_OS_ERROR	An operating system error occurred.
EPC_OUT_OF_RSRCS	The underlying operating system currently contains insufficient resources to create the specified mapping.
EPC_SUCCESS	The function completed successfully.

See Also EpcCopyData, EpcGetMappingAttributes, EpcGetSlaveMapping, EpcOpenSession, EpcUnmapSharedMemory.

Example See EpcCopyData.

EpcOpenSession

Description Creates a session.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcOpenSession( unsigned long FAR *Session_ID_Ptr);
```

Session_ID_Ptr — *Session_ID_Ptr* points to a location where a handle to the session will be placed.

Visual Basic Synopsis

```
Declare Function
EpcOpenSession% Lib "bmvxiw16.dll" (Session_ID_Ptr&)
```

Remarks **EpcOpenSession** creates a session and places a handle to the session in the location pointed to by *Session_ID_Ptr*.

By default, a newly created session does not lock shared interface hardware and has no enabled events, installed event handlers, or memory mappings.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	The specified *Session_ID_Ptr* parameter is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_OUT_OF_RSRCS	The underlying operating system currently contains insufficient resources to open a session.
EPC_SUCCESS	The function completed successfully.

See Also **EpcCloseSession, EpcLockSession, EpcMapBusMemory, EpcMapSharedMemory, EpcSetEventEnableMask, EpcSetEventHandler.**

Example See EpcCloseSession.

EpcPopData

Pops a block of data from a single memory location to consecutive memory locations.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcPopData( unsigned long *          Session_ID,
            void HUGE *              Source_ptr,
            void HUGE *              Dest_ptr,
            unsigned long            Size,
            unsigned short           Data_Width
            unsigned long  FAR *     Actual_Size_Ptr);
```

Session_ID — *Session_ID* specifies a bus location.

Source_Ptr — *Source_Ptr* specifies the address of a FIFO queue from which data will be popped.

Dest_Ptr — *Dest_Ptr* specifies the address of a data buffer into which data will be popped.

Size — *Size* specifies the number of data bytes to pop.

Data_Width — *Data_Width* specifies the number of data bits to pop per bus access.

Actual_Size_Ptr — *Actual_Size_Ptr* specifies a location where the actual number of bytes popped will be placed.

Remarks

EpcPopData efficiently pops blocks of data from a single memory location to consecutive memory locations using the attributes of pointes *Source_Ptr* and *Dest_Ptr*. The intended use of the function is popping large blocks of data from a FIFO queue.

The *Size* parameter shouold always express the number of bytes to be popped, regardless of the specified *Data_Width* parameter. Passing a zero *Size* parameter results in no data being popped.

Remarks

The following constants define valid values for the *Data_Width* parameter:

Constant	Description
EPC_8_BIT	8-bit data width
EPC_8_BIT_ODD	8-bit data width, odd bytes only
EPC_16_BIT	16-bit data width
EPC_32_BIT	32-bit data width
EPC_FASTCOPY	To increase pop performance, don't check for intermediate bus errors. This constant can not be used alone; it must be OR'd with one of the preceding constants.

The function returns the actual number of bytes popped in the location pointed to by *Actual_Size_Ptr*.

The function operates ocrrectly using both unmapped pointers and memory mapped pointers for Source_Ptr and Dest_Ptr. Local-to-local, local-to-VME, VME-to-local, and VME-to-VME pops all execute properly.

For a pop to complete, any *Source_Ptr* or *Dest_Ptr* that corresponds to a VMEbus addresses must be aligned on an address boundary equivalent to the specified *Data_Width*. Otherwise, the function returns an **EPC_INV_ALIGN** error. For example, if both *Source_Ptr* and *Dest_Ptr* correspond to VMEbus memory and *Data_Width* is **EPC_16_BIT**, then both *Source_Ptr* and *Dest_Ptr* must correspond to VMEbus addresses aligned on a 16-bit boundary for the pop to complete successfully.

For a 16-bit or 32-bit pop to complete under DOS or Windows, no individual data element may span a segment boundary. Otherwise, the function returns an **EPC_INV_ALIGN** error. For example, if *Data_Width* is **EPC_16_BIT** and *Size* is greater than 64 Kbytes, both *Source_Ptr* and *Dest_Ptr* must be aligned on a 16-bit boundary for the pop operation to complete successfully.

Return Value The function returns an EPConnect return value:

EPC_BERR	A bus error occurred during the pop.
EPC_INV_ALIGN	*Size* is not a multiple of *Data_Width*, *Source_Ptr* is mapped to a VMEbus address and is not aligned on a *Data_Width* boundary, *Dest_Ptr* is mapped to a VMEbus address and is not aligned on a *Data_Width* boundary, or a 16-bit or 32-bit data element spans a segment boundary or.
EPC_INV_PTR	One or more of *Source_Ptr*, *Dest_Ptr*, or *Actual_Size_Ptr* is invalid.
EPC_INV_RANGE	The address range defined by *Source_Ptr* and *Data_Width* and/or the address range defined by *Dest_Ptr* and *Size* contains bus addresses that are not currently mapped.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_INV_WIDTH	The *Data_Width* parameter is invalid.
EPC_SUCCESS	The function completed successfully.

See Also **EpcCopyData, EpcGetMappingAttributes, EpcMapBusMemory, EpcMapSharedMemory, EpcOpenSession, EpcPushData, EpcUnmapBusMemory, EpcUnmapSharedMemory.**

Example See **EpcCopyData.**

EpcPulseEpcLines

Description Pulses EPC control lines.

C Synopsis

#include "busmgr.h"

short FAR PASCAL
EpcPulseEpcLines(unsigned long *Session_ID*, unsigned long *Line_Mask*);

Session_ID	*Session_ID* specifies a session.
Line_Mask	*Line_Mask* specifies a mask of EPC control lines.

Visual Basic Synopsis

Declare Function
EpcPulseEpcLines% Lib "bmvxiw16.dll" (ByVal *Session_ID&*, ByVal *Line_Mask&*)

Remarks **EpcPulseEpcLines** pulses (asserts and deasserts as an atomic operation) the EPC control lines specified by *Line_Mask*.

Line_Mask is an OR'd mask of the following constants, where a set bit indicates that the function should pulse the corresponding interface control line:

Constant	Description
EPC_SYSFAIL	SYSFAIL.
EPC_SYSRESET	SYSRESET.

The function directly affects the interface control line state. Interface control line state reflects the state of bits in the interface's control line drive registers. Actual bus control line state is an OR'd combination of the states all devices on the bus. If the interface asserts a control line, the actual bus control line transitions from deasserted to asserted only if all other devices on the bus have previously deasserted the line. Likewise, if the interface deasserts a control line, the actual bus control line transitions from asserted to deasserted only if all devices on the bus have previously deasserted the line.

When pulsing the SYSRESET interface control line, the function leaves the line asserted for at least 200 milliseconds (in accordance with bus specifications). Whether pulsing the SYSRESET actual bus control line resets an EPC-7 or EPC-8 depends on the value of the interface's **EPC_SYSRESET_IN** miscellaneous attribute bit (see **EpcSetMiscAttributes**). (EPC-7 and EPC-8 only)

Whether pulsing the SYSFAIL interface control line pulses the SYSFAIL actual bus control line depends on the value of the interface's **EPC_SYSFAIL_OUT** miscellaneous attribute bit (see **EpcSetMiscAttributes**).

To pulse SYSFAIL on an EPC-7, EPC-8, or VXLink interface, the function deasserts then asserts the interface's **EPC_PASS** miscellaneous attribute bit (see **EpcSetMiscAttributes**). After pulsing SYSFAIL, the interface's **EPC_PASS** miscellaneous attribute remains asserted.

Return Value The function returns a Bus Management return value:

EPC_INV_MASK	The parameter *Line_Mask* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_SUCCESS	The function completed successfully.

See Also EpcGetBusLines, EpcGetEpcLines, EpcOpenSession, EpcSetEpcLines, EpcSetMiscAttributes.

Example See **EpcAssertInterrupt**.

EpcPulseEpcTriggers

Description Pulses interface trigger lines.

C Synopsis

```
#include "busmgr.h"

short
EpcPulseEpcTriggers(unsigned long Session_ID, unsigned long
Trigger_Mask);
```

Session_ID	*Session_ID* specifies a session.
Trigger_Mask	*Trigger_Mask* specifies a mask of interface trigger lines.

Visual Basic Synopsis

```
Declare Function
EpcPulseEpcTriggers% Lib "bmvxiw16.dll"
(ByVal Session_ID&, ByVal Trigger_Mask&)
```

Remarks **EpcPulseEpcTriggers** pulses (asserts and deasserts as an atomic operation) the interface trigger lines specified by *Trigger_Mask*.

Trigger_Mask is an OR'd mask of the following constants, where a set bit indicates that the function should pulse the corresponding interface trigger line:

Constant	Description
EPC_ECL_TRIG0	ECL trigger 0 (EPC-7 only).
EPC_ECL_TRIG1	ECL trigger 1 (EPC-7 only).
EPC_TTL_TRIG0	TTL trigger 0 (EPC-7 and VXLink only).
...	
EPC_TTL_TRIG7	TTL trigger 7 (EPC-7 and VXLink only).

The function directly affects the interface trigger line state. Interface trigger line state reflects the state of bits in the interface's trigger line drive registers. Actual bus trigger line state is an OR'd combination of the states all devices on the bus. If the interface asserts a trigger line, the actual bus trigger line transitions from deasserted to asserted only if all other devices on the bus have previously deasserted the line. Likewise, if the interface deasserts a trigger line, the actual bus trigger line transitions from asserted to deasserted only if all devices on the bus have previously deasserted the line.

Return Value The function returns a EPConnect return value:

EPC_INV_MASK	The parameter *Trigger_Mask* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_SUCCESS	The function completed successfully.

See Also **EpcGetBusTriggers, EpcGetEpcTriggers, EpcOpenSession, EpcSetEpcTriggers.**

EpcPushData

Pushes a block of data from consecutive memory locations to a single memory location.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcPushData(unsigned long          Session_ID,
            void HUGE *            Source_Ptr,
            void HUGE *            Dest_Ptr,
            unsigned long          Size,
            unsigned short         Data_Width,
            unsigned long FAR *    Actual_Size_Ptr);
```

Session_ID	*Session_ID* specifies a bus session.
Source_Ptr	*Source_Ptr* specifies the address of a data buffer from which data will be pushed.
Dest_Ptr	*Dest_Ptr* specifies the address of a FIFO queue to which data will be pushed.
Size	*Size* specifies the number of data bytes to push.
Data_Width	*Data_Width* specifies the number of data bits to push per bus access.
Actual_Size_Ptr	*Actual_Size_Ptr* specifies a location where the actual number of bytes pushed will be placed.

Remarks

EpcPushData efficiently pushes blocks of data from consecutive memory locations to a single memory location using the attributes of pointers *Source_Ptr* and *Dest_Ptr*. The intended use of the function is pushing large blocks of data to a FIFO queue.

The *Size* parameter should always express the number of bytes to be pushed, regardless of the specified *Data_Width* parameter. Passing a zero *Size* parameter results in no data being pushed.

The following constants define valid values for the *Data_Width* parameter:

Constant	Description
EPC_8_BIT	8-bit data width
EPC_8_BIT_ODD	8-bit data width, odd bytes only
EPC_16_BIT	16-bit data width
EPC_32_BIT	32-bit data width
EPC_FASTCOPY	To increase push performance, don't check for intermediate bus errors. This constant cannot be used alone; it must be OR'd with one of the preceding constants.

The function returns the actual number of bytes pushed in the location pointed to by *Actual_Size_Ptr*.

The function operates correctly using both unmapped pointers and memory mapped pointers for *Source_Ptr* and *Dest_Ptr*. local-to-local, local-to-VME, VME-to-local, and VME-to-VME pushes all execute properly.

For a push to complete, the specified *Size* must be aligned on a boundary equivalent to the specified *Data_Width*. Otherwise, the function returns an **EPC_INV_ALIGN** error. For example, if *Data_Width* is **EPC_16_BIT**, then *Size* must be a multiple of two for the push to complete successfully. If *Data_Width* is **EPC_32_BIT**, then *Size* must be a multiple of four for the push to complete successfully.

For a push to complete, any *Source_Ptr* or *Dest_Ptr* that corresponds to a VMEbus addresses must be aligned on an address boundary equivalent to the specified *Data_Width*. Otherwise, the function returns an **EPC_INV_ALIGN** error. For example, if both *Source_Ptr* and *Dest_Ptr* correspond to VMEbus memory and *Data_Width* is **EPC_16_BIT**, then both *Source_Ptr* and *Dest_Ptr* must correspond to VMEbus addresses aligned on a 16-bit boundary for the push to complete successfully.

For a 16-bit or 32-bit push to complete under DOS or Windows, no individual data element may span a segment boundary. Otherwise, the function returns an **EPC_INV_ALIGN** error. For example, if *Data_Width* is **EPC_16_BIT** and *Size* is greater than 64 Kbytes, both *Source_Ptr* and *Dest_Ptr* must be aligned on a 16-bit boundary for the push operation to complete successfully.

Return Value The function returns a EPConnect return value:

EPC_BERR	A bus error occurred during the push.
EPC_INV_ALIGN	*Size* is not a multiple of *Data_Width*, *Source_Ptr* is mapped to a VMEbus address and is not aligned on a *Data_Width* boundary, *Dest_Ptr* is mapped to a VMEbus address and is not aligned on a *Data_Width* boundary, or a 16-bit or 32-bit data element spans a segment boundary.
EPC_INV_PTR	One or more of *Source_Ptr*, *Dest_Ptr*, or *Actual_Size_Ptr* is invalid.
EPC_INV_RANGE	The address range defined by *Source_Ptr* and *Size* and/or the address range defined by *Dest_Ptr* and *Data_Width* contains bus addresses that are not currently mapped.
EPC_INV_SESSION	The specified *Session_ID* is invalid.

EPC_INV_SW	The BusManager device driver is not present.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_INV_WIDTH	The *Data_Width* parameter is invalid.
EPC_SUCCESS	The function completed successfully.

See Also **EpcCopyData, EpcGetMappingAttributes, EpcMapBusMemory, EpcMapSharedMemory, EpcOpenSession, EpcPopData, EpcUnmapBusMemory, EpcUnmapSharedMemory.**

Example See **EpcCopyData.**

EpcSetBusAttributes

Description Defines the interface's bus management attributes.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcSetBusAttributes( unsigned long  Session_ID,
                     unsigned short Bus_Enable,
                     unsigned short Bus_Arb_Mode,
                     unsigned short Bus_Arb_Priority,
                     unsigned short Bus_Release);
```

Session_ID	*Session_ID* specifies a session.
Bus_Enable	*Bus_Enable* specifies the interface's bus enable attribute.
Bus_Arb_Mode	*Bus_Arb_Mode* specifies the interface's bus arbitration mode.
Bus_Arb_Priority	*Bus_Arb_Priority* specifies the interface's bus arbitration priority.
Bus_Release	*Bus_Release* specifies the interface's bus release mode.

Visual Basic Synopsis

```
Declare Function
EpcSetBusAttributes% Lib "bmvxiw16.dll"
                     (ByVal Session_ID&,
                     ByVal Bus_Enable%,
                     ByVal Bus_Arb_Mode%,
                     ByVal Bus_Arb_Priority%,
                     ByVal Bus_Release%)
```

Remarks

EpcSetBusAttributes defines the interface's bus management attributes.

Bus_Enable specifies the interface's bus enable attribute. The interface's bus enable attribute determines whether accesses made by the interface reach the bus. Valid *Bus_Enable* values are:

Bus_Enable	Description
EPC_DISABLE_BUS	Disable bus accesses for the interface (Supported on EPC-7 and EPC-8 only).
EPC_ENABLE_BUS	Enable bus accesses for the interface.

Bus_Arb_Mode specifies the interface's bus arbitration mode. The interface's bus arbitration mode defines how the interface arbitrates bus collisions. The interface's bus arbitration mode only affects bus accesses if the interface has been designated the VMEbus slot-1 controller or VXIbus slot-0 controller. Valid *Bus_Arb_Mode* values are:

Bus_Arb_Mode	Description
EPC_PRIORITY	Priority bus arbitration.
EPC_ROUND_ROBIN	Round-robin bus arbitration.

