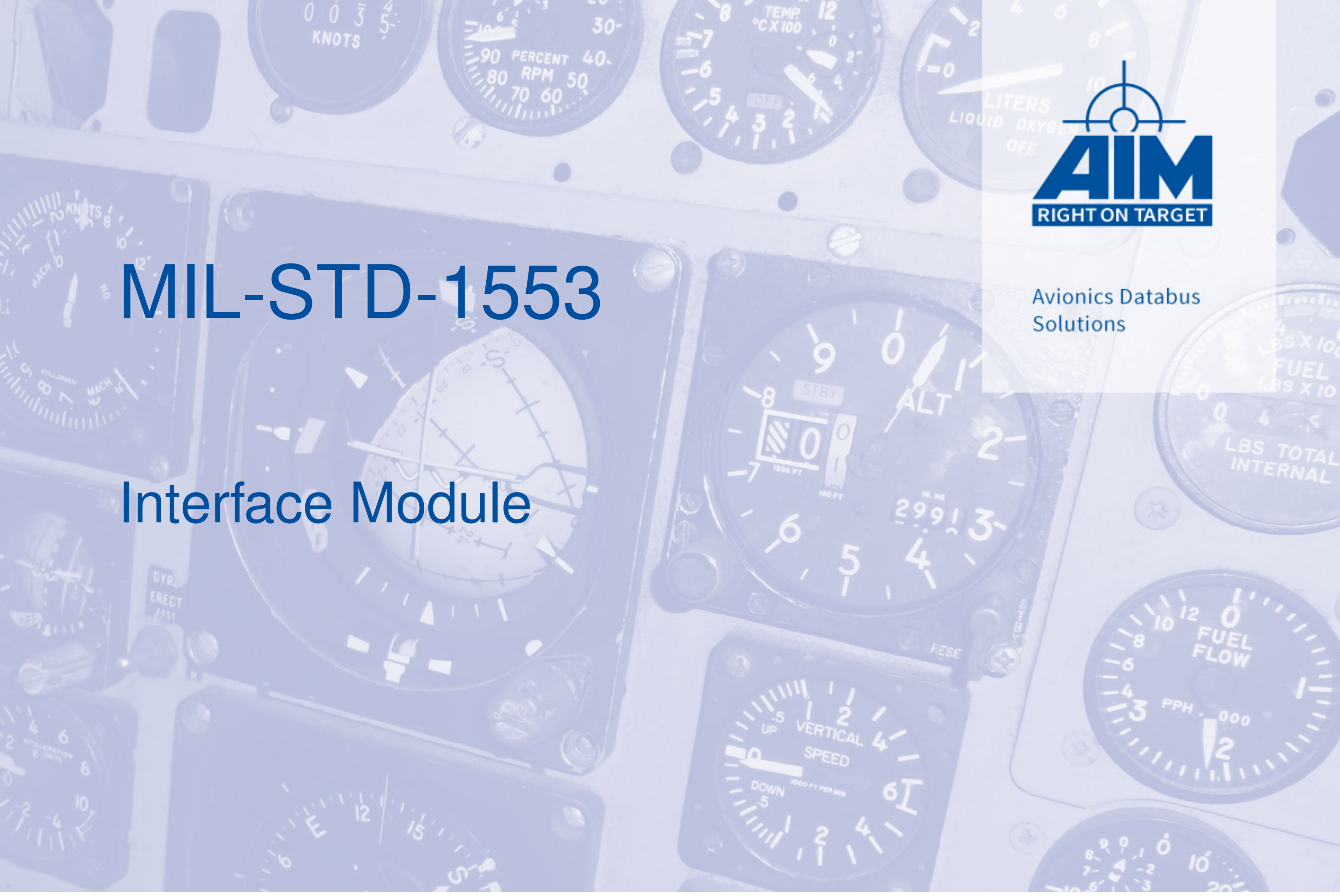


MIL-STD-1553

Interface Module

RIGHT ON TARGET

Avionics Databus Solutions

Reference Manual

Version 24.22.0
December, 2024

www.aim-online.com

MIL-STD-1553

Interface Module

Reference Manual

Version 24.22.0
December, 2024

AIM NO. 60-11900-36-24.22.0

AIM – Gesellschaft für angewandte Informatik und Mikroelektronik mbH

AIM GmbH
Sasbacher Str. 2
D-79111 Freiburg / Germany
Phone +49 (0)761 4 52 29-0
Fax +49 (0)761 4 52 29-33
sales@aim-online.com

AIM GmbH – Munich Sales Office
Terofalstr. 23a
D-80689 München / Germany
Phone +49 (0)89 70 92 92-92
Fax +49 (0)89 70 92 92-94
salesgermany@aim-online.com

AIM UK Office
Cressex Enterprise Centre, Lincoln Rd.
High Wycombe, Bucks. HP12 3RB / UK
Phone +44 (0)1494-446844
Fax +44 (0)1494-449324
salesuk@aim-online.com

AIM USA LLC
Seven Neshaminy Interplex
Suite 211 Trevose, PA 19053
Phone 267-982-2600
Fax 215-645-1580
sales@aim-online.us

DOCUMENT HISTORY

The following table defines the history of this document. Appendix A provides a more comprehensive list of changes made with each version.

Version	Cover Date	Created by	Description
23.0.x Rev. A	07.06.2016	Martin Haag	See Appendix A for details
24.0.x Rev. A	02.08.2016	Martin Haag	See Appendix A for details
24.2.x Rev. A	03.03.2017	Martin Haag	See Appendix A for details
24.2.x Rev. B	16.05.2017	Martin Haag	See Appendix A for details
24.2.x Rev. C	07.07.2017	Martin Haag	See Appendix A for details
24.3.x Rev. A	20.10.2017	Frank Schmidt	See Appendix A for details
24.4.x Rev. A	05.02.2018	Frank Schmidt	See Appendix A for details
24.4.x Rev. B	13.02.2018	Martin Haag	See Appendix A for details
24.5.x Rev. A	07.03.2018	Martin Haag	See Appendix A for details
24.5.x Rev. B	03.04.2018	Frank Schmidt	See Appendix A for details
24.5.x Rev. C	15.05.2018	Martin Haag	See Appendix A for details
24.7.x Rev. A	16.06.2018	Martin Haag	See Appendix A for details
24.7.x Rev. B	20.06.2018	Martin Haag	See Appendix A for details
24.8.x Rev. A	09.08.2018	Martin Haag	See Appendix A for details
24.9.x Rev. A	26.09.2018	Thomas Jahn	See Appendix A for details
24.10.x Rev. A	18.12.2018	Patrick Giesel	See Appendix A for details
24.11.x Rev. A	15.02.2019 11.04.2019	Patrick Giesel Thomas Jahn	See Appendix A for details
24.11.x Rev. B	28.05.2019	Patrick Giesel	See Appendix A for details
24.12.x Rev. A	19.03.2020	Thomas Jahn	See Appendix A for details
24.13.x Rev. A	08.05.2020 10.07.2020 28.07.2020	Thomas Jahn Thomas Jahn Thomas Jahn	See Appendix A for details
24.14.x Rev. A	22.03.2021	Thomas Jahn	See Appendix A for details
24.14.x Rev. B	25.08.2021	Thomas Jahn	See Appendix A for details
24.14.x Rev. C	03.12.2021	Martin Haag	See Appendix A for details
24.15.x Rev. A	02.02.2022	Martin Haag	See Appendix A for details
24.17.x Rev. A	01.08.2022 22.09.2022	Thomas Jahn Martin Haag	See Appendix A for details
24.17.x Rev. B	23.11.2022	Martin Haag	See Appendix A for details
24.18.x Rev. A	07.03.2023	Martin Haag Thomas Jahn	See Appendix A for details
24.19.x Rev. A	15.03.2024	Martin Haag Thomas Jahn	See Appendix A for details
24.21.x Rev. A	25.09.2024	Martin Haag	See Appendix A for details
24.22.x Rev. A	16.12.2024	Thomas Jahn	See Appendix A for details

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Welcome

Welcome to the Software Library Reference Manual for the AIM MIL-STD-1553 application programming interface. This reference manual, in conjunction with the Programmer's Guide, is intended to provide the software (s/w) programmer with the information needed to develop a host computer application interface to the AIM 1553 interface module. This reference manual provides the user with detailed programming information including library function call and header file details and specific troubleshooting information. The programmer's guide provides the 1553 application developer with high-level s/w development information including high level AIM 1553 interface module system design information, board support package (BSP) contents, user application system design concepts, function call guidelines, and sample programs.

1.2 How This Manual is Organized

This reference manual is divided into the following Sections:

Chapter 1
C Library Header Files

Provides helpful information about the content and structure of the Header Files used when developing C programs for the user's application interface to the AIM 1553 interface module.

Chapter 2
Library Admin Functions

Chapter 3
System Functions

Chapter 4
Calibration Functions

Chapter 5
Buffer Functions

Chapter 6
FIFO Functions

Chapters 2 - 6 comprise the system setup and support functions used when controlling an AIM 1553 interface module.

Chapter 7
Bus Controller Functions

Chapter 8
Remote Terminal Functions

Chapter 9
Bus Monitor Functions

Chapter 10
Replay Functions

Chapter 11
GenIo Functions

Sections 7 - 11 contain the function calls used to setup the main functional systems on the 1553 interface module including:

Bus Controller

Remote Terminal(s)

Bus Monitor

Replay

GenIo

Section 12 contains helpful information enabling the user to interpret error messages and steps to take to correct those errors.

Chapter 12
Trouble-shooting

App A
Document/ Software History

Provides detailed information regarding the history of the API Library reference software and documentation

1.3 Conventions Used

1.3.1 General Documentation Conventions

We use a number of different styles of text and layout in this document to help differentiate between the different kinds of information. Here are some examples of the styles we use and an explanation of what they mean:

Italics - used as a placeholder for the actual name, filename, or version of the software in use

Bold text - a function, or parameter, or used to highlight important information

Bold italics - caution, warning or note

Font - font used to show paths, directories and filenames within the body of text will be shown in blue, for example:

C:\Windows\System32\Drivers\Aim_mil.sys

A smaller version of this font will be used to list software code.

In addition to text and layout convention, there are a couple of naming conventions used to simplify the information herein. All AIM hardware systems/cards utilize the same s/w library functions also called the Application Program Interface (API). Therefore, for ease of documentation flow, this s/w library will be referred to from this point on as the **API S/W Library**. In addition, the software and firmware contained on the AIM bus interface 1553 device will be referred to as the **Target S/W**. Several references will be made within the API S/W Library function description to the Application Interface. The **Application Interface** is the software interface between the API S/W Library function calls and the Target S/W.

1.3.2 Naming Conventions

Naming conventions have been used for naming constants, structures, functions calls and data types throughout the API S/W Library. All constants, structures and functions used in the API S/W Library are defined in the **Ai1553i_def.h** header file which is contained in Appendix B. Data types used in the API S/W Library are defined in **ai_cdef.h**, also defined in Appendix B. Naming conventions used include the following

- **Constants** - For every function call, a list of constants have been defined to better describe the numerical value of the function input or output (located in Ai1553i_def.h)
- **Structures** - Named as **ty_api_*name*** where *name* is unique to the structure (located in Ai1553i_def.h)
- **Functions** - Named as either **Api*name*** or **ApiCmd*name*** -

  Apiname functions do not involve driver commands to the biu

  ApiCmdname functions involve driver commands to the biu (located in Ai1553i_def.h)
- **Data Types** - all variable are assigned an AIM equated data type as shown in Table 1.1 below (defined in ai_cdef.h):

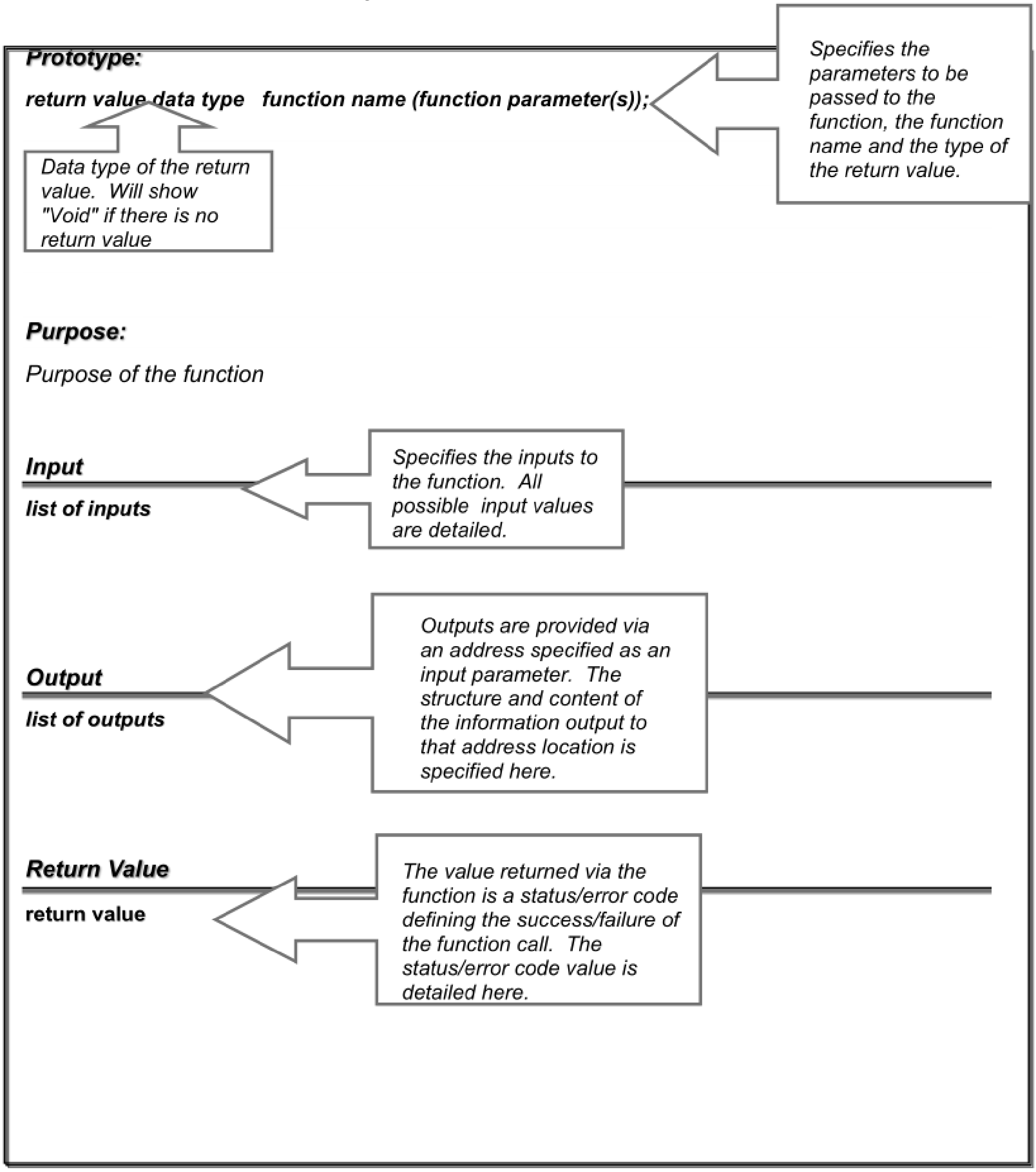
Table 1.1: API S/W Library Data type Naming Conventions

API S/W Library	Data Type	Size (in bytes)
AiInt	Int	4
AiUInt	unsigned int	4
AiInt8	char	1
AiUInt8	unsigned char	1
AiInt16	short	2
AiUnt16	unsigned short	2
AiInt32	int	4
AiUInt32	unsigned int	4
AiInt64	long long	8
AiUInt64	unsigned long long	8
AiChar	Char	1
AiUChar	unsigned char	1
AiDouble	double	8
AiFloat	float	4

1.3.3 Function Call Documentation Conventions

Each function call contained in this document uses a standard documentation format. The information included for each function call is as follows:

Figure 1.1: Document Conventions



1.3.3.1 Conventions for parameters ‘ul_ModuleHandle’ and ‘biu’

Most API S/W Library function calls require the parameters ***“ul_ModuleHandle”*** and ***“biu”***. In order to simplify the manual, these two function parameters are not detailed in the documentation for each function call, therefore, they are both defined here as follows:

- “***ul_ModuleHandle***” - usually the first function parameter. This parameter determines the AIM destination module handle. The handle is returned by ApiOpen() or ApiOpenEx().
- “***biu***” - usually the second function parameter. This parameter determines the Bus Interface Unit (biu) of the selected AIM module.

Value	Constant	Comment
1	**API_BIU_1**	Bus Interface Unit 1 on the AIM board
2	**API_BIU_2**	Bus Interface Unit 2 on the AIM board
..	**..**	Bus Interface Unit 3 on the AIM board
8	**API_BIU_8**	Bus Interface Unit 8 on the AIM board

Note:
The ‘biu’ parameter is only relevant if the command ApiOpen() was used to get the module handle. When using ApiOpenEx() the ‘biu/stream’ information is already coded within the module handle and therefore this parameter is ignored in this case.

Note:
For 3910 board application interfaces and opened with ApiOpen():
For all functions that require ’biu’ as input, only API_BIU_1 is valid, with the exception of ApiInstIntHandler/ApiDelINtHandler functions.
(For 3910 boards, BIU1 is always the LS BIU and BIU2 is the HS BIU. All 3910Hs functions automatically program API_BIU_2 for HS and API_BIU_1 [if required] for LS functionality.)

1.3.4 Function Calling Convention

For Microsoft Windows the library is created using the “_cdecl” calling convention. To avoid side effects in a Microsoft Windows operating system be sure to use also “_cdecl” as the calling convention in your application.

1.3.5 Buffer Header ID, Buffer ID and Transfer ID Ranges

In several function calls, a Buffer Header ID, Buffer ID or Transfer ID parameter is required. The ID’s maximum value can be obtained with **ApiCmdSysGetMemPartition**.

1.4 Special Board Functionality

The AIM boards can be configured with restricted functionality, thus providing the user with only the function(s) required by their application. When the AIM board is delivered with restricted functionality, the application program developer will be restricted to utilize only the API Software Library Reference calls that are applicable to the functional capabilities provided on the AIM board as defined in Table 1.2.

Table 1.2: AIM Board Type Restrictions

AIM Board FunctionalityType	Description	Restrictions
<AIM Board>	**Full Functionality**	None
<AIM Board>-**S**	**BC/RT Simulator** (Concurrent RT(s) (up to 31) and BC)	BM function calls cannot be used.
<AIM Board>-**M**	**BC+BM or RT(s)+BM**	All API Software Library calls can be used, however, only one Board function, either the BC or RT(s), and each can run a concurrent BM (relating to the start/halt commands.) There is no restriction related to simulatneously setting up the BC, RT(s) or BM including the buffers, transfer and read/write memory, etc.

1.5 Applicable Documents

1.5.1 Industry Documents

MIL-STD-1553B, Department of Defense Interface Standard for Digital Time Division Command/Response Multiplex Data Bus, Notice 1-4, January 1996

PCI Local Bus Specification, Revision 2.1, June 1991

1.5.2 AIM Document Family

AIM has developed several documents that may be used to aid the developer with other aspects involving the use of the AIM 1553 interface module(s). These documents and a summary of their contents are listed below:

MIL Programmer's Guide – Provides higher level programming guidelines along with trouble shooting tipps and migration information.

VME 1553 VxWorks Programmer's Guide - provides the 1553 application developer with high-level s/w development information including high level VME 1553 system design information, board support package contents, user application system design concepts, function call guidelines, and sample programs. This guide is to be used in conjunction with this reference manual.

MIL-STD-1553 Tutorial - provides a general overview of MIL-STD-1553 including MIL-STD-1553 history and a complete annotated version of the MIL-STD-1553B specification and an interpretation of the specification contents.

MIL-STD-1553 Getting Started Manual - assists the first time users of the AIM 1553 boards with software installation, hardware setup and starting a sample project.

Hardware Manuals -- provide the hardware user's manual for the specified modules. The documents cover the hardware installation, the board connections the technical data and a general description of the hardware architecture. The following hardware manuals are available:

- AEC1553 Hardware Manual (ExpressCard Bus modules)
- AMCX1553 Hardware Manual (PMC Bus modules)
- AXC1553 Hardware Manual (XMC Bus modules)
- AXCX1553 Hardware Manual (XMC Bus modules)
- ACE1553 Hardware Manual (PXIe Bus modules)
- ACXX1553 Hardware Manual (PXI Bus modules)
- ASC1553-1 Hardware Manual (USB-A 2.0 Bus modules)
- ASC1553-2 Hardware Manual (USB-C 3.0 Bus modules)
- ANET1553 Hardware Manual (Ethernet Bus modules)

- AME1553 Hardware Manual (Mini-PCIe modules)
- AME1553-1-AP Hardware Manual (AcroPack Mini-PCIe modules)
- AMEE1553 Hardware Manual (Mini-PCIe modules)
- AM2E1553 Hardware Manual (M.2-B-M modules)

1.6 C Header Files

This chapter introduces you to the header files required when using the API S/W Library. Once the BSP has been installed, the header files are located on your system at a platform specific location. (See Getting Started Manual for software installation instructions.) These header files include the following:

(a) **Api1553.h** - this header file provides for the inclusion of the main header files: **Ai_cdef.h** and **Ai1553i_def.h** and **Ai1553i_fnc.h** (described in b., c. and d. below) and in VME environments **AiVmeGeneric.h**. **This is the header file you will need to include in your application program.**

(b) **Ai_cdef.h** - this header file provides definition of data types used in the API S/W Library functions. It also provides for the use of different compilers supported by the PCI S/W Library (See PCI Programmers Guide for a complete list of compilers supported.).

(c) **Ai1553i_def.h** - this header file provides the inclusion of **Ai1553i_fnc.h**. All of these files provide definition of constants and structures including but not limited to function call errors, parameters used within each function, all structure definitions, boundary constants, driver command codes (also listed in chapter 12 - Troubleshooting).

(d) **Ai1553i_fnc.h** – this header file provides the function definitions

(e) **AiVmeGeneric.h** – this header file provides VME specific function prototpyes and definitions of constants and structures

There is a separate header **AiOs.h** file which may also be included to an application. It includes a operating system specific header file like **AiOsWindows.h**, which provide a set of functions that do the same for each operating system, but have to be encoded a little bit differently.

2 LIBRARY ADMINISTRATION AND INITIALIZATION FUNCTIONS

Chapter 2 defines the Library Administration function calls of the API S/W Library. The Library Administration functions provide general library initialization, and shutdown, interrupt handler setup, and error message handling setup. Table 2.1 defines the list and definition of general Library Administration functions, Table 2.2 the VME specific administration functions. The function calls in this table are listed in a functional order, however, the detailed descriptions of the Library Administration function calls in the following sections are in alphabetical order.

Table 2.1: General Library Administration Function Descriptions

Function	Description
ApiClose	Closes AIM module interface - called last
ApiConnectToServer	Establishes a network connection to a specified PC server where AIM Network Server (ANS) software is running.
ApiDelIntHandler	Removes the pointer interface to the interrupt handler function
ApiDisconnectFromServer	Disconnects the network connection
ApiExit	This function cleans up the library resources so it is safe to unload
ApiFlushIntHandler	Deletes all pending interrupts for this device within the device driver
ApiGetBoardInfo	Get information about the assigned hardware
ApiGetBoardName	Get the board name for this module handle
ApiGetDeviceConfig	Get the device config
ApiGetErrorDescription	Get a error message for a given error code
ApiGetErrorMessage	Get a error message for a given error code
ApiGetLibraryInfo	Reads extended information about the current library settings
ApiGetServerInfo	Retrieves information about AIM boards installed on the ANS PC
ApiInit	Initializes the API S/W Library Application Interface - performed first
ApiInstIntHandler	Provides a pointer to the interrupt handler function
ApiOpenEx	Initializes the AIM module & stream & provides the handle for future board commands
ApiSetDeviceConfig	Sets the device config
ApiSetDllDbgLevel	Sets the debug output level

Table 2.2: VME Specific Library Administration Function Descriptions

Function	Description
AiVmeExamineSlot	Examines the board(s) on a specific VME slot/ A16 address and runs a PCI config cycle
Ai1553CheckModule	Checks the type of module
AiPciScan	Scans the local PCI bus for known boards and returns the number of boards handled by this driver
AiPciGetHeader	Provides the PCI header information of a board on the local PCI bus.
AiVmeInitGenericInterrupt	Sets interrupt specific parameters of the board.
AiVme1553MapModule	Maps a board to the VME bus
AiVme1553UnmapModule	Removes a board from the VME bus

2.1 General Library Administration Functions

2.1.1 ApiClose

Prototype:

```
AiReturn ApiClose(AiUInt32 ul_ModuleHandle);
```

Purpose:

Closes the API S/W Library Application Interface for the specified module. This function must be called last in an application program. No function (except **ApiOpen(..)**) shall be called after **ApiClose(..)**.

Input:

None

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

2.1.2 ApiConnectToServer

Prototype:

```
AiReturn ApiConnectToServer(AiUChar *pszNetworkAdress,
                            AiInt16*w_ServerBoards);
```

Purpose:

Establishes a network connection to a specified server PC, where the AIM Network Server (ANS) is located.

Note:
This function is not thread safe. See section Symmetric Multiprocessing in the Programmers Guide for details.

Input:

`AiUChar *pszNetworkAdress`

Name of the PC, where the ANS is located.

Value	Description
<SrvName>	Name of the PC, where the ANS is located (e.g. “SW-PC-06” or “192.168.0.119”)

Output:

`AiInt16 *w_ServerBoards`

Value	Description
0	No installed AIM board found on server PC
>0	Number of AIM boards installed on server PC

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

2.1.3 ApiDelIntHandler

Prototype:

```
AiReturn ApiDelIntHandler(AiUInt32 ul_ModuleHandle,
                          AiUInt8 biu, AiUInt8 uc_Type);
```

Purpose:

"Uninstalls" an user interrupt handler function, which has been installed previously with the function **ApiInstIntHandler**. This function will remove the pointer to the interrupt handler function for the specified interrupt. It is necessary to call this function for each interrupt type (BC, RT, BM, Replay or BC-Branch) to be removed.

Note:
Under VxWorks the parameter uc_Type is not used, the function should only called once to remove the interrupt function

Input:

`AiUInt8 biu`

Biu Number or Stream Type

Board Opened With	Value	Constant	Description
ApiOpen()	**1**	**API_BIU_1**	Remove interrupt routine for BIU1
	2	**API_BIU_2**	Remove interrupt routine for BIU2
	3	**API_BIU_3**	Remove interrupt routine for BIU3
	4	**API_BIU_4**	Remove interrupt routine for BIU4
ApiOpenEx()	**1**	**API_INT_LS**	Remove interrupt routine for LS
	2	**API_INT_HS**	Remove interrupt routine for HS

`AiUInt8 uc_Type`

Interrupt Type
Defines the type of interrupt which will be "uninstalled" for the given AIM board.

Value	Constant	Description
0	**API_INT_RT**	interrupt for RT related events
1	**API_INT_BC**	interrupt for BC related events
2	**API_INT_BM**	interrupt for BM related events
3	**API_INT_REPLAY**	interrupt for Replay related events
4	**API_INT_BC_BRANCH**	interrupt for BC Branch (e.g. Skip) related events

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

2.1.4 ApiDisconnectFromServer

Prototype:

```
AiReturn ApiDisconnectFromServer(AiUChar * pszNetworkAdress);
```

Purpose:

Disconnect a previously established network connection from a specified server PC, where the AIM Network Server (ANS) is located.

Note:
This function is not thread safe. See section Symmetric Multiprocessing in the Programmers Guide for details.

Input:

`AiUChar *pszNetworkAdress`

Name of the PC, where the ANS is located.

Value	Description
<SrvName>	Name of the PC, where the ANS is located (e.g. “\\SW-PC-06” or “192.168.0.119”)

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

2.1.5 ApiExit

Prototype:

```
AiReturn ApiExit(void);
```

Purpose:

This function cleans up the library resources so it is safe to unload it.

Note:
This function is not thread safe. See section Symmetric Multiprocessing in the Programmers Guide for details.

Input:

none

Output:

none

Return Value:

Always zero.