Bus_Arb_Priority specifies the interface's bus arbitration priority. The interface's bus arbitration priority defines the priority level at which the interface arbitrates for the bus. Possible values placed at *Bus_Arb_Priority* are:

Bus_Arb_Priority	Description
EPC_PRIORITY0	Bus arbitration priority 0.
EPC_PRIORITY1	Bus arbitration priority 1.
EPC_PRIORITY2	Bus arbitration priority 2.
EPC_PRIORITY3	Bus arbitration priority 3.

Bus_Release specifies the interface's bus release mode. The interface's bus release mode determines when the interface requests and/or releases the bus. Valid *Bus_Release* values are:

Bus_Release	Description
EPC_ROR	"Release On Request" bus release mode.
EPC_RONR	"Request On No Request" bus release mode.

Return Value The function returns a Bus Management return value:

EPC_INV_ARB_MODE	The parameter *Bus_Arb_Mode* is invalid.
EPC_INV_ARB_PRIO	The parameter *Bus_Arb_Priority* is invalid.
EPC_INV_ENABLE	The parameter *Bus_Enable* is invalid.
EPC_INV_RELEASE	The parameter *Bus_Release* is invalid.
EPC_INV_SESSION	The parameter *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_INV_TIMEOUT	An invalid timeout was encountered.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_SUCCESS	The function completed successfully.

See Also **EpcGetBusAttributes, EpcOpenSession.**

Example See **EpcGetBusAttributes.**

EpcSetEpcLines

Description Defines the interface control line state.

C Synopsis

#include "busmgr.h"

short FAR PASCAL
EpcSetEpcLines(unsigned long *Session_ID*, unsigned long *Line_Mask*);

Session_ID	*Session_ID* specifies a session.
Line_Mask	*Line_Mask* specifies an interface control line state.

Visual Basic Synopsis

Declare Function
EpcSetEpcLines% Lib "bmvxiw16.dll" (ByVal *Session_ID*&, ByVal *Line_Mask*&)

Remarks **EpcSetEpcLines** defines the interface control line state as specified by *Line_Mask*.

Line_Mask is either zero or an OR'd bit mask of the following constants. A set bit indicates that the function should assert the corresponding interface control line. A clear bit indicates that the function should deassert the corresponding interface control line:

Constant	Description
EPC_SYSFAIL	SYSFAIL.
EPC_SYSRESET	SYSRESET.

The function directly affects the interface control line state. Interface control line state reflects the state of bits in the interface's control line drive registers. Actual bus control line state is an OR'd combination of the states all devices on the bus. If the interface asserts a control line, the actual bus control line transitions from deasserted to asserted only if all other devices on the bus have previously deasserted the line. Likewise, if the interface deasserts a control line, the actual bus control line transitions from asserted to deasserted only if all devices on the bus have previously deasserted the line.

Whether asserting the SYSRESET actual bus control line resets the interface on an EPC-7 or EPC-8 depends on the value of the interface's **EPC_SYSRESET_IN** miscellaneous attribute bit (see **EpcSetMiscAttributes**).

Whether asserting or deasserting the SYSFAIL interface control line asserts or deasserts the SYSFAIL actual bus control line depends on the value of the interface's **EPC_SYSFAIL_OUT** miscellaneous attribute bit (see **EpcSetMiscAttributes**).

To assert or deassert SYSFAIL on an EPC-7, EPC-8, or VXLink interface, the function deasserts or asserts the interface's **EPC_PASS** miscellaneous attribute bit (see **EpcSetMiscAttributes**). After asserting SYSFAIL, the interface's **EPC_PASS** miscellaneous attribute remains deasserted. Likewise, after deasserting SYSFAIL, the interface's **EPC_PASS** miscellaneous attribute remains asserted.

Return Value The function returns a Bus Management return value:

EPC_INV_MASK	The parameter *Line_Mask* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_SUCCESS	The function completed successfully.

See Also EpcGetBusLines, EpcGetEpcLines, EpcOpenSession, EpcPulseEpcLines, EpcSetMiscAttributes.

Example See EpcAssertInterrupt.

EpcSetEpcMODID

Description Defines interface MODID line state.

C Synopsis

```
#include "busmgr.h"

short
EpcSetEpcMODID(unsigned long Session_ID, unsigned long
MODID_Mask);
```

Session_ID	*Session_ID* specifies a session.
MODID_Mask	*MODID_Mask* specifies an interface MODID line state.

Visual Basic Synopsis

```
Declare Function
EpcSetEpcMODID% Lib "bmvxiw16.dll"
(ByVal Session_ID&, ByVal MODID_Mask&)
```

Remarks **EpcSetEpcMODID** defines the interface MODID line state as specified by *MODID_Mask*.

MODID_Mask is either zero or an OR'd bit mask of the following constants. A set bit indicates that the function should assert the corresponding interface MODID line. A clear bit indicates that the function should deassert the corresponding interface MODID line:

Constant	Description
EPC_MODID0	MODID line 0 (EPC-7 and VXLink only).
...	
EPC_MODID12	MODID line 12 (EPC-7 and VXLink only).

Only the VXIbus slot-0 controller device can assert or deassert the actual bus MODID lines. When an interface is the VXIbus slot-0 controller, defining the interface MODID line state also defines the actual bus MODID line state. When an interface is not the VXIbus slot-0 controller, defining the interface MODID line state has no effect on the actual bus MODID lines.

Return Value The function returns an EPConnect return value:

EPC_INV_MASK	The parameter *MODID_Mask* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The Bus Manager device driver is not present.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_SUCCESS	The function completed successfully.

See Also **EpcGetBusMODID, EpcOpenSession.**

EpcSetEpcTriggers

Description Defines the interface trigger line state.

C Synopsis

```
#include "busmgr.h"

short
EpcSetEpcTriggers(unsigned long Session_ID, unsigned long
Trigger_Mask);
```

Session_ID	*Session_ID* specifies a session.
Trigger_Mask	*Trigger_Mask* specifies an interface bus control line state.

Visual Basic Synopsis

```
Declare Function

EpcSetEpcTriggers% Lib "bmvxiw16.dll" (ByVal Session_ID&,
ByVal Trigger_Mask&)
```

Remarks **EpcSetEpcTriggers** defines the interface trigger line state as specified by *Trigger_Mask*.

Trigger_Mask is either zero or an OR'd bit mask of the following constants. A set bit indicates that the function should assert the corresponding interface trigger line.

A clear bit indicates that the function should deassert the corresponding interface trigger line:

Constant	Description
EPC_ECL_TRIG0	ECL trigger 0 (EPC-7 only).
EPC_ECL_TRIG1	ECL trigger 1 (EPC-7 only).
EPC_TTL_TRIG0	TTL trigger 0 (EPC-7 and VXLink only).
...	
EPC_TTL_TRIG7	TTL trigger 7 (EPC-7 and VXLink only).

The function directly affects the interface trigger line state. interface trigger line state reflects the state of bits in the interface's trigger line drive registers. Actual bus trigger line state is an OR'd combination of the states all devices on the bus. If the interface asserts a trigger line, the actual bus trigger line transitions from deasserted to asserted only if all other devices on the bus have previously deasserted the line. Likewise, if the interface deasserts a trigger line, the actual bus control line transitions from asserted to deasserted only if all devices on the bus have previously deasserted the line.

Return Value

The function returns a EPConnect return value:

EPC_INV_MASK	The parameter *Trigger_Mask* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The Bus Manager device driver is not present.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_SUCCESS	The function completed successfully.

See Also

EpcGetBusTriggers, EpcGetEpcTriggers, EpcOpenSession, EpcPulseEpcTriggers.

EpcSetEventEnableMask

Description Defines a session's enabled event mask attribute.

C Synopsis

#include "busmgr.h"

short
EpcSetEventEnableMask(unsigned long *Session_ID*, unsigned long *Event_Mask*);

Session_ID	*Session_ID* specifies a session.
Event_Mask	*Event_Mask* specifies a mask of enabled events.

Visual Basic Synopsis

Declare Function
EpcSetEventEnableMask% Lib "bmvxiw16.dll" (ByVal *Session_ID&*, ByVal *Event_Mask&*)

Remarks **EpcSetEventEnableMask** sets the specified session's enabled event mask attribute to *Event_Mask*.

The *Event_Mask* parameter is a bit mask where each bit corresponds to an event. The *Event_Mask* parameter should be either zero or an OR'd combination of the following constants:

Event	Description
EPC_MSG_INT	Message interrupt (EPC-7 and EPC-8 only)
EPC_VME1_INT	VMEbus interrupt 1
...	
EPC_VME7_INT	VMEbus interrupt 7
EPC_SIGNAL_INT	VXIbus signal FIFO interrupt
EPC_TTL_TRIG0_INT	VXIbus TTL Trigger 0 interrupt (EPC-7 only)
...	
EPC_TTL_TRIG7_INT	VXIbus TTL Trigger 7 interrupt (EPC-7 only)
EPC_SYSRESET_ERR	VMEbus SYSRESET error
EPC_ACFAIL_ERR	VMEbus power failure error
EPC_BERR_ERR	VMEbus access error
EPC_SYSFAIL_ERR	VMEbus SYSFAIL error
EPC_WATCHDOG_ERR	Watchdog timer expiration error (EPC-7 and EPC-8 only)
EPC_EXT_TRIG0_INT	External trigger 0 interrupt (VXLink only)
EPC_EXT_TRIG1_INT	External trigger 1 interrupt (VXLink only)

Return Value The function returns a Bus Management return value:

EPC_INV_MASK	*Event_Mask* contains enabled events that are not valid for this interface.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_SUCCESS	The function completed successfully.

See Also **EpcGetEventEnableMask, EpcOpenSession.**

Example See **EpcGetEventEnableMask.**

EpcSetEventHandler

Description Defines an entry in a session's event handler array.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcSetEventHandler( unsigned long  Session_ID,
                    unsigned long  Event_Mask,
                    void (FAR *    Event_Handler)
                          (unsigned long,
                          unsigned long,
                          unsigned long),
                    void FAR *     Stack_Ptr);
```

Session_ID	*Session_ID* specifies a session.
Event_Mask	*Event_Mask* specifies an event.
Event_Handler	*Event_Handler* specifies an event handler.
Stack_Ptr	*Stack_Ptr* specifies an event handler stack pointer.

Visual Basic Synopsis

```
Declare Function
EpcSetEventHandler% Lib "bmvxiw16.dll"
                    (ByVal Session_ID&,
                    ByVal Event_Mask&,
                    Event_Handler As Any,
                    Stack_Ptr As Any);
```

Remarks **EpcSetEventHandler** sets the specified session's specified event handler array entry to *Event_Handler* and the event handler's stack pointer to *Stack_Ptr*.

The *Event_Mask* parameter is a bit mask where each bit corresponds to an event. The *Event_Mask* parameter should be one of the following constants:

Event	Description
EPC_MSG_INT	Message interrupt (EPC-7 and EPC-8 only)
EPC_VME1_INT	VMEbus interrupt 1
...	
EPC_VME7_INT	VMEbus interrupt 7
EPC_SIGNAL_INT	VXIbus signal FIFO interrupt
EPC_TTL_TRIG0_INT	VXIbus TTL Trigger 0 interrupt (EPC-7 only)
...	
EPC_TTL_TRIG7_INT	VXIbus TTL Trigger 7 interrupt (EPC-7 only)
EPC_SYSRESET_ERR	VMEbus SYSRESET error
EPC_ACFAIL_ERR	VMEbus power failure error
EPC_BERR_ERR	VMEbus access error
EPC_SYSFAIL_ERR	VMEbus SYSFAIL error
EPC_EXT_TRIG0_INT	External trigger 0 interrupt (VXLink only)
EPC_EXT_TRIG1_INT	External trigger 1 interrupt (VXLink only)
EPC_WATCHDOG_ERR	Watchdog timer expiration error (EPC-7 and EPC-8 only)

The *Event_Handler* parameter is a pointer to an event handler function with the following call semantics:

```
void FAR
EventHandlerFunction(unsigned long Session_ID,
            unsigned long Handler_Mask,
            unsigned long Handler_Data);
```

The event handler function should return to the caller using a normal RET instruction. It should not attempt to return using an IRET instruction.

The value passed in the event handler function's *Session_ID* parameter specifies the session that received the event. The value passed in the event handler function's *Handler_Mask* parameter specifies the event that caused execution of the event handler function.

Whether or not the event handler function receives a meaningful *Handler_Data* parameter depends on the value of *Handler_Mask*:

Handler_Mask	*Handler_Data*
EPC_MSG_INT	0
EPC_VME1_INT ... EPC_VME7_INT	VMEbus interrupt status/id (zero-extended to 32 bits)
EPC_SIGNAL_INT	VXIbus signal data (zero extended to 32 bits)
EPC_TTL_TRIG0_INT ... EPC_TTL_TRIG7_INT	0
EPC_ACFAIL_ERR	0
EPC_BERR_ERR	0
EPC_SYSFAIL_ERR	0
EPC_WATCHDOG_ERR	0
EPC_SYSRESET_ERR	0
EPC_EXT_TRIG0_INT	0
EPC_EXT_TRIG1_INT	0

The *Stack_Ptr* parameter is a pointer to the bottom of a block of memory reserved for use as a stack.

Defining a NULL event handler and/or event handler stack pointer effectively removes any previously assigned event handler and event handler stack pointer.

Defining an event handler and an event handler stack pointer does not enable or disable reception of the corresponding event. A separate call to **EpcSetEventEnableMask** is required.

Bus Management for Windows calls an event handler exactly once for each occurrence of its corresponding event and disables virtual processor interrupts before an event handler is called. The table below describes the algorithm used by EPConnect/VXI in processing each event type:

Event	Algorithm
EPC_MSG_INT	For each session with the event enabled and a handler installed, the IRQ handler disables the event and calls the installed event handler. To receive additional message interrupt events, a session must re-enable the event. To avoid redundant message interrupt events, a session should only re-enable the event after receiving a word serial command (using **EpcSrvReceiveWSCommand**) or sending a word serial command response (using **EpcSrvSendWSResponse**).
EPC_VME1_INT ... EPC_VME7_INT	The IRQ handler acknowledges the VMEbus interrupt and gets the status/id data. For each session with the event enabled and a handler installed, the IRQ handler calls the installed event handler. Additional events occur whenever additional VMEbus interrupts are asserted on the bus.

EPC_SIGNAL_INT	The IRQ handler gets the signal data from the signal FIFO. For each session with the event enabled and a handler installed, the IRQ handler calls the installed event handler Additional events occur whenever a device writes to the interface's signal FIFO.
EPC_TTL_TRIG0_INT ... **EPC_TTL_TRIG7_INT**	For each session with the event enabled and a handler installed, the IRQ handler disables the event and calls the installed event handler. To receive additional TTL trigger interrupt events for a specific TTL trigger, a session must re-enable the event. To ensure that a previous TTL trigger assertion does not cause redundant events, a session should wait for the deassertion of the corresponding TTL trigger latch bit (using **EpcGetMiscAttributes**) before re-enabling a TTL trigger interrupt event.

EPC_ACFAIL_ERR	For each session with the event enabled and a handler installed, the IRQ handler disables the event and calls the installed event handler. To receive additional ACFAIL error events, a session must re-enable the event. To ensure that the previous ACFAIL assertion does not cause redundant events, a session should wait for the deassertion of the ACFAIL bus control line (using **EpcGetBusLines**) before re-enabling the ACFAIL error event.
EPC_BERR_ERR	The IRQ handler clears the BERR condition. For each session with the event enabled and a handler installed, the IRQ handler calls the installed event handler. Additional BERR error events occur whenever the interface makes a bus access that terminates with a BERR condition.

EPC_SYSFAIL_ERR	For each session with the event enabled and a handler installed, the IRQ handler disables the event and calls the installed event handler. To receive additional SYSFAIL error events, a session must re-enable the event. To ensure that the previous SYSFAIL assertion does not cause a redundant event, a session should wait for the deassertion of the SYSFAIL bus control line (using **EpcGetBusLines**) before re-enabling the SYSFAIL error event.
EPC_WATCHDOG_ERR	For each session with the event enabled and a handler installed, the IRQ handler disables the event and calls the installed event handler. To receive additional watchdog timer error events, a session must re-enable the watchdog timer error event. To ensure that the previous watchdog timer expiration does not cause redundant events, a session should reset the watchdog timer (using **EpcWatchdogTimer**) before re-enabling the watchdog timer error event.

EPC_SYSRESET_ERR	The IRQ handler re-initializes the hardware interface. For each session with the event enabled and a handler installed, the IRQ handler disables the event and calls the installed event handler. To receive additional SYSRESET error events, a session must re-enable the event. To ensure that the previous SYSRESET assertion does not cause redundant events, a session should wait for the deassertion of the SYSRESET bus control line (using **EpcGetBusLines**) before re-enabling the SYSRESET error event.
EPC_EXT_TRIG0_INT EPC_EXT_TRIG1_INT	For each session with the event enabled and a handler installed, the IRQ handler disables the event and calls the installed event handler. Additional events occur whenever additional external trigger events are detected.

Return Value The function returns a Bus Management return value:

EPC_INV_MASK	*Event_Mask* contains more than one event or contains an event that is not valid for this EPC.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_SUCCESS	The function completed successfully.

See Also **EpcGetBusLines, EpcGetEventHandler, EpcGetMiscAttributes, EpcOpenSession, EpcSetEventEnableMask, EpcSrvReceiveWSCommand, EpcSrvSendWSResponse, EpcWatchdogTimer.**

Example See **EpcGetEventEnableMask.**

EpcSetLockingTimeout

Description Defines a session's locking timeout.

C Synopsis

#include "busmgr.h"

short FAR PASCAL
EpcSetLockingTimeout(unsigned long *Session_ID*, unsigned long *Timeout*);

Session_ID	*Session_ID* specifies a session.
Timeout	*Timeout* specifies a locking timeout.

Visual Basic Synopsis

Declare Function
EpcSetLockingTimeout% Lib "bmvxiw16.dll" (ByVal *Session_ID&*, ByVal *Timeout&*)

Remarks **EpcSetLockingTimeout** defines the specified session's locking timeout.

Timeout specifies the session's locking timeout, in milliseconds.

By default, a session has a locking timeout of zero milliseconds. When the session encounters a locking conflict, an **EPC_LOCKED** error is returned immediately.

Return Value The function returns a Bus Management return value:

EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_SUCCESS	The function completed successfully.

See Also **EpcGetLockingTimeout**, **EpcLockSession**, **EpcOpenSession**.

Example See **EpcGetLockingTimeout**.

EpcSetMiscAttributes

Description Defines the interface's miscellaneous configuration attributes.

C Synopsis

#include "busmgr.h"

short FAR PASCAL
EpcSetMiscAttributes(unsigned long *Session_ID*, unsigned long *Misc_Mask*);

Session_ID	*Session_ID* specifies a session.
Misc_Mask	*Misc_Mask* specifies miscellaneous interface configuration attributes.

Visual Basic Synopsis

Declare Function
EpcSetMiscAttributes% Lib "bmvxiw16.dll" (ByVal *Session_ID&*, ByVal *Misc_Mask&*)

Remarks **EpcSetMiscAttributes** defines miscellaneous interface configuration attributes.

Misc_Mask is either zero or an OR'd bit mask of the following constants, where a set bit indicates that the function should assert the corresponding miscellaneous interface attribute bit. A clear bit indicates that the function should deassert the corresponding miscellaneous interface attribute bit:

Constant	Description
EPC_DIR	Word serial byte transfer protocol DIR bit. Asserting the bit indicates that the interface is ready to receive data from its commander device. Supported on EPC-7 and EPC-8 only.

EPC_DOR	Word serial byte transfer protocol DOR bit. Asserting the bit indicates that the interface is ready to send data to its commander device. Supported on EPC-7 and EPC-8 only.
EPC_ERR	Word serial protocol ERR* bit. Asserting the bit indicates to the commander device that the interface has detected a word serial protocol error. Supported on EPC-7 and EPC-8 only.
EPC_LOCK	Message-based device Locked* bit. Asserting the bit indicates that the commander device has locked access to the interface from other local sources. Supported on EPC-7 and EPC-8 only.
EPC_MULTIPLE_LOCK	Word serial protocol extension multiple commander lock bit. When asserted, the first commander to read the asserted bit from the interface's Response register can safely send a word serial command. Supported on EPC-7 and EPC-8 only.
EPC_PASS	Device initialization PASSED bit. Asserting the bit indicates that the interface has passed self-test.
EPC_READY	Device initialization READY bit. Asserting the bit indicates that the interface is ready to begin normal operation.
EPC_RESET	Interface reset bit. Asserting the bit places the interface in VXI "soft reset" state.

EPC_RRDY	Word serial protocol Read Ready bit. Asserting the bit indicates to a commander device that the interface has a word serial response in its message register. Supported on EPC-7 and EPC-8 only.
EPC_RSRC_MGR	Interface resource manager execution bit. Asserting the bit indicates that resource manager execution is complete.
EPC_STICKY_BERR	"Sticky" bus error bit. When asserted, the bit indicates that a bus error has occurred since the bit was last deasserted. This bit cannot be asserted directly by software; it can only be deasserted.
EPC_SYSFAIL_OUT	SYSFAIL output enable bit. When asserted, the interface can assert SYSFAIL. When deasserted, the interface cannot assert SYSFAIL.
EPC_SYSRESET_IN	SYSRESET input enable bit. When asserted, asserting SYSRESET resets the interface. When deasserted, asserting SYSRESET does not reset the interface.
EPC_WRDY	Word serial protocol Write Ready bit. Asserting the bit indicates to a commander device that the interface is ready to receive a word serial command. Supported on EPC-7 and EPC-8 only.

Deasserting **EPC_PASS** while asserting **EPC_SYSFAIL_OUT** causes the interface to assert SYSFAIL on the bus.

Return Value The function returns a Bus Management return value:

EPC_INV_MASK	The parameter *Misc_Mask* is invalid.
EPC_INV_SESSION	The parameter *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_INV_TIMEOUT	An invalid timeout was encountered.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_SUCCESS	The function completed successfully.

See Also **EpcGetMiscAttributes**, **EpcOpenSession**.

Example See **EpcGetBusAttributes**.

EpcSetSessionData

Description Defines a session's application-specified data.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcSetSessionData( unsigned long Session_ID,
                   unsigned long Session_Data);
```

Session_ID	*Session_ID* specifies an open session.
Session_Data	*Session_Data* specifies the session's application-specified data.

Visual Basic Synopsis

```
Declare Function
EpcSetSessionData% Lib "bmvxiw16.dll" (ByVal Session_ID&,
ByVal Session_Data&)
```

Remarks **EpcSetSessionData** defines the specified session's application-specified data.

The application-specified data is a 4-byte quantity.

Typically, an application uses **EpcSetSessionData** to store a pointer to one of its data structures. Later, the application uses **EpcGetSessionData** to quickly retrieve the pointer during performance-critical operations (like event handling).

Return Value The function returns a Bus Management return value:

EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_SUCCESS	The function completed successfully.

See Also **EpcGetSessionData, EpcOpenSession.**

Example See **EpcCloseSession.**

EpcSetSlaveMapping

Description Defines the interface's slave memory mapping.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcSetSlaveMapping( unsigned long  Session_ID,
                    unsigned short Address_Space,
                    unsigned long  Base_Address);
```

Session_ID	*Session_ID* specifies a session.
Address_Space	*Address_Space* specifies a slave memory address space.
Base_Address	*Base_Address* specifies a slave memory base address.

Visual Basic Synopsis

```
Declare Function
EpcSetSlaveMapping% Lib "bmvxiw16.dll" (ByVal Session_ID&,
                    ByVal Address_Space%,
                    ByVal Base_Address&)
```

Remarks **EpcSetSlaveMapping** defines the mapping of the interface's slave memory to the bus.

Address_Space specifies whether the interface's slave memory appears on the bus, and if so, in which address space. *Base_Address* specifies the base address of the interface's slave memory in the given *Address_Space*.

Valid combinations of *Address_Space* and *Base_Address* are dependent on the interface type:

Interface Type	*Address_Space*	*Base_Address*
EPC-7	**EPC_DISABLED**	N/A
	EPC_A24	0x00000000, 0x00400000, ..., 0x00C00000
	EPC_A32	0x00000000, 0x01000000, ..., 0xFF000000
EPC-8	**EPC_DISABLED**	N/A
VXLink	**EPC_DISABLED**	N/A

A24 base addresses are aligned on a 4 Mbyte boundary, and only the first 4 Mbytes of the interface's slave memory is mapped to the bus. A32 base addresses are aligned on a 16 Mbyte boundary, and only the first 16 Mbytes of the interface's slave memory is mapped to the bus.

Return Value The function returns a Bus Management return value:

EPC_INV_BASE	The parameter *Base_Address* is invalid.
EPC_INV_SESSION	The parameter *Session_ID* is invalid.
EPC_INV_SPACE	The parameter *Address_Space* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_INV_TIMEOUT	An invalid timeout was encountered.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_SUCCESS	The function completed successfully.

See Also **EpcGetSlaveMapping, EpcOpenSession.**

Example See **EpcGetBusAttributes.**

EpcSetULA

Description Defines the interface's unique logical address.

C Synopsis

#include "busmgr.h"

short FAR PASCAL
EpcSetULA(unsigned long *Session_ID*, unsigned short *ULA*);

Session_ID	*Session_ID* specifies a session.
ULA	*ULA* specifies the interface's unique logical address.

Visual Basic Synopsis

Declare Function
EpcSetULA% Lib "bmvxiw16.dll" (ByVal *Session_ID&*, ByVal *ULA%*)

Remarks **EpcSetULA** defines the interface's unique logical address.

Valid unique logical address values are 0x00 through 0xFF.

Return Value The function returns a Bus Management return value:

EPC_INV_SESSION	The parameter *Session_ID* is invalid.
EPC_INV_TIMEOUT	An invalid timeout was encountered.
EPC_INV_ULA	The parameter *ULA* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_SUCCESS	The function completed successfully.

See Also EpcGetULA, EpcOpenSession.

Example See EpcGetBusAttributes.

EpcSrvEnableWSCommand

Description Enables word serial command reception.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcSrvEnableWSCommand
        (unsigned long Session_ID,
        unsigned short Enable_Next_Command);
```

Session_ID	*Session_ID* specifies a session.
Enable_Next_Command	*Enable_Next_Command* specifies the type of word serial command reception to enable.

Visual Basic Synopsis

```
Declare Function
EpcSrvEnableWSCommand% Lib "bmvxiw16.dll" (ByVal
Session_ID&, ByVal Enable_Next_Command%)
```

Remarks **EpcSrvEnableWSCommand** configures the interface hardware to receive a word serial command.

The following constants specify valid values for the *Enable_Next_Command* parameter:

Constant	Description
EPC_DISABLE_ALL	Disable word serial command reception.
EPC_ENABLE_WRDY	Enable word serial command reception by asserting WRDY and deasserting both DIR and DOR.
EPC_ENABLE_DIR	Enable word serial command reception and data input by asserting both WRDY and DIR and deasserting DOR (EPC-7 and EPC-8 only).
EPC_ENABLE_DOR	Enable word serial command reception and data output by asserting both WRDY and DOR and deasserting DIR (EPC-7 and EPC-8 only).
EPC_ENABLE_ALL	Enable word serial command reception, data input, and data output by asserting WRDY, DIR, and DOR (EPC-7 and EPC-8 only).

Disabling word serial command reception when it is already enabled and without receiving a word serial command can result in a word serial protocol violation by allowing the commander device to write an unexpected word serial command.

On an EPC-7, enabling word serial command reception when an outgoing word serial command response remains unread in the interface's message registers can result in a word serial protocol violation by allowing the commander device to write over the word serial command response.

Enabling word serial command reception when it is already enabled can result in a word serial protocol violation. In particular, enabling word serial command reception with DIR deasserted when word serial command reception is already enabled with DIR asserted can generate a DIR violation. Likewise, enabling word serial command reception with DOR deasserted when word serial command reception is already enabled with DOR asserted can generate a DOR violation.

EPConnect does not support enabling word serial command reception on a VXLink interface. Attempting to use the function on a VXLink interface results in an **EPC_INV_HW** error.

Return Value The function returns a Bus Management return value:

EPC_INV_ENABLE	The parameter *Enable_Next_Command* is invalid.
EPC_INV_HW	The interface does not support enabling word serial command reception.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_SUCCESS	The function completed successfully.

See Also **EpcOpenSession, EpcSrvReceiveWSCommand.**

EpcSrvReceiveWSCommand

Description Receives a word serial command from a commander device.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcSrvReceiveWSCommand
        (unsigned long          Session_ID,
        void FAR *              Command_Ptr,
        unsigned short FAR *    Command_Size_Ptr,
        unsigned short          Enable_Next_Command,
        unsigned long           Timeout);
```

Session_ID	*Session_ID* specifies a session.
Command_Ptr	*Command_Ptr* specifies a location where the word serial command will be placed.
Command_Size_Ptr	*Command_Size_Ptr* specifies a location where the size of the word serial command will be placed.
Enable_Next_Command	*Enable_Next_Command* specifies whether to enable the interface hardware to receive the another word serial command.
Timeout	*Timeout* specifies the number of milliseconds to wait for a word serial command.

Visual Basic Synopsis

```
Declare Function
EpcSrvReceiveWSCommand% Lib "bmvxiw16.dll"
                        (ByVal Session_ID&,
                        Command_Ptr As Any,
                        Command_Size_Ptr%,
                        ByVal Enable_Next_Command%,
                        ByVal Timeout&)
```

Remarks

EpcSrvReceiveWSCommand receives a word serial command and places the command and its size in the locations pointed to by *Command_Ptr* and *Command_Size_Ptr*, respectively. The function then configures the interface hardware for future word serial command reception.

Command_Size_Ptr points to a location where the function places the size of the received word serial command:

**Command_Size_Ptr*	Description
EPC_16_BIT	Received a 16-bit word serial command.
EPC_32_BIT	Received a 32-bit long word serial command (EPC-7 only).

The following constants specify valid values for the *Enable_Next_Command* parameter:

Constant	Description
EPC_DISABLE_ALL	Disable word serial command reception.
EPC_ENABLE_WRDY	Enable word serial command reception by asserting WRDY and deasserting both DIR and DOR.
EPC_ENABLE_DIR	Enable word serial command reception and data input by asserting both WRDY and DIR and deasserting DOR.
EPC_ENABLE_DOR	Enable word serial command reception and data output by asserting both WRDY and DOR and deasserting DIR .
EPC_ENABLE_ALL	Enable word serial command reception, data input, and data output by asserting WRDY, DIR, and DOR.

Word serial command reception must be enabled before attempting to receive a word serial command. Otherwise, **EpcSrvReceiveWSCommand** returns invalid word serial command data. Use **EpcSrvEnableWSCommand** to enable initial word serial command reception.

Occasionally, it's useful to receive a word serial command without destroying the contents of the interface's message registers. To receive a word serial command without destroying the contents of the interface's message registers, use **EpcSrvReceiveWSCommand** with an *Enable_Next_Command* parameter value of **EPC_DISABLE_ALL**. This allows a subsequent call to **EpcSrvReceiveWSCommand** to receive the same word serial command. Note that either enabling word serial command reception or (on an EPC-7) sending a word serial command response overwrites the contents of the interface's message registers, destroying any data preserved there.

On an EPC-7 or EPC-8, the function returns **EPC_DIR_ERR** when a Byte Available or Trigger word serial command is received and the interface is not enabled for data input (e.g., interface's DIR bit is clear). Likewise, on an EPC-7 or EPC-8, the function returns **EPC_DOR_ERR** when a Byte Request word serial command is received and the interface is not enabled for data output (e.g., the interface's DOR bit is clear). Use **EpcSrvSendWSProtocolError** to send protocol errors to the commander device.

EPConnect does not support receiving a word serial command on a VXLink interface. Attempting to use the function on a VXLink interface results in an **EPC_INV_HW** error.

Return Value

The function returns a Bus Management return value:

EPC_DIR_ERR	A word serial command protocol DIR violation error occurred.
EPC_DOR_ERR	A word serial command protocol DOR violation error occurred.
EPC_INV_ENABLE	The parameter *Enable_Next_Command* is invalid.
EPC_INV_HW	The interface does not support enabling word serial command reception.
EPC_INV_PTR	One or more of parameters *Command_Ptr* and *Command_Size_Ptr* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_RECV_TIMEOUT	A timeout occurred waiting for a word serial command.
EPC_SUCCESS	The function completed successfully.

See Also EpcOpenSession, EpcSrvEnableWSCommand, EpcSrvSendWSResponse.

EpcSrvSendProtocolEvent

Description Sends a protocol event to the commander device.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcSrvSendProtocolEvent(  unsigned long   Session_ID,
                          unsigned short  ULA,
                          unsigned long   Method_Mask,
                          unsigned short  Protocol_Event);
```

Session_ID	*Session_ID* specifies a session.
ULA	*ULA* specifies the unique logical address of the commander device.
Method_Mask	*Method_Mask* specifies the method to use for sending the protocol event.
Protocol_Event	*Protocol_Event* specifies a protocol event.