2.1.6 ApiFlushIntHandler

Prototype:

```
AiReturn ApiFlushIntHandler(AiUInt32 ul_ModuleHandle);
```

Purpose:

Clears pending interrupts for the device from the driver. In the rare event of an application crash, the device can continue to generate interrupts to the driver even after the application is crashed. The device stops generating interrupts when the application is restarted and **ApiCmdReset** is called for each stream. If the application installs the interrupt handler with **ApiInstIntHander** in this case, the handler will immediately begin to be called with the previously received and unhandled interrupt events. To avoid this it can be necessary to call this function before **ApiInstIntHander** to clear the list of pending interrupt events.

Note:
Because the interrupts are handled on a device level this function clears all interrupts of the device for all streams.

Note:
This function is not thread safe. See section Symmetric Multiprocessing in the Programmers Guide for details.

Input:

None

Output:

None

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

2.1.7 ApiGetBoardInfo

Prototype:

```
AiReturn ApiGetBoardInfo(AiUInt32 ul_Module,
                         TY_API_BOARD_INFO *px_BoardInfo);
```

Purpose:

Reads extended information about the board capabilities.

Input:

none

Output:

TY_API_BOARD_INFO *px_BoardInfo

Library Info Structure

```
typedef union {
  AiUInt32 ul_All;
  struct {
    AiUInt32 ul_Res: 26;
    AiUInt32 b_DigitalWrap: 1;
    AiUInt32 b_Packed: 1;
    AiUInt32 b_Network: 1;
    AiUInt32 b_Transformer: 1;
    AiUInt32 b_Direct: 1;
    AiUInt32 b_Isolated: 1;
  }
  b;
}
TY_COUPLING_CAPABILITIES;

typedef union {
  AiUInt32 ul_All;
  struct {
    AiUInt32 ul_Res: 27;
    AiUInt32 b_NoExternalSignalOnInternalMode : 1;
    AiUInt32 b_InstantIrigSetOnInternalMode : 1;
    AiUInt32 b_Sinusoidal: 1;
    AiUInt32 b_FreeWheeling: 1;
    AiUInt32 b_IrigSwitch: 1;
  }
  b;
```

```
}
TY_IRIG_CAPABILITIES;

typedef struct ty_api_board_capabilities {
  AiUInt32 ul_CanChangeAmplitude;
  TY_COUPLING_CAPABILITIES x_CouplingCapabilities;
  TY_IRIG_CAPABILITIES x_IrigCapabilities;
}
TY_API_BOARD_CAPABILITIES;

typedef struct ty_api_board_info {
  AiUInt32 ul_DeviceType;
  AiUInt32 ul_NumberOfChannels;
  AiUInt32 ul_NumberOfBiu;
  AiUInt32 ul_NovRamBoardType;
  AiUInt32 ul_NovRamBoardConfig;
  AiUInt32 ul_SerialNumber;
  AiUInt32 ul_PartNumber;
  AiUInt32 ul_GlobalRamSize;
  AiUInt32 ul_SharedRamSize;
  AiUInt32 ul_GlobalRamStartOffset[MAX_BIU];
  TY_API_BOARD_CAPABILITIES x_BoardCapabilities;
}
TY_API_BOARD_INFO;
```

AiUInt32 ul_DeviceType

Indicates the device board type.

Value	Description
74	AEC1553-1
75	AEC1553-2
76	AXC1553-1
77	AXC1553-2
78	AXC1553-4
79	AMCX1553-1
80	AMCX1553-2
81	AMCX1553-4
82	APE1553-1
83	APE1553-2
84	APE1553-4
85	ANET1553-1
86	ANET1553-2
87	ANET3910-1
88	ANET3910Xp-1
89	ASC1553-1
90	APXX3910
91	APXX3910Xp
92	APEX3910
93	APEX3910Xp
94	AME1553
95	ACEX3910
96	ACEX3910Xp
97	AXE1553-1
98	AXE1553-2
99	AXE1553-4
100	AMCE1553-1
101	AMCE1553-2
102	AMCE1553-4
103	ASE1553-1
104	ASE1553-2
105	ASE1553-4
106	AXC3910
107	AXC3910Xp
108	AMEE1553-1
109	AMEE1553-2
110	ASC1553-2
111	AXCX1553-1
112	AXCX1553-2
113	AM2E1553-1
114	AM2E1553-2
115	AXCX1553-4

AiUInt32 ul_NumberOfChannels

Indicates the number of channels on the board. In case of a 3910 board, LS and HS part will be indicated as one channel. e.g. API1553-2 has 2 channels, API3910 has one channel

AiUInt32 ul_NumberOfBiu

Indicates the number of BIU processors on the board

AiUInt32 ul_NovRamBoardType

Indicates the Board Type that is setup onboard

AiUInt32 ul_NovRamBoardConfig

Indicates the Board Configuration that is setup onboard

AiUInt32 ul_SerialNumber

Indicates the Board Serial Number

AiUInt32 ul_PartNumber

Indicates the AIM Part Number of the board

AiUInt32 ul_GlobalRamSize

Indicates the Global RAM size (in 64kBytes steps)

AiUInt32 ul_SharedRamSize

Indicates the Shared RAM size (in 64kBytes steps)

AiUInt32 ul_GlobalRamStartOffset[MAX_BIU]

Indicates for each BIU on the board with which offset its Global RAM starts (relative to the Global RAM Start).

AiUInt32 x_BoardCapabilities.ul_CanChangeAmplitude

Indicates the board output amplitude capabilities

Value	Constant	Description
0	-	Output amplitude cannot be changed
1	-	Output amplitude can be changed

AiUInt32 x_BoardCapabilities.x_CouplingCapabilities.ul_All

Indicates the board coupling capabilities

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
ISO	DIR	TRA	NET	PAK	DIG	reserved (0)	

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
reserved (0)							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
reserved (0)							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
reserved (0)							

PAK

If set to 1, channel A and B of a given stream are packed and cannot be changed independently.

NET

If set to 1, Network coupling mode is available

TRA

If set to 1, Tranformer coupling mode is available

DIR

If set to 1, Direct coupling mode is available

ISO

If set to 1, Isolated coupling mode is available

DIG

If set to 1, Digital wrap around mode is available

`AiUInt32 x_BoardCapabilities.x_IrigCapabilities.ul_All`

Indicates the board IRIG capabilities

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
SW	FRW	SIN	INST	NOEXT	reserved (0)		

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
reserved (0)							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
reserved (0)							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
reserved (0)							

NOEXT

If set to 1, no externael IRIG signal available if Irig set to internal

INST

If set to 1, IRIG signal is available without delay

SIN

If set to 1, IRIG signal is sinusoidal

FRW

If set to 1, Free Wheeling is available

SW

If set to 1, IRIG can be switched between intern / extern

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

2.1.8 ApiGetBoardName

Prototype:

```
const char * ApiGetBoardName(AiUInt32 ul_ModuleHandle);
```

Purpose:

Return board name for this module handle.

Input:

None

Output:

None

Return Value:

`const char *`

The name of the board.

2.1.9 ApiGetDeviceConfig

Prototype:

```
AiReturn ApiGetDeviceConfig(AiUInt32 ul_Module,
                            TY_API_DEVICE_CONFIG* px_Config);
```

Purpose:

Returns the device configuration in px_Config.

Input:

none

Output:

TY_API_DEVICE_CONFIG * px_Config

The device config struct.

```
typedef struct ty_api_device_config {
  AiUInt8 uc_DmaEnabled;
  AiUInt8 uc_DataQueueMemoryType;
  AiUInt8 uc_DataQueueMode;
  AiUInt8 uc_ReservedB4;
  AiUInt16 uw_ReservedW1;
  AiUInt16 uw_ReservedW2;
  AiUInt32 ul_DmaMinimumSize;
  AiUInt32 ul_IntRequestCount;
  AiUInt32 ul_DriverFlags;
  AiUInt32 ul_ReservedLW4;
  AiUInt32 ul_ReservedLW5;
  AiUInt32 ul_ReservedLW6;
  AiUInt32 ul_ReservedLW7;
  AiUInt32 ul_ReservedLW8;
}
TY_API_DEVICE_CONFIG;
```

AiUInt8 uc_DmaEnabled

Enable or disable DMA read transfers. This value is only applicable for system driver version 12 or higher.

Value	Description
0	DMA disabled
1	DMA enabled

AiUInt32 ul_DmaMinimumSize

The memory size limit for DMA transfers. If a requested memory block is bigger or equal to this value a DMA transfer will be issued. This value is only applicable for system driver version 12 or higher.

AiUInt8 uc_DataQueueMemoryType

The memory type of the memory where the data queue headers are located.

Value	Define	Description
1	API_MEMTYPE_SHARED	Shared memory
2	API_MEMTYPE_LOCAL	Local memory (AyI)

AiUInt8 uc_DataQueueMode

The data queue mode determines how the data is copied to the host.

Value	Define	Description
0	-	data will copied by the ASP
1	-	data will be copied direct to the host

AiUInt32 ul_IntRequestCount

The number of interrupt requests sent to the driver.

AiUInt32 ul_DriverFlags

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
						INT_ENA	

INT_ENA (write only)

This flag might be 0, 1 or 2 depending on what was set in a previous call to ApiSetDeviceConfig. This flag does however not reflect changes that have been done by other processes. By default the interrupts are physically enabled and the flag will return 0.

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

2.1.10 ApiGetDriverInfo

Prototype:

```
AiReturn ApiGetDriverInfo(AiUInt32 ul_Module,
                          TY_API_DRIVER_INFO * px_Info);
```

Purpose:

Returns information from the system driver.This function can be called even if the on board components are not running or the device is not yet initialized. This function is internal and only used for update purpose.

Input:

none

Output:

TY_API_DRIVER_INFO * px_Info

The driver info struct.

```
typedef struct ty_api_driver_info {
  AiUInt8 uc_DeviceGroup;
  AiUInt8 uc_ReservedB2;
  AiUInt8 uc_ReservedB3;
  AiUInt8 uc_ReservedB4;
  AiUInt16 uw_ReservedW1;
  AiUInt16 uw_ReservedW2;
  AiUInt32 ul_DriverFlags;
  AiUInt32 ul_SN;
  AiUInt32 ul_BoardConfig;
  AiUInt32 ul_BoardType;
  AiUInt32 ul_OpenConnections;
  AiUInt32 ul_ReservedLW6;
  AiUInt32 ul_ReservedLW7;
  AiUInt32 ul_ReservedLW8;
}
TY_API_DRIVER_INFO;
```

AiUInt8 uc_DeviceGroup

The hardware platform group that the device belongs to.

Value	Define	Description
0	AI_DEVICE_UNKNOWN	Unrecognized
1	AI_DEVICE_AYI	AyI
2	AI_DEVICE_AYX	AyX
3	AI_DEVICE_AMC	Amc
4	AI_DEVICE_AMC_ASP	Amc-Asp
5	AI_DEVICE_AYE	AyE
6	AI_DEVICE_AYE_ASP	AyE-Asp
7	AI_DEVICE_AYX_GNET	AyX-Gnet
8	AI_DEVICE_USB	USB
9	AI_DEVICE_AYS_ASP	AyS-Asp
10	AI_DEVICE_AYS_ASP_MA	AyS-Asp Mixed
11	AI_DEVICE_ZYNQMP_ASP	ZynqMP
12	AI_DEVICE_AYS	AyS

AiUInt32 ul_DriverFlags

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
							TOUT

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
					DNBA	DNBD	ASP

TOUT

This flag indicates “1” that the timout value is used. If “0” the driver does not time out on ASP commands.

DNBA

This flag indicates “1” that the driver dows boot the device but does not start the virtual ASP emulation. Do Not Boot ASP.

DNBD

This flag indicates “1” that the driver does not boot the device or start the virtual ASP emulation. Do Not Boot Device.

ASP

This flag indicates “1” if the device ASP processor is available and activated.

AiUInt32 ul_SN

The device serial number. Only valid vor AyE and USB devices.

AiUInt32 ul_BoardConfig

The device board config. Only valid vor AyE and USB devices.

AiUInt32 ul_BoardType

The device board type. Only valid vor AyE and USB devices.

AiUInt32 ul_OpenConnections

The number of connected applications including the current connection.

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

2.1.11 ApiGetErrorMessage

Prototype:

```
const char * ApiGetErrorMessage(AiReturn error_code);
```

Purpose:

Returns a string with a description for the error code passed as input. This function might be called with a return value from any API function.

All API calls return API_OK in a no error condition. The return value of each API call should be checked if it is not equal to API_OK.

Note:
This function replaces the obsolete function ApiGetErrorDescription. The old function is still working but is limited to handle 16 bit error codes.

Input:

`AiReturn error_code`

The error code to be described.

Output:

none

Return Value:

`const char *`

A pointer to a string with the error description.

2.1.12 ApiGetLibraryInfo (obsolete)

Prototype:

```
AiReturn ApiGetLibraryInfo(TY_API_LIB_INFO *px_LibInfo);
```

Purpose:

Reads extended information about the current library settings.

Input:

none

Output:

TY_API_LIB_INFO *px_LibInfo

Library Info Structure

```
typedef struct ty_api_lib_info {
  AiUInt32 ul_AttachedApps;
}
TY_API_LIB_INFO;
```

AiUInt32 ul_AttachedApps

Number of applications (processes) that are currently attached to the library.

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

2.1.13 ApiGetServerInfo

Prototype:

`AiReturn ApiGetServerInfo(TY_SERVER_INFO *pServerInfo);`

Purpose:

Reads extended information about the AIM boards installed on the AIM Network Server (ANS) PC.

Note:
The network connection to the server PC already has to be established with the ApiConnectToServer call.

Input:

none

Output:

`TY_SERVER_INFO *pServerInfo`

Server Info Structure

```
typedef struct ty_tagserverinfo {
  TY_VER_INFO server_version;
  AiUInt32 protocol_major;
  AiUInt32 protocol_minor;
  AiChar application_name[128];
  AiChar description[128];
  AiChar host_name[128];
  AiChar os_info[128];
}
TY_SERVERINFO;
```

`TY_VER_INFO server_version`

The version information of the ANS server

`AiUInt32 protocol_major`

The major version of the ANS protocol

`AiUInt32 protocol_minor`

The minor version ot the ANS protocol

`AiChar application_name[128]`

The name of the server application.

`AiChar description[128]`

The server description.

`AiChar host_name[128]`

The server host name.

`AiChar os_info[128]`

Server operating system information.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

2.1.14 ApiInit

Prototype:

```
AiReturn ApiInit(void);
```

Purpose:

This function initializes the entire Application Interface and must be called first in an application program, before any other API S/W Library function is applied.

Note:
This function is not thread safe. See section Symmetric Multiprocessing in the Programmers Guide for details.

Input:

none

Output:

none

Return Value:

Operating System	Value
Windows	Amount of boards found
Linux	Amount of boards found
VxWorks	Always 1, due to the fixed memory mapping

2.1.15 ApiInstIntHandler

Prototype:

```
AiReturn ApiInstIntHandler(AiUInt32 ul_ModuleHandle, AiUInt8 biu,
                           AiUInt8 uc_Type, TY_INT_FUNC_PTR pf_IntFunc);
```

Purpose:

This function is used to install a user-defined interrupt handler function. It is possible to define interrupt handler functions for BC, RT, BM, Replay and BC-Branch related interrupts. If there is the need for an interrupt handler function that handles several interrupt types (BC, RT, BM, Replay, BC-Branch), it is necessary to call this function for all different interrupt types (BC, RT, BM, Replay or BC-Branch), each with the same given interrupt handler function "**pf_IntFunc**".

Note:
To install a HS-Interrupt-Handler when working with a 3910-Board, the "biu" parameter shall be set to "API_BIU_2" !!!

Note:
USB/ANET devices: This devices do not have a dedicated interrupt line. Event callbacks might be subject to increased latency, because of the extended processing from the additional USB-/Ethernet layers.

Note:
Under VxWorks the parameter uc_Type is not used, it is only possile to install one user-defined interrupt handler function

Input:

AiUInt8 biu

Biu Number or Stream Type

Board Opened With	Value	Constant	Description
ApiOpen()	**1**	**API_BIU_1**	Set interrupt routine for BIU1
	2	**API_BIU_2**	Set interrupt routine for BIU2
	3	**API_BIU_3**	Set interrupt routine for BIU3
	4	**API_BIU_4**	Set interrupt routine for BIU4
ApiOpenEx()	**1**	**API_INT_LS**	Set interrupt routine for LS
	2	**API_INT_HS**	Set interrupt routine for HS

AiUInt8 uc_Type

Interrupt Type Defines the type of interrupt which will be connected to the interrupt handler function given in "pf_IntFunc".

Value	Constant	Description
0	**API_INT_RT**	interrupt for RT related events
1	**API_INT_BC**	interrupt for BC related events
2	**API_INT_BM**	interrupt for BM related events
3	**API_INT_REPLAY**	interrupt for Replay related events
4	**API_INT_BC_BRANCH**	interrupt for BC Branch (e.g. Skip) related events

TY_INT_FUNC_PTR pf_IntFunc

Pointer to the interrupt handler function of the user application.

```
Typedef void(_cdecl * TY_INT_FUNC_PTR)(AiUInt32 ul_Module,
  AiUInt8 uc_Biu, AiUInt8 uc_Type,
  TY_API_INTR_LOGLIST_ENTRY * x_Info);
```

The interrupt function will receive the following parameters, which identify exactly the type of interrupt.

AiUInt32 ul_Module

Module Number of the AIM board that generated the interrupt.

Value	Constant	Description
0	**API_MODULE_1**	AIM board 1
1	**API_MODULE_2**	AIM board 2
2	**API_MODULE_3**	AIM board 3
3	**API_MODULE_4**	AIM board 4
4	**API_MODULE_5**	AIM board 5
5	**API_MODULE_6**	AIM board 6
6	**API_MODULE_7**	AIM board 7
7	**API_MODULE_8**	AIM board 8

AiUInt8 uc_Biu

BIU Number of the AIM board that generated the interrupt.

Value	Constant	Description
1	**API_BIU_1**	Bus Interface Unit 1 on the AIM board
2	**API_BIU_2**	Bus Interface Unit 2 on the AIM board
..	**..**	..
8	**API_BIU_8**	Bus Interface Unit 8 on the AIM board

AiUInt8 uc_Type

Interrupt type as defined in parameter “uc_Type” above. Contains the type of interrupt that the AIM board has generated.

Value	Constant	Description
0	**API_INT_RT**	interrupt for RT related events
1	**API_INT_BC**	interrupt for BC related events
2	**API_INT_BM**	interrupt for BM related events
3	**API_INT_REPLAY**	interrupt for Replay related events
4	**API_INT_BC_BRANCH**	interrupt for BC Branch (e.g. Skip) related events

TY_API_INTR_LOGLIST_ENTRY x_Info

Contains detailed information about the cause of the interrupt.

```
Typedef union {
  AiUInt32 ul_All;
```

```
  struct {
    AiUInt32 ul_Info: 24;
    AiUInt32 ul_IntType: 8;
  }
  t;

  struct {
    AiUInt32 ul_Info: 24;
    AiUInt8 uc_Biu1: 1;
    AiUInt8 uc_Biu2: 1;
    AiUInt8 uc_Dma: 1;
    AiUInt8 uc_Target: 1;
    AiUInt8 uc_Cmd: 1;
    AiUInt8 uc_Biu3: 1;
    AiUInt8 uc_Biu4: 1;
    AiUInt8 res: 1;
  }
  b;
}
TY_API_INTR_LOGLIST_LLC;

typedef union {
  AiUInt32 ul_All;
  struct {
    AiUInt32 ul_Index: 16;
    AiUInt32 uc_Res: 8;
    AiUInt32 uc_IntSrc: 8;
  }
  t;
}
TY_API_INTR_LOGLIST_LLD;

typedef struct {
  AiUInt32 ul_Lla;
  AiUInt32 ul_Llb;
  TY_API_INTR_LOGLIST_LLC x_Llc;
  TY_API_INTR_LOGLIST_LLD x_Lld;
  AiUInt32 ul_Lld;
}
TY_API_INTR_LOGLIST_ENTRY;
```

AiUInt32 ul_Lla

Interrupt Loglist Event, Entry Word 1

TYPE	Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
RT	INT_TYPE			UDF			AEI	IXI
BC	INT_TYPE			UDF	BCH	UXI	AEI	IXI
BM	INT_TYPE			UDF	METI	MTI	MBF	MSI
REPLAY	INT_TYPE			UDF				
BC_BRANCH	INT_TYPE			UDF				

TYPE	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
RT		PGI						
BC		PGI						
BM	MST							
REPLAY			RSO	RPI				
BC_BRANCH								

TYPE	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
RT								
BC								
BM								
REPLAY								
BC_BRANCH								

TYPE	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RT	RBI							
BC	RBI							
BM	TCBI							
REPLAY								
BC_BRANCH								

INT_TYPE

Value	Constant	Description
0	**API_INT_RT**	RT Interrupt Type
1	**API_INT_BC**	BC Interrupt Type
2	**API_INT_BM**	BM Interrupt Type
3	**API_INT_REPLAY**	Replay Interrupt Type
4	**API_INT_BC_BRANCH**	BC Branch Interrupt Type

UPF **Update Flag**
If set to 1, Interrupt Loglist Entry was updated

AEI **Any Error Interrupt**
If set to 1, an interrupt was asserted if any error was detected during transfer.

IXI **Index Interrupt**
If set to 1, an interrupt was asserted due to the current buffer index.

PGI **Programmed Transfer (BC) or SA (RT) Interrupt**
For BC: If set to 1, an interrupt was asserted due to programmed BC Iinterrupt. For RT: If set to 1, an interrupt was asserted due to programmed RTSA Iinterrupt

BCH **Bus Controller Halt**

If set to 1, an interrupt was asserted due to BC Halt (1553 protocol only)

UXI **Unexpected (Status Word) Response Interrupt**
If set to 1, an unexpected Status Word response interrupt was asserted (1553 protocol only)

METI **Monitor External Trigger Event during bus idle time**
If set to 1, an interrupt was asserted if no bus traffic takes place and an external trigger event was detected. This trigger type provides neither a trigger control block index (TCBI) nor a monitor buffer pointer (MBP) to the interrupt loglist entry (1553 protocol only)

MTI **Monitor Trigger Interrupt**
If set to 1, an interrupt was asserted if a trigger event becomes valid during the trigger control block processing (1553 protocol only)

MBF **Monitor Buffer Full Interrupt** (or Half Buffer Full Interrupt in Recording Mode)
If set to 1, an interrupt was asserted due to the Monitor Buffer Full event is standard or selective data capture mode and due to the Half Buffer Full event in recording mode

MSI **Monitor Start Interrupt**
If set to 1, an interrupt was asserted due to a Monitor start trigger event

MST **Monitor Stop Interrupt**
If set to 1, an interrupt was asserted due to a Monitor stop trigger event (1553 protocol only)

RSO **Replay Stop Interrupt**
If set to 1, an interrupt was asserted if the replay operation expired due to an expired count.

RPI **Replay Half Buffer Interrupt**
If set to 1, an interrupt was asserted if one half buffer was replayed and indicates a buffer reload request.

RBI **Relative Buffer Index**

Value	Description
0..255	Indicates the buffer index of the currently used buffer that is related to this interrupt

TCBI **Trigger Control Block Index**

Value	Description
0..255	Indicates the index of the trigger control block that created the interrupt. This field is only updated if the interrupt is asserted by the trigger control block processing (METI or MTI). This field is not used on MBF, MSI, or MST events.

Note:
On occurrence of the METI event, the TCBI field will not be initialized.

AiUInt32 ul_Llb

Interrupt Loglist Event, Entry Word 2

For RT (1553) Interrupt Type:	26-bit RT Subaddress / Mode code Descriptor Pointer
For RT (3910) Interrupt Type:	26-bit RT Message ID / Mode code Descriptor Pointer
For BC Interrupt Type:	26-bit BC Transfer Descriptor Pointer
For BM Interrupt Type:	26-bit Monitor Buffer Pointer
For Replay Interrupt Type:	26-bit Replay Buffer Pointer
For BC-Branch Interrupt Type:	26-bit BC Instruction List Pointer to Relative Branch

TY_API_INTR_LOGLIST_LLC x_Llc

AiUInt32 x_Llc.ul_All

<table>
<tr><th colspan="2">TYPE</th><th>Bit 31</th><th>Bit 30</th><th>Bit 29</th><th>Bit 28</th><th>Bit 27</th><th>Bit 26</th><th>Bit 25</th><th>Bit 24</th></tr>
<tr><td rowspan="2">RT</td><td>1553</td><td colspan="3" rowspan="6">reserved</td><td rowspan="6">CMD</td><td rowspan="6">ASP</td><td rowspan="6">DMA</td><td rowspan="6">BIU2</td><td rowspan="6">BIU1</td></tr>
<tr><td>3910</td></tr>
<tr><td colspan="2">BC</td></tr>
<tr><td colspan="2">BM</td></tr>
<tr><td colspan="2">REPLAY</td></tr>
<tr><td colspan="2">BC_BRANCH</td></tr>
</table>

<table>
<tr><th colspan="2">TYPE</th><th>Bit 23</th><th>Bit 22</th><th>Bit 21</th><th>Bit 20</th><th>Bit 19</th><th>Bit 18</th><th>Bit 17</th><th>Bit 16</th></tr>
<tr><td rowspan="2">RT</td><td>1553</td><td colspan="8">reserved</td></tr>
<tr><td>3910</td><td colspan="8">reserved</td></tr>
<tr><td colspan="2">BC</td><td colspan="8">TRANS_ID</td></tr>
<tr><td colspan="2">BM</td><td colspan="8">reserved</td></tr>
<tr><td colspan="2">REPLAY</td><td colspan="8">reserved</td></tr>
<tr><td colspan="2">BC_BRANCH</td><td colspan="8">reserved</td></tr>
</table>

<table>
<tr><th colspan="2">TYPE</th><th>Bit 15</th><th>Bit 14</th><th>Bit 13</th><th>Bit 12</th><th>Bit 11</th><th>Bit 10</th><th>Bit 9</th><th>Bit 8</th></tr>
<tr><td rowspan="2">RT</td><td>1553</td><td colspan="5">RT_ADDR</td><td>T/R</td><td colspan="2">RT_SUB</td></tr>
<tr><td>3910</td><td colspan="5">RT_ADDR</td><td>T/R</td><td colspan="2">reserved</td></tr>
<tr><td colspan="2">BC</td><td colspan="8">TRANS_ID</td></tr>
<tr><td colspan="2">BM</td><td colspan="8">reserved</td></tr>
<tr><td colspan="2">REPLAY</td><td colspan="8">reserved</td></tr>
<tr><td colspan="2">BC_BRANCH</td><td colspan="8">reserved</td></tr>
</table>

<table>
<tr><th colspan="2">TYPE</th><th>Bit 7</th><th>Bit 6</th><th>Bit 5</th><th>Bit 4</th><th>Bit 3</th><th>Bit 2</th><th>Bit 1</th><th>Bit 0</th></tr>
<tr><td rowspan="2">RT</td><td>1553</td><td colspan="3">RT_SUB</td><td colspan="5">MODE_CODE / WORD_COUNT</td></tr>
<tr><td>3910</td><td colspan="8">MID</td></tr>
<tr><td colspan="2">BC</td><td colspan="8">TRANS_ID</td></tr>
<tr><td colspan="2">BM</td><td colspan="8">reserved</td></tr>
<tr><td colspan="2">REPLAY</td><td colspan="8">reserved</td></tr>
<tr><td colspan="2">BC_BRANCH</td><td colspan="8">reserved</td></tr>
</table>

TRANS_ID

AIM Card	Value	Description
1553	**1..511**	Transfer Identifier
3910	**1..255**	Transfer Identifier

Note:
See Section 1.3.5 for the range allowed for this parameter.