ULA is unused when *Method_Mask* specifies a VMEbus interrupt.

Visual Basic Synopsis

```
EpcSrvSendProtocolEvent% Lib "bmvxiw16.dll"
                         (ByVal Session_ID&,
                         ByVal ULA%,
                         ByVal Method_Mask&,
                         ByVal Protocol_Event%)
```

Remarks **EpcSrvSendProtocolEvent** sends the VMEbus protocol event *Protocol_Event* to the commander device at unique logical address *ULA*.

Method_Mask specifies the method to use for sending the protocol event. Valid values are:

Method_Mask	Description
EPC_VME1_INT	VMEbus interrupt 1 (EPC-7 only).
...	
EPC_VME7_INT	VMEbus interrupt 7 (EPC-7 only).
EPC_SIGNAL_REG	Write to the commander device's signal register.

The function always overwrites the lower eight bits of the specified *Protocol_Event* parameter with the unique logical address of the interface.

On an EPC-7, using a VMEbus interrupt to send a protocol event requires the use of the EPC-7's message high register. Receiving 32-bit long word serial commands also requires the use of the EPC-7's message high register. Therefore, using a VMEbus interrupt to send a protocol event while simultaneously receiving 32-bit long word serial commands can have unpredictable results. Note, however, that no conflict occurs when using a VMEbus interrupt to send a protocol event while simultaneously receiving 16-bit word serial commands.

Return Value The function returns a Bus Management return value:

EPC_BERR	A bus error occurred writing the protocol event into the commander device's signal register.
EPC_INV_ASSERT	The interface is already asserting a VMEbus interrupt.
EPC_INV_EVENT	The parameter *Protocol_Event* is invalid.
EPC_INV_METHOD	The parameter *Method_Mask* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_INV_ULA	The parameter *ULA* is invalid.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_SUCCESS	The function completed successfully.

See Also **EpcOpenSession.**

EpcSrvSendWSProtocolError

Description Sends a word serial protocol error to the commander device.

C Synopsis

```
#include "busmgr.h"

short
EpcSrvSendWSProtocolError
                    (unsigned long Session_ID,
                    unsigned short Protocol_Error,
                    unsigned short Enable_Next_Command,
                    unsigned long Timeout);
```

Session_ID	*Session_ID* specifies a session.
Protocol_Error	*Protocol_Error* specifies a *Read Protocol Error* word serial command response.
Enable_Next_Command	*Enable_Next_Command* specifies whether to enable the interface hardware to receive the another word serial command.
Timeout	*Timeout* specifies the number of milliseconds to wait for a word serial command. *Timeout* also specifies the number of milliseconds to wait for a commander to read a word serial command response.

Visual Basic Synopsis

```
Declare Function
EpcSrvSendWSProtocolError% Lib "bmvxiw16.dll"
                    (ByVal Session_ID&,
                    ByVal Protocol_Error%,
                    ByVal Enable_Next_Command%,
                    ByVal Timeout&)
```

Remarks

EpcSrvSendWSProtocolError notifies a commander device that a word serial protocol error has occurred by asserting the interface's response register Write Ready and Err* bits. The function then either:

- receives a READ PROTOCOL ERROR word serial command, deasserts the interface's response register Err* bit, sends the specified *Protocol_Error* response, and optionally enables reception of the next word serial command, or
- receives an ABORT NORMAL OPERATION word serial command, deasserts the interface's response register Err* bit, and returns **EPC_RECV_ANO.**
- receives a CLEAR word serial command, deasserts the interface's response register Err* bit, and returns **EPC_RECV_CLEAR.**
- receives an END NORMAL OPERATION word serial command, deasserts the interface's response register Err* bit, and returns **EPC_RECV_ENO.**

All word serial commands received while the interface is waiting for either a READ PROTOCOL ERROR word serial command, an ABORT NORMAL OPERATION, a CLEAR, or an END NORMAL OPERATION are discarded.

The following constants specify valid values for the *Enable_Next_Command* parameter:

Constant	Description
EPC_DISABLE_ALL	Disable word serial command reception.
EPC_ENABLE_WRDY	Enable word serial command reception by asserting WRDY and deasserting both DIR and DOR.
EPC_ENABLE_DIR	Enable word serial command reception and data input by asserting both WRDY and DIR and deasserting DOR.
EPC_ENABLE_DOR	Enable word serial command reception and data output by asserting both WRDY and DOR and deasserting DIR.
EPC_ENABLE_ALL	Enable word serial command reception, data input, and data output by asserting WRDY, DIR, and DOR.

On an EPC-7, any outgoing word serial command response must be read from the interface's message registers by the interface's commander device before attempting to notify a commander device that a word serial protocol error has occurred. Otherwise, additional word serial protocol violations can occur. Successful completion of **EpcSrvSendWSResponse** indicates that an outgoing word serial command response has been read from the interface's message registers.

EPConnect does not support sending a word serial protocol error on VXLink interfaces. Attempting to use the function on an unsupported interface results in an **EPC_INV_HW** error.

Return Value The function returns a EPConnect return value:

EPC_INV_ENABLE	The parameter *Enable_Next_Command* is invalid.
EPC_INV_ERROR	The parameter *Protocol_Error* is invalid.

EPC_INV_HW	The interface hardware does not support sending a word serial protocol error.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The Bus Manager device driver is not present.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_RECV_ANO	The interface received an ABORT NORMAL OPERATION command.
EPC_RECV_CLEAR	The interface received a CLEAR word serial command.
EPC_RECV_ENO	The interface received an END NORMAL OPERATION word serial command.
EPC_RECV_TIMEOUT	A timeout occurred receiving a word serial command.
EPC_SEND_TIMEOUT	A timeout occurred sending a word serial command response.
EPC_SUCCESS	The function completed successfully.

See Also **EpcOpenSession, EpcSrvSendWSResponse.**

EpcSrvSendWSResponse

Description Sends a word serial command response to the commander device.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcSrvSendWSResponse
        (unsigned long   Session_ID,
        void  FAR *      Response_Ptr,
        unsigned short   Response_Size,
        unsigned short   Enable_Next_Command,
        unsigned long    Timeout);
```

Session_ID	*Session_ID* specifies a session.
Response_Ptr	*Response_Ptr* specifies the location of a word serial command response.
Response_Size	*Response_Size* specifies the size of the word serial command response.
Enable_Next_Command	*Enable_Next_Command* specifies whether to enable the interface hardware to receive the another word serial command.
Timeout	*Timeout* specifies the number of milliseconds to wait for a commander to read the word serial command response.

Visual Basic Synopsis

```
Declare Function
EpcSrvSendWSResponse% Lib "bmvxiw16.dll"
                (ByVal Session_ID&,
                Response_Ptr As Any,
                ByVal Response_Size%,
                ByVal Enable_Next_Command%,
                ByVal Timeout&)
```

Remarks

EpcSrvSendWSResponse optionally sends the word serial command response at the location pointer to by *Response_Ptr* to a commander device. The function then configures the interface hardware for future word serial command reception.

Response_Size specifies the size of the word serial command response:

Response_Size	Description
EPC_16_BIT	Send a 16-bit word serial command response.
EPC_32_BIT	Send a 32-bit long word serial command response. (EPC-7 only)

The following constants specify valid values for the *Enable_Next_Command* parameter:

Constant	Description
EPC_DISABLE_ALL	Disable word serial command reception.
EPC_ENABLE_WRDY	Enable word serial command reception by asserting WRDY and deasserting both DIR and DOR.
EPC_ENABLE_DIR	Enable word serial command reception and data input by asserting both WRDY and DIR and deasserting DOR.
EPC_ENABLE_DOR	Enable word serial command reception and data output by asserting both WRDY and DOR and deasserting DIR.
EPC_ENABLE_ALL	Enable word serial command reception, data input, and data output by asserting WRDY, DIR, and DOR.

On an EPC-7, sending a word serial command response while word serial command reception is enabled can result in a word serial protocol violation by allowing the commander device to write over the word serial command response.

Occasionally, it is useful to ensure that a word serial response has been read without destroying the contents of the interface's message registers. To ensure that a word serial response has been read without destroying the contents of the interface's message registers, use **EpcSrvSendWSResponse** with a *Response_Ptr* parameter value of null and an *Enable_Next_Command* parameter value of **EPC_DISABLE_ALL**. Such a call tests that a word serial command response has been read from the interface's message registers without destroying the contents of the registers. Note that either enabling word serial command reception or sending a word serial command response overwrites the contents of the interface's message registers, destroying any data preserved there.

The function returns **EPC_MULTIPLE_ERR** if the *Response_Ptr* parameter is not null and previously sent response data remains unread in the interface's message registers.

The function returns **EPC_SEND_TIMEOUT** if a commander device does not read the word serial command response within the specified timeout time. If this error occurs, the word serial command response remains in the interface message register.

EPConnect does not support sending a word serial command response on a VXLink interface. Attempting to use the function on a VXLink interface results in an **EPC_INV_HW** error.

Return Value The function returns a Bus Management return value:

EPC_INV_ENABLE	The parameter *Enable_Next_Command* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_HW	The interface does not support sending a word serial command response.
EPC_INV_SIZE	The parameter *Response_Size* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_LOCKED	Shared interface hardware is locked by another session.
EPC_MULTIPLE_ERR	A word serial protocol multiple queries error occurred.
EPC_SEND_TIMEOUT	A timeout occurred waiting for a commander to read the word serial command response.
EPC_SUCCESS	The function completed successfully.

See Also **EpcOpenSession, EpcSrvEnableWSCommand.**

EpcSwap16

Description Byte-swaps a 16-bit value.

C Synopsis

#include "busmgr.h"

short FAR PASCAL
EpcSwap16(unsigned short **FAR** **Value_Ptr*);*

Value_Ptr	*Value_Ptr* specifies a location containing a 16-bit value to byte-swap.

Visual Basic Synopsis

Declare Function
EpcSwap16% Lib "bmvxiw16.dll" (*Value_Ptr%*)

Remarks **EpcSwap16** byte-swaps the 16-bit value in the location pointed to by *Value_Ptr*.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	The parameter *Value_Ptr* is invalid.
EPC_SUCCESS	The function completed successfully.

See Also **EpcSwapBuffer, EpcSwap32, EpcSwap48, EpcSwap64, EpcSwap80.**

Example See **EpcSwapBuffer.**

EpcSwap32

Description Byte-swaps a 32-bit value.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcSwap32(unsigned long FAR*Value_Ptr);
```

Value_Ptr — *Value_Ptr* specifies a location containing a 32-bit value to byte-swap.

Visual Basic Synopsis

```
Declare Function
EpcSwap32% Lib "bmvxiw16.dll" (Value_Ptr&)
```

Remarks **EpcSwap32** byte-swaps the 32-bit value in the location pointed to by *Value_Ptr*.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	The parameter *Value_Ptr* is invalid.
EPC_SUCCESS	The function completed successfully.

See Also **EpcSwapBuffer, EpcSwap16, EpcSwap48, EpcSwap64, EpcSwap80.**

Example See **EpcSwapBuffer**.

EpcSwap48

Description Byte-swaps a 48-bit value.

C Synopsis

#include "busmgr.h"

short FAR PASCAL
EpcSwap48(void **FAR***Value_Ptr*);

Value_Ptr	*Value_Ptr* specifies a location containing a 48-bit value to byte-swap.

Visual Basic Synopsis

Declare Function
EpcSwap48% Lib "bmvxiw16.dll" (*Value_Ptr* As Any)

Remarks **EpcSwap48** byte-swaps the 48-bit value in the location pointed to by *Value_Ptr*.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	The parameter *Value_Ptr* is invalid.
EPC_SUCCESS	The function completed successfully.

See Also **EpcSwapBuffer, EpcSwap16, EpcSwap32, EpcSwap64, EpcSwap80.**

Example See **EpcSwapBuffer.**

EpcSwap64

Description Byte-swaps a 64-bit value.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcSwap64(void FAR*Value_Ptr);
```

Value_Ptr — *Value_Ptr* specifies a location containing a 64-bit value to byte-swap.

Visual Basic Synopsis

```
Declare Function
EpcSwap64% Lib "bmvxiw16.dll" (Value_Ptr As Any)
```

Remarks **EpcSwap64** byte-swaps the 64-bit value in the location pointed to by *Value_Ptr*.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	The parameter *Value_Ptr* is invalid.
EPC_SUCCESS	The function completed successfully.

See Also **EpcSwapBuffer**, **EpcSwap16**, **EpcSwap32**, **EpcSwap48**, **EpcSwap80**.

Example See **EpcSwapBuffer**.

EpcSwap80

Description Byte-swaps an 80-bit value.

C Synopsis

```
#include "busmgr.h"

short
EpcSwap80(void FAR*Value_Ptr);
```

Value_Ptr — *Value_Ptr* specifies a location containing a 80-bit value to byte-swap.

Visual Basic Synopsis

```
Declare Function
EpcSwap80% Lib "bmvxiw16.dll" (Value_Ptr As Any)
```

Remarks **EpcSwap80** byte-swaps the 80-bit value in the location pointed to by *Value_Ptr*.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	The parameter *Value_Ptr* is invalid.
EPC_SUCCESS	The function completed successfully.

See Also **EpcSwapBuffer**, **EpcSwap16**, **EpcSwap32**, **EpcSwap48**, **EpcSwap64**.

Example See **EpcSwapBuffer**.

EpcSwapBuffer

Description Byte-swaps a buffer of data.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcSwapBuffer(void FAR    * Buffer_Ptr,
              unsigned long  Buffer_Size,
              unsigned short Width);
```

Buffer_Ptr	*Buffer_Ptr* specifies the location of a buffer of data elements to byte-swap.
Buffer_Size	*Buffer_Size* specifies the size of the specified data buffer, in bytes.
Width	*Width* specifies the width of the individual data elements in the specified data buffer.

Visual Basic Synopsis

Declare Function
EpcSwapBuffer% Lib "bmvxiw16.dll" (*Buffer_Ptr* As Any, ByVal *Buffer_Size*&, ByVal *Width*%)

Remarks **EpcSwapBuffer** byte-swaps the array of data elements at the location pointed to by *Buffer_Ptr*.

Width specifies the width of the individual data elements in the specified data buffer, in bytes. Valid values are:

Width	Description
EPC_8_BIT	The specified buffer contains *Buffer_Size* 8-bit data elements.
EPC_16_BIT	The specified buffer contains *Buffer_Size*/2 16-bit data elements.
EPC_32_BIT	The specified buffer contains *Buffer_Size*/4 32-bit data elements.
EPC_48_BIT	The specified buffer contains *Buffer_Size*/6 48-bit data elements.
EPC_64_BIT	The specified buffer contains *Buffer_Size*/8 64-bit data elements.
EPC_80_BIT	The specified buffer contains *Buffer_Size*/10 80-bit data elements.

The function assumes that all of the data elements in the specified buffer are the same size.

The function does not limit *Buffer_Size* to less than 64 Kbytes, nor does it make any attempt to detect the end of the buffer segment. The function wraps around to the beginning of the buffer segment if *Buffer_Size* is too large.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	The parameter *Buffer_Ptr* is invalid.
EPC_INV_SIZE	The parameter *Buffer_Size* is not a multiple of *Width*.
EPC_INV_WIDTH	The parameter *Width* is invalid.
EPC_SUCCESS	The function completed successfully.

See Also **EpcSwap16, EpcSwap32, EpcSwap48, EpcSwap64, EpcSwap80.**

Example

```
/*
 * Copyright 1994 by RadiSys Corporation.  All rights reserved.
 */

/*
 * byteord.c -- Bus Management Library byte order functions sample code.
 */

#include "busmgr.h"

/*
 * FUNCTION PROTOTYPES...
 */

short FAR
ByteOrdSample(void);

int FAR
WinPrintf(char FAR *Format_Ptr, ...);

/*
 * GLOBAL DATA...
 */

unsigned char Value16[] = { 0x00, 0x11 };
unsigned char Value32[] = { 0x00, 0x11, 0x22, 0x33 };
unsigned char Value48[] = { 0x00, 0x11, 0x22, 0x33, 0x44, 0x55 };
unsigned char Value64[] = { 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77 };
unsigned char Value80[] = { 0x00, 0x11, 0x22, 0x33, 0x44,
                            0x55, 0x66, 0x77, 0x88, 0x99 };

/*
 * CODE...
 */

short FAR
ByteOrdSample(void)
{
    /*
     * Byte-swap a 16-bit value, then byte-swap it back to its original order.
     *
     * NOTES:
     *      1.  For the sake of example, the code uses both EpcSwap16() and
     *          EpcSwapBuffer() to swap the data.
     */

    EpcSwap16((unsigned short FAR *) Value16);
    EpcSwapBuffer((void FAR *) Value16, sizeof(Value16), EPC_16_BIT);

    /*
     * Byte-swap a 32-bit value, then byte-swap it back to its original order.
     *
     * NOTES:
     *      1.  For the sake of example, the code uses both EpcSwap32() and
     *          EpcSwapBuffer() to swap the data.
     */

    EpcSwap32((unsigned long FAR *) Value32);
    EpcSwapBuffer((void FAR *) Value32, sizeof(Value32), EPC_32_BIT);

    /*
     * Byte-swap a 48-bit value, then byte-swap it back to its original order.
     *
```

```
     * NOTES:
     *      1.  For the sake of example, the code uses both EpcSwap48() and
     *          EpcSwapBuffer() to swap the data.
     */

    EpcSwap48((void FAR *) Value48);
    EpcSwapBuffer((void FAR *) Value48, sizeof(Value48), EPC_48_BIT);

    /*
     * Byte-swap a 64-bit value, then byte-swap it back to its original order.
     *
     * NOTES:
     *      1.  For the sake of example, the code uses both EpcSwap64() and
     *          EpcSwapBuffer() to swap the data.
     */

    EpcSwap64((void FAR *) Value64);
    EpcSwapBuffer((void FAR *) Value64, sizeof(Value64), EPC_64_BIT);

    /*
     * Byte-swap a 80-bit value, then byte-swap it back to its original order.
     *
     * NOTES:
     *      1.  For the sake of example, the code uses both EpcSwap80() and
     *          EpcSwapBuffer() to swap the data.
     */

    EpcSwap80((void FAR *) Value80);
    EpcSwapBuffer((void FAR *) Value80, sizeof(Value80), EPC_80_BIT);
    WinPrintf("SUCCESS:  ByteOrdSample() complete.\n");
    return (EPC_SUCCESS);
}
```

EpcUnlockSession

Description Unlocks shared interface hardware for a session.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcUnlockSession(unsigned long Session_ID);
```

Session_ID	*Session_ID* specifies a session.

Visual Basic Synopsis

```
Declare Function
EpcUnlockSession% Lib "bmvxiw16.dll" (ByVal Session_ID&)
```

Remarks **EpcUnlockSession** unlocks shared interface hardware for the specified *Session_ID*.

Return Value The function returns a Bus Management return value:

EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_NOT_LOCKED	Shared interface hardware is not locked by the specified session.
EPC_SUCCESS	The function completed successfully.

See Also **EpcLockSession**, **EpcOpenSession**.

Example See **EpcGetLockingTimeout**.

EpcUnmapBusMemory

Description Destroys a bus memory mapping.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcUnmapBusMemory( unsigned long          Session_ID,
                   volatile void HUGE *Mapped_Ptr);
```

Session_ID	*Session_ID* specifies a bus session.
Mapped_Ptr	*Mapped_Ptr* specifies a pointer to mapped bus memory.

Visual Basic Synopsis

```
Declare Function
EpcUnmapBusMemory% Lib "bmvxiw16.dll" (ByVal
Session_ID&, ByVal Mapped_Ptr As Any)
```

Remarks **EpcUnmapBusMemory** destroys a bus memory mapping.

Return Value The function returns a Bus Management return value:

EPC_INV_MAP	The specified *Mapped_Ptr* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_USAGE	The mapping specified by *Mapped_Ptr* is in use by another thread.
EPC_OS_ERROR	An operating system error occurred.
EPC_SUCCESS	The function completed successfully.

See Also **EpcMapBusMemory**, **EpcOpenSession**.

Example See **EpcCopyData**.

EpcUnmapSharedMemory

Description Destroys a shared memory mapping.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcUnmapSharedMemory( unsigned long          Session_ID,
                      volatile void HUGE *Mapped_Ptr);
```

Session_ID	*Session_ID* specifies a bus session.
Mapped_Ptr	*Mapped_Ptr* specifies a pointer to mapped shared memory.

Visual Basic Synopsis

```
Declare Function
EpcUnmapSharedMemory% Lib "bmvxiw16.dll" (ByVal
Session_ID&, ByVal Mapped_Ptr As Any)
```

Remarks **EpcUnmapSharedMemory** destroys a shared memory mapping.

Return Value The function returns a Bus Management return value:

EPC_INV_MAP	The specified *Mapped_Ptr* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_INV_USAGE	The mapping specified by *Mapped_Ptr* is in use by another thread.
EPC_OS_ERROR	An operating system error occurred.
EPC_SUCCESS	The function completed successfully.

See Also **EpcMapSharedMemory, EpcOpenSession.**

Example See **EpcCopyData.**

EpcValidateBusMapping

Validates a bus memory mapping that uses dynamically configured bus window hardware.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcValidateBusMapping( unsigned long          Session_ID,
                       volatile void HUGE *Mapped_Ptr);
```

Session_ID	*Session_ID* specifies a bus session.
Mapped_Ptr	*Mapped_Ptr* specifies a pointer to mapped bus memory.

Remarks

EpcValidateBusMapping configures an interface's dynamically configured bus window hardware with the attributes of the specified bus memory mapping.

The function supports direct bus access for bus memory mappings created using **EpcMapBusMemoryExt**. The function is a no-op for bus memory mappings created using **EpcMapBusMemory**.

Direct bus access using dynamically configured bus window hardware requires using **EpcValidateBusMapping** before a group direct bus accesses to ensure that the bus window references the desired bus memory.

On an operating system that supports multiple processes or threads (any non-DOS operating system), direct bus access using dynamically configured bus windows also requires a mechanism for protecting the bus window hardware from reconfiguration during the bus accesses. In a non-preemptive environment (like Windows), simply insuring that all direct bus accesses complete before giving up the processor is sufficient.

In an environment with a preemptive scheduling algorithm (like LynxOS or OS/2), direct bus access using dynamically configured bus window hardware also requires using **EpcLockSession** and **EpcValidateBusMapping** before a group of direct bus accesses and **EpcUnlockSession** after the direct bus access is complete. Using **EpcLockSession** and **EpcUnlockSession** insures that another thread does not reconfigure the dynamically configured bus window hardware during the access.

Return Value The function returns a EPConnect return value:

EPC_INV_MAP	The specified *Mapped_Ptr* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present.
EPC_LOCKED	Shared interface hardware is locked by another bus session.
EPC_OS_ERROR	An operating system error occurred.
EPC_SUCCESS	The function completed successfully.

See Also **EpcMapBusMemory, EpcMapBusMemoryExt, EpcLockSession, EpcOpenSession, EpcUnlockSession.**

EpcVerifyEnvironment

Description Verifies and queries the EPConnect environment.

C Synopsis

#include "busmgr.h"

short FAR PASCAL
EpcVerifyEnvironment(struct **EpcEnvironment FAR** **Environment_Ptr*);

Environment_Ptr — *Environment_Ptr* specifies a location where data describing the EPConnect environment will be placed.

Visual Basic Synopsis

Declare Function
EpcVerifyEnvironment% Lib "bmvxiw16.dll" (*Environment_Ptr* As Any)

Remarks **EpcVerifyEnvironment** verifies the EPConnect environment and places data describing the environment in the structure pointed to by *Environment_Ptr*.

The returned EPConnect environment structure contains a complete description of the underlying hardware and software. The structure also contains a complete description of the Bus Management Library features supported by the underlying software and hardware:

```
struct EpcEnvironment
{
   /* Hardware, firmware, and software revision attributes.          */

   unsigned char HWRevision;          /* Hardware revision number:    */
                                      /*     EPC_7                    */
                                      /*     EPC_8                    */
                                      /*     VXLink_ISA               */
   unsigned char BIOSMajorRevision;   /* BIOS major revision number.  */
   unsigned char BIOSMinorRevision;   /* BIOS minor revision number.  */
   unsigned char SWMajorRevision;     /* Software major revision number. */
   unsigned char SWMinorRevision;     /* Software minor revision number. */
```

```
                                /* Memory mapping attributes.                  */

unsigned char IsHWByteSwap;     /* Is the interface capable of hardware byte */
                                /* swapping?                                   */
                                /*     TRUE                                    */
                                /*     FALSE                                   */
unsigned short   AddressMod;    /* Valid address modifiers (OR'd               */
                                /* combination of):                            */
                                /*     EPC_A16S                                */
                                /*     EPC_A24SD                               */
                                /*     EPC_A32SD                               */
unsigned short   ByteOrder;     /* Valid byte ordering values (OR'd            */
                                /* combination of):                            */
                                /*     EPC_IBO                                 */
                                /*     EPC_MBO                                 */
unsigned short   DataWidth;     /* Valid data widths (OR'd combination of):    */
                                /*     EPC_8_BIT                               */
                                /*     EPC_8_BIT_ODD                           */
                                /*     EPC_16_BIT                              */
                                /*     EPC_32_BIT                              */
                                /*     EPC_FASTCOPY                            */
unsigned short MinA16Address;   /* Minimum accessible A16 address.             */
unsigned short MaxA16Address;   /* Maximum accessible A16 address.             */
unsigned long MinA24Address;    /* Minimum accessible A24 address.             */
unsigned long MaxA24Address;    /* Maximum accessible A24 address.             */
unsigned long MinA32Address;    /* Minimum accessible A32 address.             */
unsigned long MaxA32Address;    /* Maximum accessible A32 address.             */

/* Event attributes. */

unsigned long EventMask;        /* Valid events (OR'd combination of):         */
                                /*     EPC_MSG_INT             */
                                /*     EPC_VME1_INT            */
                                /*     ...                     */
                                /*     EPC_VME7_INT            */
                                /*     EPC_SIGNAL_INT          */
                                /*     EPC_TTL_TRIG0_INT       */
                                /*     ...                     */
                                /*     EPC_TTL_TRIG7_INT       */
                                /*     EPC_ACFAIL_ERR          */
                                /*     EPC_BERR_ERR            */
                                /*     EPC_SYSFAIL_ERR         */
                                /*     EPC_WATCHDOG_ERR        */
                                /*     EPC_SYSRESET_ERR        */

/* EPC bus configuration attributes. */

unsigned short   BusEnable;         /* Valid bus enable attributes (OR'd       */
                                    /* combination of):                        */
                                    /*     EPC_DISABLE_BUS                     */
                                    /*     EPC_ENABLE_BUS                      */
unsigned short   BusArbMode;        /* Valid bus arbitration mode attributes   */
                                    /* (OR'd combination of):                  */
                                    /*     EPC_PRIORITY                        */
                                    /*     EPC_ROUND_ROBIN                     */
unsigned short   BusArbPriority;    /* Valid bus arb priority attributes       */
                          /* (OR'd combination of):                            */
                          /*     EPC_PRIORITY0                                 */
                          /*     EPC_PRIORITY1                                 */
                          /*     EPC_PRIORITY2                                 */
                          /*     EPC_PRIORITY3                                 */
unsigned short   BusRelease;/* Valid bus release attributes (OR'd              */
                          /* combination of):                                  */
```

```
                        /*     EPC_ROR                                  */
                        /*     EPC_RONR                                 */

/* Miscellaneous configuration attributes. */

unsigned char IsBERRAssertion; /* Is  software capable of asserting the    */
                        /* sticky BERR bit?              */
                        /*     TRUE                      */
                        /*     FALSE                     */
unsigned long GetMiscMask;  /* Miscellaneous attributes that can be     */
                        /* queried (OR'd combination of):           */
                        /*     EPC_DIR                   */
                        /*     EPC_DOR                   */
                        /*     EPC_ERR                   */
                        /*     EPC_LOCK                  */
                        /*     EPC_PASS                  */
                        /*     EPC_PIPELINE_BUSY         */
                        /*     EPC_READY                 */
                        /*     EPC_RRDY                  */
                        /*     EPC_RSRC_MGR              */
                        /*     EPC_STICKY_BERR           */
                        /*     EPC_SYSFAIL_OUT           */
                        /*     EPC_SYSRESET_IN           */
                        /*     EPC_TTL_LATCH0            */
                        /*     ...                       */
                        /*     EPC_TTL_LATCH7            */
                        /*     EPC_WATCHDOG              */
                        /*     EPC_WRDY                  */
unsigned long SetMiscMask;  /* Miscellaneous attributes that can be     */
                        /* defined (OR'd combination of):           */
                        /*     EPC_DIR                   */
                        /*     EPC_DOR                   */
                        /*     EPC_ERR                   */
                        /*     EPC_LOCK                  */
                        /*     EPC_PASS                  */
                        /*     EPC_READY                 */
                        /*     EPC_RESET                 */
                        /*     EPC_RRDY                  */
                        /*     EPC_RSRC_MGR              */
                        /*     EPC_STICKY_BERR           */
                        /*     EPC_SYSFAIL_OUT           */
                        /*     EPC_SYSRESET_IN           */
                        /*     EPC_WRDY                  */


/* Slave memory configuration attributes. */

unsigned char IsSelfAccess; /* Is  the I/F capable of slave memory   */
                        /* self-accesses (via the VMEbus)?          */
                        /*     TRUE                                 */
                        /*     FALSE                                */
unsigned short SlaveSpace;  /* Valid slave memory address spaces    */
                        /* (OR'd combination of):                   */
                        /*     EPC_DISABLED                         */
                        /*     EPC_A24                              */
                        /*     EPC_A32                              */
unsigned short SlaveWidth;  /* Valid slave memory access widths (OR'd*/
                        /* combination of):                         */
                        /*     EPC_8_BIT                            */
                        /*     EPC_16_BIT                           */
                        /*     EPC_32_BIT                           */
unsigned long MinA24Slave;  /* Minimum A24 slave memory base        */
                        /* address.                                 */
```

```
unsigned long MaxA24Slave;          /* Maximum A24 slave memory base   */
                                    /* address.                        */
unsigned long A24SlaveBaseInc;      /* A24 slave memory base increment*/
unsigned long MinA32Slave;          /* Minimum A32 slave memory base   */
                                    /* address.                        */
unsigned long MaxA32Slave;          /* Maximum A32 slave memory base   */
                                    /* address.                        */
unsigned long A32SlaveBaseInc;      /* A32 slave memory base increment*/

/* unique logical address configuration attributes.                    */
unsigned char MinULA;               /* Minimum valid ULA.              */
unsigned char MaxULA;               /* Maximum valid ULA.              */

/* Bus line attributes. */

unsigned long GetBusLineMask;   /* Bus control lines that can be queried */
                                /* (OR'd combination of):                */
                                /*     EPC_ACFAIL                        */
                                /*     EPC_SYSFAIL                       */
                                /*     EPC_SYSRESET                      */
unsigned long GetBusMODIDMask;  /* Bus MODID lines that can be queried   */
                                /* (OR'd combination of):                */
                                /*     EPC_SLOT_MODID                    */
                                /*     EPC_MODID0                        */
                                /*     ...                               */
                                /*     EPC_MODID12                       */
unsigned long GetBusTriggerMask;   /* Bus trigger lines that can be queried */
                                /* (OR'd combination of):                */
                                /*     EPC_ECL_TRIG0                     */
                                /*     EPC_ECL_TRIG1                     */
                                /*     EPC_TTL_TRIG0                     */
                                /*     ...                               */
                                /*     EPC_TTL_TRIG7                     */

/* EPC line attributes. */

unsigned long GetEpcLineMask;   /* I/f control lines that can be queried */
                                /* (OR'd combination of):                */
                                /*     EPC_SYSFAIL                       */
                                /*     EPC_SYSRESET                      */
unsigned long SetEpcLineMask;   /* I/f control lines that can be defined */
                                /* (OR'd combination of):                */
                                /*     EPC_SYSFAIL                       */
                                /*     EPC_SYSRESET                      */
unsigned long SetEpcMODIDMask;  /* I/f MODID lines that can be defined   */
                                /* (OR'd combination of):                */
                                /*     EPC_MODID0                        */
                                /*     ...                               */
                                /*     EPC_MODID12                       */
unsigned long GetEpcTriggerMask;   /* I/f trigger lines that can be queried */
                                /* (OR'd combination of):                */
                                /*     EPC_ECL_TRIG0                     */
                                /*     EPC_ECL_TRIG1                     */
                                /*     EPC_TTL_TRIG0                     */
                                /*     ...                               */
                                /*     EPC_TTL_TRIG7                     */
unsigned long SetEpcTriggerMask;   /* I/f trigger lines that can be queried */
                                /* (OR'd combination of):                */
                                /*     EPC_ECL_TRIG0                     */
                                /*     EPC_ECL_TRIG1                     */
                                /*     EPC_TTL_TRIG0                     */
                                /*     ...                               */
                                /*     EPC_TTL_TRIG7                     */
unsigned long InTriggerMask;    /* I/f trigger lines used as an          */
```

```
                                /* input trigger in a trigger mapping     */
                                /* operation (OR'd combination of):       */
                                /*     EPC_TTL_TRIG0                      */
                                /*     ...                                */
                                /*     EPC_TTL_TRIG7                      */
unsigned long OutTriggerMask[EPC_TRIGGER_CNT];
                                /* I/f trigger lines that can be used as */
                                /* output triggers in a trigger mapping  */
                                /* operation (one array element for each */
                             /* potential input trigger;  each entry is an   */
                             /* OR'd combination of):                        */
                             /*     EPC_TTL_TRIG0                            */
                             /*     ...                                      */
                             /*     EPC_TTL_TRIG7                            */

/* Watchdog timer attributes. */

unsigned short   WatchdogCfg;  /* Valid watchdog timer configuration       */
                             /* constants (OR'd combination of):            */
                             /*     EPC_WDT_RESET                           */
                             /*     EPC_WDT_FAST_ERROR                      */
                             /*     EPC_WDT_FAST_RESET                      */
                             /*     EPC_WDT_SLOW_ERROR                      */
                             /*     EPC_WDT_SLOW_RESET                      */

/* Servant attributes. */

unsigned char IsProtocolError;     /* Is the EPC capable of signaling a     */
                             /* protocol error to its commander?            */
                             /*     TRUE                                    */
                             /*     FALSE                                   */
unsigned short   WSSize;     /* Valid word serial command/response          */
                             /* sizes (OR'd combination of):                */
                             /*     EPC_16_BIT                              */
                             /*     EPC_32_BIT                              */
                             /*     EPC_48_BIT                              */
unsigned short   EnableNextCommand;   /* Valid word serial command enable   */
                             /* constants (OR'd combination of):            */
                             /*     EPC_DISABLE_ALL                         */
                             /*     EPC_ENABLE_WRDY                         */
                             /*     EPC_ENABLE_DIR                          */
                             /*     EPC_ENABLE_DOR                          */
                             /*     EPC_ENABLE_ALL                          */
unsigned long MethodMask;    /* Valid methods for sending protocol          */
                             /* events (OR'd combination of):               */
                             /*     EPC_VME1_INT                            */
                             /*     ...                                     */
                             /*     EPC_VME7_INT                            */
                             /*     EPC_SIGNAL_REG                          */
unsigned short   IRQ;        /* PC-AT IRQ used.                             */
unsigned short   IOBase;     /* I/O base address.                           */
unsigned long WindowBase;    /* Bus window base address.                    */
unsigned long WindowSize;    /* Bus window size, in bytes.                  */
```

```
    /* Dynamic memory mapping attributes. */

    unsigned char IsHWByteSwapExt; /* Is  the interface  capable of hardware    */
                                   /* byte swapping?                           */
                                   /*     TRUE                                 */
                                   /*     FALSE                                */
    unsigned short    AddressModExt;       /* Valid address modifiers (OR'd
    */
                                   /* combination of):                         */
                                   /*     EPC_A16N                             */
                                   /*     EPC_A16S                             */
                                   /*     EPC_A24ND                            */
                                   /*     EPC_A24NP                            */
                                   /*     EPC_A24SD                            */
                                   /*     EPC_A24SP                            */
                                   /*     EPC_A32ND                            */
                                   /*     EPC_A32NP                            */
                                   /*     EPC_A32SD                            */
                                   /*     EPC_A32SP                            */
    unsigned short    ByteOrderExt,     /* Valid byte ordering values (OR'd      */
                                   /* combination of):                         */
                                   /*     EPC_IBO                              */
                                   /*     EPC_MBO                              */
    unsigned short    DataWidthExt;     /* Valid data widths (OR'd combination of):
    */
                                   /*     EPC_8_BIT                            */
                                   /*     EPC_8_BIT_ODD                        */
                                   /*     EPC_16_BIT                           */
                                   /*     EPC_32_BIT                           */
                                   /*     EPC_FASTCOPY                         */
    unsigned short    MinA16AddressExt;/* Minimum accessible A16 address. */
    unsigned short    MaxA16AddressExt;/* Maximum accessible A16 address. */
    unsigned long MinA24AddressExt;/* Minimum accessible A24 address.    */
    unsigned long MaxA24AddressExt;/* Maximum accessible A24 address.    */
    unsigned long MinA32AddressExt;/* Minimum accessible A32 address.    */
    unsigned long MaxA32AddressExt;/* Maximum accessible A32 address.    */

    unsigned long Reserved[6];         /* Reserved area (for future expansion). */
};
```