RT_ADDR

Value	Description
0..31	RT Address

T/R

Value	Description
0	Receive
1	Transmit

RT_SUB

Value	Description
0..31	RT Subaddress

MODE_CODE / WORD_COUNT

Value	Description
0..31	Mode code for RT Subaddress 0, 31
0..31	Word count for RT Subaddress 1..30

MID

Value	Description
1..127	Message Identifier

BIU1

Interrupt has been generated from BIU1

BIU2

Interrupt has been generated from BIU2

DMA

Interrupt has been generated from onboard DMA controller

ASP

Interrupt has been generated from onboard ASP (Target-SW)

CMD

Reserved for internal use

TY_API_INTR_LOGLIST_LLC x_LId

AiUInt32 x_LId.ul_All

TYPE		Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
RT	1553	INT_SOURCE							
	3910								
BC									
BM									
REPLAY									
BC_BRANCH									

TYPE		Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
RT	1553	reserved							
	3910	reserved							
BC		reserved							
BM		reserved							
REPLAY		reserved							
BC_BRANCH		reserved							

TYPE		Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
RT	1553	BUF_INDEX							
	3910	BUF_INDEX							
BC		BUF_INDEX							
BM		reserved							
REPLAY		reserved							
BC_BRANCH		reserved							

TYPE		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RT	1553	BUF_INDEX							
	3910	BUF_INDEX							
BC		BUF_INDEX							
BM		reserved							
REPLAY		reserved							
BC_BRANCH		reserved							

INT_SOURCE

Value	Description
0	Interrupt has been generated from BIU1
1	Interrupt has been generated from BIU2
2	Interrupt has been generated from onboard DMA controller
3	Interrupt has been generated from onboard ASP (Target-SW)
4	Reserved for internal use
5	Interrupt has been generated from BIU3
6	Interrupt has been generated from BIU4
7	Interrupt has been generated from BIU5
8	Interrupt has been generated from BIU6
9	Interrupt has been generated from BIU7
10	Interrupt has been generated from BIU8

BUF_INDEX

Data Buffer Index

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

2.1.16 ApiOpen (obsolete)

Prototype:

```
AyI/AyX: AiReturn ApiOpen(AiUInt8 uc_Module,
                          AiChar *ac_SrvName,
                          AiUInt32 *pul_ModuleHandle);
AVI/AVX: short ApiOpen(UINT8 bModule);
```

Purpose:

Note:
This command is obsolete and only maintained for compatibility reasons! We recommend to use ApiOpenEx() instead!

This function initializes the Application Interface for the specified module and must be called first, before any other API S/W Library function is used for the specified module. This function establishes connectivity between the Application Interface and the AIM board by calling operating system routines to open the AIM board and initialize a shared memory area for host-to-target communication.

Note:
To open the board communication you need to call either ApiOpen() or ApiOpenEx(), but not both!

Input:

`AiUInt8 uc_Module`

Board Module Number to access

Value	Constant	Description
0	**API_MODULE_1**	Module 1
1	**API_MODULE_2**	Module 2
2	**API_MODULE_3**	Module 3
3	**API_MODULE_4**	Module 4
...		
31	**API_MODULE_32**	Module 32

`AiChar *ac_SrvName`

Name of the PC, where the ANS Server is running.

Value	Description
“local”	Local use of the board
<SrvName>	Name of the PC, where the ANS (AIM Network Server) is running (e.g. “\\SW-PC-06” or “192.168.0.119”)

Output:

`AiUInt32 *pul_ModuleHandle`

API Board Module Access Handle

> **Note:**
> pul_ModuleHandle must be used as input for all other functions.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

2.1.17 ApiOpenEx

Prototype:

```
AiReturn ApiOpenEx(TY_API_OPEN *px_ApiOpen,
                   AiUInt32 *pul_ModuleHandle);
```

Purpose:

This function initializes the Application Interface for the specified module and must be called first, before any other API S/W Library function is used for the specified module. This function establishes connectivity between the Application Interface and the AIM board by calling operating system routines to open the AIM board and initialize a shared memory area for host-to-target communication.

> **Note:**
> When using this function to open the board communication, the 'biu' parameter described in all other functions is not needed and its value will be ignored! By using this function this information is already coded within the returned 'pul_ModuleHandle'.

> **Note:**
> To open the board communication you need to call either ApiOpen() or ApiOpenEx(), but not both!

Input:

`TY_API_OPEN *px_ApiOpen`

Synchronization Counter get structure

```
typedef struct ty_api_open {
  AiUInt32 ul_Module;
  AiUInt32 ul_Stream;
  AiChar ac_SrvName[28];
}
TY_API_OPEN
```

`AiUInt32 ul_Module`

Board Module Number to access

Value	Constant	Description
0	**API_MODULE_1**	Module 1
1	**API_MODULE_2**	Module 2
2	**API_MODULE_3**	Module 3
3	**API_MODULE_4**	Module 4
...		
31	**API_MODULE_32**	Module 32

`AiUInt32 ul_Stream`

Stream Number of Board to be opened This number reflects the MILbus or STANAG3910 channel to be opened on the board given by parameter 'ul_Module'. On 1553 boards the stream number corresponds to the former ‘biu’ parameter.

Value	Constant	Description
1	**API_STREAM_1**	Stream 1
2	**API_STREAM_2**	Stream 2
3	**API_STREAM_3**	Stream 3
4	**API_STREAM_4**	Stream 4
...		
8	**API_STREAM_8**	Stream 8

`AiChar ac_SrvName[28]`

Name of the PC, where the ANS Server is running

Value	Description
“local”	Local use of the board
<SrvName>	Name of the PC, where the ANS (AIM Network Server) is running (e.g. “\\SW-PC-06” or “192.168.0.119”)

Output:

`AiUInt32 *pul_ModuleHandle`

API Board Module Access Handle

> **Note:**
> pul_ModuleHandle must be used as input for all other functions.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

2.1.18 ApiSetDeviceConfig

Prototype:

```
AiReturn ApiSetDeviceConfig(AiUInt32 ul_Module,
                            TY_API_DEVICE_CONFIG* px_Config);
```

Purpose:

Set the device configuration from px_Config. The configuration should be read with ApiGetDeviceConfig. After the changes have been done the configuration can be written back with this function.

Input:

TY_API_DEVICE_CONFIG * px_Config

The device config struct.

```
Typedef struct ty_api_device_config {
  AiUInt8 uc_DmaEnabled;
  AiUInt8 uc_DataQueueMemoryType;
  AiUInt8 uc_DataQueueMode;
  AiUInt8 uc_ReservedB4;
  AiUInt16 uw_ReservedW1;
  AiUInt16 uw_ReservedW2;
  AiUInt32 ul_DmaMinimumSize;
  AiUInt32 ul_IntRequestCount;
  AiUInt32 ul_DriverFlags;
  AiUInt32 ul_ReservedLW4;
  AiUInt32 ul_ReservedLW5;
  AiUInt32 ul_ReservedLW6;
  AiUInt32 ul_ReservedLW7;
  AiUInt32 ul_ReservedLW8;
}
TY_API_DEVICE_CONFIG;
```

AiUInt8 uc_DmaEnabled

Enable or disable DMA read transfers.
This value is only applicable for system driver version 12 or higher.

Value	Description
0	DMA disabled
1	DMA enabled

AiUInt32 ul_DmaMinimumSize

The memory size limit for DMA transfers. If a requested memory block is bigger or equal to this value a DMA transfer will be issued. This value is only applicable for system driver version 12 or higher.

`AiUInt8 uc_DataQueueMemoryType`

The memory type of the memory where the data queue headers are located.

Value	Define	Description
1	**API_MEMTYPE_SHARED**	Shared memory
2	**API_MEMTYPE_LOCAL**	Local memory (AyI)

`AiUInt8 uc_DataQueueMode`

The data queue mode determines how the data is copied to the host.

Value	Define	Description
0	-	data will copied by the ASP
1	-	data will be copied direct to the host

`AiUInt32 ul_IntRequestCount`

The number of interrupt requests sent to the driver.

`AiUInt32 ul_DriverFlags`

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
						INT_ENA	

INT_ENA (write only)

Value	Description
0	Do not change setting (Interrupts are enabled by default)
1	Enable interrupts
2	Disable interrupts

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

2.1.19 ApiSetDllDbgLevel

Prototype:

```
AiReturn ApiSetDllDbgLevel(AiUInt32 ul_DllDbgLevel);
```

Purpose:

This function sets the current debug output level of the Application Interface.

Note:
This function is not thread safe. See section Symmetric Multiprocessing in the Programmers Guide for details.

Input:

`AiUInt32 ul_DllDbgLevel`

Debug Level

Note:
Under Windows the debug output is done with the Windows API call of OutputDebugString. The output can be achieved e.g. via the utility "DbgView" (see http://www.sysinternals.com for download).

Value	Constant	Description
0x00000000	**DBG_DISABLED**	Force no debug output
0x00000001	**DBG_INT**	Force interrupt related debug output
0x00000002	**DBG_INIT**	Force initialization related debug output
0x00000004	**DBG_OPEN**	Force module open related debug output
0x00000008	**DBG_CLOSE**	Force module close related debug output
0x00000010	**DBG_IO**	Force module I/O related debug output
0x00000020	**DBG_READREC**	Force recording related debug output
0x00000040	**DBG_WRITEREP**	Force replay related debug output
0x00000080	**DBG_MUTEX**	Force mutex related debug output
0x00000400	**DBG_PARAMCHK**	Force parameter range check
0x04000000	**DBG_TRACE_PBAPRO**	For internal use only
0x08000000	**DBG_TRACE**	Force a function call trace log into file "aim_mil.log"
0x10000000	**DBG_INFO**	Force informational debug output
0x20000000	**DBG_ERROR**	Force general error related debug output (e.g. range check errors)
0x40000000	**DBG_ERRORMSG**	Force error message box, if I/O to the board fails with error or range check fails
0xFFFFFFFF	**DBG_ALL**	Force all available debug output

Note:
The default debug level after program start is set to 0x30000000 (DBG_ERROR | DBG_INFO | DBG_PARAMCHK)!

Output:

None

Return Value:

Returns always 0.

2.2 VME Intialization Functions

These functions are for the use of VME environments only. They are not available for Windows, Linux or similar environments.

2.2.1 AiVmeExamineSlot

Prototype:

```
AiReturn AiVmeExamineSlot(TY_VME_EXAMINE_SLOT *in,
                          TY_PCI_INFO *px_PCI_Info1,
                          TY_PCI_INFO *px_PCI_Info2);
```

Purpose:

Runs the PCI config cycle on the AIM board on the A16 address specified in in->ul_A16Addr, regardless of its ransfer and writes its data into the output parameters px_PCI_Info1 and px_PCI_Info2. Using these output parameters an AVI1553, AVX1553 board or an AMC1553 board on an AVC-2 carrier can be initialized using the function AiVme1553MapModule().

Note:
this function is not supported for all boards. Please see Table B.3 – Function Support By Boards With ASP for details

Input:

TY_VME_EXAMINE_SLOT *in

```
typedef struct ty_vme_examine_slot {
  AiUInt32 ul_A16Addr;
  AiUInt32 ul_Force;
  /* only needed for PMC on AVC */
  AiUInt32 ul_TempA32Addr;
  AiUInt32 ul_TempA32UserAccess;
}
TY_VME_EXAMINE_SLOT;
```

AiUInt32 ul_A16Addr

Is a User defined address in the VME A16 address space, where the board is mapped with the boards DIP-switch. This address is the A16 address where the CPU accesses the A16 space in his local memory and can be different to the real physical A16 address (Typically there is a qualifier in the upper half of the long word) . The size of the address space must be 4 kByte.

AiUInt32 ul_Force

Force overwrite of already initialized boards. This forces a PCI Config Cycle on AVI/AVC boards. For most cases this value should be set to zero.

AiUInt32 ul_TempA32Addr

VME A32 bus bus address the configuration window should temporarily be mapped to.

Only used for PMC on AVC

AiUInt32 ul_TempA32UserAccess

The ‘virtual’ address the PMC board is temporarily be mapped to

Only used for PMC on AVC

Output:

TY_PCI_INFO *px_PCI_Info1

Used as input for functions AiVme1553MapModule(), AiVme1553UnmapModule() or Ai1553CheckModule(). For two PMC boards on one AVC-2 carrier, this contains the configuration data of the first PMC board.

```
Typedef struct {
  TY_PCI_CONFIGSPACE_HEADER x_PCIConfHd;
  TY_PCI_BAR_INFO x_PCIBarInfo[6];
  AiUInt32 ul_PCITotalMemorySize;
  AiUInt32 ul_PCIStartAddress;
  AiUInt32 ul_A16Address;
  AiUInt8 uc_VmeHandleCount;
  AiUInt busNo;
  AiUInt8 deviceNo;
  AiUInt8 funcNo;
}
TY_PCI_INFO;

TY_PCI_CONFIGSPACE_HEADER x_PCIConfHd
typedef struct {
  AiUInt16 uw_DeviceID; // Device ID
  AiUInt16 uw_VendorID; // Vendor ID
  AiUInt16 uw_Status; // PCI status register
  AiUInt16 uw_Command; // PCI command register
  AiUInt32 ul_ClassCode_RevID; // PCI Class code / Revision ID
  AiUInt8 uc_Bist; // PCI BIST register
  AiUInt8 uc_HeaderType; // PCI header type
  AiUInt8 uc_LatencyTimer; // PCI latency timer
  AiUInt8 uc_CacheLineSize; // PCI cache line size
  AiUInt32 ul_BAR[6]; // Base address registers
```

```
  AiUInt32 ul_CardbusCisPtr; // Card Bus CIS Ptr
  AiUInt16 uw_SubsystemID; // Subsystem ID
  AiUInt16 uw_SubsystemVendID; // Subsystem Vendor ID
  AiUInt32 ul_ExpRomBaseAddr; // Expansion ROM Base Address
  AiUInt16 uw_Reserved1;
  AiUInt8 uc_Reserved2;
  AiUInt8 uc_CapabilitiesPtr; // Capabilities Ptr
  AiUInt32 ul_Reserved3;
  AiUInt8 uc_MaxLat; // Max latency
  AiUInt8 uc_MinGnt; // Min grant
  AiUInt8 uc_intr_pin; // Interrupt pin
  AiUInt8 uc_intr_line; // Interrupt Line
}
TY_PCI_CONFIGSPACE_HEADER;

typedef struct {
  AiUInt32 ul_size; // Requested size of the BAR reg
  AiUInt32 ul_BarBaseAddress; // PCI Base Address of BAR reg
}
TY_PCI_BAR_INFO;
```

TY_PCI_INFO *px_PCI_Info2

As px_PCI_Info1, but it contains the configuration data of the second PMC board of an AVC-2 carrier. For all other boards, this value may be ignored.

Return Value:

0 in case of success.

2.2.2 Ai1553CheckModule

Prototype:

```
AiReturn Ai1553CheckModule(TY_VME_MAP_MODULE_IN *in);
```

Purpose:

This commands checks if the input parameter refers to a known board.

> **Note:**
> this function is not supported for all boards. Please see Table B.3 – Function Support By Boards With ASP for details

Input:

TY_VME_MAP_MODULE *in

Parameters used to map this board to the VME bus in function AiVme1553MapModule().

Output:

None

Return Value:

Constant	Description
MODULE_TYPE_OTHER_AVI	Unknown board
MODULE_TYPE_PMC_1553	Board identified as AMC1553-1/-2/-4
MODULE_TYPE_PMC_1553_ASP	Board identified as AMC1553-T
MODULE_TYPE_AVX_1553	Board identified as AVX1553-1/-2/-4/-8
MODULE_TYPE_APM_1553	Board identified as APM1553
MODULE_TYPE_AVX_3910	Board identified as AVX3910-1/-2
MODULE_TYPE_AVX_3910_1553	Board identified as AVX-EFA-1/-2/-4
MODULE_TYPE_AVX_EFEX	Board identified as AVX3910-1/-2, configured to EFEX
MODULE_TYPE_AVX_EFEX_1553	Board identified as AVX-EFA-1/-2/-4 configured to EFEX

2.2.3 AiPciScan

Prototype:

```
AiReturn AiPciScan( void );
```

Purpose:

This commands scan the local PCI bus for known devices and internally stores the PCI headers of all boards found. It allows to use the AiPciGetHeader() command to get the PCI header of any board found.

Note:
this function is not supported for all boards. Please see Table B.3 – Function Support By Boards With ASP for details

Input:

None

Output:

None

Return Value:

The number of boards found

2.2.4 AiPciGetHeader

Prototype:

```
TY_PCI_INFO* AiPciGetHeader(AiUInt32 ulModuleIndex);
```

Purpose:

This commands returns the PCI header of a board, which can be used to call AiVme1553MapModule(). Before calling this command, AiPciScan() has to be called first.

Note:
this function is not supported for all boards. Please see Table B.3 – Function Support By Boards With ASP for details

Input:

`AiUInt32 ulModuleIndex`

This is an index to the PCI module, from which the PCI header shall be returned. An index of zero returns the first board that was found.

Output:

None

Return Value:

The PCI header of the board identified by the ulModuleIndex.

```
Typedef struct {
  TY_PCI_CONFIGSPACE_HEADER x_PCIConfHd;
  TY_PCI_BAR_INFO x_PCIBarInfo[6];
  AiUInt32 ul_PCITotalMemorySize;
  AiUInt32 ul_PCIStartAddress;
  AiUInt32 ul_A16Address;
  AiUInt8 uc_VmeHandleCount;
  AiUInt busNo;
  AiUInt8 deviceNo;
  AiUInt8 funcNo;
}
TY_PCI_INFO;
```

2.2.5 AiVmeInitGenericInterrupt

Prototype:

```
void AiVmeInitGenericInterrupt(TY_PCI_INFO *px_PCI_Info,
                               TY_INIT_VMEGENERIC_INT *in);
```

Driver Command:

None

Purpose:

This function applies interrupt specific parameters to the board, specified in px_PCI_Info. This command can be called after AiVmeExamineSlot() for a board on the VME bus or after AiPciGetHeader() for a board on a local PCI bus.

Note: this function is not supported for all boards. Please see Table B.3 – Function Support By Boards With ASP for details

Input:

TY_PCI_INFO *px_PCI_Info

Pointer to the PCI info element for this board. It determines for which board the additional settings are. To get this parameter please use the output of AiVmeExamineSlot() or AiPciGetHeader()

TY_INIT_VMEGENERIC_INT *in

```
typedef struct ty_init_vmegeneric_int {
  AiUInt32 ul_IrLevel;
  AiUInt32 ul_IrVector;
  INTERRUPT_SET_FUNC * intSetFunction;
  INTERRUPT_SET_FUNC * intDeinstallFunction;
}
TY_INIT_VMEGENERIC_INT;
```

AiUInt32 ul_IrLevel

Using this parameter the interrupt level can be applied to the board.

Note: if two PMC boards are driven on the same AVC-2 carrier, both must have the same interrupt level and vector.

AiUInt32 ul_IrVector

Using this parameter the interrupt vector can be applied to the board.

> **Note:**
> if two PMC boards are driven on the same AVC-2 carrier, both must have the same interrupt level and vector.

INTERRUPT_SET_FUNC *intSetFunction

According to specifics of your VME system it can be very different to set and enable the interrupt to the specific level and vector. So this parameter is a function pointer to a routine, which can set the interrupt vector to the interrupt vector table in the host VME system for selected interrupt level number. This function must be from the type INTERRUPT_SET_FUNC which is defined in 'API1553.h' as follows:

```
typedef AiUint8 INTERRUPT_SET_FUNC(AiUint8 vector,
  AiUint8 level, VOID_FUNC * intFuntion);
```

This is a callback function which will be called to make the interrupt settings. The parameter '*intFuntion' of type VOID_FUNC is the function pointer which should be called in case of interrupt. This function is a driver internal interrupt function which handles the ransfer interrupt on the AIM board and distributes to the user interrupt functions.

The type VOID_FUNC is defined as follows:

```
typedef void VOID_FUNC(void);
```

INTERRUPT_SET_FUNC *intDeinstallFunction

This is a function pointer to a user function to do deinstall an interrupt from the system. This function must be from the type INTERRUPT_FUNC which is defined as follows: :

```
typedef AiUint8 INTERRUPT_SET_FUNC(AiUint8 vector,
  AiUint8 level, VOID_FUNC * intFuntion);
```

Output:

None

Return Value:

0 in case of success

2.2.6 AiVme1553MapModule

Prototype:

`AiReturn AiVme1553MapModule(TY_VME_MAP_MODULE_IN *in)`

Purpose:

This function initializes the Application Interface for the specified module and returns a module handle that has to be used for most functions. This function establishes connectivity between the Application Interface and the AIM memory area for host-to-target communication.

Note: most commands require also ApiOpen() to be called

Note: this function is not supported for all boards. Please see Table B.3 – Function Support By Boards With ASP for details

Input:

`TY_VME_MAP_MODULE_IN *in`

```
typedef struct ty_vme_map_module_in {
  AiUInt32 ul_A32Addr;
  AiUInt32 ul_A32UserAccess;
  AiUInt32 ul_Force;
  AiUInt32 ul_cPCI;
  TY_PCI_INFO * px_PCI_Info;
}
TY_VME_MAP_MODULE_IN;
```

`AiUInt32 A32Address`

Is a User defined address in the VME A32 address space where the user wants to see the memory of the VME Carrier's AMC1553 Module or the AVI1553 module in the VME address space (real, physical A32 address). The memory size for the module in the A32 space is dependant on the amount of memory on the module. The real memory depends on the settings of the PCI-BAR register of the AIM Module. For all requested memory of the AIM Module one image is mapped on the VME-bus.

To access this mapped VME-memory for each BAR register memory request one VME base pointer is returned.

Note: this ransfe has to be 16MB ransfe.

`AiUInt32 A32UserAccess`

According to your VME-CPU system or operating system, the access address of your CPU to the VME A32 range can be different to the real physical A32 address. This parameter defines the CPU VME A32 access address.

> **Note:**
> this parameter may be set to zero, if the command 'sysBusToLocalAdrs' works in the VxWorks BSP.

`AiUInt32 ul_Force`

Force mapping process, even if already mapped. For most cases this value should be set to zero.

`AiUInt32 ul_cPCI`

This parameter has to be set to '1' if the boards are located on a cPCI bus. AMCX1553 on ACC carrier. For AVI1553 boards, for AMCX1553 on AVC-2 carrier or for AMCX1553 located in the local PMC slots of a VME CPU, this value shall be set to zero.

`TY_PCI_INFO *px_PCI_Info`

Pointer to the PCI info element for this board. It determines for which board the additional settings are. To get this parameter please use the output of AiVmeExamineSlot() or AiPciGetHeader()

Output:

None

Return Value:

0..32 Module ID. To be used for further A1553 commands

Value	Description
0..32	Module Handle, to be used for further commands
0xff	General IO error

2.2.7 AiVme1553UnmapModule

Prototype:

```
AiReturn AiVme1553UnmapModule(TY_VME_MAP_MODULE_IN *in)
```

Purpose:

This function undoes the mapping of a board to the VME bus or local PCI bus.

Note: this function is not supported for all boards. Please see Table B.3 – Function Support By Boards With ASP for details

Input:

`TY_VME_MAP_MODULE_IN *in`

This is the structure that was used to map the board to the bus.

Output:

None

Return Value:

0 in case of success

3 SYSTEM FUNCTIONS

Chapter 3 defines the System function calls of the API S/W Library. The System functions provide general device control, response timeout setup, IRIG setup, board configuration status and control of the generation of dynamic data words/datasets. Table 3.1 defines the list and definition of System function calls within this group. The function calls in this table are listed in a functional order, however, the detailed descriptions of the System function calls in the following sections are in alphabetical order.

Table 3.1: System Function Descriptions

Function	Description
ApiCmdIni	Initializes AIM board and returns board configuration
ApiCmdReset	Resets the AIM board and ASP driver software data to initial state
ApiCmdBite	Performs a selftest on the AIM board
ApiCmdDefRespTout	Defines the response timeout value (default is 14 μsec)
ApiReadAllVersions	Reads all versions applicable for this board
ApiCmdExecSys	Executes a system-related function on the AIM board
ApiCmdSetIrigTime	Sets the time of the on-board IRIG timecode encoder
ApiCmdGetIrigTime	Reads the time on the on-board IRIG timecode encoder
ApiCmdDefMilbusProtocol	Defines MILbus Protocol type (A or B) for individual or single RTs
ApiWriteRepData	Writes and copies replay data
ApiCmdSystagDef	Defines the generation of dynamic data words or datasets in BC and RT mode
ApiCmdSystagCon	Suspends or resumes the generation of dynamic data words or datasets in BC and RT mode.
ApiCmdSysTriggerEdgeInputSet	Configure edge sensitivity of the input triggers
ApiCmdSysTriggerEdgeInputGet	Get edge sensitivity of the input triggers
ApiCmdTrackDef	Defines an area (track) in the 1553 Data Buffer to be copied and stored in a Shared memory area and multiplexed with tracks from subsequent buffers transmitted/received with the same XID/RT SA
ApiCmdTrackRead	Reads the multiplexed 1553 message data defined as a track with ApiCmdTrackDef
ApiCmdSysSetMemPartition	Configures the Global RAM of the board
ApiCmdSysFree	Free a allocate a target software memory block
ApiCmdSysGetBoardInfo	Read board information from the target software
ApiCmdSysGetMemPartition	Reads the configuration of the Global RAM of the board
ApiCmdSysMalloc	Allocate a target software memory block
ApiCmdReadDiscretes	Reads from the onboard discrete register
ApiCmdReadDiscretesConfig	Reads back the discrete configuration set up by ApiCmdInitDiscretes
ApiCmdWriteDiscretes	Writes to the onboard discrete register
ApiCmdInitDiscretes	Initializes the onboard discrete behaviour
ApiCmdReadDiscretesInfo	Reads the configuration of the discrete channels
ApiCmdSysPXICon	Combine PXI specific trigger lines with the trigger lines of AIM boards
ApiCmdSysPXIGeographicalAddressGet	Get the geographical address of the PXI slot

3.1 Low Speed Functions

3.1.1 ApiCmdBite

Prototype:

```
AiReturn ApiCmdBite(AiUInt32 ul_ModuleHandle, AiUInt8 biu, AiUInt8 sc,
                    AiUInt8 bite_status[2]);
```

Purpose:

This function is used to perform a selftest on the AIM board. The selftest result is reported as part of the returning information. After a Selftest Command the currently active board setups are lost, due to the performed RAM Test.

Note:
When calling this function, a board reset is executed automatically after the Selftest (at the end of this command) using the ApiCmdReset function.

Note 2:
When operating with a 3910 board, the "biu" parameter setting "API_BIU_2" is not allowed!!! The execution of this command using "biu" parameter set to "API_BIU_1" will execute all 3910 related high speed tests.

Input:

AiUInt8 sc

Selftest control

Value	Constant	Description
0	**API_BITE_ALL**	Execute all tests
1	**API_BITE_BOARD_ENABLE**	Execute Board Enable test
2	**API_BITE_INTERN_SELFTEST**	Execute Internal Selftest
3	**API_BITE_GLOBAL_RAM_BIU**	Execute Global RAM test for BIU Control Block area
4	**API_BITE_GLOBAL_RAM_BCRT**	Execute Global RAM test for BIU BC/RT Descriptor area
5	**API_BITE_GLOBAL_RAM_BM**	Execute Global RAM test for BIU BM area
6	**API_BITE_SHARED_RAM**	Execute Shared RAM test (beginning at 0x100000 offset)
7	**API_BITE_SHARED_RAM_CMD**	Execute Shared RAM test for CMD area
8	**API_BITE_SHARED_RAM_ACK**	Execute Shared RAM test for ACK area
9	**API_BITE_TRANSFER**	Execute Transfer Test
10	**API_BITE_INTERRUPT**	Execute Interrupt test
11	**API_BITE_DA_CONV_BUSA**	Execute D/A Converter test, Bus A
12	**API_BITE_DA_CONV_BUSB**	Execute D/A Converter test, Bus B
13	**API_BITE_TIMING**	Execute Timing Test

Output:

`AiUInt8 bite_status[]`

Selftest results The two returning selftest status bytes are comprising the following information:

Description	[0]	[1]	Constant
Selftest Passed	**0**	**0**	**API_BITE_PASSED**
Board Enable Test error	**1**		**API_BITE_ERR_BOARD_ENABLE**
Internal Selftest error	**2**		**API_BITE_ERR_INTERN_SELFTEST**
Memory Tests			
- Addressing Test		**16**	**API_BITE_ERR_INT_ADDR**
- Pattern Test 0x5555		**17**	**API_BITE_ERR_INT_PAT_55**
- Pattern Test 0xAAAA		**18**	**API_BITE_ERR_INT_PAT_AA**
- Pattern Test 0xFFFF		**19**	**API_BITE_ERR_INT_PAT_FF**
- Pattern Test 0x0000		**20**	**API_BITE_ERR_INT_PAT_00**
- Walking Zero Test		**21**	**API_BITE_ERR_INT_WALK_0**
- Walking One Test		**22**	**API_BITE_ERR_INT_WALK_1**
PBI Test			
- Wrong PBI ID Code		**23**	**API_BITE_ERR_INT_WRONG_PBI**
MILBus Test		**24**	**API_BITE_ERR_INT_MILBUS**
Global RAM Test error	**3**		**API_BITE_ERR_GLOBAL_RAM_BIU**
BIU Control Block area			
- Data Pattern Test		**1**	**API_BITE_ERR_DATA_PATTERN**
- Walking Bit Test		**2**	**API_BITE_ERR_WALKING_BIT**
- Addressing Test		**3**	**API_BITE_ERR_ADDRESSING**
- Bus Test		**4**	**API_BITE_ERR_BUS**
Global RAM Test error	**4**		**API_BITE_ERR_GLOBAL_RAM_BCRT**
BIU BC/RT Descriptor area			
- Data Pattern Test		**1**	**API_BITE_ERR_DATA_PATTERN**
- Walking Bit Test		**2**	**API_BITE_ERR_WALKING_BIT**
- Addressing Test		**3**	**API_BITE_ERR_ADDRESSING**
- Bus Test		**4**	**API_BITE_ERR_BUS**
Global RAM Test error	**5**		**API_BITE_ERR_GLOBAL_RAM_BM**
BIU BM area			
- Data Pattern Test		**1**	**API_BITE_ERR_DATA_PATTERN**
- Walking Bit Test		**2**	**API_BITE_ERR_WALKING_BIT**
- Addressing Test		**3**	**API_BITE_ERR_ADDRESSING**
- Bus Test		**4**	**API_BITE_ERR_BUS**
Shared RAM Test error	**6**		**API_BITE_ERR_SHARED_RAM**
(at 0x100000 offset)			
Paging Test		**1**	**API_BITE_ERR_PAGING**

Description	[0]	[1]	Constant
Shared RAM Test error	**7**		**API_BITE_ERR_SHARED_RAM_CMD**
(CMD area)			
- Data Pattern Test		**1**	**API_BITE_ERR_DATA_PATTERN**
- Walking Bit Test		**2**	**API_BITE_ERR_WALKING_BIT**
- Addressing Test		**3**	**API_BITE_ERR_ADDRESSING**
- Bus Test		**4**	**API_BITE_ERR_BUS**
Shared RAM Test error	**8**		**API_BITE_ERR_SHARED_RAM_ACK**
(ACK area)			
- Data Pattern Test		**1**	**API_BITE_ERR_DATA_PATTERN**
- Walking Bit Test		**2**	**API_BITE_ERR_WALKING_BIT**
- Addressing Test		**3**	**API_BITE_ERR_ADDRESSING**
- Bus Test		**4**	**API_BITE_ERR_BUS**
Transfer Test error	**9**		**API_BITE_ERR_TRANSFER**
BC Broadcast			
- electrical wrap, Bus A		**1**	**API_BITE_ERR_BC_BROAD_ELECT_A**
- electrical wrap, Bus B		**2**	**API_BITE_ERR_BC_BROAD_ELECT_B**
- electrical wrap, Bus A,BM		**3**	**API_BITE_ERR_BC_BROAD_ELECT_A_BM**
- electrical wrap, Bus B,BM		**4**	**API_BITE_ERR_BC_BROAD_ELECT_B_BM**
- isol.coupling, Bus A,BM		**5**	**API_BITE_ERR_BC_BROAD_ISOL_A_BM**
- isol.coupling, Bus B,BM		**6**	**API_BITE_ERR_BC_BROAD_ISOL_B_BM**
BC-RT, RT-BC			
- isol.coupling, Bus A+B, BM		**7**	**API_BITE_ERR_BCRT_ISOL_AB_BM**
RT-RT			
- isol.coupling, Bus A,BM		**8**	**API_BITE_ERR_RTRT_ISOL_A_BM**
- isol.coupling, Bus B,BM		**9**	**API_BITE_ERR_RTRT_ISOL_B_BM**
Mode codes			
- isol.coupling, Bus A,BM		**10**	**API_BITE_ERR_MODE_ISOL_A_BM**
- isol.coupling, Bus B,BM		**11**	**API_BITE_ERR_MODE_ISOL_B_BM**
BC Timeout			
- isol.coupling, Bus A,BM		**12**	**API_BITE_ERR_BC_TIMEOUT_ISOL_A_BM**
- isol.coupling, Bus B,BM		**13**	**API_BITE_ERR_BC_TIMEOUT_ISOL_B_BM**
Interrupt Test error	**10**		**API_BITE_ERR_INTERRUPT**
D/A Converter Test, Bus A	**11**		**API_BITE_ERR_DA_CONV_BUSA**
D/A Converter Test, Bus B	**12**		**API_BITE_ERR_DA_CONV_BUSB**
Timing Test error	**13**		**API_BITE_ERR_TIMING**
LS BC Minor Framing			
- Time 50ms		**1**	**API_BITE_ERR_TIMING_MFRM_50**
- Time 10ms		**2**	**API_BITE_ERR_TIMING_MFRM_10**
- Time 1ms		**3**	**API_BITE_ERR_TIMING_MFRM_1**
LS RT Response Time		**10**	**API_BITE_ERR_TIMING_RT_RESPONSE**
LS Intermessage Gap		**20**	**API_BITE_ERR_TIMING_IMG**

Description	[0]	[1]	Constant
HS Internal Selftest error	**101**		**API_BITE_ERR_HS_INTERN_SELFTEST**
Memory Tests			
- Addressing Test		**16**	**API_BITE_ERR_HS_INT_ADDR**
- Pattern Test 0x5555		**17**	**API_BITE_ERR_HS_INT_PAT_55**
- Pattern Test 0xAAAA		**18**	**API_BITE_ERR_HS_INT_PAT_AA**
- Pattern Test 0xFFFF		**19**	**API_BITE_ERR_HS_INT_PAT_FF**
- Pattern Test 0x0000		**20**	**API_BITE_ERR_HS_INT_PAT_00**
- Walking Zero Test		**21**	**API_BITE_ERR_HS_INT_WALK_0**
- Walking One Test		**22**	**API_BITE_ERR_HS_INT_WALK_1**
PBI Test			
- Wrong PBI ID Code		**23**	**API_BITE_ERR_HS_INT_WRONG_PBI**
Loop Tests			
- Simulator Transfer Count mismatch		**24**	**API_BITE_ERR_HS_INT_SIM_TX_CNT**
- Simulator Error Count mismatch		**25**	**API_BITE_ERR_HS_INT_SIM_ERR_CNT**
- Monitor Activity Transfer Count mismatch		**26**	**API_BITE_ERR_HS_INT_MON_TX_CNT**
- Monitor Activity Transfer Error mismatch		**27**	**API_BITE_ERR_HS_INT_MON_ERR_CNT**
- Simulator BC to RT data mismatch		**28**	**API_BITE_ERR_HS_INT_SIM_BCRT**
- Simulator RT to BC data mismatch		**29**	**API_BITE_ERR_HS_INT_SIM_RTBC**
- Monitor data mismatch		**30**	**API_BITE_ERR_HS_INT_MON_DATA**
- Monitor Transfer Count mismatch during Replay Test		**31**	**API_BITE_ERR_HS_INT_MON_TX_CNT_REP**
- Monitor Error Count mismatch during Replay Test		**32**	**API_BITE_ERR_HS_INT_MON_ERR_CNT_REP**
- Monitor data mismatch during Replay Test		**33**	**API_BITE_ERR_HS_INT_MON_DATA_REP**
Timetag Counter Tests			
- Time Tag too slow		**40**	
- Time Tag too fast		**41**	**API_BITE_ERR_HS_INT_TT_SLOW**
- Time Tag Timeout		**42**	**API_BITE_ERR_HS_INT_TT_FAST**
(not counting or very slow)			**API_BITE_ERR_HS_INT_TT_TIMEOUT**

Description	[0]	[1]	Constant
HS Transfer Test error	**102**		**API_BITE_ERR_HS_TRANSFER**
BC Broadcast			
- isol.coupling, LS Channel A, HS Channel A		**1**	**API_BITE_ERR_HS_BC_BROAD_ISOL_AA**
- isol.coupling, LS Channel B, HS Channel A		**2**	**API_BITE_ERR_HS_BC_BROAD_ISOL_BA**
- isol.coupling, LS Channel A, HS Channel B		**3**	**API_BITE_ERR_HS_BC_BROAD_ISOL_AB**
- isol.coupling, LS Channel B, HS Channel B		**4**	**API_BITE_ERR_HS_BC_BROAD_ISOL_BB**
- electrical wrap, LS Channel A, HS Channel A		**5**	**API_BITE_ERR_HS_BC_BROAD_ELECT_AA**
- electrical wrap, BM, LS Channel A, HS Channel A		**6**	**API_BITE_ERR_HS_BC_BROAD_ELECT_AA_BM**
BC-RT			
- isol.coupling, LS Channel A, HS Channel A		**10**	**API_BITE_ERR_HS_BCRT_ISOL_AA**
- isol.coupling, LS Channel A, HS Channel B		**11**	**API_BITE_ERR_HS_BCRT_ISOL_AB**
- isol.coupling, LS Channel B, HS Channel A		**12**	**API_BITE_ERR_HS_BCRT_ISOL_BA**
- isol.coupling, LS Channel B, HS Channel B		**13**	**API_BITE_ERR_HS_BCRT_ISOL_BB**
RT-BC			
- isol.coupling, LS Channel A, HS Channel A		**20**	**API_BITE_ERR_HS_RTBC_ISOL_AA**
- isol.coupling, LS Channel A, HS Channel B		**21**	**API_BITE_ERR_HS_RTBC_ISOL_AB**
- isol.coupling, LS Channel B, HS Channel A		**22**	**API_BITE_ERR_HS_RTBC_ISOL_BA**
- isol.coupling, LS Channel B, HS Channel B		**23**	**API_BITE_ERR_HS_RTBC_ISOL_BB**
RT-RT			
- isol.coupling,LS Channel A, HS Channel A		**30**	**API_BITE_ERR_HS_RTRT_ISOL_AA**
HS Timing Test error	**103**		**API_BITE_ERR_HS_TIMING**
Transmitter Initialize Time		**1**	**API_BITE_ERR_HS_TIMING_INIT**
LS-HS Correlation Time		**2**	**API_BITE_ERR_HS_TIMING_CORR**
Receiver Timeout		**3**	**API_BITE_ERR_HS_TIMING_RX_TO**

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.2 ApiCmdDefMilbusProtocol

Prototype:

```
AiReturn ApiCmdDefMilbusProtocol(AiUInt32 ul_ModuleHandle,
                                 AiUInt8 biu,
                                 AiUInt8 mil_prot,
                                 AiUInt8 mode, AiUInt8 rt);
```

Purpose:

This function is used to define the MILBus Protocol type. The MILBus Protocol type is set as default to MILbus 1553B using the **ApiCmdReset** function.

Input:

`AiUInt8 mil_prot`

Value	Constant	Description
0	**API_PROTOCOL_1553_A**	MILbus 1553A protocol **Note:** This mode is not available on embedded devices! (see also chapter B.1.7 “Limitations for embedded board variants”)
1	**API_PROTOCOL_1553_B**	MILbus 1553B protocol

`AiUInt8 mode`

Value	Constant	Description
0	**API_MODE_ALL_RT**	All RTs
1	**API_MODE_SINGLE_RT**	Single RT

`AiUInt8 rt`

‘mode’	Value	Description
0	**0**	-
1	**0..31**	RT number

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.3 ApiCmdDefRespTout

Prototype:

```
AiReturn ApiCmdDefRespTout(AiUInt32 ul_ModuleHandle,
                           AiUInt8 biu,
                           float resp_tout);
```

Purpose:

This function is used to define the Response Timeout value for Simulator and Bus Monitor operation of the AIM board. The Response Timeout value is the maximum time the Bus Controller will wait for a Status word response from the RT. The Response Timeout value is set to a default of 14μs using the **ApiCmdReset** function.

Input:

`AiFloat resp_tout`

Response Timeout value
(Range: 0..63.75 μs in steps of 0.25 μs)

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.4 ApiCmdExecSys

Prototype:

```
AiReturn ApiCmdExecSys(AiUInt32 ul_ModuleHandle,
                        AiUInt8 mode, AiUInt8 con);
```

Purpose:

This function is used to execute a system related function on the AIM board target.

Input:

AiUInt8 mode

Value	Constant	Description
1	**API_MODE_TIMETAG_SOURCE**	Timetag source*
2	**API_MODE_DSUB_CONNECT**	Connection type of D-Sub connector
8	**API_MODE_HS_ADDR**	Set HS Subaddress
13	**API_MODE_SET_TRANSP_INTR**	Switch ASP interrupt handling to transparent state (no processing, pass-through only)

***Note:**
It takes several seconds (ransfe. 2 seconds) until the IRIG generator has synchronized on the new IRIG time!!!

AiUInt8 con

'mode'	Value	Constant	Description
1	**0**	**API_IRIG_ON_BOARD**	Use on-board timetag source (IRIG compatible)
	1	**API_IRIG_EXTERN**	Use external IRIG source for timetag
2	**0**	**API_DSUB_RS232**	RS-232 interface (ASP) is connected with the corresponding bits of the D-Sub connector of the PBI
	1	**API_DSUB_TRG_IN**	RT trigger inputs are connected with the corresponding bits of the D-Sub connector of the PBI
8	**0**		Disable HS Subaddress (all sub addresses are acting like pure 1553 sub 63ransferr)
	1..30		HS Subaddress to be set
13	**0**	**API_OFF**	Disable transparent interrupt mode. ASP interrupts are processed normally.
	1	**API_ON**	Enable transparent interrupt mode. ASP interrupts are processed in a pass-through manner, where normal processing is not performed.

> **Note:**
> Mode 2 is only supported on AyI devices. For limitations of the Irig time mode 1 please refere to the chapter “Board functionality overview”.

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.5 ApiCmdGetIrigStatus

Prototype:

```
AiReturn ApiCmdGetIrigStatus(AiUInt32 ul_ModuleHandle,
                             TY_API_IRIG_SOURCE *source,
                             AiBoolean * in_sync);
```

Purpose:

This function is used to read the on-board IRIG timecode encoder status.

Input:

none

Output:

`TY_API_IRIG_SOURCE *source`

Enum	Description
API_IRIG_INTERN	Board is switched to internal IRIG time source
API_IRIG_EXTERN	Board is switched to external IRIG time source

`AiBoolean *in_sync`

Returns wether the irig time is in sync with the source or not.

Note:
The IRIG time is e.g. not synchronized, if the board is switched to external IRIG time generator and no external IRIG signal is detected.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.6 ApiCmdGetIrigTime

Prototype:

```
AiReturn ApiCmdGetIrigTime(AiUInt32 ul_ModuleHandle,
                           TY_API_IRIG_TIME * time);
```

Purpose:

This function is used to read the on-board IRIG timecode encoder time.

Input:

none

Output:

`TY_API_IRIG_TIME *time`

IRIG Timecode structure

```
typedef struct ty_api_irig_time {
  AiUInt32 day;
  AiUInt32 hour;
  AiUInt32 minute;
  AiUInt32 second;
  AiUInt32 microsecond;
}
TY_API_IRIG_TIME
```

`AiUInt32 day`

The IRIG day field

`AiUInt32 hour`

Value of the IRIG hour field (0..23)

`AiUInt32 minute`

Value of the IRIG minute field (0..59)

`AiUInt32 seconds`

Value of the IRIG second field (0..59)

`AiUInt32 microsecond`

Value of the IRIG microsecond field

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.7 ApiCmdIni (obsolete)

Prototype:

```
AiReturn ApiCmdIni(AiUInt32 ul_ModuleHandle,
                   AiUInt8 mode,
                   TY_API_INI_INFO *pini);
```

Purpose:

This function is obsolete and should be used with care. If mode 3 is specified this function resets the board to the powerup state.

Input:

AiUInt8 mode

Initialization mode

Value	Constant	Description
0	**API_INIT_MODE_READ**	Read initialize parameters only
1	**API_INIT_MODE_READ**	Read initialize parameters only
2	**API_INIT_MODE_ALL**	Initialize global variables for BIU1 and BIU2

Output:

TY_API_INI_INFO *pini

Board initialization parameter

```
typedef struct ty_api_ini_info {
  AiUInt8 bt[4];
  AiUInt8 chns;
  AiUInt8 prot;
  AiUInt8 emod;
  AiUInt8 irg;
  AiUInt8 res1;
  AiUInt8 padd1;
  AiUInt16 padd2;
  AiUInt16 pbi_id_biu1;
  AiUInt16 asp_mon_id;
  AiUInt16 asp_bite_id;
  AiUInt16 pbi_id_biu2;
  AiUInt32 board_config;
  AiUInt32 glb_mem_size;
  AiUInt32 glb_mem_addr;
  AiUInt32 loc_dram_size;
  AiUInt32 loc_dram_addr;
  AiUInt32 shared_dram_size;
```

```
    AiUInt32 shared_dram_addr;
    AiUInt32 flash_ram_size;
    AiUInt32 flash_ram_addr;
    AiUInt32 pci[16];
    AiUInt32 board_type;
    AiUInt32 board_sub_type;
    AiUInt32 hardware_variant;
}
TY_API_INI_INFO;
```

AiUInt8 bt[4]

Board Type Bit Field (auc_Bt[0] ->BIU1, auc_Bt[1] ->BIU2, auc_Bt[2] ->BIU3, auc_Bt[3] ->BIU4)

Value	Bit	Constant	Description
0	**-**	**API_DEVICE_MODE_FULL**	Simulator & Monitor
1	**0**	**API_DEVICE_MODE_SIM**	Simulator only
2	**1**	**API_DEVICE_MODE_SF**	Single Function
4	**2**	**API_DEVICE_MODE_EMBEDDED**	Embedded Flag
8	**3**	**API_DEVICE_MODE_NOREPLAY_NOERROR**	No replay, No error injection
16	**4**	**API_DEVICE_MODE_DISABLE_BC**	BC disabled
0xFF	**-**	**API_DEVICE_MODE_NA**	not present

AiUInt8 chns

Amount of supported channels

'prot'	Value	Constant	Description
1	**1**	**API_CHN_SINGLE_1553**	MIL-STD-1553B Single stream
1	**2**	**API_CHN_DUAL_1553**	MIL-STD-1553B Dual Stream
1	**4**	**API_CHN_QUAD_1553**	MIL-STD-1553B Quad Stream
2	**1**	**API_CHN_SINGLE_3910**	STANAG3910 Single Stream
2	**2**	**API_CHN_DUAL_3910**	STANAG3910 Dual Stream
3	**1**	**API_CHN_SINGLE_EFEX**	EFEX Single Stream
3	**2**	**API_CHN_DUAL_EFEX**	EFEX Dual Stream
4	**2**	**API_CHN_SINGLE_3910_SINGLE_1553**	MIL-STD-1553B Single Stream + STANAG3910 Single Stream
4	**3**	**API_CHN_SINGLE_3910_DUAL_1553**	MIL-STD-1553B Dual Stream + STANAG3910 Single Stream

4	**5**	**API_CHN_SINGLE_3910_QUAD_1553**	MIL-STD-1553B Quad Stream + STANAG3910 Single Stream
5	**2**	**API_CHN_SINGLE_EFEX_SINGLE_1553**	MIL-STD-1553B Single Stream + EFEX Single Stream
5	**3**	**API_CHN_SINGLE_EFEX_DUAL_1553**	MIL-STD-1553B Dual Stream + EFEX Single Stream
5	**5**	**API_CHN_SINGLE_EFEX_QUAD_1553**	MIL-STD-1553B Quad Stream + EFEX Single Stream

AiUInt8 prot

Protocol Type

Value	Constant	Description
1	**API_PROTOCOL_1553**	MIL-STD-1553B
2	**API_PROTOCOL_3910**	STANAG3910
3	**API_PROTOCOL_EFEX**	EFEX
4	**API_PROTOCOL_1553_3910**	MIL-STD-1553B + STANAG3910
5	**API_PROTOCOL_1553_EFEX**	MIL-STD-1553B + EFEX

AiUInt8 emod

0 (Reserved)

AiUInt8 irg

IRIG on-board mode

Value	Constant	Description
0	**API_IRIG_NOT_PRESENT**	IRIG not implemented
1	**API_IRIG_PRESENT**	IRIG present

AiUInt8 res1

0 (Reserved)

AiUInt8 padd1

0 (Reserved)

AiUInt16 padd2

0 (Reserved)

AiUInt16 pbi_id_biu1

Identifier of the BIU1 PBI

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
0 (reserved)							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 (reserved)			SUB	PBI			

PBI

PBI Identification Code

Value	Constant	Description
1	**API_PBI_SINGLE_1553**	MIL1553 single channel
2	**API_PBI_DUAL_1553**	MIL1553 dual channel
4	**API_PBI_3910**	STANAG3910
9	**API_PBI_APX**	MIL1553 APX-PBI
10	**API_PBI_DUAL_1553_ONE_BIU**	MIL1553 dual channel with one BIU
16	**API_PBI_SINGLE_1553_DBTE**	MIL1553 single channel for DBTE

SUB

PBI Sub-Identification Code

PBI	Value	Constant	Description
1, 2, 10	**0**	**API_PBI_STANDARD**	Standard PBI
	1	**API_PBI_PROGRAMMABLE**	Programmable PBI
4	**0**	**API_PBI_ELECTRICAL**	Electrical interface
	1	**API_PBI_ONBOARD_FOFE**	Onboard FOFE
9	**0**	**API_PBI_APX_MILSCOPE**	With MIL-Scope
	1	**API_PBI_APX_STANDARD**	Without MIL-Scope
16	**0**	**-**	reserved
	1	**API_PBI_PROGRAMMABLE**	Programmable PBI

`AiUInt16 asp_mon_id`

ASP Monitor Software Version Number (4 digit BCD format)

`AiUInt16 asp_bite_id`

ASP Built-In Test Version Number (4 digit BCD format)

`AiUInt16 pbi_id_biu2`

Identifier of the BIU2 PBI

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
0 (reserved)							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 (reserved)			SUB	PBI			

PBI

PBI Identification Code

Value	Constant	Description
1	**API_PBI_SINGLE_1553**	MIL1553 single channel
2	**API_PBI_DUAL_1553**	MIL1553 dual channel
4	**API_PBI_3910**	STANAG3910
9	**API_PBI_APX**	MIL1553 APX-PBI
10	**API_PBI_DUAL_1553_ONE_BIU**	MIL1553 dual channel with one BIU
16	**API_PBI_SINGLE_1553_DBTE**	MIL1553 single channel for DBTE

SUB

PBI Sub-Identification Code

PBI	Value	Constant	Description
1, 2, 10	**0**	**API_PBI_STANDARD**	Standard PBI
	1	**API_PBI_PROGRAMMABLE**	Programmable PBI
4	**0**	**API_PBI_ELECTRICAL**	Electrical interface
	1	**API_PBI_ONBOARD_FOFE**	Onboard FOFE
9	**0**	**API_PBI_APX_MILSCOPE**	With MIL-Scope
	1	**API_PBI_APX_STANDARD**	Without MIL-Scope
16	**0**	**-**	reserved
	1	**API_PBI_PROGRAMMABLE**	Programmable PBI

`AiUInt32 board_config`

Board Configuration

Bit 31	...	Bit 16
Main Board HW Version		

Bit 15 ... Bit 8	Bit 7 ... Bit 0
Main Board HW Revision	Platform

HW Version	Description (four digit BCD)
0000	reserved (unset)
...	
0100	Version V01.00
0200	Version V02.00
...	

HW Revision	Description (two digit BCD)
00	reserved (unset)
01	A
02	B
. . .	
09	I
10	J
11	K
...	
26	Z
27..30	reserved
31..56	A1-Z1
57..60	reserved
61..86	A2-Z2
87..99	reserved

Platform	Define	Name
0x14	**AI_PLATFORM_PCI_SHORT_ZYNQMP**	APXX
0x1B	**AI_PLATFORM_PCI_E_1L_ZYNQMP**	APEX
0x21	**AI_PLATFORM_VMEX_B**	Reserved
0x23	**AI_PLATFORM_VMEX_A**	Reserved
0x41	**AI_PLATFORM_CPCIX_3U**	Reserved
0x43	**AI_PLATFORM_CPCIX_6U**	Reserved
0x44	**AI_PLATFORM_CPCIE_3U**	ACE
0x45	**AI_PLATFORM_CPCIX_3U_PCIE_BASED**	ACXX
0x48	**AI_PLATFORM_CPCIE_3U_ZYNQMP**	ACE
0x50	**AI_PLATFORM_PMC_32**	Reserved
0x51	**AI_PLATFORM_PMC_32_QUAD**	Reserved
0x56	**AI_PLATFORM_PMC_64_ASP**	Reserved
0x58	**AI_PLATFORM_PMC_XMC_BASED**	AMCX
0x70	**AI_PLATFORM_PC_CARD**	Reserved
0x75	**AI_PLATFORM_PCIE_CARD**	AEC
0x76	**AI_PLATFORM_MINI_PCIE_CARD**	AME
0x80	**AI_PLATFORM_PC104**	Reserved
0x90	**AI_PLATFORM_PCIX**	Reserved
0x91	**AI_PLATFORM_PCIE_PCIX_BASED**	Reserved
0x98	**AI_PLATFORM_PCIE**	APE
0x99	**AI_PLATFORM_PCIX_PCIE_BASED**	APXX
0xA0	**AI_PLATFORM_USB**	Reserved
0xA8	**AI_PLATFORM_ASC**	ASC (USB-A 2.0)
0xA8	**AI_PLATFORM_USB3**	ASC (USB-C 3.0)
0xB0	**AI_PLATFORM_XMC**	AXC
0xB3	**AI_PLATFORM_XMC_NOREPLAY_NOERROR**	ASE
0xB4	**AI_PLATFORM_XMC_ZYNQMP**	AXC (ZynqMp Based)
0xB5	**AI_PLATFORM_XMC_ARTIXUS**	AXCX (ARTIXUS+ Based)
0xC0	**AI_PLATFORM_ANET**	ANET
0xC1	**AI_PLATFORM_ANET_AYS**	ANET (AyS Based)
0xC2	**AI_PLATFORM_ANET_AYS_MA**	ANET (Mixed)

`AiUInt32 glb_mem_size`

Global Memory Size (in 64kBytes steps)

`AiUInt32 glb_mem_addr`

Global Memory Base Address

`AiUInt32 loc_dram_size`

ASP Local Memory size (in 64kBytes steps)

`AiUInt32 loc_dram_addr`

ASP Local Memory Base Address

`AiUInt32 shared_dram_size`

Shared RAM size (in 64kBytes steps)

`AiUInt32 shared_dram_addr`

Shared RAM Base Address

`AiUInt32 flash_ram_size`

Flash RAM size (in 64kBytes steps)

`AiUInt32 flash_ram_addr`

Flash RAM Base Address

`AiUInt32 pci[16]`

Reserved

`AiUInt32 board_type`

Internal use only.

`AiUInt32 board_sub_type`

Internal use only.

`AiUInt32 hardware_variant`

Internal use only.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.8 ApiCmdInitDiscretes

Prototype:

```
AiReturn ApiCmdInitDiscretes(AiUInt32 ul_ModuleHandle,
                             AiUInt32 ul_DiscreteSetup);
```

Purpose:

This command is used to configure the discretes.

Note:
this function is not supported for all boards. Please see Table B.3 – Function Support By Boards With ASP for details

Input:

`AiUInt32 ul_DiscreteSetup`

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
Reserved (0)							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
Reserved (0)							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Reserved (0)							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IN/OUT	IN/OUT	IN/OUT	IN/OUT	IN/OUT	IN/OUT	IN/OUT	IN/OUT

IN/OUT

Each of the 8 discretes can be programmed to Input or Output

Value	Description
0	Discrete is used as Input
1	Discrete is used as Output

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.9 ApiCmdReadDiscretes

Prototype:

```
AiReturn ApiCmdReadDiscretes(AiUInt32 ul_ModuleHandle,
                             AiUInt32 *pul_Value);
```

Purpose:

This command is used to read from the discrete inputs.