Return Value The function returns a Bus Management return value:

EPC_INV_HW	EPConnect does not support this revision of interface hardware.
EPC_INV_PTR	The parameter *Environment_Ptr* is invalid.
EPC_INV_SW	The BusManager device driver is not present or there is a revision mismatch between the Bus Management Library and the BusManager VxD.
EPC_SUCCESS	The function completed successfully.

Example See **EpcGetErrorString**.

EpcWaitForEvent

Description Wait for an event to occur.

C Synopsis

```
#include "busmgr.h"

short FAR PASCAL
EpcWaitForEvent(  unsigned long          Session_ID,
                  unsigned long          Timeout,
                  unsigned long          Wait_Mask,
                  unsigned long FAR *    Event_Mask_Ptr,
                  unsigned long FAR *    Event_Data_Ptr);
```

Session_ID	*Session_ID* specifies a session.
Timeout	*Timeout* specifies the number of milliseconds to wait for an enabled event to occur.
Wait_Mask	*Wait_Mask* specifies the events to await.
Event_Mask_Ptr	*Event_Mask_Ptr* specifies a location where an event mask that specifies the occurring event will be placed.
Event_Data_Ptr	*Event_Data_Ptr* specifies a location where event data from the occurring event will be placed.

Visual Basic Synopsis

```
Declare Function
EpcWaitForEvent% Lib "bmvxiw16.dll"
                        (ByVal Session_ID&,
                        ByVal Timeout&,
                        ByVal Wait_Mask&,
                        Event_Mask_Ptr&,
                        Event_Data_Ptr&)
```

Remarks

EpcWaitForEvent waits at least *Timeout* milliseconds for one of the events specified by *Wait_Mask* to occur, then places an event mask identifying the event and the event's data in the locations pointed to by *Event_Mask_Ptr* and *Event_Data_Ptr*, respectively.

The *Wait_Mask* parameter is a bit mask where each bit corresponds to an event. The *Wait_Mask* parameter should be an OR'd combination of the following constants:

Event	Description
EPC_MSG_INT	Message (EPC-7 and EPC-8 only)
EPC_VME1_INT	VMEbus interrupt 1
...	
EPC_VME7_INT	VMEbus interrupt 7
EPC_SIGNAL_INT	VXIbus signal FIFO interrupt
EPC_TTL_TRIG0_INT	VXIbus TTL Trigger 0 interrupt (EPC-7 only)
...	
EPC_TTL_TRIG7_INT	VXIbus TTL Trigger 7 interrupt (EPC-7 only)
EPC_SYSRESET_ERR	VMEbus SYSRESET error
EPC_ACFAIL_ERR	VMEbus power failure error
EPC_BERR_ERR	VMEbus access error
EPC_SYSFAIL_ERR	VMEbus SYSFAIL error
EPC_WATCHDOG_ERR	Watchdog timer expiration error (EPC-7 and EPC-8 only)
EPC_EXT_TRIG0_INT	External trigger 0 interrupt (VXLink only)
EPC_EXT_TRIG1_INT	External trigger 1 interrupt (VXLink only)

The value that **EpcWaitForEvent** places in the location pointed to by *Event_Mask_Ptr* can be one the following constants:

**Event_Mask_Ptr*	Description
EPC_MSG_INT	Message interrupt (EPC-7 and EPC-8 only)
EPC_VME1_INT	VMEbus interrupt 1
...	
EPC_VME7_INT	VMEbus interrupt 7
EPC_SIGNAL_INT	VXIbus signal FIFO interrupt
EPC_TTL_TRIG0_INT	VXIbus TTL Trigger 0 interrupt (EPC-7 only)
...	
EPC_TTL_TRIG7_INT	VXIbus TTL Trigger 7 interrupt (EPC-7 only)
EPC_SYSRESET_ERR	VMEbus SYSRESET error
EPC_ACFAIL_ERR	VMEbus power failure error
EPC_BERR_ERR	VMEbus access error
EPC_SYSFAIL_ERR	VMEbus SYSFAIL error
EPC_WATCHDOG_ERR	Watchdog timer expiration error (EPC-7 and EPC-8 only)
EPC_EXT_TRIG0_INT	External trigger 0 interrupt (VXLink only)
EPC_EXT_TRIG1_INT	External trigger 1 interrupt (VXLink only)

Whether the value that **EpcWaitForEvent** places in the location pointed to by *Event_Data_Ptr* is meaningful depends on the event:

**Event_Mask_Ptr*	**Event_Data_Ptr*
EPC_MSG_INT	0
EPC_VME1_INT	VMEbus interrupt
...	status/id
EPC_VME7_INT	(zero-extended to 32 bits)
EPC_SIGNAL_INT	VXIbus signal data (zero-extended to 32 bits)
EPC_TTL_TRIG0_INT	0
...	0
EPC_TTL_TRIG7_INT	0
EPC_SYSRESET_ERR	0
EPC_ACFAIL_ERR	0
EPC_BERR_ERR	0
EPC_SYSFAIL_ERR	0
EPC_WATCHDOG_ERR	0
EPC_EXT_TRIG0_INT	0
EPC_EXT_TRIG1_INT	0

When the specified timeout expires before an event occurs, the locations pointed to by *Event_Mask_Ptr* and *Event_Data_Ptr* contain undefined values.

If a session has an event handler and event handler stack pointer defined for an enabled event, an occurrence of the event satisfies the wait condition and invokes the event handler. The order of execution of the two threads is undefined.

Waiting for an event does not enable reception of the corresponding event. A separate, preceding call to **EpcSetEventEnableMask** is required.

Return Value The function returns a Bus Management return value:

EPC_INV_MASK	*Wait_Mask* contains an event that is not valid for this interface.
EPC_INV_PTR	One or more of the parameters *Event_Mask_Ptr* and *Event_Data_Ptr* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_SUCCESS	The function completed successfully.
EPC_TIMEOUT	The specified timeout period expired before an enabled event occurred.

See Also **EpcOpenSession, EpcSetEventEnableMask.**

Example See **EpcGetEventEnableMask.**

EpcWatchdogTimer

Description Modifies the interface's watchdog timer configuration.

C Synopsis

#include "busmgr.h"

short FAR PASCAL
EpcWatchdogTimer(unsigned long *Session_ID*, unsigned short *WatchdogCfg*);

Session_ID	*Session_ID* specifies a session.
WatchdogCfg	*WatchdogCfg* specifies a watchdog timer configuration.

Visual Basic Synopsis

Declare Function
EpcWatchdogTimer% Lib "bmvxiw16.dll" (ByVal *Session_ID&*, ByVal *Watchdog_Cfg%*)

Remarks **EpcWatchdogTimer** modifies the configuration of the interface's watchdog timer.

WatchdogCfg specifies the configuration of the interface's watchdog timer.

Valid values are:

Constant	Description
EPC_WDT_RESET	Reset the watchdog timer without modifying either the watchdog timer period or the operation that occurs upon watchdog timer expiration expiration (EPC-7 and EPC-8 only).
EPC_WDT_FAST_ERROR	Reset the watchdog timer, use the short watchdog timer period, and generate a watchdog timer error event upon watchdog timer expiration expiration (EPC-7 and EPC-8 only).
EPC_WDT_FAST_RESET	Reset the watchdog timer, use the short watchdog timer period, and reset the EPC upon watchdog timer expiration expiration (EPC-7 and EPC-8 only).
EPC_WDT_SLOW_ERROR	Reset the watchdog timer, use the long watchdog timer period, and generate a watchdog timer error event upon watchdog timer expiration expiration (EPC-7 and EPC-8 only).
EPC_WDT_SLOW_RESET	Reset the watchdog timer, use the long watchdog timer period, and reset the EPC upon watchdog timer expiration expiration (EPC-7 and EPC-8 only).

The actual length of an interface's watchdog timer period varies depending on the type of the interface:

Interface Type	Short Watchdog Timer Period	Long Watchdog Timer Period
EPC-7	210 milliseconds	6.7 seconds
EPC-8	128 milliseconds	8.2 seconds

If an interface's watchdog timer is not reset within the current watchdog timer period, a watchdog timer error event occurs. An application can enable the event (using **EpcSetEventEnableMask**) and install an event handler (using **EpcSetEventHandler**) to gracefully handle the error.

The interface's watchdog timer is typically reset in sections of code that execute frequently and/or execute at regular time intervals. Note that other bus operations may reset the watchdog timer as a side-effect of their execution.

By default, an interface's watchdog timer is configured to use the long watchdog timer period and generate a watchdog error event upon expiration.

EPC-5 and VXLink hardware do not include watchdog timer support. Attempting to use the function on an EPC-5 or VXLink interface results in an **EPC_INV_CFG** error.

Return Value

The function returns a Bus Management return value:

EPC_INV_CFG	The parameter *WatchdogCfg* is invalid.
EPC_INV_SESSION	The specified *Session_ID* is invalid.
EPC_INV_SW	The BusManager device driver is not present or there is a revision mismatch between the Bus Management Library and the BusManager VxD.
EPC_SUCCESS	The function completed successfully.

See Also

EpcOpenSession, EpcSetEventEnableMask, EpcSetEventHandler.

Example

```
/*
 * Copyright 1994 by RadiSys Corporation.  All rights reserved.
 */

/*
 * watchdog.c -- Bus Management Library EPC watchdog timer functions sample
 *               code.
 */

#include "busmgr.h"

/*
 * FUNCTION PROTOTYPES...
 */

short FAR
WatchdogSample(void);

int FAR
WinPrintf(char FAR *Format_Ptr, ...);

/*
 * CODE...
 */

short FAR
WatchdogSample(void)
{
    char                  err_string[ERROR_STRING_SZ];
    short                 err_num;
    unsigned long         event_data;
    unsigned long         event_mask;
    unsigned long         session_id;
    struct EpcEnvironment environment;

    /*
     * Verify the EPConnect environment.
     */

    if ((err_num = EpcVerifyEnvironment(&environment)) != EPC_SUCCESS)
    {
        EpcGetErrorString(err_num, err_string);
        WinPrintf("FAILURE:  EpcVerifyEnvironment() error == %s (%d).\n",
                  err_string,
                  err_num);
        return (err_num);
    }

    /*
     * Open a session.
     */

    if ((err_num = EpcOpenSession(&session_id)) != EPC_SUCCESS)
    {
        EpcGetErrorString(err_num, err_string);
        WinPrintf("FAILURE:  EpcOpenSession() error == %s (%d).\n",
                  err_string,
                  err_num);
        return (err_num);
    }

    /*
     * If watchdog timer functionality is supported on this EPC, configure the
```

```
     * EPC to generate a watchdog timer error event using a short timeout, then
     * wait for up to one second (1000 ms) for the event to occur.
     */

    if ((environment.WatchdogCfg & EPC_WDT_FAST_ERROR) != 0 &&
        (environment.WatchdogCfg & EPC_WDT_SLOW_ERROR) != 0  )
    {
        EpcWatchdogTimer(session_id, EPC_WDT_FAST_ERROR);
        EpcWaitForEvent(session_id,
                        1000,
                        EPC_WATCHDOG_ERR,
                        &event_mask,
                        &event_data);
        if ((event_mask & EPC_WATCHDOG_ERR) == 0)
        {
            WinPrintf("Watchdog timer event DID NOT occurred.\n");
        }
        else
        {
            WinPrintf("Watchdog timer event occurred.\n");
        }
        EpcWatchdogTimer(session_id, EPC_WDT_SLOW_ERROR);
    }

    /*
     * Close the session and return.
     */

    EpcCloseSession(session_id);
    WinPrintf("SUCCESS:  WatchdogSample() complete.\n");
    return (EPC_SUCCESS);
}
```

3. On-Line Resource Manager

EPConnect provides an On-Line Resource Manager (OLRM) interface for querying configuration and state information about an instrument system and devices within that instrument system. Configuration information is typically static (i.e., established by a Start-Up Resource Manager (SURM) at system initialization time, and not changed thereafter). Device state information is typically dynamic (i.e., reflects the run-time state changes of a device as it is used by an executing application).