Note:
Since the discretes are programmable, be sure to have setup the discrete inputs with the function ApiCmdInitDiscretes()!

Note:
this function is not supported for all boards. Please see Table B.3 – Function Support By Boards With ASP for details

Input:

None

Output:

AiUInt32 *pul_Value

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
Reserved (0)							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
Reserved (0)							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Reserved (0)							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IN	IN	IN	IN	IN	IN	IN	IN

Note:
The bits are only valid if the corresponding discretes are configured as Input with the function ApiCmdInitDiscretes!

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.10 ApiCmdReadDiscretesConfig

Prototype:

```
AiReturn ApiCmdReadDiscretesConfig(AiUInt32 ul_ModuleHandle,
                                   AiUInt32 * ul_DiscreteSetup);
```

Purpose:

This command is used to read back the discrete configuration set up by ApiCmdInitDiscretes.

Note:
this function is not supported for all boards. Please see Table B.3 – Function Support By Boards With ASP for details

Input:

None

Output:

AiUInt32 * ul_DiscreteSetup

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
Reserved (0)							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
Reserved (0)							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Reserved (0)							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IN/OUT	IN/OUT	IN/OUT	IN/OUT	IN/OUT	IN/OUT	IN/OUT	IN/OUT

IN/OUT

Each of the 8 discretes can be programmed to Input or Output

Value	Description
0	Discrete is used as Input
1	Discrete is used as Output

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.11 ApiCmdReadDiscretesInfo

Prototype:

```
AiReturn ApiCmdReadDiscretesInfo(AiUInt32 ul_ModuleHandle,
                                 TY_API_DISCR_INFO *px_DiscrInfo);
```

Purpose:

This command is used to read the configuration of the discrete channels.

Input:

None

Output:

TY_API_DISCR_INFO* px_DiscrInfo

```
struct ty_api_discr_info {
  AiUInt32 channels;
  AiUInt32 canIn;
  AiUInt32 canOut;
}
TY_API_DISCR_INFO
```

AiUInt32 channels

Number of discrete channels

AiUInt32 canIn

Channel can be a Input if the corresponding bit is 1

AiUInt32 canOut

Channel can be a Output if the corresponding bit is 1

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.12 ApiCmdReadSWVersion

Prototype:

```
AiReturn ApiCmdReadSWVersion(AiUInt32 ul_ModuleHandle,
                             AiUInt8 biu,
                             AiUInt16 *fw_id,
                             AiUInt16 *sw_id,
                             AiUInt16 *lca_id,
                             AiUInt32 *lca_chks);
```

Purpose:

This function is used to read the Software version numbers of the AIM board target software parts.

Input:

none

Output:

AiUInt16 *fw_id

Firmware ID (4 digit BCD format)

AiUInt16 *sw_id

Driver Software version number (4 digit BCD format)

AiUInt16 *lca_id

LCA Software ID (1 digit BCD format)

AiUInt32 *lca_chks

LCA Software Checksum value (32-bit)

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.13 ApiCmdReset

Prototype:

```
AiReturn ApiCmdReset(AiUInt32 ul_ModuleHandle,
                     AiUInt8 biu, AiUInt8 rc,
                     TY_API_RESET_INFO *pres);
```

Purpose:

This function initializes the associated streams data structures and variables to an initial state. The initialization includes writing the simulation buffer area with zeros. Since the simulation buffer is shared between all streams, calling ApiCmdReset does affect the data transmitted and received by all streams of the board. To avoid this use the Reset Control option API_RESET_WITHOUT_SIMBUF.
The **ApiCmdReset** function reports memory configuration information and has to be applied before using any of the API S/W library functions described in the following sections. **MILBus protocol** will be set to a default value of Type **B** (1553B). **Response Timeout** will be set to a default value of **14μs**.

Note:
The LS Monitor Buffer size is set to 512 Kbytes, the LS Simulator Buffer Size is set to 256 Kbytes!

Note:
When using a 3910 board this command also sets default size values for HS Monitor and HS Simulator Area. The HS Monitor Buffer size is set to the half size of the onboard Global RAM (which is currently available with 4 or 8 Mbytes). The default HS Simulator Buffer Size is set to 0.5 Mbytes (with 4 Mbytes Global RAM) or 2.5 Mbytes (with 8 Mbytes Global RAM)!

Input:

AiUInt8 rc

Reset Control

Value	Constant	Description
0	**API_RESET_ALL**	**Complete Reset**
1	**API_RESET_WITHOUT_MONITOR**	**Partial Reset excluding:** - transmitter amplitude - bus connection and coupling - bus monitor
2	**API_RESET_WITHOUT_SIMBUF**	**Partial Reset excluding:** - BC/RT buffer area
3	**API_RESET_WITHOUT_SIMBUF_MON**	**Partial Reset excluding:** - transmitter amplitude - bus connection and coupling - bus monitor - BC/RT buffer area
0x40	**API_RESET_USE_COUNTER_MODE_2**	**Use global BC and RT counters increment also on errors**
0x80	**API_RESET_ENABLE_1760**	**Enable 1760 simulation mode** Refer to ApiCmdBufC1760Con

Output:

`TY_API_RESET_INFO *pres`

Reset information

```
typedef struct ty_api_reset_info {
  AiUInt8 mbufs;
  AiUInt8 sbufs;
  AiUInt32 mon_addr;
  AiUInt32 sim_addr;
}
TY_API_RESET_INFO;
```

`AiUInt8 mbufs`

Allocated bus monitor buffer size (in 64kBytes steps)

`AiUInt8 sbufs`

Allocated simulator buffer size (in 64kBytes steps)

`AiUInt32 mon_addr`

26-bit global memory pointer (byte address) to bus monitor buffer start address

`AiUInt32 sim_addr`

26-bit global memory pointer (byte address) to bus simulator buffer start address

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.14 ApiCmdSetIrigStatus

Prototype:

```
AiReturn ApiCmdSetIrigStatus(AiUInt32 ul_ModuleHandle,
                             TY_API_IRIG_SOURCE new_source);
```

Purpose:

This function is used to set the on-board IRIG timecode encoder status.

> **Note:**
> The IRIG timecode encoder needs up to three seconds before changing the time. Please make sure no operations that need the IRIG time are made during the next three seconds after ApiCmdSetIrigTime was called.

Input:

TY_API_IRIG_SOURCE *new_source

Enum	Description
API_IRIG_INTERN	Board is switched to internal IRIG time source
API_IRIG_EXTERN	Board is switched to external IRIG time source

> **Note:**
> This parameter is not possible on all AIM boards. Please check chapter "Board functionality overview". If your board does not support INTERN/EXTERN switching it uses "Free Wheeling Mode"

Output:

none

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.15 ApiCmdSetIrigTime

Prototype:

```
AiReturn ApiCmdSetIrigTime(AiUInt32 ul_ModuleHandle,
                           TY_API_IRIG_TIME *time);
```

Purpose:

This function sets the IRIG-B time on the on-board IRIG timecode encoder.

Note:
IRIG time starts with ‘DAY one’ (First of January) not with ‘DAY zero’.

Note:
Most cards need up to three seconds before changing the time. Please make sure no operations that need the IRIG time are made during the next three seconds after ApiCmdSetIrigTime was called. Some new cards are able to set the time instandly if the internal irig mode is use. See TY_IRIG_CAPABILITIES 2.1.7

Note:
The IRIG time is only set according to day, hour, minute and second fields. The millisecond field is ignored. Some new cards are able to set the milliseconds. See TY_IRIG_CAPABILITIES 2.1.7

Input:

TY_API_IRIG_TIME *time

IRIG Timecode structure

```
typedef struct ty_api_irig_time {
  AiUInt32 day;
  AiUInt32 hour;
  AiUInt32 minute;
  AiUInt32 second;
  AiUInt32 microsecond;
}
TY_API_IRIG_TIME
```

AiUInt32 day

IRIG day field

AiUInt32 hour

IRIG hour field (0..23)

AiUInt32 minute

IRIG minute field (0..59)

`AiUInt32 second`

IRIG second field (0..59)

`AiUInt32 microsecond`

This value is ignored when the time is set.

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.16 ApiCmdSyncCounterGet

Prototype:

```
AiReturn ApiCmdSyncCounterGet(AiUInt32 ul_ModuleHandle,
                              AiUInt8 uc_Biu,
                              TY_API_SYNC_CNT_GET *px_SyncCntGet);
```

Purpose:

This function reads all synchronization counter values.

BC Mode Code Handling:
The synchronization counter is used for the data word of a synchronize mode code (MC17).

RT Mode Code Handling:

MC1 (Synchronize)	This mode command resets the synchronization counter and updates the synchronization init time tags.
MC17 (Synchronize with DW)	This mode command initializes the synchronization counter with the received data value and updates the synchronization time tags.

Note:
On BC side the functionaliy is only applicaple if enabled with the function ApiCmdBCModeCtrl()!

Note:
On RT side the functionaliy is only applicaple if the BC is not running!

Note:
This function is not available on devices with a multi channel firmware. Please see chapter “Limitations for specific boards” for details.

Input:

None

Output:

`TY_API_SNYC_CNT_GET *px_SyncCntGet`

Synchronization Counter get structure

```
typedef struct ty_api_sync_cnt_set {
  AiUInt32 ul_SyncCntVal;
  AiUInt32 ul_SyncCntInit;
  AiUInt32 ul_SyncCntInitLow;
  AiUInt32 ul_SyncCntInitHigh;
}
TY_API_SYNC_CNT_SET
```

AiUInt32 ul_SyncCntVal

Actual Synchronization Counter Value

Value	Description
0..65535	Actual Synchronization Counter value in steps of 64μs (range from 0..4,19 sec)

AiUInt32 ul_SyncCntInit

Synchronization Counter Init Value
This value is either set with the function ApiCmdSyncCounterSet() or if an RT receives a synchronize mode code (MC1 or MC17). If MC1 is received by an RT, this value is reset to 0.

Note:
The RT only modifies the synchronization counter on receiving a MC1 or MC17, if the BC is not running!

Value	Description
0..65535	Synchronization Counter Init value in steps of 64μs (range from 0..4,19 sec)

AiUInt32 ul_SyncCntInitLow

Synchronization Counter Init Timetag Low

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
MINUTES						SECONDS	

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
SECONDS				MICROSECONDS			

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
MICROSECONDS							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MICROSECONDS							

MINUTES	Minutes of hour(0..59)
SECONDS	Seconds of minute (0..59)
MICROSECONDS	Microseconds of second (0..999999)

AiUInt32 ul_SyncCntInitHigh

Synchronization Counter Init Timetag High

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
0 (reserved)							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0 (reserved)				DAYS			

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
DAYS					HOURS		

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HOURS		MINUTES					

DAYS	Days of year (1..365)
HOURS	Hours of day (0..23)
MINUTES	Minutes of hour(0..59)

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.17 ApiCmdSyncCounterSet

Prototype:

```
AiReturn ApiCmdSyncCounterSet(AiUInt32 ul_ModuleHandle,
                              AiUInt8 uc_Biu,
                              TY_API_SYNC_CNT_SET *px_SyncCntSet);
```

Purpose:

This function sets synchronization counter init value. This counter is used for the data word of a synchronize mode code (MC17).

Note:
The functionaliy is only applicaple if enabled with the function ApiCmdBCModeCtrl().

Note:
This function is not available on devices with a multi channel firmware. Please see chapter "Limitations for specific boards" for details.

Input:

TY_API_SNYC_CNT_SET *px_SyncCntSet

Synchronization Counter set structure

```
typedef struct ty_api_sync_cnt_set {
  AiUInt32 ul_SyncCntVal;
}
TY_API_SYNC_CNT_SET
```

AiUInt32 ul_SyncCntVal

Synchronization Counter Init Value

Value	Description
0..65535	Synchronization Counter in steps of 64μs (range from 0..4,19 sec)

Output:

none

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.18 ApiCmdSysFree

Prototype:

```
AiReturn ApiCmdSysFree(AiUInt32 ul_Module,
                       AiUInt8 uc_MemType,
                       AiUInt32 ul_Offset,
                       AiUInt32 ul_Tag);
```

Purpose:

Free a memory block that was allocated with the **ApiCmdSysMalloc** function. This function is currently only implemented on AEC/AXC/AMCX1553 boards for update purpose.

Input:

AiUInt8 uc_MemType

The memory type from where the block was allocated.

Value	Define	Description
0	**API_MEMTYPE_GLOBAL**	Global memory
1	**API_MEMTYPE_SHARED**	Shared memory
2	**API_MEMTYPE_LOCAL**	Local memory

AiUInt32 ul_Offset

The start offset of the memory block.

AiUInt32 ul_Tag

Name of the request for debug and bug check purpose.

Output:

none

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.19 ApiCmdSysGetBoardInfo

Prototype:

```
AiReturn ApiCmdSysGetBoardInfo(AiUInt32 ulModHandle,
                               AiUInt32 ulDataStart,
                               AiUInt32 ulDataCount,
                               AiUInt32 * ulOutput,
                               AiUInt32 * ulOutCount);
```

Purpose:

This function can be used to get information about the board from the target software. The interpretation of the fields is analog to the function ApiGetBoardInfo description.

Input:

`AiUInt32 ulDataStart`

Identifies the first board information element to read.

Define	Interpretations
TY_BOARD_INFO_DEVICE_TYPE	ul_DeviceType
TY_BOARD_INFO_CHANNEL_COUNT	ul_NumberOfChannels
TY_BOARD_INFO_BIU_COUNT	ul_NumberOfBiu
TY_BOARD_INFO_BOARD_TYPE	ul_NovRamBoardType
TY_BOARD_INFO_BOARD_CONFIG	ul_NovRamBoardConfig
TY_BOARD_INFO_SERIAL	ul_SerialNumber
TY_BOARD_INFO_PARTNO	Partnumber
TY_BOARD_INFO_SIZE_GLOBAL	Size of the global RAM
TY_BOARD_INFO_SIZE_SHARED	Size of the shared RAM
TY_BOARD_INFO_OFFS_GLOBAL	Global RAM offset for second PBI
TY_BOARD_INFO_CHANGE_AMPL	ul_CanChangeAmplitude
TY_BOARD_INFO_COUPLING	TY_COUPLING_CAPABILITIES
TY_BOARD_INFO_IRIG	TY_IRIG_CAPABILITIES
TY_BOARD_INFO_DISCRETE_CNT	Number of discretes available
TY_BOARD_INFO_DISCRETE_CONFIG	Configuration of discretes
TY_BOARD_INFO_IR	Reserved for interrupts on gpio
TY_BOARD_INFO_IS_MULTI_CHANNEL	This value is 1 for multichannel FW
TY_BOARD_INFO_IS_HS_REDUNDANT	This value is 1 if HS channel B is not available.
TY_BOARD_INFO_CAN_HIGH_RES _ZERO_CROSSING	This value is 1 if high Res Zero Crossing is supported.
TY_BOARD_INFO_HAS_ELECTRICAL _INTERFACE	This value is 1 if it has an electrical interface
TY_BOARD_INFO_MILSCOPE_TYPE	This value is 1 for AYX MilScope and 2 for AyE MilScope
TY_BOARD_INFO_IS_DBTE	Special PBI with removed relais.

TY_BOARD_INFO_HAS_EXTERNAL_IRIG	Has external irig connector.
TY_BOARD_INFO_HAS_EXTERNAL _TRIGGER	Has external trigger connector.
TY_BOARD_INFO_PROTOCOL	See ApiCmdIni / prot
TY_BOARD_INFO_APPLICATION_TYPE	See ApiCmdIni / bt[4]
TY_BOARD_INFO_CHANGE_AMPL _HIGH_RES	AiTrue if this device has a high resolution amplitude range.
TY_BOARD_INFO_CAN_OVERRIDE _ TRAFFIC	AITrue if this device has the traffic override feature.
TY_BOARD_INFO_HAS_FAST_GAP	AiTrue if this device has fast gap support
TY_BOARD_INFO_HAS_LOW_PRIORITY _LIST	AiTrue if this device has low priority list support
TY_BOARD_INFO_HAS_BC_INSTR _ RETURN_ON_XFER_ERROR	AiTrue if this device has return on xfer error support
TY_BOARD_INFO_USB_AIM_POWER _ CLASS	The AIM Power Class for USB-C devices: Class 0: 100% bus load Class 1: max. 50% bus load Class 2: max. 10% bus load

`AiUInt32 ulDataCount`

Defines the number of board information elements to read. This value is in the range from 1 to TY_BOARD_INFO_MAX.

Output:

`AiUInt32 * ulOutput`

This array must be preallocated with the size of ulDataCount. This array contains the requested values.

`AiUInt32 * ulOutCount`

The count of valid intries in the ulOutput array.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.20 ApiCmdSysGetMemPartition

Prototype:

```
AiReturn ApiCmdSysGetMemPartition(AiUInt32 ul_ModuleHandle,
                                  AiUInt8 uc_Mode,
                                  TY_API_GET_MEM_INFO *px_MemInfo);
```

Purpose:

This function is used to return the actual partitioning of the structures setup within the Board Global memory (e.g. Start Addresses of descriptor areas, Amount of descriptors, IDs, etc)

Note:
All returned addresses are relative to the start of the Global RAM!

Note:
To get the default values of the board memory settings, just call ApiCmdSysGetMemPartition as the first call in your application after a reboot of your computer!

Input:

AiUInt8 uc_Mode

Value	Description
0	reserved

Output:

TY_API_GET_MEM_INFO *px_MemInfo

```
typedef struct ty_api_get_mem_layout {
  TY_API_MEM_BIU_ADDR ax_BiuAddr[8];
  TY_API_MEM_SIM_BUF x_Sim[2];
  TY_API_MEM_BIU_SIZE ax_BiuSize[8];
  TY_API_MEM_BIU_COUNT ax_BiuCnt[8];
  TY_API_MEM_BIU_INFO ax_BiuInfo[8];
  AiUInt32 aul_GlobalMemSize[2];
}
TY_API_GET_MEM_INFO;
```

TY_API_MEM_BIU_ADDR ax_BiuAddr[8]

Start addresses of structures (logical BIUs 1..8)

```
typedef struct ty_api_mem_biu_addr {
  AiUInt32 ul_Cb;
  AiUInt32 ul_IrLog;
  AiUInt32 ul_RtDesc;
```

```
  AiUInt32 ul_BmTcb;
  AiUInt32 ul_BmAct;
  AiUInt32 ul_RtSaDesc;
  AiUInt32 ul_RtBhArea;
  AiUInt32 ul_RtSqArea;
  AiUInt32 ul_RtEqArea;
  AiUInt32 ul_BcBhArea;
  AiUInt32 ul_BcSqArea;
  AiUInt32 ul_BcEqArea;
  AiUInt32 ul_BcXferDesc;
  AiUInt32 ul_BcHipInstr;
  AiUInt32 ul_BcLipInstr;
  AiUInt32 ul_BcAcycInstr;
  AiUInt32 ul_RepBuf;
  AiUInt32 ul_BmBuf;
}
TY_API_MEM_BIU_ADDR;
```

AiUInt32 ul_Cb
Start address of the system control block

AiUInt32 ul_IrLog
Start address of the interrupt log list

AiUInt32 ul_RtDesc
Start address of the RT descriptor area

AiUInt32 ul_BmTcb
Start address of the BM trigger control block area

AiUInt32 ul_BmAct
Start address of the BM activity recording / message filtering page

AiUInt32 ul_RtSaDesc
Start address of the RT subaddress/modecode descriptor table

AiUInt32 ul_RtBhArea
Start address of the RT buffer header area

AiUInt32 ul_RtSqArea
Start address of the RT status queue area

AiUInt32 ul_RtEqArea
Start address of the RT event queue area

AiUInt32 ul_BcBhArea
Start address of the BC buffer header area

AiUInt32 ul_BcSqArea
Start address of the BC status queue area **AiUInt32 ul_BcEqArea**

Start address of the BC event queue area

AiUInt32 ul_BcXferDesc
Start address of the BC transfer descriptor area

AiUInt32 ul_BcHipInstr
Start address of the BC high priority instruction list

AiUInt32 ul_BcLipInstr
Start address of the BC low priority instruction list

AiUInt32 ul_BcAcycInstr
Start address of the BC acyclic instruction list

AiUInt32 ul_RepBuf
Start address of the Replay buffer

AiUInt32 ul_BmBuf
Start address of the Bus Monitor buffer

TY_API_MEM_SIM_BUF x_Sim[2]

Start addresses and sizes of Simulator data buffers for every Global Memory bank (0..1)

```
typedef struct ty_api_mem_sim_buf {
  AiUInt32 ul_BufBaseAddr;
  AiUInt32 ul_BufSize;
  AiUInt32 ul_BufCount;
  AiUInt32 ul_HsBufBaseAddr;
  AiUInt32 ul_HsBufSize;
  AiUInt32 ul_HsRes;
}
TY_API_MEM_SIM_BUF;
```

AiUInt32 ul_BufBaseAddr
Start address of the simulator buffer

AiUInt32 ul_BufSize
Size of the simulator buffer in bytes

AiUInt32 ul_BufCount
Number of data buffers (32 words) in simulator buffer

AiUInt32 ul_HsBufBaseAddr
Start address of the HS simulator buffer

AiUInt32 ul_HsBufSize
Size of the HS simulator buffer in bytes

AiUInt32 ul_HsRes
0 (reserved)

TY_API_MEM_BIU_SIZE ax_BiuSize[8]

Size of BC instruction list areas, Replay and BM buffer area (logical BIUs 1..8)

```
typedef struct ty_api_mem_biu_size {
  AiUInt32 ul_BcHipInstr;
  AiUInt32 ul_BcLipInstr;
  AiUInt32 ul_BcAcycInstr;
  AiUInt32 ul_RepBuf;
  AiUInt32 ul_BmBuf;
}
TY_API_MEM_BIU_SIZE;
```

AiUInt32 ul_BcHipInstr
Size if the high priority instruction list in bytes

AiUInt32 ul_BcLipInstr
Size if the low priority instruction list in bytes

AiUInt32 ul_BcAcycInstr
Size if the acyclic instruction list in bytes

AiUInt32 ul_RepBuf
Size of the Replay buffer in bytes

AiUInt32 ul_BmBuf
Size of the Bus Monitor buffer in bytes

TY_API_MEM_BIU_COUNT ax_BiuCnt[8]

Amount of BC & RT simulator related IDs (logical BIUs 1..8)

```
typedef struct ty_api_mem_biu_count {
  AiUInt32 ul_RtBhArea;
  AiUInt32 ul_RtSqArea;
  AiUInt32 ul_RtEqArea;
  AiUInt32 ul_BcBhArea;
  AiUInt32 ul_BcSqArea;
  AiUInt32 ul_BcEqArea;
  AiUInt32 ul_BcXferDesc;
}
TY_API_MEM_BIU_COUNT;
```

AiUInt32 ul_RtBhArea
Maximum number of RT buffer header IDs

AiUInt32 ul_RtSqArea
Number of RT status queues

AiUInt32 ul_RtEqArea
Number of RT event queues

AiUInt32 ul_BcBhArea
Maximum number of BC buffer header IDs

AiUInt32 ul_BcSqArea
Number of BC status queues

AiUInt32 ul_BcEqArea
Number of BC event queues

AiUInt32 ul_BcXferDesc
Maximum number of transfer IDs

`TY_API_MEM_BIU_INFO ax_BiuInfo[8]`

Information structure how to use the other structure parts of this function

```
typedef struct ty_api_mem_biu_info {
  AiUInt32 ul_Protocol;
  AiUInt32 ul_StreamNb;
  AiUInt32 ul_MemoryBank;
}
TY_API_MEM_BIU_INFO;
```

AiUInt32 ul_Protocol
Protocol of the logical BIU referenced by the index (0..7)

Value	Constant	Description
1	**API_PROTOCOL_1553**	MIL-STD-1553B
2	**API_PROTOCOL_3910**	STANAG3910
3	**API_PROTOCOL_EFEX**	EFEX

AiUInt32 ul_StreamNb
Stream number of the logical BIU referenced by the index (0..7)

Value	Constant	Description
1	**API_STREAM_1**	Stream 1
2	**API_STREAM_2**	Stream 2
3	**API_STREAM_3**	Stream 3
4	**API_STREAM_4**	Stream 4
5	**API_STREAM_5**	Stream 5
6	**API_STREAM_6**	Stream 6
7	**API_STREAM_7**	Stream 7
8	**API_STREAM_8**	Stream 8

AiUInt32 ul_MemoryBank
Global Memory Bank of the logical BIU referenced by the index (0..7)

Value	Constant	Description
0	**-**	Memory Bank 0
1	**-**	Memory Bank 1

`AiUInt32 aul_GlobalMemSize[2]`

Size of the Global RAM in Mbytes for every Global Memory bank (0..1)

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.21 ApiCmdSysMalloc

Prototype:

```
AiReturn ApiCmdSysMalloc(AiUInt32 ul_Module,
                         AiUInt8 uc_MemType,
                         AiUInt32 ul_Size,
                         AiUInt32 ul_Tag,
                         AiUInt32* pul_Offset);
```

Purpose:

Allocate a memory block with the given size in the given memory. This function is currently only implemented on AEC/AXC/AMCX1553 boards for update purpose. Only global memory is accepted and the function will always return an offset of 16MB.

Input:

AiUInt8 uc_MemType

The memory type from where the block should be allocated.

Value	Define	Description
0	**API_MEMTYPE_GLOBAL**	Global memory
1	**API_MEMTYPE_SHARED**	Shared memory
2	**API_MEMTYPE_LOCAL**	Local memory

AiUInt32 ul_Size

The size of the memory block.

AiUInt32 ul_Tag

Name of the request for debug and bug check purpose.

Output:

AiUInt32 * pul_Offset

The offset where the allocated buffer starts.

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.22 ApiCmdSysSetMemPartition

Prototype:

```
AiReturn ApiCmdSysSetMemPartition(AiUInt32 ul_ModuleHandle,
                                  AiUInt8 uc_Mode,
                                  TY_API_SET_MEM_INFO *px_MemInfo,
                                  AiUInt32 *pul_Status,
                                  AiUInt32 aul_MemUsed[2]);
```

Purpose:

This function is used to configure the actual partitioning of the structures setup within the Board Global memory (e.g. Start Addresses of descriptor areas, Amount of descriptors, IDs, etc).

Note:
If this function is used, it must be called before the function ApiCmdIni() is called in the application!

Note:
To get the default values of the board memory settings, just call ApiCmdSysGetMemPartition as the first call in your application after a reboot of your computer!

Input:

AiUInt8 uc_Mode

Value	Description
0	Normal use case
1	Reserved
2	Enhanced BM activity page

Note:
This mode is not available on embedded devices! (see also chapter B.1.7 "Limitations for embedded board variants")

TY_API_SET_MEM_INFO *px_MemInfo

```
typedef struct ty_api_set_mem_layout {
  AiUInt32 aul_SimBufSize[2];
  TY_API_MEM_BIU_SIZE ax_BiuSize[8];
  TY_API_MEM_BIU_COUNT ax_BiuCnt[8];
}
TY_API_SET_MEM_INFO;
```

AiUInt32 aul_SimBufSize[2]

Sizes of Simulator data buffers for every Global Memory bank (0..1)

TY_API_SET_MEM_BIU_SIZE ax_BiuSize[8]

Size of BC instruction list areas, Replay and BM buffer area area (logical BIUs 1..8).

```
typedef struct ty_api_set_mem_biu_size {
  AiUInt32 ul_BcHipInstr;
  AiUInt32 ul_BcLipInstr;
  AiUInt32 ul_BcAcycInstr;
  AiUInt32 ul_RepBuf;
  AiUInt32 ul_BmBuf;
}
TY_API_SET_MEM_BIU_SIZE;
```

AiUInt32 ul_BcHipInstr

Size of the high priority instruction list in entries

AiUInt32 ul_BcLipInstr

Size of the low priority instruction list in entries

AiUInt32 ul_BcAcycInstr

Size of the acyclic instruction list in entries

AiUInt32 ul_RepBuf

Size of the Replay buffer in bytes

AIM Card	Value	Description
1553	**0**	Replay is not used for the relating BIU
	20000h	For BIU1
	20000h	For BIU2 (available only on 1553-2 cards)
3910	**0**	Replay is not used for the relating BIU **Note:** To disable replay on 3910 cards, this value has to be set to 0 for all BIUs on the HS Global Memory bank!
	20000h	For BIU1
	60000h	For BIU2

Note:
To be able to use the replay mechanism of the board the above sizes must be set!!!

AiUInt32 ul_BmBuf Size of the Bus Monitor buffer in bytes

TY_API_SET_MEM_BIU_COUNT ax_BiuCnt[8]

Amount of BC & RT simulator related IDs (logical BIUs 1..8)

```
typedef struct ty_api_set_mem_biu_count {
  AiUInt32 ul_RtBhArea;
  AiUInt32 ul_RtSqArea;
  AiUInt32 ul_RtEqArea;
  AiUInt32 ul_BcBhArea;
```

```
  AiUInt32 ul_BcSqArea;
  AiUInt32 ul_BcEqArea;
  AiUInt32 ul_BcXferDesc;
}
TY_API_SET_MEM_BIU_COUNT;
```

AiUInt32 ul_RtBhArea

Maximum number of RT buffer header Ids
This value shall be a multiple of 256!

AiUInt32 ul_RtSqArea

Number of RT status queues

> **Note:**
> Only for LS memory banks: When the LS status queue is intended to be used, this value must be set to 'aul_SimBufSize[n]' / 32 (where n is the Global Memory bank [0..1])

AiUInt32 ul_RtEqArea

Number of RT event queues
This value shall be a multiple of 256!

AiUInt32 ul_BcBhArea

Maximum number of BC buffer header Ids
This value shall be a multiple of 256!

AiUInt32 ul_BcSqArea

Number of BC status queues

> **Note:**
> Only for LS memory banks: When the LS status queue is intended to be used, this value must be set to 'aul_SimBufSize[n]' / 32 (where n is the Global Memory bank [0..1])

AiUInt32 ul_BcEqArea

Number of BC event queues
This value shall be a multiple of 256!

AiUInt32 ul_BcXferDesc

Maximum number of transfer Ids
This value shall be a multiple of 256!

Output:

AiUInt32 *pul_Status

Value	Constant	Description
0	**API_MEM_PART_OK**	Successful Global Memory partitioning
1	**API_MEM_PART_ERR**	Memory allocation for partitioning failed (requested layout size too big)
2	**API_MEM_PART_PARAM_ERR**	Invalid Input parameter value

`AiUInt32 pul_MemUsed[2]`

Amount of Global RAM area used by requested partitioning in bytes for each Global Memory bank (0..1)

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.23 ApiCmdSysPXICon

Prototype:

```
AiReturn ApiCmdSysPXICon(AiUInt32 ul_ModuleHandle,
                         TY_API_PXI_CON *px_PXICon);
```

Purpose:

This function is used to combine PXI specific trigger lines with the trigger lines of AIM boards. Any of the 8 PXI trigger lines can be set once to one trigger line of the AIM board, thus a maximum number of 8 trigger line combinations are possible.

It also provides support to switch the IRIG TT source.

> **Note:**
> this function is not supported for all boards. Please see Table B.3 – Function Support By Boards With ASP for details

Input:

TY_API_PXI_CON *px_PXICon

```
typedef struct ty_api_pxi_con {
  AiUInt32 ul_Mode;
  AiUInt32 ul_TrgSource;
  AiUInt32 ul_TrgDest;
  AiUInt32 ul_TTClear;
}
TY_API_PXI_CON;
```

AiUInt32 ul_Mode

Value	Constant	Description
0	**API_PXI_SET_TRG**	Define a new Trigger combination using trigger source and destination given in parameters 'ul_TrgSource' and 'ul_TrgDest'
1	**API_PXI_CLR_TRG**	Clear all previously defined trigger combinations
2	**API_PXI_SET_TTSRC_BACKPLANE**	The external 10MHz PXI-Reference Clock from the PXI-Rack Backplane is used for Time Tagging.
3	**API_PXI_SET_TTSRC_FRONT**	The external 1KHz analog IRIG-B input signal via Frontpanel-I/O is used for Time Tagging.

AiUInt32 ul_TrgSource

Value	Constant	Restriction	Description
0	**API_TRG_PXI0**		PXI Trigger Line 0
1	**API_TRG_PXI1**		PXI Trigger Line 1
2	**API_TRG_PXI2**		PXI Trigger Line 2
3	**API_TRG_PXI3**		PXI Trigger Line 3
4	**API_TRG_PXI4**		PXI Trigger Line 4
5	**API_TRG_PXI5**		PXI Trigger Line 5
6	**API_TRG_PXI6**		PXI Trigger Line 6
7	**API_TRG_PXI7**		PXI Trigger Line 7
8	**API_TRG_BC_CHN1**		Onboard BC Trigger of Channel 1
9	**API_TRG_BC_CHN2**	Only on ACX1553-2-3U and ACX1553-4-3U available	Onboard BC Trigger of Channel 2
10	**API_TRG_BC_CHN3**	Only on ACX1553-4-3U available	Onboard BC Trigger of Channel 3
11	**API_TRG_BC_CHN4**	Only on ACX1553-4-3U available	Onboard BC Trigger of Channel 4
12	**API_TRG_RT_CHN1**		Onboard RT Trigger of Channel 1
13	**API_TRG_RT_CHN2**	Only on ACX1553-2-3U and ACX1553-4-3U available	Onboard RT Trigger of Channel 2
14	**API_TRG_RT_CHN3**	Only on ACX1553-4-3U available	Onboard RT Trigger of Channel 3
15	**API_TRG_RT_CHN4**	Only on ACX1553-4-3U available	Onboard RT Trigger of Channel 4
16	**API_TRG_BM_CHN1**		Onboard BM Trigger of Channel 1
17	**API_TRG_BM_CHN2**	Only on ACX1553-2-3U and ACX1553-4-3U available	Onboard BM Trigger of Channel 2
18	**API_TRG_BM_CHN3**	Only on ACX1553-4-3U available	Onboard BM Trigger of Channel 3
19	**API_TRG_BM_CHN4**	Only on ACX1553-4-3U available	Onboard BM Trigger of Channel 4
20	**API_TRG_BM_HS**	Only on ACX3910-3U and ACE3910 available	Onboard BM HS Trigger

AiUInt32 ul_TrgDest

Value	Constant	Restriction	Description
0	**API_TRG_PXI0**		PXI Trigger Line 0
1	**API_TRG_PXI1**		PXI Trigger Line 1
2	**API_TRG_PXI2**		PXI Trigger Line 2
3	**API_TRG_PXI3**		PXI Trigger Line 3
4	**API_TRG_PXI4**		PXI Trigger Line 4
5	**API_TRG_PXI5**		PXI Trigger Line 5
6	**API_TRG_PXI6**		PXI Trigger Line 6
7	**API_TRG_PXI7**		PXI Trigger Line 7
8	**API_TRG_BC_CHN1**		Onboard BC Trigger of Channel 1
9	**API_TRG_BC_CHN2**	Only on ACX1553-2-3U and ACX1553-4-3U available	Onboard BC Trigger of Channel 2
10	**API_TRG_BC_CHN3**	Only on ACX1553-4-3U available	Onboard BC Trigger of Channel 3
11	**API_TRG_BC_CHN4**	Only on ACX1553-4-3U available	Onboard BC Trigger of Channel 4
12	**API_TRG_RT_CHN1**		Onboard RT Trigger of Channel 1
13	**API_TRG_RT_CHN2**	Only on ACX1553-2-3U and ACX1553-4-3U available	Onboard RT Trigger of Channel 2
14	**API_TRG_RT_CHN3**	Only on ACX1553-4-3U available	Onboard RT Trigger of Channel 3
15	**API_TRG_RT_CHN4**	Only on ACX1553-4-3U available	Onboard RT Trigger of Channel 4
16	**API_TRG_BM_CHN1**		Onboard BM Trigger of Channel 1
17	**API_TRG_BM_CHN2**	Only on ACX1553-2-3U and ACX1553-4-3U available	Onboard BM Trigger of Channel 2
18	**API_TRG_BM_CHN3**	Only on ACX1553-4-3U available	Onboard BM Trigger of Channel 3
19	**API_TRG_BM_CHN4**	Only on ACX1553-4-3U available	Onboard BM Trigger of Channel 4
20	**API_TRG_BM_HS**	Only on ACX3910-3U and ACE3910 available	Onboard BM HS Trigger

`AiUInt32 ul_TTClear`

Enable / disable Time Tag Clear functionality via PXI Trigger0 or PXI Start Trigger Input

'ul_Mode'	Value	Constant	Description
0, 1	**0**		This functionality is not available with these modes
2, 3	**0**	**API_DIS**	Time Tag Clear over PXI TRIGGER INPUT 0 or via the PXI STAR TRIGGER INPUT line is disabled.
2,3	**1**	**API_ENA**	Time Tag Clear over PXI TRIGGER INPUT 0 or via the PXI STAR TRIGGER INPUT line is enabled. The Time Tag can be cleared via an electrical pulse on PXI TRIGGER INPUT 0 or PXI STAR TRIGGER INPUT line.

Note:
If enabled, the PXI Trigger Line 0 cannot be used as Trigger source or destination when defining a trigger combination!

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.24 ApiCmdSysPXIGeographicalAddressGet

Prototype:

```
AiReturn ApiCmdSysPXIGeographicalAddressGet(AiUInt32 ul_ModuleHandle,
                                            AiUInt32 *pxiGeographicalAddress);
```

Purpose:

This function will get the geographical address of the PXI slot where the PXI board is plugged in.

> **Note:**
> this function is not supported for all boards. Please see Table B.3 – Function Support By Boards With ASP for details

Input:

None

Output:

`AiUInt32 *pxiGeographicalAddress`

Geographical address of used PXI slot, value 0..31.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.25 ApiCmdSystagCon

Prototype:

```
AiReturn ApiCmdSystagCon(AiUInt32 ul_ModuleHandle,
                         AiUInt8 biu,
                         AiUInt8 id,
                         AiUInt8 con);
```

Purpose:

This function is used to control (Suspend & Resume) the generation of System Dynamic Data words in BC and RT mode as defined with function **ApiCmdSystagDef**.

Input:

`AiUInt8 id`

Identifies the System Dynamic Data generation scenario defined when ApiCmdSystagDef is issued.

Value	Description
1..255	System Dynamic Data identifier global for the module

`AiUInt8 con`

System Dynamic Data Control

Value	Constant	Description
0	**API_SYSTAG_SUSPEND**	Suspend System Dynamic Data Generation
1	**API_SYSTAG_RESUME**	Resume System Dynamic Data Generation
4	**API_SYSTAG_RECALC_CHECKSUM**	Re-calculate checksum, if the systag referred to with 'id' was defined with type API_SYSTAG_FCT_CHECKSUM in function **ApiCmdSystagDef()**

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.26 ApiCmdSystagDef

Prototype:

```
AiReturn ApiCmdSystagDef(AiUInt32 ul_ModuleHandle,
                         AiUInt8 biu, AiUInt8 id,
                         AiUInt8 con, AiUInt8 mode,
                         TY_API_SYSTAG *psystag);
```

Purpose:

This function defines a scenario to insert System Dynamic Data words/buffers into the data transmitted by the simulated BC and/or RT. Up to 255 scenarios can be created per module. This function can also enable/disable the defined System Dynamic Data scenario. If dataset buffers are to be transmitted by the simulated BC/RT, these buffers should first be filled with data using the **ApiCmdRamWriteDataset** function.

On BC side, System Dynamic Data insertion is performed in the BC Transmit Buffer within an interrupt service routine, whenever the selected BC Transfer is executed. The BC transfer already has to be defined using the library functions **ApiCmdBCXferDef** and **ApiCmdBCBHDef** when System Dynamic Data generation is enabled.

On RT side the System Dynamic Data insertion is performed in the RT Transmit Subaddress Buffer within an interrupt service routine, whenever the selected Transmit Subaddress is accessed by a Bus Controller Transmit command. The RT Transmit Subaddress already has to be defined using the library functions **ApiCmdRTSACon** and **ApiCmdRTBHDef** when System Dynamic Data generation is enabled.

Input:

`AiUInt8 id`

Identifies the System Dynamic Data generation scenario containing the data generation scheme defined when this function is issued.

Value	Description
1..255	System Dynamic Data identifier global for the module

`AiUInt8 con`

System Dynamic Data Control

Value	Constant	Description
0	**API_DIS**	Disable System Dynamic Data Generation
1	**API_ENA**	Enable System Dynamic Data Generation
2		Reserved
3	**API_ENA_INIT**	Enable System Dynamic Data Generation and initialize System Dynamic Data Word

AiUInt8 mode

BC or RT System Dynamic Data Mode

Value	Constant	Description
1	**API_BC_MODE**	BC System Dynamic Data mode
2	**API_RT_MODE**	RT System Dynamic Data mode

TY_API_SYSTAG *psystag

System Dynamic Data description

```
typedef struct ty_api_systag {
  AiUInt16 xid_rtsa;
  AiUInt16 fct;
  AiUInt16 min;
  AiUInt16 max;
  AiUInt16 step;
  AiUInt16 wpos;
  AiUInt16 bpos;
}
TY_API_SYSTAG;
```

AiUInt16 xid_rtsa

'mode'	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
1 (BC)	XFER_ID							
2 (RT)	RT_ADDR							

'mode'	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1 (BC)	XFER_ID							
2 (RT)	SUBADDR_MODECODE							

XFER_ID

Transfer ID

Note:
See Section 1.3.5 for the range allowed for this parameter.

RT_ADDR

Value	Value
0..31	Remote Terminal Address

SUBADDR_MODECODE

Value	Value
1..30	RT Subaddress
0	Mode code
31	('wpos' determines the Mode code Number)

AiUInt16 fct

System Dynamic Data Word Generation Function

Value	Constant	Description
0	**API_SYSTAG_FCT_DISABLE**	Disable
1	**API_SYSTAG_FCT_POS_RAMP**	Positive Ramp Function
2	**API_SYSTAG_FCT_NEG_RAMP**	Negative Ramp Function
3	**API_SYSTAG_FCT_POS_TRIANGLE**	Positive Triangle Function
4	**API_SYSTAG_FCT_NEG_TRIANGLE**	Negative Triangle Function
5	**API_SYSTAG_FCT_DATASET**	Dynamic Dataset Function
6	**API_SYSTAG_FCT_STATES**	User States Function
7	**API_SYSTAG_FCT_COMP**	Complement Function
8	**API_SYSTAG_FCT_CHECKSUM**	Checksum Function

`AiUInt16 min`

'fct'	Value	Value
1..4	**0..n**	Lower Limit of the System Dynamic Data Word
5	**0..4095**	Dataset Buffer Start ID (refer to function ApiCmdRamWriteDataset)
6	**0..4095**	Dataset Buffer ID (refer to function ApiCmdRamWriteDataset)
7	**0..31**	Word Position to build the complement from
8	**0..31**	Checksum Start Word Position

`AiUInt16 max`

'fct'	Value	Value
1..4	**0..n**	Upper Limit of the System Dynamic Data Word
5	**0..n**	Amount of contiguous Dataset Buffers
6	**1..32**	Amount of contiguous words from specified Dataset Buffer ID
7	**0**	Reserved
8	**0..31**	Checksum End Word Position

`AiUInt16 step`

'fct'	Value	Constant	Value
1..4	**0..n**		Stepsize used to increment or decrement the System Dynamic Data Word
5	**0**	**API_SYSTAG_STEP_CYCLIC**	Cyclic operation
	1	**API_SYSTAG_STEP_KEEP_LAST**	Keep last Buffer of Dataset
6	**0**	**API_SYSTAG_STEP_CYCLIC**	Cyclic operation
	1	**API_SYSTAG_STEP_KEEP_LAST**	Keep last Word of States
7	**1**	**API_SYSTAG_STEP_1S_COMP**	1's Complement
	2	**API_SYSTAG_STEP_2S_COMP**	2's Complement
8	**0**	**API_SYSTAG_STEP_CHECKSUM_PLUS**	Checksum is calculated by adding all data word values from word position 'min' to 'max'

1	**API_SYSTAG_STEP_CHECKSUM_XOR**	Checksum is calculated by xoring all data word values from word position 'min' to 'max'
2	**API_SYSTAG_STEP_CHECKSUM_1760**	1760 Checksum is calculated from all data words preceding the specified Checksum word position 'wpos' ('min' & 'max' are n/a)

`AiUInt16 wpos`

'tag_tct'	Value	Value
1..4, 6, 7, 8	**0..31**	Word Position of the System Dynamic Data Word in the Transmit Buffer
5	**0**	Reserved

`AiUInt16 bpos`

'fct'	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
1..4, 6	BIT_POS							
5, 7, 8	0 (Reserved)							

'fct'	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1..4, 6	BIT_NB							
5, 7, 8	0 (Reserved)							

BIT_POS

Value	Value
0..15	Position of LSB in System Dynamic Data Word

BIT_NB

Value	Value
1..16	Amount of bits in System Dynamic Data Word
0	Identical to 'BIT_NB' = 16

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.27 ApiCmdSysTriggerEdgeInputSet

Prototype:

```
AiReturn ApiCmdSysTriggerEdgeInputSet(AiUInt32 ul_ModuleHandle,
                                      AiUInt8 biu,
                                      AiUInt32 edge_flags);
```

Purpose:

This function can be used to control the edge sensitivity of input trigger lines. All triggers are rising edge sensitive by default. Specifying a 1 for a trigger can be used to invert the logic for falling edge sensitivity.

Note:
This function is not available for all hardware platforms. Please see Appendix B "Functionality Overview" for details.

Note:
This will also affect the behavior of the PXI trigger lines.

Input:

AiUINt32 edge_flags

Bit field containing the BC, RT and BM setting of the current stream.

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
0 (Reserved)							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0 (Reserved)							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
0 (Reserved)							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 (Reserved)					BM_E	RT_E	BC_E

BC_E

Value	Value
0	BC Trigger Input Rising Edge (default)
1	BC Trigger Input Falling Edge

RT_E

Value	Value
0	RT Trigger Input Rising Edge (default)
1	RT Trigger Input Falling Edge

BM_E

Value	Value
0	BM Trigger Input Rising Edge (default)
1	BM Trigger Input Falling Edge

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.28 ApiCmdSysTriggerEdgeInputGet

Prototype:

```
AiReturn ApiCmdSysTriggerEdgeInputGet(AiUInt32 ul_ModuleHandle,
                                      AiUInt8 biu,
                                      AiUInt32 * edge_flags);
```

Purpose:

This function can be used to get the edge sensitivity of input trigger lines.

> **Note:**
> This function is not available for all hardware platforms. Please see Appendix B "Functionality Overview" for details.

> **Note:**
> This will also affect the behavior of the PXI trigger lines.

Input:

None

Output:

`AiUInt32 * edge_flags`

Bit field containing the BC, RT and BM setting of the current stream.

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
0 (Reserved)							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0 (Reserved)							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
0 (Reserved)							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 (Reserved)					BM_E	RT_E	BC_E

BC_E

Value	Value
0	BC Trigger Input Rising Edge (default)
1	BC Trigger Input Falling Edge

RT_E

Value	Value
0	RT Trigger Input Rising Edge (default)
1	RT Trigger Input Falling Edge

BM_E

Value	Value
0	BM Trigger Input Rising Edge (default)
1	BM Trigger Input Falling Edge

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.29 ApiCmdTrackDef

Prototype:

```
AiReturn ApiCmdTrackDef(AiUInt32 ul_ModuleHandle,
                        AiUInt8 biu,
                        AiUInt8 uc_TrackId,
                        AiUInt8 uc_BT,
                        AiUInt16 uw_XidRtSa,
                        AiUInt8 uc_Mode,
                        AiUInt16 uw_TrackStartPos,
                        AiUInt8 uc_TrackBPos,
                        AiUInt8 uc_TrackBLen,
                        AiUInt16 uw_TrackLen,
                        AiUInt16 uw_MultiplexedTrackNb,
                        AiUInt16 uw_ContinousTrackNb,
                        AiUInt32 *pul_TrackBufferStartAddr);
```

Purpose:

This function provides a method for the user to continually store up to 32 words (Track) from pre-defined 1553 Data message transfers received at either the BC or RT. Storage will occur in a Track Multiplex Buffer at the Track Multiplex Buffer Index contained within the 1553 Data message. This function is used to define up to 32 Track Multiplex Buffers and the Track and Track Multiplex Buffer Index length and locations within the 1553 Data message. The Target software will continually monitor the user-defined Xfer ID or RT SA/Mode code for Data messages and when received will use the Track Multiplex Buffer Index contained within the Data message to index into the Track Multiplex Buffer to store the 1553 data. Thus, providing a means to Transfer specific filtered 1553 data transferred over the MILbus.

Input:

`AiUInt8 uc_TrackId`

Value	Description
0..255	Track Multiplex Buffer Identifier

`AiUInt8 uc_BT`

Buffer type

Value	Constant	Description
1	**API_TRACK_TYPE_BC**	BC Message Buffer
2	**API_TRACK_TYPE_RT**	RT Message Buffer
3	**API_TRACK_TYPE_MUX_TRACK**	Multiplexed Track (used for 2nd level multiplexing)

AiUInt16 uw_XidRtSa

'uc_BT'	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
1 (BC)	XFER_ID							
2 (RT)	TR	RT_ADDR						
3 (MUX)	TRACK_ID							

'uc_BT'	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1 (BC)	XFER_ID							
2 (RT)	SUBADDR_MODECODE							
3 (MUX)	TRACK_ID							

XFER_ID

BC Transfer Identifier

Note:
See Section 1.3.5 for the range allowed for this parameter.

TR

Type of the RT subaddress / modecode

Value	Value
0	Receive (Simulation and Mailbox)
1	Transmit (Mailbox only)

RT_ADDR

Value	Value
0..31	Remote Terminal Address

SUBADDR_MODECODE

Value	Value
1..30	RT Subaddress
0, 31	Modecode

TRACK_ID

Value	Value
0..255	Track Multiplex Buffer Identifier

AiUInt8 uc_Mode

Value	Constant	Description
0	**API_DIS**	Delete / Disable Track Definition
1	**API_ENA**	Create Track Definition

AiUInt16 uw_TrackStartPos

Value	Description
0..31	Data buffer word number of the Track starting location

AiUInt8 uc_TrackBPos

Value	Description
0..15	Start bit position of the Track Multiplex Buffer Index in the Track's first word (defined by 'uw_TrackStartPos')

AiUInt8 uc_TrackBLen

Value	Description
1..16	Number of bits used for defining the Track Multiplex Buffer Index in the Track's first word (defined by 'uw_TrackStartPos')

Note:
The number of bits selected (uc_TrackBLen) controls the range of (uc_MultiplexedTrackNb) (as shown in the table at right.)

Range of Number of Multiplexed Tracks	
uc_TrackBLen	uw_MultiplexedTrackNb (max)
1	2
2	4
..	
16	65536

AiUInt16 uw_TrackLen

Value	Description
1..32	Size of the track in words (starting from the word specified in parameter uw_TrackStartPos)

AiUInt16 uw_MultiplexedTrackNb

Value	Description
0	Memory is allocated by calling ApiCmdTrackPreAlloc after this function. **Note:** This could previously be used to let the software allocate the buffer upon receiption of the data. This mode is now obsolete. Please always pre-allocate the buffers necessary.
2..65536	Number of multiplexed tracks in the Track Multiplex Buffer (See Table at right for maximum range) for which memory is already allocated

AiUInt16 uw_ContinousTrackNb

Value	Description
1..32	Number of continuous tracks in the Data buffer. This number defines the number of times the track is repeated in the Data buffer.

Note:
uw_ContinousTrackNb X uw_TrackLen $\leq$ 32

Output:

AiUInt32 *pul_TrackBufferStartAddr

Start Address of the Track Multiplex Buffer located in Local Memory on the Target. This address can be used as an input parameter for the functions ApiReadMemData and ApiWriteMemData. However, the Track Multiplex Buffers should only be read using the function **ApiCmdTrackRead**.

If zero is returned, there was not enough memory available on the Target.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.30 ApiCmdTrackDefEx

Prototype:

```
AiReturn ApiCmdTrackDefEx(AiUInt32 ul_ModuleHandle,
                          AiUInt8 biu,
                          TY_API_TRACK_DEF_IN * px_TrackDefIn,
                          TY_API_TRACK_DEF_OUT *px_TrackDefOut);
```

Purpose:

This function provides a method for the user to continually store up to 32 words (Track) from pre-defined 1553 Data message transfers received at either the BC or RT. Storage will occur in a Track Multiplex Buffer at the Track Multiplex Buffer Index contained within the 1553 Data message. This function is used to define up to 32 Track Multiplex Buffers and the Track and Track Multiplex Buffer Index length and locations within the 1553 Data message. The Target software will continually monitor the user-defined Xfer ID or RT SA/Mode code for Data messages and when received will use the Track Multiplex Buffer Index contained within the Data message to index into the Track Multiplex Buffer to store the 1553 data. Thus, providing a means to Transfer specific filtered 1553 data transferred over the MILbus.

Input:

TY_API_TRACK_DEF_IN * px_TrackDefIn

Track definition structure

```
typedef struct ty_api_track_def_in {
  AiUInt32 ul_TrackId;
  AiUInt32 ul_BT;
  AiUInt32 ul_XidRtSa;
  AiUInt32 ul_Mode;
  AiUInt32 ul_TrackStartPos;
  AiUInt32 ul_TrackBPos;
  AiUInt32 ul_TrackBLen;
  AiUInt32 ul_TrackLen;
  AiUInt32 ul_MultiplexedTrackNb;
  AiUInt32 ul_ContinousTrackNb;
  AiInt32 l_Offset;
}
TY_API_TRACK_DEF_IN;
```

AiUInt32 ul_TrackId

Value	Description
0..255	Track Multiplex Buffer Identifier

AiUInt32 ul_BT

Buffer type

Value	Constant	Description
1	**API_TRACK_TYPE_BC**	BC Message Buffer
2	**API_TRACK_TYPE_RT**	RT Message Buffer
3	**API_TRACK_TYPE_MUX_TRACK**	Multiplexed Track (used for 2nd level multiplexing)

AiUInt32 ul_XidRtSa

'uc_BT'	Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
1 (BC)	Reserved							
2 (RT)	Reserved							
3 (MUX)	Reserved							

'uc_BT'	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
1 (BC)	Reserved							
2 (RT)	Reserved							
3 (MUX)	Reserved							

'uc_BT'	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
1 (BC)	XFER_ID							
2 (RT)	TR	RT_ADDR						
3 (MUX)	TRACK_ID							

'uc_BT'	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1 (BC)	XFER_ID							
2 (RT)	SUBADDR_MODECODE							
3 (MUX)	TRACK_ID							

XFER_ID

BC Transfer Identifier

> **Note:**
> See Section 1.3.5 for the range allowed for this parameter.

TR

Type of the RT subaddress / modecode

Value	Value
0	Receive (Simulation and Mailbox)
1	Transmit (Mailbox only)

RT_ADDR

Value	Value
0..31	Remote Terminal Address

SUBADDR_MODECODE

Value	Value
1..30	RT Subaddress
0, 31	Modecode

TRACK_ID

Value	Value
0..255	Track Multiplex Buffer Identifier

AiUInt32 ul_Mode

Value	Constant	Description
0	**API_DIS**	Delete / Disable Track Definition
1	**API_ENA**	Create Track Definition

AiUInt32 ul_TrackStartPos

Value	Description
0..31	Data buffer word number of the Track starting location

AiUInt32 ul_TrackBPos

Value	Description
0..15	Start bit position of the Track Multiplex Buffer Index in the Track's first word (defined by 'ul_TrackStartPos')

AiUInt32 ul_TrackBLen

Value	Description
1..16	Number of bits used for defining the Track Multiplex Buffer Index in the Track's first word (defined by 'ul_TrackStartPos')

Note:
The number of bits selected (ul_TrackBLen) controls the range of (ul_MultiplexedTrackNb) (as shown in the table at right.)