The following OLRM functions are available:

Function	Description
OlrmGetArbAttr	Queries an arbitrary string attribute.
OlrmGetBoolAttr	Queries a boolean attribute.
OlrmGetNmByGPA	Queries the device name corresponding to a GPIB address.
OlrmGetNmByNA	Queries the device name corresponding to a network address.
OlrmGetNmByULA	Queries the device name corresponding to a VXIbus unique logical address.
OlrmGetNumAttr	Queries a numeric attribute.
OlrmGetStrAttr	Queries a string attribute.

To use OLRM to query a specific device's attributes, an application must first know the name of the device. An application can use **OlrmGetNmByGPA**, **OlrmGetNmByNA**, or **OlrmGetNmByULA** to query a device's name from its address. Once the application knows the device's name, it can use **OlrmGetArbAttr**, **OlrmGetBoolAttr**, **OlrmGetNumAttr**, and **OlrmGetStrAttr** to query individual device attributes.

3

3.1 Functions By Name

This section contains an alphabetical listing of the OLRM library functions. Each listing describes the function, gives its invocation sequence and arguments, discusses its operation, and lists its returned values. Where usage of the function may not be clear, an example with comments is given.

OlrmGetArbAttr

Description Queries an arbitrary string attribute.

C Synopsis

```
#include "busmgr.h"
#include "olrm.h"

short
OlrmGetArbAttr(charFAR * Name_Ptr,
               charFAR * Arb_Attribute_Ptr,
               charFAR * Arb_Result_Ptr);
```

Name_Ptr	*Name_Ptr* specifies a device name.
Arb_Attribute_Ptr	*Arb_Attribute_Ptr* specifies an arbitrary string attribute.
Arb_Result_Ptr	*Arb_Result_Ptr* specifies the location of a buffer where the specified string attribute will be placed.

Visual Basic Synopsis

```
Declare Function
OlrmGetNmByGPA% Lib "olrmw16.lib"
                        (ByVal Primary%,
                        ByVal Secondary%
                        ByVal Name_Ptr$)
```

Remarks **OlrmGetArbAttr** queries an arbitrary string attribute of the specified device and places the result in the buffer pointed to by *Arb_Result_Ptr*. The function allows an application to obtain attribute information that is not accessible via the standard set of integer search keys, particularly attribute information about non-GPIB/non-VME devices.

Name_Ptr is a null-terminated ASCII string specifying a device name.

Arb_Attribute_Ptr is a null-terminated ASCII string specifying the string attribute to query.

Arb_Result_Ptr specifies the location a buffer where the function places the result of the arbitrary string attribute query. The buffer must be at least **ATTRIBUTE_SZ** bytes long.

The result of an arbitrary string attribute query is always a null-terminated ASCII character string. Depending on the specified string attribute, the result string may represent a decimal number, a hexadecimal number, a binary number, a bit mask, or a string of characters. It is the responsibility of the application to interpret the result string.

Return Value The function returns a Bus Management return value:

EPC_INV_ATTR	The parameter *Arb_Attribute_Ptr* is invalid.
EPC_INV_NAME	A device with the specified name does not exist.
EPC_INV_PTR	One or more of the parameters *Name_Ptr* and *Arb_Result_Ptr* is invalid.
EPC_OS_ERROR	The resource manager database file could not be read.
EPC_SUCCESS	The function completed successfully.

See Also **OlrmGetBoolAttr, OlrmGetNumAttr, OlrmGetStrAttr.**

OlrmGetBoolAttr

Description Queries a boolean attribute.

C Synopsis

```
#include "busmgr.h"
#include "olrm.h"

short
OlrmGetBoolAttr(char FAR * Name_Ptr,
                short Bool_Attribute,
                unsigned short FAR * Bool_Result_Ptr);
```

Name_Ptr	*Name_Ptr* specifies a device name.
Bool_Attribute	*Bool_Attribute* specifies a boolean attribute.
Bool_Result_Ptr	*Bool_Result_Ptr* specifies a location where the specified boolean attribute will be placed.

Visual Basic Synopsis

```
Declare Function
OlrmGetBoolAttr% Lib "olrmw16.dll"
                 (ByVal Name_Ptr$$,
                 (ByVal Bool_Attribute%
                 Bool_Result_Ptr%)
```

Remarks **OlrmGetBoolAttr** queries a boolean attribute of the specified device and places the result in the location pointed to by *Bool_Result_Ptr*.

Name_Ptr is a null-terminated ASCII string specifying a device name.

Bool_Attribute specifies the boolean attribute to query. Valid values for VXIbus devices are:

Bool_Attribute	**Bool_Result_Ptr*
VXI_ISMEMACT	Memory active (accessible).
VXI_ISMODID	MODID line asserted.
VXI_ISREADY	Ready for normal operation.
VXI_ISPASSED	Passed self test.
VXI_MEM_ISNONPRIV	Non-privileged access capability (memory devices only).
VXI_MEM_ISBLKTR	Block transfer capability (memory devices only).
VXI_MEM_ISNONVOL	Non-volatile RAM memory (memory devices only).
VXI_MEM_ISELPROG	EPROM memory (memory devices only).
VXI_MEM_ISD32TR	D32 transfer capability (memory devices only).
VXI_MPR_ISCMDR	Commander capability (message-based devices only).
VXI_MPR_ISSIG	Signal register present (message-based devices only).
VXI_MPR_ISMSTR	VXIbus master capability (message-based devices only).
VXI_MPR_ISINTR	Interrupter capability (message-based devices only).
VXI_MPR_ISFHS	FHS capability (message-based devices only).
VXI_MPR_ISSHMEM	Shared memory protocol capability (message-based devices only).
VXI_MRP_ISRG	Response generation capability (message-based devices only).

VXI_MRP_ISEG	Event generation capability (message-based devices only).
VXI_MRP_ISPI	Programmable interrupter capability (message-based devices only).
VXI_MRP_ISPH	Programmable handler capability (message-based devices only).
VXI_MRP_ISTRG	Supports word serial *Trigger* command (message-based devices only).
VXI_MRP_ISI4	Supports IEEE 488.2 instrument protocol (message-based devices only).
VXI_MRP_ISINST	Supports VXIbus instrument protocol (message-based devices only).
VXI_MRP_ISELW	Extended long word serial capability (message-based devices only).
VXI_MRP_ISLW	Long word serial capability (message-based devices only).
VXI_MRR_ISDOR	DOR asserted (message-based devices only).
VXI_MRR_ISDIR	DIR asserted (message-based devices only).
VXI_MRR_ISERR	Word serial protocol error detected (message-based devices only).
VXI_MRR_ISRRDY	Read Ready asserted (message-based devices only).
VXI_MRR_ISWRDY	Write Ready asserted (message-based devices only).
VXI_MRR_ISFHSACT	FHS protocol active (message-based devices only).

VXI_MRR_ISLOCKED	Local lockout active (message-based devices only).

No valid *Bool_Attribute* values are defined for GPIB network devices.

If the requested boolean attribute is false, the function places a zero at the location pointed to by *Bool_Result_Ptr*. Otherwise, the function places a non-zero value at the location.

Return Value The function returns a Bus Management return value:

EPC_INV_ATTR	The parameter *Bool_Attribute* is invalid.
EPC_INV_NAME	A device with the specified name does not exist.
EPC_INV_PTR	One or more of the parameters *Name_Ptr* and *Bool_Result_Ptr* is invalid.
EPC_NO_DATA	The resource management database does not contain the requested attribute.
EPC_SUCCESS	The function completed successfully.

See Also **OlrmGetArbAttr, OlrmGetNumAttr, OlrmGetStrAttr.**

OlrmGetNmByGPA

Description Queries the device name corresponding to a GPIB address.

C Synopsis

```
#include "busmgr.h"
#include "olrm.h"

short
OlrmGetNmByGPA(short Primary, short Secondary, char FAR
*Name_Ptr);
```

Primary	*Primary* specifies a GPIB primary address.
Secondary	*Secondary* specifies a GPIB secondary address.
Name_Ptr	*Name_Ptr* specifies the location of a buffer where the name of the device corresponding to the specified GPIB address will be placed.

Visual Basic Synopsis

```
Declare Function
OlrmGetNmByGPA% Lib "olrmw16.dll"
                        (ByVal Primary%,
                        ByVal Secondary%,
                        ByVal Name_Ptr$)
```

Remarks **OlrmGetNmByGPA** queries the name of the device corresponding to the specified GPIB address and places the name in the buffer pointed to by *Name_Ptr*.

Primary specifies a GPIB primary address. Valid values are 0 through 30, inclusive.

Secondary specifies a GPIB secondary address. Valid values are -1 and 0 through 30, inclusive. If *Secondary* is -1, the function searches for a GPIB device with the specified primary address and no secondary address. Otherwise, the function searches for a GPIB device with the specified primary address and the specified secondary address.

Name_Ptr specifies the location of a buffer where a device's name will be placed. The buffer must be at least **DEVNAME_SZ** byte long.

Return Value The function returns a Bus Management return value:

EPC_INV_GPA	One or more of the parameters *Primary* and *Secondary* is invalid.
EPC_INV_PTR	The parameter *Name_Ptr* is invalid.
EPC_NO_DEVICE	A device corresponding to the specified GPIB address does not exist.
EPC_SUCCESS	The function completed successfully.

See Also **OlrmGetNmByNA, OlrmGetNmByULA.**

OlrmGetNmByNA

Description Queries the device name corresponding to a network address.

C Synopsis

#include "busmgr.h"
#include "olrm.h"

short
OlrmGetNmByNA(char **FAR** **Net_Address_Ptr*, char **FAR** **Name_Ptr*);

Net_Address_Ptr	*Net_Address_Ptr* specifies a network address string.
Name_Ptr	*Name_Ptr* specifies the location of a buffer where the name of the device corresponding to the specified network address will be placed.

Visual Basic Synopsis

Declare Function
OlrmGetNmByNA% Lib "olrmw16.dll"
(ByVal *Net_Address_Ptr*$,
ByVal *Name_Ptr*$)

Remarks **OlrmGetNmByNA** queries the name of the device corresponding to the specified network address and places the name in the buffer pointed to by *Name_Ptr*.

Net_Address_Ptr specifies the location of a null-terminated ASCII string representing a network address.

Name_Ptr specifies the location of a buffer where a device's name will be placed. The buffer must be at least **DEVNAME_SZ** byte long.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	One or more of the parameters *Net_Address_Ptr* and *Name_Ptr* is invalid.
EPC_NO_DEVICE	A device corresponding to the specified network address does not exist.
EPC_SUCCESS	The function completed successfully.

See Also **OlrmGetNmByGPA, OlrmGetNmByULA.**

OlrmGetNmByULA

Description Queries the device name corresponding to a VXIbus unique logical address.

C Synopsis

```
#include "busmgr.h"
#include "olrm.h"

short
OlrmGetNmByULA(unsigned short ULA, char FAR *Name_Ptr);
```

ULA	*ULA* specifies a VXIbus unique logical address.
Name_Ptr	*Name_Ptr* specifies the location of a buffer where the name of the device corresponding to the specified VXIbus unique logical address will be placed.

Visual Basic Synopsis

```
Declare Function
OlrmGetNmByULA% Lib "olrmw16.dll"
                        (ByVal ULA%,
                        ByVal Name_Ptr$)
```

Remarks **OlrmGetNmByULA** queries the name of the device corresponding to the specified VXIbus unique logical address and places the name in the buffer pointed to by *Name_Ptr*.

ULA specifies a VXIbus unique logical address. Valid values are 0 through 255, inclusive.

Name_Ptr specifies the location of a buffer where a device's name will be placed. The buffer must be at least **DEVNAME_SZ** byte long.

Return Value The function returns a Bus Management return value:

EPC_INV_PTR	The parameter *Name_Ptr* is invalid.
EPC_NO_DEVICE	A device corresponding to the specified VMEbus unique logical address does not exist.
EPC_SUCCESS	The function completed successfully.

See Also **OlrmGetNmByGPA, OlrmGetNmByNA.**

OlrmGetNumAttr

Description Queries a numeric attribute.

C Synopsis

```
#include "busmgr.h"
#include "olrm.h"

short
OlrmGetNumAttr(char FAR * Name_Ptr,
               short Num_Attribute,
               unsigned long FAR * Num_Result_Ptr);
```

Name_Ptr	*Name_Ptr* specifies a device name.
Num_Attribute	*Num_Attribute* specifies a numeric attribute.
Num_Result_Ptr	*Num_Result_Ptr* specifies a location where the specified numeric attribute will be placed.

Visual Basic Synopsis

```
Declare Function
OlrmGetNumAttr% Lib "olrmw16.dll"
                (ByVal Name_Ptr$$,
                ByVal Num_Attribute%,
                ByVal Num_Result_Ptr&)
```

Remarks **OlrmGetNumAttr** queries a numeric attribute of the specified device and places the result in the location pointed to by *Num_Result_Ptr.*

Name_Ptr is a null-terminated ASCII string specifying a device name.

Num_Attribute specifies the numeric attribute to query. Valid values for VXIbus devices are:

Num_Attribute	**Num_Result_Ptr*
VXI_ULA	Unique logical address.
VXI_IDREG	ID register.
VXI_DTREG	Device type register.
VXI_STREG	Status register.
VXI_OFFREG	Offset register.
VXI_MNFID	Manufacturer ID.
VXI_MODCOD	Model code.
VXI_DEVCLASS	Device class. 0=memory, 1=extended, 2=message-based, 3=register-based.
VXI_ADRSP	Address space. 0=A16/A24, 1=A16/A32, 3=A16 only.
VXI_A16BASE	A16 memory base.
VXI_A24BASE	A24 memory base (A16/A24 devices only).
VXI_A24SIZE	A24 memory size, in bytes (A16/A24 devices only).
VXI_A32BASE	A32 memory base (A16/A32 devices only).
VXI_A32SIZE	A32 memory size, in bytes (A16/A32 devices only).
VXI_STATDD	Status register device dependent bits. Defined and reserved bits are masked to zero.
VXI_BUSNUM	Bus mainframe number.
VXI_SLOTNUM	Slot number.
VXI_CMDRLA	Unique logical address of commander device.

VXI_S0LA	Unique logical address of slot-0 device.
VXI_SVARSZ	Servant area size.
VXI_HDLRMAP1 ... VXI_HDLRMAP7	Interrupt line assigned to handlers 1 through 7. A zero result indicates the handler is not assigned for the device.
VXI_INTRMAP1 ... VXI_INTRMAP7	Interrupt line assigned to interrupters 1 through 7. A zero result indicates the interrupter is not assigned for the device.
VXI_MEM_ATTREG	Attribute register (memory devices only).
VXI_MEM_TYPE	Memory type (memory devices only). 1 = ROM, 2 = other, 3 = RAM.
VXI_MEM_SPEED	Minimum memory access time, in nanoseconds (memory devices only).
VXI_MEM_DD	Attribute register device dependent bits (memory devices only). Defined and reserved bits are masked to zero.
VXI_SBC_REG	Subclass register (extended devices only)
VXI_SBC_RES	Reserved subclass ID (extended devices only).
VXI_SBC_MNFID	Subclass manufacturer ID (extended devices only).
VXI_SBC_MFSBC	Manufacturer subclass (extended devices only).
VXI_MSG_PTOREG	Protocol register (message-based devices only).
VXI_MSG_RSPREG	Response register (message-based devices only).
VXI_MSG_DHIREG	Data high register (message-based devices only). Warning: querying this register may modify device state.

VXI_MSG_DLOREG	Data low register (message-based devices only). Warning: querying this register may modify device state.
VXI_MSG_PRDD	Protocol register device dependent bits (message-based devices only). Defined and reserved bits are masked to zero.
VXI_MSG_RDPR	Word serial *Read Protocol* command response (message-based devices only).
VXI_MSG_RDPRDD	Word serial *Read Protocol* command response device dependent bits (message-based devices only) Defined and reserved bits are masked to zero.
VXI_MSG_RRDD	Response register device dependent bits (message-based devices only) Defined and reserved bits are masked to zero.