Range of Number of Multiplexed Tracks	
ul_TrackBLen	ul_MultiplexedTrackNb (maximum range)
1	2
2	4
3	8
4	16
..	..
16	65536

AiUInt32 ul_TrackLen

Value	Description
1..32	Size of the track in words (starting from the word specified in parameter ul_TrackStartPos)

AiUInt16 uw_MultiplexedTrackNb

Value	Description
0	Memory is allocated by calling ApiCmdTrackPreAlloc after this function. **Note:** This could previously be used to let the software allocate the buffer upon receiption of the data. This mode is now obsolete. Please always pre-allocate the buffers necessary.
2..65536	Number of multiplexed tracks in the Track Multiplex Buffer (See Table at right for maximum range) for which memory is already allocated

AiUInt32 ul_ContinousTrackNb

Value	Description
1..32	Number of continuous tracks in the Data buffer. This number defines the number of times the track is repeated in the Data buffer.

Note:
ul_ContinousTrackNb X ul_TrackLen $\leq$ 32

AiInt32 l_Offset

Value	Description
-31..+31	Negative or positive offset in words relative to the calculation start position of the track defined with parameter 'ul_TrackStartPos' With this offset the start position of the data to be copied can be set, so the mux word of the track may be outside of these data words.

Output:

TY_API_TRACK_DEF_OUT x_TrackDefOut

Track definition output structure

```
typedef struct ty_api_track_def_out {
  AiUInt32 ul_TrackBufferStartAddr;
}
TY_API_TRACK_DEF_OUT;
```

AiUInt32 *ul_TrackBufferStartAddr

Start Address of the Track Multiplex Buffer located in Local Memory on the Target. This address can be used as an input parameter for the functions ApiReadMemData and ApiWriteMemData. However, the Track Multiplex Buffers should only be read using the function **ApiCmdTrackRead** / **ApiCmdTrackReadEx**.
If zero is returned, there was not enough memory available on the Target.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.31 ApiCmdTrackPreAlloc

Prototype:

```
AiReturn ApiCmdTrackPreAlloc(AiUInt32 ul_ModuleHandle,
                             AiUInt8 biu,
                             TY_API_TRACK_PREALLOC_IN *px_TrackPreAllocIn,
                             TY_API_TRACK_PREALLOC_OUT *px_TrackPreAllocOut);
```

Purpose:

This function is used to pre-allocate the memory of a list of multiplex states. This function may be used more than once to pre-allocate different multiplex states. This function is only useful, if a track was previously defined in ApiCmdTrackDefEx() with parameter 'ul_MultiplexedTrackNb' set to 0.

Note:
Dynamic memory allocation is obsolete. When using 'ul_MultiplexedTrackNb' set to 0 in ApiCmdTrackDef and ApiCmdTrackDefEx always allocate the buffer with this function afterwards. Multiplex states that are not pre-allocated with this function will not be stored.

Input:

TY_API_TRACK_PREALLOC_IN *px_TrackPreAllocIn

Track pre-allocate structure

```
typedef struct ty_api_track_prealloc_in {
  AiUInt32 ul_TrackId;
  AiUInt32 ul_PreAllocNb;
  AiUInt32 * pul_MuxStates;
}
TY_API_TRACK_PREALLOC_IN;
```

AiUInt32 ul_TrackId

Value	Description
0..255	Track Multiplex Buffer Identifier

AiUInt32 ul_PreAllocNb

Value	Description
0..n	The number of mux states where the memory shall be pre-allocated

AiUInt32 *pul_MuxStates

List of multiplex states where the data buffer memory shall be pre-allocated.

Note:
It is the responsibility of the user application that this list is of the size given with parameter 'ul_PreAllocNb'

Output:

`TY_API_TRACK_PREALLOC_OUT *px_TrackPreAllocOut`

Track pre-allocate output structure

```
typedef struct ty_api_track_prealloc_out {
  AiUInt32 * pul_TrackBufferStartAddr;
}
TY_API_TRACK_PREALLOC_OUT;
```

`AiUInt32 *pul_TrackBufferStartAddr`

Array of start addresses of each of the allocated Track Multiplex Buffers.
If zero is returned, there was not enough memory available on the Target for the Track Multiplex Buffer.

Note:
It is the responsibility of the user application that this list is of the size given with parameter 'ul_PreAllocNb'

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.32 ApiCmdTrackRead

Prototype:

```
AiReturn ApiCmdTrackRead(AiUInt32 ul_ModuleHandle,
                         AiUInt8 biu,
                         AiUInt8 uc_TrackId,
                         AiUInt16 uw_MultiplexedTrackIndex,
                         AiUInt8 uc_ReadMode,
                         AiUInt8 *puc_DataValid,
                         AiUInt16 *puw_TrackDataWords);
```

Purpose:

This function is used to read all or specific tracks from a previously defined Track Multiplex Buffer defined using **ApiCmdTrackDef**/**ApiCmdTrackDefEx**.

Input:

AiUInt8 uc_TrackId

Value	Description
0..255	Track Multiplex Buffer Identifier

AiUInt16 uw_MultiplexedTrackIndex

Value	Description
0..65535	Track Index into the Track Multiplex Buffer to read from

Note:
It is the responsibility of the application to make sure that the Track Index does not exceed the length of the Track Multiplex Buffer (uw_MultiplexedTrackNb) defined in ApiCmdTrackDef.

AiUInt8 uc_ReadMode

Value	Constant	Description
0	**API_TRACK_DONT_CLR_UDF**	Do not clear track update flag
1	**API_TRACK_CLR_UDF**	Clear track update flag after read

Output:

AiUInt8 *puc_DataValid

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
					NOMEM		VALID

NOMEM

If set to 1, the requested multiplex state could not be defined due to low memory

VALID

Value	Constant	Description
0	**API_DATA_NOT_VALID**	Returned data is not valid (e.g. track buffer did not receive anything yet)
1	**API_DATA_VALID**	Data is valid

AiUInt16 *puw_TrackDataWords

Pointer to the Track Multiplex Buffer at the requested Track Index.

Note:
It is the responsibility of the application to make sure that enough memory is provided with this pointer!

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.33 ApiCmdTrackReadEx

Prototype:

```
AiReturn ApiCmdTrackReadEx(AiUInt32 ul_ModuleHandle,
                           AiUInt8 biu,
                           TY_API_TRACK_READ_IN *x_TrackReadIn,
                           TY_API_TRACK_READ_OUT *px_TrackReadOut);
```

Purpose:

This function is used to read all or specific tracks from a previously defined Track Multiplex Buffer defined using **ApiCmdTrackDef**/**ApiCmdTrackDefEx**.

Input:

TY_API_TRACK_READ_IN *x_TrackReadIn

Track read structure

```
typedef struct ty_api_track_read_in {
  AiUInt32 ul_TrackId;
  AiUInt32 ul_MultiplexedTrackIndex;
  AiUInt32 ul_ReadMode;
}
TY_API_TRACK_READ_IN;
```

AiUInt32 ul_TrackId

Value	Description
0..255	Track Multiplex Buffer Identifier

AiUInt32 ul_MultiplexedTrackIndex

Value	Description
0..65535	Track Index into the Track Multiplex Buffer to read from

Note:
It is the responsibility of the application to make sure that the Track Index does not exceed the length of the Track Multiplex Buffer (uw_MultiplexedTrackNb) defined in ApiCmdTrackDef.

AiUInt32 ul_ReadMode

Value	Constant	Description
0	**API_TRACK_DONT_CLR_UDF**	Do not clear track update flag
1	**API_TRACK_CLR_UDF**	Clear track update flag after read

Output:

`TY_API_TRACK_READ_OUT *px_TrackReadOut`

Track read output structure

```
typedef struct ty_api_track_read_out {
  AiUInt32 ul_DataValid;
  AiUInt32 ul_LastTT;
  AiUInt16 * puw_TrackDataWords;
}
TY_API_TRACK_READ_OUT;
```

`AiUInt32 ul_DataValid`

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
					NOMEM		VALID

NOMEM

If set to 1, the requested multiplex state could not be defined due to low memory

VALID

Value	Constant	Description
0	**API_DATA_NOT_VALID**	Returned data is not valid (e.g. track buffer did not receive anything yet)
1	**API_DATA_VALID**	Data is valid

`AiUInt32 ul_LastTT`

The time tag low of the last reception of this multiplex state

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
MINUTES_OF_HOUR (0..59)						SEC_OF_MIN	

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
SEC_OF_MINUTE (0..59)				MICROSEC_OF_SEC			

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
MICROSEC_OF_SEC (0..999999)							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MICROSEC_OF_SEC (0..999999)							

AiUInt16 *puw_TrackDataWords

Pointer to the Track Multiplex Buffer at the requested Track Index.

Note:
It is the responsibility of the application to make sure that enough memory is provided with this pointer!

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.34 ApiCmdTrackScan

Prototype:

```
AiReturn ApiCmdTrackScan(AiUInt32 ul_ModuleHandle,
                         AiUInt8 biu,
                         TY_API_TRACK_SCAN_IN *x_TrackScanIn,
                         TY_API_TRACK_SCAN_OUT *px_TrackScanOut);
```

Purpose:

This function is used to read a list of all currently valid multiplex states.
A multiplex state is valid, if it had been received at least once. It is not valid, if it either was not defined or if **ApiCmdTrackRead**/**ApiCmdTrackReadEx** was called with "**ReadMode**" = **API_TRACKCLR_UDF** for the related mux state, and the state was not received since then.

Input:

TY_API_TRACK_SCAN_IN *x_TrackScanIn

Track scan structure

```
typedef struct ty_api_track_scan_in {
  AiUInt32 ul_TrackId;
  AiUInt32 ul_ChunkNb;
  AiUInt32 ul_ChunkSize;
}
TY_API_TRACK_SCAN_IN;
```

AiUInt32 ul_TrackId

Value	Description
0..255	Track Multiplex Buffer Identifier

AiUInt32 ul_ChunkNb

Value	Description
0..n	The chunk number to read, when more multiplex states are available than wanted to be read. The number of states wanted to be read is given with parameter 'ul_ChunkSize'.

Example:
Available multiplex states = 35

ul_ChunkNb = 0
ul_ChunkSize = 16

→Function returns the first 16 multiplex states (1..16) and ul_MoreData=1

ul_ChunkNb = 1
ul_ChunkSize = 16

→Function returns the second 16 multiplex states (17..32) and ul_MoreData=1

ul_ChunkNb = 2
ul_ChunkSize = 16

→Function returns the third 16 multiplex states (only 33..35 are valid, 36..48 are set to 0xFFFF) and ul_MoreData = 0

AiUInt32 ul_ChunkSize

Value	Description
1..256	The maximum number of multiplex states wanted to be read

Output:

TY_API_TRACK_SCAN_OUT *px_TrackScanOut

Track scan output structure

```
typedef struct ty_api_track_scan_out {
  AiUInt32 ul_NumberOfReturnedStates;
  AiUInt32 ul_MoreData;
  AiUInt32 * pul_ReturnedStates;
}
TY_API_TRACK_SCAN_OUT;
```

AiUInt32 ul_NumberOfReturnedStates

Number of multiplex states returned in 'puw_ReturnedStates'

AiUInt32 ul_MoreData

Set to 1, if more multiplex states are available.

AiUInt32 *pul_ReturnedStates

List of returned multiplex states

Value	Description
0..0xFFFF	Valid Multiplex State
0xFFFFFFFF	Invalid Multiplex State

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.35 ApiCmdWriteDiscretes

Prototype:

```
AiReturn ApiCmdWriteDiscretes(AiUInt32 ul_ModuleHandle,
                              AiUInt32 ul_Mask,
                              AiUInt32 ul_Value);
```

Purpose:

This command is used to write to the discrete outputs.

Note: Since the discretes are programmable, be sure to have setup the discrete outputs with the function ApiCmdInitDiscretes()!

Note: this function is not supported for all boards. Please see Table B.3 – Function Support By Boards With ASP for details

Input:

AiUInt32 ul_Mask

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
Reserved (0)							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
Reserved (0)							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Reserved (0)							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MASK							

This ApiCmdWriteDiscretes() function takes the mask value as a “qualifier” for a read-modify-write instruction to set/reset individual Discretes. First the current Register value will be read,
step2: the mask is used to identify the bit(s) which should be changed,
step3: the(se) bit(s) will be set according to the new Discrete Register value, last step is to write the new value to the Discrete Register.
A “1” marks the discrete output to be written.

AiUInt32 ul_Value

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
Reserved (0)							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
Reserved (0)							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Reserved (0)							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT

OUT

Note:
The bits are only valid if the corresponding discretes are configured as Output with the function ApiCmdInitDiscretes!

Output:

None

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.36 ApiReadAllVersions

Prototype:

```
AiReturn ApiReadAllVersions(AiUInt32 ul_ModuleHandle,
                            AiUInt32 count,
                            TY_VER_INFO versions[],
                            TY_VERSION_OUT * info);
```

Purpose:

This function returns the version numbers of all board software package components for the AIM board.

Input:

AiUInt32 count

The size of the version array and the numer of versions to read. This usually is AI_MAX_VERSIONS.

The versions array needs to be pre-allocated by the application to store count versions structs.

Output:

TY_VER_INFO versions[]

The versions obtained for this board.

TY_VERSION_OUT *info

Additional information on how many entries of the versions aray are valid and how many version the board has.

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.37 ApiReadBSPVersion (Obsolete)

Purpose:

This function is obsolete and might be removed in future BSP versions. Please use ApiReadAllVersions or ApiReadVersion instead.

3.1.38 ApiReadBSPVersionEx (obsolete)

Purpose:

This function is obsolete and might be removed in future BSP versions. Please use function ApiReadAllVersions to read the version numbers of all board software package components for the AIM board.

3.1.39 ApiReadRecData (obsolete)

Prototype:

```
AiReturn ApiReadRecData(AiUInt32 ul_ModuleHandle,
                        AiUInt8 biu,
                        TY_API_BM_REC *api_rec_stat,
                        void *lpBuf,
                        AiUInt32 *lBytesRead);
```

Purpose:

This function copies the 1553 MILBus data recorded by the Bus Monitor while in Recording mode from the Bus Monitor Buffer Global RAM area into an application buffer specified by the user. This application buffer can later be used for replay (see System function **ApiWriteRepData**) or post-analysis.

Input:

`void *lpBuf`

Pointer to application buffer area (used to store recording data)

Output:

`TY_API_BM_REC *api_rec_stat`

Recording status information structure.

```
Typedef struct ty_api_bm_rec {
  AiUInt8 status;
  AiUInt8 padding1;
  AiUInt16 padding2;
  AiUInt32 hfi_cnt;
  AiUInt32 saddr;
  AiUInt32 size;
}
TY_API_BM_REC;
```

`AiUInt8 status`

Bus Monitor Status

Value	Description
1	Bus Monitor halted
2	Bus Monitor busy

`AiUInt8 padding1`

0 (Reserved)

AiUInt16 padding2

0 (Reserved)

AiUInt32 hfi_cnt

Actual value of the Half Buffer Full Interrupt counter (incremented each time a Half Buffer Full Interrupt occurs).

When the hfi_cnt value is incremented, the BM buffer contents have been copied from the Global RAM to the application buffer (see parameter '**lpBuf**').

AiUInt32 saddr

Start Address (Byte-Address) of the AIM board BM recording buffer which contains the recording buffer entries to be copied from the Global RAM to the application buffer area.

AiUInt32 size

Amount of 32-bit BM recording buffer entries that have been copied from the Global RAM area of the AIM board.

> **Note:**
> For a description of a 32-bit BM recording buffer entry see the Programmer's Guide.

AiUInt32 *lBytesRead

Amount of bytes read

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.40 ApiReadVersion

Prototype:

```
AiReturn ApiReadVersion(AiUInt32 ul_ModuleHandle,
                        TY_E_VERSION_ID eId,
                        TY_VER_INFO * pxVerion);
```

Purpose:

This function returns the version number of a software package component for the AIM board.

Input:

`TY_E_VERION_ID eId`

Target software package component from AIM board

Output:

`TY_VER_INFO * pxVerion`

The version obtained for this board.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

3.1.41 ApiWriteRepData

Prototype:

```
AiReturn ApiWriteRepData(AiUInt32 ul_ModuleHandle,
                         AiUInt8 biu,
                         TY_API_REP_STATUS *api_rep_stat,
                         void *lpBuf,
                         AiUInt32 *lBytesWritten);
```

Purpose:

This function copies Bus Monitored recorded entries (previously copied to an application buffer (see System function **ApiReadRecData**)) from an application buffer into the Replay Buffer Global RAM area. The replay of this data can then be started using the **ApiCmdReplayStart** function.

Input/Output:

`TY_API_REP_STATUS *api_rep_stat (INPUT / OUTPUT)`

Replay status information structure.

```
Typedef struct ty_api_rep_status {
  AiUInt8 status;
  AiUInt8 padding1;
  AiUInt16 padding2;
  AiUInt32 rpi_cnt;
  AiUInt32 saddr;
  AiUInt32 size;
  AiUInt32 entry_cnt;
}
TY_API_REP_STATUS;
```

`AiUInt8 status (OUTPUT)`

Replay Status

Value	Constant	Description
0	**API_REP_HALTED**	Replay halted
1	**API_REP_BUSY**	Replay busy

`AiUInt8 padding1`

0 (Reserved)

`AiUInt16 padding2`

0 (Reserved)

AiUInt32 rpi_cnt (OUTPUT)

Actual value of the Half Buffer Transmitted Interrupt counter (incremented each time a High Speed Half Buffer Transmitted Interrupt occurs).

When the **rpi_cnt** value is incremented new Replay data given in parameter '**lpBuf**' has been reloaded to the Global RAM of the AIM board area.

AiUInt32 saddr (INPUT)

Start Address of the AIM board Replay buffer in the Global RAM area to copy the replay buffer entries from the application buffer area (parameter '**lpBuf**') to the Global RAM area of the AIM board.

The value of the start address may be read with the function **ApiCmdReplayStatus**

AiUInt32 size (INPUT)

Amount of Bytes to be copied

Value	Description
1..10000h	Bytes to be copied into LS Replay area

Note:
To write a replay half buffer, the application should always copy a full replay half buffer ('size' set to maximum value), except the last one!

AiUInt32 entry_cnt

0 (reserved)

AiUInt32 *lpbuf (INPUT)

Pointer to application buffer area (replay data)

Output:

AiUInt32 *lBytesWritten

Amount of bytes written

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

4 CALIBRATION FUNCTIONS

Chapter 4 defines the Calibration function calls of the API S/W Library. The Calibration functions provide configuration of the physical MILbus including coupling mode, transmitter amplitude output and data rate (default 1 Mbps). Table 4.1 defines the list and definition of Calibration function calls. The function calls in this table are listed in a functional order, however, the detailed descriptions of the Calibration function calls in the following sections are in alphabetical order.

Table 4.1: Calibration Function Descriptions

Function	Description
ApiCmdCalCplCon	Sets up the physical MILbus coupling mode
ApiCmdCalSigCon	Enables/disables 1 Mhz square wave calibration test signal
ApiCmdCalTransCon	Controls the data rate of MILbus (500 kbps or 1 Mbps)
ApiCmdCalXmtCon	Modifies the output amplitude of the MILbus/test signal

4.1 Low Speed Functions

4.1.1 ApiCmdCalCplCon

Prototype:

```
AiReturn ApiCmdCalCplCon(AiUInt32 ul_ModuleHandle,
                         AiUInt8 biu,
                         AiUInt8 bus,
                         AiUInt8 cpl);
```

Purpose:

This function is used to select the physical MILbus coupling mode for the selected MILBus channel on the AIM board.

Input:

`AiUInt8 bus`

Value	Constant	Description
1	**API_CAL_BUS_PRIMARY**	Primary MILbus
2	**API_CAL_BUS_SECONDARY**	Secondary MILbus

`AiUInt8 cpl`

Note:
Not all coupling modes are available on all AIM devices. Please check the capter “Board functionality overview” for details.

Note:
On boards where API_CAL_CPL_ISOLATED is not available it is internally wrapped to the mode API_CAL_CPL_WRAP_AROUND_LOOP.

Note:
For boards where only external and wrap around is supported the external coupling depends on the configuration of the hardware. Per default, this is normally transformer coupled. For the other boards the default coupling is set by factory and must be explicitly declared when ordered. If not declared, always transformer coupled is the default coupling.

Value	Constant	Description
0	**API_CAL_CPL_ISOLATED**	Isolated, on-board termination
1	**API_CAL_CPL_TRANSFORM**	Transformer coupled MILbus
2	**API_CAL_CPL_DIRECT**	Direct coupled MILbus
3	**API_CAL_CPL_EXTERNAL**	Externally coupled MILbus with on-board transformer-coupled MILbus termination
4	**API_CAL_CPL_WRAP_AROUND_LOOP**	On-board, digital wrap-around loop between MILbus Encoder and Decoder without relying on the external MILbus.

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

4.1.2 ApiCmdCalSigCon

Prototype:

```
AiReturn ApiCmdCalSigCon(AiUInt32 ul_ModuleHandle,
                         AiUInt8 biu,
                         AiUInt8 bus,
                         AiUInt8 con);
```

Purpose:

This function is used to enable/disable a 1 MHz square wave calibration test signal on the selected MILbus of the AIM board. The amplitude of the square wave signal depends on the setting of the appropriate D/A converter controlled by the library function ApiCmdCalXmtCon.

Input:

AiUInt8 bus

Value	Constant	Description
1	**API_CAL_BUS_PRIMARY**	Primary MILbus
2	**API_CAL_BUS_SECONDARY**	Secondary MILbus

AiUInt8 con

Value	Constant	Description
0	**API_DIS**	Disable
1	**API_ENA**	Enable

Output:

none

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

4.1.3 ApiCmdCalTransCon

Prototype:

```
AiReturn ApiCmdCalTransCon(AiUInt32 ul_ModuleHandle,
                           AiUInt8 biu,
                           AiUInt8 trans_mode);
```

Purpose:

This function is used to control the MIL-Bus data transmission rate.

> **Note:**
> The 500 kBit data transmission rate is only provided for special test applications and does not conform to the MIL-STD-1553.

> **Note:**
> This function is not available for all hardware platforms. Please see Appendix B "Functionality Overview" for details.

Input:

AiUInt8 trans_mode

Value	Constant	Description
0	**API_CAL_TRANS_1M**	MIL-Bus Data Transmission Mode (default)
1	**API_CAL_TRANS_500K**	500 kBit Data Transmission Mode

Output:

none

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

4.1.4 ApiCmdCalXmtCon

Prototype:

```
AiReturn ApiCmdCalXmtCon(AiUInt32 ul_ModuleHandle, AiUInt8 biu,
                         AiUInt8 bus, AiUInt8 amp);
```

Purpose:

This function is used to control the transmitter output amplitude of the selected MILBus transceiver on the AIM board via the voltage control pins.

Input:

AiUInt8 bus

Value	Constant	Description
1	**API_CAL_XMT_FULL_RANGE_PRI**	Primary MILbus full range
2	**API_CAL_XMT_FULL_RANGE_SEC**	Secondary MILbus full range
3	**API_CAL_XMT_HIGHRES_RANGE**	Primary and Secondary bus in high resolution range.

Note:
1) Not all devices support a variable amplitude. See Appendix B for limitations.
2) Not all devices support switching the primary and the secondary bus independently.
3) Not all devices support a high resolution variable amplitude range. Please read the hardware manual of you device for details.

AiUInt8 amp

Value	Description
0..255	Register value for Output amplitude control of selected MILbus (0 = 0%, 255 = 100%)

Note:
The relationship between the physical output amplitude and the value given with this parameter is nearly linear. The output amplitude value depends on the adjusted coupling mode.

Output:

none

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

5 BUFFER FUNCTIONS

Chapter 5 defines the Buffer function calls of the API S/W Library. The Buffer functions provide setup and status of the RT/BC global RAM message buffer memory area and the ASP shared RAM dataset buffer area used for low speed and high speed message transfers. Table 5.1 defines the list and definition of Buffer functions. The function calls in this table are listed in a functional order, however, the detailed descriptions of the Calibration function calls in the following sections are in alphabetical order.

Table 5.1: Buffer Function Descriptions

Function	Description
ApiCmdBufDef	Defines the contents of the BC/RT transmit/receive message buffer
ApiCmdBufHostControl	Switches the buffer index to the next buffer index
ApiCmdBufRead	Reads the contents of the BC/RT transmit/receive message buffer
ApiCmdBufWrite	Writes a data word/bits to variable positions of the BC/RT transmit/receive message buffer
ApiCmdBufWriteBlock	Writes a datablock to the BC/RT transmit/receive message buffer
ApiCmdRamWriteDataset	Writes 32-word dataset to ASP Local RAM when in Dynamic Dataset mode
ApiCmdRamReadDataset	Reads 32-word dataset from ASP Local RAM when in Dynamic Dataset mode
ApiReadMemData	Reads a byte/word/longword from AIM board memory bypassing AIM board cmd/ack interface
ApiWriteMemData	Writes a byte/word/longword to AIM board memory bypassing AIM board cmd/ack interface
ApiReadBlockMemData	Reads a datablock from AIM board memory bypassing AIM board cmd/ack interface
ApiWriteBlockMemData	Writes a datablock to AIM board memory bypassing AIM board cmd/ack interface
ApiCmdBufC1760Con	Enables/Disables the generation of MIL-STD-1760 checksum
ApiBHModify	Modifies the BC/RT buffer header on-the-fly

5.1 Low Speed Functions

5.1.1 ApiBHModify

Prototype:

```
AiReturn ApiBHModify(AiUInt32 ul_ModuleHandle,
                     TY_API_BH_MODIFY *px_BHInfo);
```

Purpose:

This function is used to modify the BC/RT buffer header on-the-fly. The buffer header is addressed via the output address that can be achieved from the command **ApiCmdBCBHDef** or **ApiCmdRTBHDef**.

Input:

`TY_API_BH_MODIFY *px_BHInfo`

Buffer header modify description

```
typedef struct ty_api_bh_modify {
  AiUInt32 ul_BHAddr;
  AiUInt32 ul_DataBufferStoreMode;
  AiUInt32 ul_BufferSizeMode;
  AiUInt32 ul_BufferSize;
  AiUInt32 ul_BufferQueueMode;
  AiUInt32 ul_BufferQueueSize;
  AiUInt32 ul_StatusQueueSize;
  AiUInt32 ul_EventQueueSize;
  AiUInt32 ul_CurrentBufferIndex;
}
TY_API_BH_MODIFY;
```

`AiUInt32 ul_BHAddr`

BC/RT buffer header address relative to the start of the Global RAM. This address is achieved via the command ApiCmdBCBHDef for BC buffer header and ApiCmdRTBHDef for RT buffer header.

`AiUInt32 ul_DataBufferStoreMode`

Data Buffer Store Mode
For Receive Operation the following options are valid:

Value	Constant	Description
0	**API_BSM_RX_DISCARD**	Discard error messages from the current Data Buffer
1	**API_BSM_RX_KEEP_CURRENT**	Keep error messages at the current Data Buffer
0xFFFFFFFF	**-**	Do not modify this parameter

For Transmit Operation the following options are valid:

Value	Constant	Description
0	**API_BSM_TX_KEEP_SAME**	Keep the same Data Buffer at transfer error
1	**API_BSM_TX_GO_ON_NEXT**	Go on with the next Data Buffer in the Buffer Queue
0xFFFFFFFF	**-**	Do not modify this parameter

Note:
The data buffer store mode is always evaluated at the ‘Transfer-End’:
‘bsm’ = 0: If a transfer error is detected at the ‘Transfer-End’, the Buffer is not valid!
‘bsm’ = 1: At ‘Transfer-End’ the Buffer is always valid and it does not matter, if a transfer error is detected or not. If an error occurred during a RT-BC or RT-RT transfer, the BC terminates the transfer with the error detection. Thus, the possible remainder of the transfer after the error will not be stored in the buffer.

AiUInt32 ul_BufferSizeMode

0xFFFFFFFF (reserved)

AiUInt32 ul_BufferSize

Buffer size definition A full-size MILBus transfer needs 16 longwords. A maximum buffer size of 31 longwords allows the application software to store application dependant data up to 15 longwords (60 bytes) in the buffer area.