No valid *Num_Attribute* values are defined for GPIB network devices.

Return Value The function returns a Bus Management return value:

EPC_INV_ATTR	The parameter *Num_Attribute* is invalid.
EPC_INV_NAME	A device with the specified name does not exist.
EPC_INV_PTR	One or more of the parameters *Name_Ptr* and *Num_Result_Ptr* is invalid.
EPC_NO_DATA	The resource management database does not contain the requested attribute.
EPC_SUCCESS	The function completed successfully.

See Also **OlrmGetArbAttr, OlrmGetBoolAttr, OlrmGetStrAttr.**

OlrmGetStrAttr

Description Queries a string attribute.

C Synopsis

```
#include "busmgr.h"
#include "olrm.h"

short
OlrmGetStrAttr(char FAR * Name_Ptr,
               short Str_Attribute,
               char FAR * Str_Result_Ptr);
```

Name_Ptr	*Name_Ptr* specifies a device name.
Str_Attribute	*Str_Attribute* specifies a string attribute.
Str_Result_Ptr	*Str_Result_Ptr* specifies the location of a buffer where the specified string attribute will be placed.

Visual Basic Synopsis

```
Declare Function
OlrmGetStrAttr% Lib "olrmw16.dll"
                    (ByVal Name_Ptr$,
                    ByVal Str_Attribute%,
                    ByVal Str_Result_Ptr$)
```

Remarks **OlrmGetStrAttr** queries a string attribute of the specified device and places the result in the buffer pointed to by *Str_Result_Ptr*.

Name_Ptr is a null-terminated ASCII string specifying a device name.

Str_Attribute specifies the string attribute to query. Valid values for VXIbus devices are:

Str_Attribute	**Str_Result_Ptr*
VXI_CMDRNM	Name of commander device.
VXI_MFNM	Manufacturer name.
VXI_MODNM	Model name.
VXI_S0NM	Name of slot-0 device.

Valid *Str_Attribute* values for network devices are:

Str_Attribute	**Str_Result_Ptr*
NET_ADDRESS	Network address.

No valid *Str_Attribute* values are defined for GPIB devices.

Str_Result_Ptr specifies the location a buffer where the function places the result of the string attribute query. The buffer must be at least **ATTRIBUTE_SZ** bytes long.

Return Value The function returns a Bus Management return value:

EPC_INV_ATTR	The parameter *Str_Attribute* is invalid.
EPC_INV_NAME	A device with the specified name does not exist.
EPC_INV_PTR	One or more of the parameters *Name_Ptr* and *Str_Result_Ptr* is invalid.
EPC_NO_DATA	The resource management database does not contain the requested attribute.
EPC_SUCCESS	The function completed successfully.

See Also **OlrmGetArbAttr, OlrmGetBoolAttr, OlrmGetNumAttr.**

NOTES

4. Advanced Topics

This chapter discusses topics of interest to advanced application programmers. Topics include:

- Byte Ordering and Data Representation
- Handler Operations
- Event Handler Execution Under Windows
- Event Handler Implementation
- TTL Trigger Interrupt Handling on an EPC-7
- Using the backward-compatibility library

4.1 Byte Ordering and Data Representation

Byte ordering adds complexity to the VXIbus interface. Many VXIbus devices use the data formats of Motorola microprocessors. Others, including RadiSys EPC controllers, use the data format of Intel microprocessors. Although the Motorola and Intel microprocessors use the same data types, the hardware representations of these data types differ.

Figure 4-1 shows how the same sequence of bytes in memory is interpreted by Intel and Motorola microprocessors. Memory value 11 is at the lowest address and memory value AA is at the highest address. The data widths shown correspond to the data operand sizes found on both microprocessors.

Memory Value	Intel Order	Data Width	Motorola Order
11	11	8 bits	11
22	2211	16 bits	1122
33			
44	44332211	32 bits	11223344
55			
66	665544332211	48 bits	112233445566
77			
88	8877665544332211	64 bits	1122334455667788
99			
AA	AA998877665544332211	80 bits	112233445566778AA

Figure 4-1. Byte Order Example.

4.1.1 Byte Swapping Functions

The **EpcSwap*** functions convert 16-bit, 32-bit, 48-bit, 64-bit and 80-bit data between Intel and Motorola byte orders (8-bit data does not require conversion).

4.1.2 Correcting Data Structure Byte Ordering

Even if byte-swapping occurs during a transfer, byte ordering problems occur when data is copied between Motorola and Intel memory using a different data width than the width of the operand itself. This situation occurs when a data structure containing mixed-type fields is copied in a single operation.

The following code fragment illustrates how to use the **EpcSwap*** functions to correct the byte order in the local copy of the data structure:

```
struct DataStructure
{
        char        Field8;
        char        Field16[2];
        char        Field32[4];
        char        Field48[6];
        char        Field64[8];
        char        Field80[10];
} data;

/* Copy the data structure to EPC memory from the VXIbus. */

(void) EpcCopyData(Session_ID,
                   (void HUGE *) Mapped_Ptr,
                   (void HUGE *) &data,
                   sizeof(struct DataStructure),
                   EPC_8_BIT,
                   &Actual_Size);

/* Byte-swap the individual structure fields (data.Field8 */
/* is an 8-bit field, so it is already correct).          */

(void) EpcSwap16((unsigned short FAR  *) data.Field16);
(void) EpcSwap32((unsigned long FAR *) data.Field32);
(void) EpcSwap48((void FAR *) data.Field48);
(void) EpcSwap64((void FAR *) data.Field64);
(void) EpcSwap80((void FAR *) data.Field80);
```

In the above example, the data structure was copied from VXIbus memory one byte at a time. To copy data from EPC memory to Motorola-ordered VXIbus memory, byte-swap the fields of the structure in local memory (using the above byte swapping functions) and copy the data using the **EpcCopyData** function.

It is sometimes more efficient to copy blocks of data using data transfer width greater than the expected data width. If you use a greater data transfer width to copy data structures containing mixed-type fields to/from Motorola-order memory, do not use the EPC's hardware byte-swapping feature. Swap the data structure fields individually.

4.2 Event Handler Execution

These conditions must be true before an application's event handlers can execute:

- The application must use **EpcSetEventHandler** to install an error handler.
- The application must call **EpcSetEventEnableMask** to enable event reception.
- An event must occur.

The Bus Management API discards all events that occur before the application installs an event handler.

When an application installs an event handler and enables event reception, the event handler processes events as soon as they are received. The installed event handler executes as part of an interrupt thread, with virtual processor interrupts disabled, and using the installed event handler stack.

4.3 Event Handler Operations Under Windows

Event handlers can execute as part of an interrupt thread under Windows. This feature implies that an event handler can only call fully reentrant Bus Management, "C" library, Windows, DOS, and BIOS support functions.

Bus Management Library and ORLM library functions are fully reentrant and may be called from an event handler or any application code that executes as part of an interrupt thread. Note, however, that when called reentrantly, EPConnect Bus Management Library functions cannot yield the processor to other tasks while in a timing loop.

The following "C" library functions are reentrant under Microsoft "C" Version 6.0, and may be called from an event handler or any application code that executes as part of an interrupt thread (it is likely that this list is different for other releases of the Microsoft "C" compiler and for compilers from other vendors):

abs	**memccpy**	**strct**	**strnset**
atoi	**memchr**	**strchr**	**strrchr**
atol	**memcmp**	**strcmp**	**strrev**
bsearch	**memcpy**	**strcmpi**	**strset**
chdir	**memicmp**	**strcpy**	**strstr**
getpid	**memmove**	**stricmp**	**strupr**
halloc	**memset**	**strlen**	**swab**
hfree	**mkdir**	**strlwr**	**tolower**
itoa	**movedata**	**strncat**	**toupper**
labs	**putch**	**strncmp**	
lfind	**rmdir**	**strncpy**	
lsearch	**segread**	**strnicmp**	

No Windows functions are fully reentrant. As such, none of the Windows functions should be called from an event handler.

Not all DOS and BIOS functions are fully reentrant. However, mechanisms exist (the "InDos" and "CriticalError" flags) for avoiding DOS reentrancy by delaying background processing until DOS is not in use.

4.4 Event Handler Implementation

An event handler is called as part of an interrupt thread with its own stack. Care must be taken during implementation to avoid several pitfalls.

Since an event handler function is called as part of an interrupt thread, the event handler function must reload the data segment register (DS) with its data segment upon entry. Any of the following three methods will correctly load the data segment register for an event handler function:

1. Explicitly declare the event handler function to be an exported function in the EXPORTS section of the application's module definition (.DEF) file.

2. Explicitly declare the event handler function to be an exported function using the "C" language "_export" function declaration.

3. Explicitly declare that the event handler function reloads the data segment register upon entry using the "C" language "_loadds" function declaration.

Since an event handler function is called using the installed event handler stack, an event handler function written in "C" must be compiled with the assumption that the data segment is not equivalent to the stack segment (DS != SS). Otherwise, a catastrophic failure can occur when the event handler function is called. For Microsoft compilers, use the "/Alfw" memory model parameter. For Borland compilers, use the "-ml" memory model parameter.

Since an event handler function is called using the installed event handler stack, an event handler function written in "C" must be compiled with automatic stack checking disabled. Otherwise, a catastrophic failure will occur when the event handler is called. For Microsoft compilers, use the "/Gs" parameter. For Borland compilers, avoid using the "-N" parameter.

Since an event handler function is generally performance-critical, its code and data segments should be carefully defined for maximum performance. For maximum performance, define the event handler function's code and data segments to be PRELOAD, FIXED, and NONDISCARDABLE in the application's module definition (.DEF) file.

4.5 TTL Trigger Interrupts on an EPC-7

Receiving and processing TL trigger interrupts on an EPC-7 requires software intervention. EPC-7 hardware generates a TTL trigger interrupt when all of the following conditions are true:

- A bit in the TTL trigger interrupt enable register is set. The Bus Management Library function **EpcSetEventEnableMask** sets and/or clears the register's bits.
- The corresponding bit in the TTL trigger latch register is clear.
- The corresponding TTL trigger line is asserted for at least 30 nanoseconds.

The main complication in this scenario is that a bit in the TTL trigger latch register cannot be cleared until the corresponding TTL trigger line is deasserted. In order to clear a bit in the register, the register must be read while the corresponding TTL trigger line is deasserted. TTL trigger line assertion is not necessarily under EPC control.

The operation of the EPC-7 TTL trigger latch register has three potential side effects for software:

- If a TTL trigger interrupt remains enabled after receiving the initial interrupt and clearing the TTL trigger latch register, the CPU can be monopolized by redundant TTL trigger interrupts.
- If a TTL trigger latch register bit is not cleared before enabling the corresponding TTL trigger interrupt, it is possible to receive an interrupt for a TTL trigger that was asserted, latched, and deasserted long before the TTL trigger interrupt was enabled.
- If a TTL trigger latch register bit is not cleared after receiving the corresponding TTL trigger interrupt, the EPC will not latch subsequent TTL trigger line assertions and, therefore, will miss subsequent TTL trigger interrupts.

To avoid the first side effect, the Bus Management for Windows implementation globally disables a TTL trigger interrupt upon reception. In addition, the Bus Manager Library implementation provides sufficient functionality to avoid the other two side effects.

To avoid the side effect of receiving extraneous TTL trigger interrupts, execute **EpcGetMiscAttributes** before calling **EpcGetEventEnableMask** and **EpcSetEventEnableMask** to enable TTL trigger interrupts for a session.

For example:


```
void FAR PASCAL
EnableTTLTriggerInterrupts(unsigned long Session_ID, unsigned
long Event_Mask)
{
       unsigned long mask1;
       unsigned long mask2;

        /*
         * Wait for corresponding TTL trigger latch register
         * bits to clear, then enable the TTL trigger
         * interrupts.
         */

       mask1 = Event_Mask << 4;
       for (;;)
       {
              EpcGetMiscAttributes(Session_ID, &mask2);
              if ((mask1 & mask2) == 0)
              {
                     break;
              }
       }
       EpcGetEventEnableMask(Session_ID, &mask1);
       EpcSetEventEnableMask(Session_ID, mask1 | Event_Mask);
}
```

To avoid the side effect of missing multiple TTL trigger interrupts from the same TTL trigger, re-enable the interrupt immediately after receiving a TTL trigger interrupt, preferably as part of the event handler function itself. For example:

```
       void FAR
       TTLTriggerInterruptHandler(  unsigned long Session_ID,
                                    unsigned long Event_Mask,
                                    unsigned long Event_Data)
       {
            /*
             *Re-enable the TTL trigger interrupt.
```

```
        */
          EnableTTLTriggerInterrupts(Session_ID,Event_Mask;
        /*
         * Execute other event handler tasks...
         */
}
```

4.6 Backward-Compatibility Library

The Backward Compatibility Library (**EPCDICW.DLL**) and its corresponding import library (**EPCDICW.LIB**) provide a level of compatibility between the Windows and DOS programming interfaces. Most of the functions available in the DOS Bus Management Library are available in the Windows Backward-Compatibility Library with identical calling conventions. However, the functionality the two libraries provide is not strictly identical. Differences between the Windows Backward-Compatibility Library and the DOS Bus Management Library include the following:

- No Message Delivery System (MDS) functionality is available in the Windows Backward-Compatibility Library. If using MDS support under Windows, the application should be ported to SICL. If MDS support under Windows is a **requirement**, users should not upgrade beyond EPConnect/VXI for DOS version 3.11.

- The Windows Backward Compatibility Library supports **A16S**, **A24SD**, **A24S**, **A32SD**, and **A32S** VMEbus address modifiers only. Attempting to use other VMEbus address modifiers (**A16N**, **A16U**, **A24ND**, **A24NP**, **A24N**, **A24U**, **A24SP**, **A32ND**, **A32NP**, **A32N**, **A32U** and **A32SP**) under Windows results in an **ERR_FAIL** error and/or a null mapped pointer.

- The Windows Backward Compatibility Library can access the first gigabyte of A32 space only (addresses 0x00000000 through 0x3FFFFFFF). Attempting to map higher A32 space addresses under Windows results in an **ERR_FAIL** error and/or a null mapped pointer.

- The Windows Backward Compatibility Library automatically disables persistent interrupt and error events when they are received. Automatic disabling of persistent events prevents the generation of multiple,

redundant events. Persistent events include **EPC_MSG_INTR**, **EPC_TTL_TRIG*_INTR**, **EPC_SYSFAIL_ERR**, **EPC_ACFAIL_ERR**, and **EPC_WATCHDOG_ERR**. Under Windows, when one of these persistent events occurs, it must be re-enabled by the application before it will be received again.

- The Windows Backward Compatibility Library supports standard servant word serial communications only. The RadiSys-specific protocol for multiple-commander word serial communications is not supported. Under Windows, the multiple-commander arming codes (**EPC_WSRCV_DISARM**, **EPC_WSRCV_ARM**, and **EPC_WSRCV_ARMandENABLE**) and the single-commander arming codes (**EPC_WSRCV_FDISARM**, **EPC_WSRCV_FARM**, and **EPC_WSRCV_FARMandENABLE**) are equivalent.

5. Support and Service

5.1 In North America

5.1.1 Technical Support

RadiSys maintains a technical support phone line at (503) 646-1800 that is staffed weekdays (except holidays) between 8 AM and 5 PM Pacific time. If you have a problem outside these hours, you can leave a message on voice-mail using the same phone number. You can also request help via electronic mail or by FAX addressed to RadiSys Technical Support. The RadiSys FAX number is (503) 646-1850. The RadiSys E-mail address on the Internet is *support@radisys.com*. If you are sending E-mail or a FAX, please include information on both the hardware and software being used and a detailed description of the problem, specifically how the problem can be reproduced. We will respond by E-mail, phone or FAX by the next business day.

Technical Support Services are designed for customers who have purchased their products from RadiSys or a sales representative. If your RadiSys product is part of a piece of OEM equipment, or was integrated by someone else as part of a system, support will be better provided by the OEM or system vendor that did the integration and understands the final product and environment.

6.1.2 Bulletin Board

RadiSys operates an electronic bulletin board (BBS) 24 hours per day to provide access to the latest drivers, software updates and other information. The bulletin board is not monitored regularly, so if you need a fast response please use the telephone or FAX numbers listed above.

The BBS operates at up to 14400 baud. Connect using standard settings of eight data bits, no parity, and one stop bit (8, N, 1). The telephone number is (503) 646-8290.

5.2 Other Countries

Contact the sales organization from which you purchased your RadiSys product for service and support.

Index

I

NOTES