Value	Description
0	Reserved
1..31	Buffer size in longwords
0xFFFFFFFF	Do not modify this parameter

AiUInt32 ul_BufferQueueMode

Buffer Queue Mode

Value	Constant	Description
0	**API_BQM_CYCLIC**	Cyclic data storage
1	**API_BQM_STAY_LAST**	Store once and stop at end of queue (last index) and reuse last buffer
2	**API_BQM_HOST_CONTROLLED**	Do not advance automatically to next buffer (host controlled)
3		Reserved until status queue is supported
0xFFFFFFFF	**-**	Do not modify this parameter

Note:
The Buffer Queue Mode is only processed, if the “Buffer is valid”. The Buffer Queue Modes affect the Current Buffer Index and thus (in the future) the Status- and Event Queue Operation of the BC!

AiUInt32 ul_BufferQueueSize

Buffer Queue size definition (Amount of contiguous Data Buffers)

Value	Constant	Description
0	**API_QUEUE_SIZE_1**	Queue size 1
1	**API_QUEUE_SIZE_2**	Queue size 2
2	**API_QUEUE_SIZE_4**	Queue size 4
3	**API_QUEUE_SIZE_8**	Queue size 8
4	**API_QUEUE_SIZE_16**	Queue size 16
5	**API_QUEUE_SIZE_32**	Queue size 32
6	**API_QUEUE_SIZE_64**	Queue size 64
7	**API_QUEUE_SIZE_128**	Queue size 128
8	**API_QUEUE_SIZE_256**	Queue size 256
0xFFFFFFFF	**-**	Do not modify this parameter

AiUInt32 ul_StatusQueueSize

0xFFFFFFFF (reserved)

AiUInt32 ul_EventQueueSize

0xFFFFFFFF (reserved)

AiUInt32 ul_CurrentBufferIndex

The current buffer index indicates the index of the current data buffer to be used relative to the buffer queue start. The BIU processor increments or clears the current buffer index in accordance with the setting of ul_BufferQueueMode (“buffer is valid”).

Value	Description
0..255	Current Buffer Index (maximum value is restricted with the current buffer queue size)
0xFFFFFFFF	Do not modify this parameter

Output:

none

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

5.1.2 ApiCmdBufC1760Con

Prototype:

```
AiReturn ApiCmdBufC1760Con(AiUInt32 ul_ModuleHandle,
                           AiUInt8 biu,
                           TY_API_C1760_CON *pc1760);
```

Purpose:

This function is used to control the 1760 checksum mode of the API1553-x software. When enabled with the **ApiCmdReset** instruction, this function can be used to enable / disable the 1760 checksum generation (checksum Word and up to 16 Critical Authority checksums) on a selected message buffer in BC and RT mode. Up to 64 different message buffers per MIL-STD-1553 stream can be defined to generate 1760 checksums. Whenever a message buffer which has been enabled for 1760 handling is written by the application (**ApiCmdBufDef** and **ApiCmdBufWrite** instruction) or via dynamic data functions of the software the selected checksums are automatically created.

> **Note:**
> To avoid an error, the 1760 simulation mode generally has to be enabled with the function ApiCmdReset before using this function!

> **Note:**
> It is the responsibility of the application to disable the 1760 mode on a buffer which is no longer used by the application.

> **Note:**
> 1760 checksumms are not supported in combination with ApiCmdBcDytagDef. Please use ApiCmdSystagDef instead.

Input:

none

Output:

`TY_API_C1760_CON *pc1760`

1760 Control description

```
typedef struct ty_api_c1760_con {
  AiUInt8 mode;
  AiUInt16 buf_id;
  AiUInt8 c01;
  AiUInt8 c02[16];
}
TY_API_C1760_CON;
```

`AiUInt8 mode`

1760 mode for the selected message buffer

Value	Constant	Description
0	**API_DIS**	disable
1	**API_ENA**	enable

`AiUInt16 buf_id`

Buffer identifier (absolute: 0..2047) of message buffer used to apply the 1760 Checksum generation algorithms

`AiUInt8 c01`

1760 Checksum Word position. The checksum word is generated using the words in position 1 to C01 – 1.

Value	Constant	Description
0	**API_DIS**	disable
1..32		Wordposition in selected message buffer

`AiUInt8 c02[]`

1760 Critical Authority Word. The Critical Authority Word is calculated from the word before C02.

Value	Constant	Description
0	**API_DIS**	disable
2..32		Wordposition in selected message buffer

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

5.1.3 ApiCmdBufDef

Prototype:

```
AiReturn ApiCmdBufDef(AiUInt32 ul_ModuleHandle,
                      AiUInt8 biu, AiUInt8 bt,
                      AiUInt16 hid,
                      AiUInt16 bid,
                      AiUInt8 len,
                      AiUInt16 *data,
                      AiUInt16 *rid, AiUInt32 *raddr);
```

Purpose:

This function defines the data buffer contents for transmit or receive messages in BC and RT mode.

Input:

AiUInt8 bt

Buffer type

Value	Constant	Description
1	**API_BUF_BC_MSG**	BC Message Buffer
2	**API_BUF_RT_MSG**	RT Message Buffer

AiUInt16 hid

Buffer Header ID

Note:
See Section 1.3.5 for the range allowed for this parameter.

AiUInt16 bid

'hid'	Value	Constant	Description
0	**1..n**		Write Buffer Data Words to absolute Data Buffer Index **Note:** See section 1.3.5 for the range allowed for this parameter.
>0	**0**	**API_BUF_WRITE_TO_CURRENT**	Write to current Data Buffer Index
	>0		Write Buffer Data Words to Data Buffer Index relative to the Data Buffer Queue Base Index of the specified Buffer Header Identifier.

AiUInt8 len

Value	Description
1..32	Amount of Data Buffer Words to write

AiUInt16 *data

Buffer Data Words to write

Output:

AiUInt16 *rid

Value	Description
1..2047	Returning absolute Data Buffer Index

AiUInt32 *raddr

Corresponding Data Buffer Address

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

5.1.4 ApiCmdBufHostControl

Prototype:

```
AiReturn ApiCmdBufHostControl(AiUInt32 ul_ModuleHandle,
                              AiUInt8 biu,
                              AiUInt8 bt,
                              AiUInt16 hid,
                              AiUInt32 con);
```

Purpose:

This function is used to switch the buffer index to the next buffer index.

Input:

`AiUInt8 bt`

Buffer type

Value	Constant	Description
1	**API_BUF_BC_MSG**	BC Message Buffer
2	**API_BUF_RT_MSG**	RT Message Buffer

`AiUInt16 hid`

Buffer Header ID

> **Note:**
> See Section 1.3.5 for the range allowed for this parameter.

`AiUInt32 con`

Value	Description
API_BUF_NEXT_BUFFER	Switch to next buffer

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

5.1.5 ApiCmdBufRead

Prototype:

```
AiReturn ApiCmdBufRead(AiUInt32 ul_ModuleHandle,
                       AiUInt8 biu,
                       AiUInt8 bt,
                       AiUInt16 hid,
                       AiUInt16 bid,
                       AiUInt8 len,
                       AiUInt16 *data,
                       AiUInt16 *rid, AiUInt32 *raddr);
```

Purpose:

This function reads the data buffer contents for transmit or receive messages in BC and RT mode.

Input:

`AiUInt8 bt`

Buffer type

Value	Constant	Description
1	**API_BUF_BC_MSG**	BC Message Buffer
2	**API_BUF_RT_MSG**	RT Message Buffer

`AiUInt16 hid`

Buffer Header ID

Note:
See Section 1.3.5 for the range allowed for this parameter.

`AiUInt16 bid`

'hid'	Value	Constant	Description
0	**1..n**		Read Buffer Data Words from absolute Data Buffer Index **Note:** See section 1.3.5 for the range allowed for this parameter.
>0	**0**	**API_BUF_READ_FROM_CURRENT**	Read from current Data Buffer Index
	>0		Read Buffer Data Words from Data Buffer Index relative to the Data Buffer Queue Base Index of the specified Buffer Header Identifier.
	0xffff	**API_BUF_READ_FROM_LAST**	Read from last Data Buffer Index that received data

AiUInt8 len

Value	Description
1..32	Amount of Data Buffer Words to read

AiUInt16 *data

Buffer Data Words to read

Output:

AiUInt16 *rid

Value	Description
1..2047	Returning absolute Data Buffer Index corresponding to parameter ‘bid’

AiUInt32 *raddr

Corresponding Data Buffer Address to parameter ‘rid’

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

5.1.6 ApiCmdBufWrite

Prototype:

```
AiReturn ApiCmdBufWrite(AiUInt32 ul_ModuleHandle,
                        AiUInt8 biu,
                        AiUInt8 bt,
                        AiUInt16 hid,
                        AiUInt16 bid,
                        AiUInt8 data_pos,
                        AiUInt8 bit_pos,
                        AiUInt8 bit_len,
                        AiUInt16 data);
```

Purpose:

This function is used to write a data word/bits to variable positions of the specified BC/RT transmit/receive message buffer.

Input:

`AiUInt8 bt`

Buffer type

Value	Constant	Description
1	**API_BUF_BC_MSG**	BC Message Buffer
2	**API_BUF_RT_MSG**	RT Message Buffer

`AiUInt16 hid`

Buffer Header ID

> **Note:**
> See Section 1.3.5 for the range allowed for this parameter.

`AiUInt16 bid`

'hid'	Value	Constant	Description
0	**1..n**		Write Buffer Data Words to absolute Data Buffer Index **Note:** See section 1.3.5 for the range allowed for this parameter.
>0	**0**	**API_BUF_WRITE_TO_CURRENT**	Write to current Data Buffer Index
	>0		Write Buffer Data Words to Data Buffer Index relative to the Data Buffer Queue Base Index of the specified Buffer Header Identifier.

`AiUInt8 data_pos`

Value	Description
1..32	Data Word position in Buffer to write

`AiUInt8 bit_pos`

Value	Description
0..15	Bit position in Data Word

`AiUInt8 bit_len`

Value	Description
1..16	Amount of Bits to write

`AiUInt16 data`

Data Word to write

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

5.1.7 ApiCmdBufWriteBlock

Prototype:

```
AiReturn ApiCmdBufWriteBlock(AiUInt32 ul_ModuleHandle,
                             AiUInt8 biu,
                             AiUInt8 bt,
                             AiUInt16 hid,
                             AiUInt16 bid,
                             AiUInt8 offset,
                             AiUInt8 len,
                             AiUInt16 *data);
```

Purpose:

This function is used to write a datablock to the specified BC/RT transmit/receive message buffer.

Input:

`AiUInt8 bt`

Buffer type

Value	Constant	Description
1	**API_BUF_BC_MSG**	BC Message Buffer
2	**API_BUF_RT_MSG**	RT Message Buffer

`AiUInt16 hid`

Buffer Header ID

> **Note:**
> See Section 1.3.5 for the range allowed for this parameter.

`AiUInt16 bid`

'hid'	Value	Constant	Description
0	**1..n**		Write Buffer Data Words to absolute Data Buffer Index. **Note:** See section 1.3.5 for the range allowed for this parameter.
>0	**0**	**API_BUF_WRITE_TO_CURRENT**	Write to current Data Buffer Index.
	1..x		Write Buffer Data Words to Data Buffer Index relative to the Data Buffer Queue Base Index of the specified Buffer Header Identifier. 1 is the first buffer and x is the last buffer in the Buffer Queue.
	0xFFFF	**API_BUF_WRITE_TO_NEXT**	Write Buffer Data Words to next Data Buffer Index.

`AiUInt8 offset`

Value	Description
0..31	Buffer Offset for writing

`AiUInt8 len`

Value	Description
1..32	Amount of words to write

`AiUInt16 *data`

Data block to write

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

5.1.8 ApiCmdRamReadDataset

Prototype:

```
AiReturn ApiCmdRamReadDataset(AiUInt32 ul_ModuleHandle,
                              AiUInt16 dsid,
                              AiUInt16 *data);
```

Purpose:

This function is used to read the 32-word dataset buffer contents of the specified Dataset Buffer ID from ASP Local RAM when in Dynamic Dataset mode. For BC and RT simulations the Dynamic Dataset mode is enabled using library function **ApiCmdSystagDef**.

Input:

`AiUInt16 dsid`

Value	Description
0..4095	Dataset Buffer Identifier

Output:

`AiUInt16 *data`

32 Dataset Buffer Words to read

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

5.1.9 ApiCmdRamWriteDataset

Prototype:

```
AiReturn ApiCmdRamWriteDataset(AiUInt32 ul_ModuleHandle,
                               AiUInt16 dsid,
                               AiUInt16 *data);
```

Purpose:

This function is used to write a 32-word dataset to ASP Local RAM when in Dynamic Dataset mode. For BC and RT simulations the Dynamic Dataset mode is enabled using library function **ApiCmdSystagDef**.

Input:

`AiUInt16 dsid`

Value	Description
0..4095	Dataset Buffer Identifier

`AiUInt16 *data`

32 Dataset Buffer Words to write

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

5.1.10 ApiReadBlockMemData

Prototype:

```
AiReturn ApiReadBlockMemData(AiUInt32 ul_ModuleHandle,
                             AiUInt8 memtype,
                             AiUInt32 offset,
                             AiUInt8 width,
                             void* data_p,
                             AiUInt32 size,
                             AiUInt32 *pul_BytesRead);
```

Purpose:

This function is used to read a data block from AIM board memory in avoidance of AIM board command and acknowledge interface access. This is necessary to access the board memories in case of interrupt. The function does a direct access to the AIM board memory at the specified offset address .

Input:

AiUInt8 memtype

Memory Type to be accessed

Value	Constant	Description
0	**API_MEMTYPE_GLOBAL**	Global Memory access
1	**API_MEMTYPE_SHARED**	Shared Memory access
2	**API_MEMTYPE_LOCAL**	Local Memory access
3	**API_MEMTYPE_IO**	IO/Galileo Memory access
4	**API_MEMTYPE_GLOBAL_DIRECT**	Direct access to Global Memory even if there is a mirror
5	**API_MEMTYPE_GLOBAL_EXTENSION**	Global Memory Extension access

Note:
not all memory types are available for all boards.

AiUInt32 offset

Byte offset address relative to the start of a specific onboard memory described in parameter ‘memtype’.

AiUInt8 width

Data width of access

Value	synonym	Description
1	**sizeof(AiUInt8)**	Byte data access
2	**sizeof(AiUInt16)**	Word data access
4	**sizeof(AiUInt32)**	Longword data access

AiUInt32 size

Amount of data block elements

Output:

void *data_p

Data to read to. This pointer should match to the data size given in the parameter width.

> **Note:**
> Be sure to have allocated enough memory referenced by this pointer within the application!

Void *pul_BytesRead

Amount of bytes actually read.

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

5.1.11 ApiReadMemData

Prototype:

```
AiReturn ApiReadMemData(AiUInt32 ul_ModuleHandle,
                        AiUInt8 memtype,
                        AiUInt32 offset,
                        AiUInt8 width,
                        void* data_p);
```

Purpose:

This function is used to read a byte/word/longword data value from AIM board memory in avoidance of AIM board command and acknowledge interface access. This is necessary to access the board memories in case of interrupt. The function does a direct access to the AIM board memory at the specified offset address .

Input:

AiUInt8 memtype

Memory Type to be accessed

Value	Constant	Description
0	**API_MEMTYPE_GLOBAL**	Global Memory access
1	**API_MEMTYPE_SHARED**	Shared Memory access
2	**API_MEMTYPE_LOCAL**	Local Memory access
3	**API_MEMTYPE_IO**	IO/Galileo Memory access
4	**API_MEMTYPE_GLOBAL_DIRECT**	Direct access to Global Memory even if there is a mirror
5	**API_MEMTYPE_GLOBAL_EXTENSION**	Global Memory Extension access

Note:
not all memory types are available for all boards.

AiUInt32 offset

Byte offset address relative to the start of a specific onboard memory described in parameter 'memtype'.

AiUInt8 width

Data width of access

Value	synonym	Description
1	**sizeof(AiUInt8)**	Byte data access
2	**sizeof(AiUInt16)**	Word data access
4	**sizeof(AiUInt32)**	Longword data access

Output:

`void *data_p`

Data to read to. This pointer should match to the data size given in the parameter width.

> **Note:**
> Be sure to have allocated enough memory referenced by this pointer within the application!

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

5.1.12 ApiWriteBlockMemData

Prototype:

```
AiReturn ApiWriteBlockMemData(AiUInt32 ul_ModuleHandle,
                              AiUInt8 memtype,
                              AiUInt32 offset,
                              AiUInt8 width,
                              void* data_p,
                              AiUInt32 size,
                              AiUInt32 *pul_BytesWritten);
```

Purpose:

This function is used to write a data block to AIM board memory in avoidance of AIM board command and acknowledge interface access. This is necessary to access the board memories in case of interrupt. The function does a direct access to the AIM board memory at the specified offset address .

Input:

AiUInt8 memtype - Memory Type to be accessed

Value	Constant	Description
0	**API_MEMTYPE_GLOBAL**	Global Memory access
1	**API_MEMTYPE_SHARED**	Shared Memory access
2	**API_MEMTYPE_LOCAL**	Local Memory access
3	**API_MEMTYPE_IO**	IO/Galileo Memory access
4	**API_MEMTYPE_GLOBAL_DIRECT**	Direct access to Global Memory even if there is a mirror
5	**API_MEMTYPE_GLOBAL_EXTENSION**	Global Memory Extension access

Note:
not all memory types are available for all boards.

AiUInt32 offset

Byte offset address relative to the start of a specific onboard memory described in parameter 'memtype'.

AiUInt8 width

Data width of access

Value	synonym	Description
1	**sizeof(AiUInt8)**	Byte data access
2	**sizeof(AiUInt16)**	Word data access
4	**sizeof(AiUInt32)**	Longword data access

`AiUInt32 size`

Amount of data block elements

`void *data_p`

Data to write from. This pointer should match the data size given in the parameter width.

> **Note:**
> Be sure to have allocated enough memory referenced by this pointer within the application!

Output:

`void *pul_BytesWritten`

Amount of bytes actually written.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

5.1.13 ApiWriteMemData

Prototype:

```
AiReturn ApiWriteMemData(AiUInt32 ul_ModuleHandle,
                         AiUInt8 memtype,
                         AiUInt32 offset,
                         AiUInt8 width,
                         void* data_p);
```

Purpose:

This function is used to write a data value to AIM board memory in avoidance of AIM board command and acknowledge interface access. This is necessary to access the board memories in case of interrupt. The function does a direct access to the AIM board memory at the specified offset address .

Input:

AiUInt8 memtype

Memory Type to be accessed

Value	Constant	Description
0	**API_MEMTYPE_GLOBAL**	Global Memory access
1	**API_MEMTYPE_SHARED**	Shared Memory access
2	**API_MEMTYPE_LOCAL**	Local Memory access
3	**API_MEMTYPE_IO**	IO/Galileo Memory access
4	**API_MEMTYPE_GLOBAL_DIRECT**	Direct access to Global Memory even if there is a mirror
5	**API_MEMTYPE_GLOBAL_EXTENSION**	Global Memory Extension access

> **Note:**
> not all memory types are available for all boards.

AiUInt32 offset

Byte offset address relative to the start of a specific onboard memory described in parameter ‘memtype’.

AiUInt8 width

Data width of access

Value	synonym	Description
1	**sizeof(AiUInt8)**	Byte data access
2	**sizeof(AiUInt16)**	Word data access
4	**sizeof(AiUInt32)**	Longword data access

`void *data_p`

Data to write from. This pointer should match to the data size given in the parameter width.

> **Note:**
> Memory must be allocated by the applicaton.

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

6 FIFO FUNCTIONS

Chapter 6 defines the FIFO function calls of the API S/W Library. The FIFO functions provide setup and status for FIFO buffers used for 1553 (LS) message transfers. Table 6.1 defines the list and definition of FIFO functions. The function calls in this table are listed in a functional order, however, the detailed descriptions of the FIFO function calls in the following sections are in alphabetical order.

Table 6.1: FIFO Function Descriptions

Function	Description
ApiCmdFifoIni	Initializes up to 32 FIFOs, each with up to 128 32-word buffers, in shared ASP Local RAM for 1553 transfers
ApiCmdFifoWrite	Loads/reloads buffers of a FIFO with data
ApiCmdFifoReadStatus	Reads the status of the number of 16-bit words in FIFO to reload
ApiCmdBCAssignFifo	Links a BC transfer to a FIFO. The FIFO becomes the source of the BC message buffer data transmitted.
ApiCmdRTSAAssignFifo	Links an RT data transmission to a FIFO. The FIFO becomes the source of the RT message buffer data transmitted.

6.1 ApiCmdBCAssignFifo

Prototype:

```
AiReturn ApiCmdBCAssignFifo(AiUInt32 ul_ModuleHandle,
                            AiUInt8 biu,
                            AiUInt8 con,
                            AiUInt8 f_id, AiUInt16 xid);
```

Purpose:

This function is used to assign a BC-RT transfer type or BC Broadcast transfer type to a specified FIFO in the FIFO pool. The FIFO becomes the source of the BC message buffer data transmitted. The FIFO pool must already be defined using the library function **ApiCmdFifoIni**. FIFO data insertion into the BC Transmit Data Buffer is performed in within an internal interrupt service routine, whenever the selected BC Transfer is executed.

Input:

`AiUInt8 con`

BC FIFO Control

Value	Constant	Description
0	**API_DIS**	Disable
1	**API_ENA**	Enable

`AiUInt8 f_id`

Value	Description
1..32	FIFO Id

`AiUInt16 xid`

BC Transfer ID

Note:
See Section 1.3.5 for the range allowed for this parameter.

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

6.2 ApiCmdFifoIni

Prototype:

```
AiReturn ApiCmdFifoIni(AiUInt32 ul_ModuleHandle,
                       AiUInt8 biu,
                       AiUInt8 fifo_nbr,
                       AiUInt16 buf_nbr);
```

Purpose:

This function allocates and initializes up to 32 FIFOs in the shared ASP Local RAM Area. Each FIFO can consist of a minimum of 2 and a maximum of 128 32-word buffers. Once initialized, the FIFO can become the source of either RT and/or BC message buffer data transmission by using either the **ApiCmdBCAssignFifo** and/or the **ApiCmdRTSAAssignFifo** functions.

Input:

AiUInt8 fifo_nbr

Value	Description
1..32	Number of FIFOs in Buffer pool

AiUInt16 buf_nbr

Value	Description
2..128	Number of Buffer's in one FIFO

Output:

none

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

6.3 ApiCmdFifoReadStatus

Prototype:

```
AiReturn ApiCmdFifoReadStatus(AiUInt32 ul_ModuleHandle,
                              AiUInt8 biu,
                              AiUInt8 f_id,
                              AiUInt16 *status);
```

Purpose:

This function is used to read the status of the transmitted buffer's of the specified FIFO. The **status** value indicates the number of 16-bit words to reload. This function may be used to reload the buffer's of the FIFOs (see **ApiCmdFifoWrite** function).

Input:

AiUInt8 f_id

Value	Description
1..32	FIFO Identifier

Output:

AiUInt16 *status

Status of transmitted Buffer's

Value	Description
0	Half buffer not yet sent
>0	Amount of 16-bit word's to reload into the free half buffer
0xFFFF	Reload under run

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

6.4 ApiCmdFifoWrite

Prototype:

```
AiReturn ApiCmdFifoWrite(AiUInt32 ul_ModuleHandle,
                         AiUInt8 biu,
                         AiUInt8 f_id,
                         AiUInt16 size, AiUInt16 *data);
```

Purpose:

This function is used to load/reload the buffers of the specified FIFO ID. If half of the FIFO Buffers have been transmitted, the FIFO buffers should be reloaded. A single **ApiCmdFifoWrite** function call can only load/reload half of the buf_nbr specified with the **ApiCmdFifoIni** function. To load/reload the buffers in the FIFO two function calls are necessary. First, use the **status** value returned with the **ApiCmdFifoReadStatus** function to determine the number of 16-bit words to reload for a specified FIFO ID. Second, use this **status** value as the size input parameter for the **ApiCmdFifoWrite** function for the specified FIFO ID.

Input:

`AiUInt8 f_id`

Value	Description
1..32	FIFO Identifier

`AiUInt16 size`

Amount of Data Word's to reload

`AiUInt16 *data`

Data to write

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

6.5 ApiCmdRTSAAssignFifo

Prototype:

```
AiReturn ApiCmdRTSAAssignFifo(AiUInt32 ul_ModuleHandle,
                              AiUInt8 biu,
                              AiUInt8 con,
                              AiUInt8 f_id,
                              AiUInt8 rt,
                              AiUInt8 sa);
```

Purpose:

This function is used to assign an RT Transmit Subaddress to a specified FIFO in the FIFO pool. The FIFO becomes the source of the RT Transmit Subaddress message buffer data transmitted. The FIFO pool has to be already defined using the library function **ApiCmdFifoIni**. FIFO data insertion into the RT Transmit Subaddress Message Buffer is performed within an internal interrupt service routine, whenever the selected Transmit Subaddress is accessed by a Bus Controller transmit command. The RT Transmit Subaddress must already be enabled when this command is applied.

Input:

`AiUInt8 con`

Value	Constant	Description
0	**API_DIS**	Disable
1	**API_ENA**	Enable

`AiUInt8 f_id`

Value	Description
1..32	FIFO Id

`AiUInt8 rt`

Value	Description
0..31	Remote Terminal Address

`AiUInt8 sa`

Value	Description
1..30	RT Subaddress

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7 BUS CONTROLLER FUNCTIONS

Chapter 7 defines the Bus Controller function calls of the API S/W Library. The BC functions provide definition of 1553 transfers within the minor frame(s) and setup of the minor frame(s) within the major frame(s) including definition of minor frame timing. The BC functions also provide definition BC transfer properties and real-time BC transfer control including insertion of acyclic messages. The function calls in this table are listed in a functional order, however, the detailed descriptions of the BC function calls in the following sections are in alphabetical order.

Table 7.1: Bus Controller Function Descriptions

Function	Description
ApiCmdBCAcycPrep	Defines the properties of the acyclic "on-the-fly" BC transfers to be inserted into the BC framing sequence
ApiCmdBCAcycPrepAndSendTransferBlocking	Defines a single transfer, sends it as acyclic, blocks until the transfer status is updated and returns the data buffer.
ApiCmdBCAcycSend	Starts the insertion of the acyclic transfers into the BC framing sequence "on-the-fly" or at a pre-defined time
ApiCmdBCBHDef	Defines a BC Buffer Header ID, Buffer Queue size, Queue mode & error protocol
ApiCmdBCBHRead	Read the Data Buffer ID, Buffer Queue size, Queue mode, and error protocol of a BC Buffer Header ID
ApiCmdBCDytagDef	Defines the generation of dynamic data words for BC transmissions
ApiCmdBCFrameDef	Defines the sequence of 1553 transfers within a minor frame with options for inserting delays, strobe pulse outputs, and skip transfer instructions
ApiCmdBCGetDytagDef	Read Dytag settings for the generation of dynamic data words for BC-RT transfer type or for BC broadcast transfer type
ApiCmdBCGetMajorFrameDefinition	Read the sequence of Minor Frames within the Major Frame
ApiCmdBCGetMinorFrameDefinition	Read the sequence of Bus Controller Transfers within a Minor Frame sequence
ApiCmdBCGetXferBufferHeaderInfo	Get the buffer header id of given transfer
ApiCmdBCGetXferDef	Get all transfer properties of a Bus Controller Transfer
ApiCmdBCHalt	Stops BC transfers
ApiCmdBCIni	Initializes the BC with information controlling # of retries and bus switching
ApiCmdBCInstrTblGen	Provides an alternate method of defining minor and major frame sequences
ApiCmdBCInstrTblGetAddrFromLabel	Obtains the address of a BC Instruction Table entry pre-defined by the user using the **ApiCmdBCInstrTblGen** function
ApiCmdBCInstrTblIni	Initializes the memory area associated with creating a BC Instruction Table for major and minor frame sequencing

Function	Description
ApiCmdBCMFrameDef	Defines the sequence of minor fames within the major frame
ApiCmdBCSrvReqVecCon	Set the sub address where the modecode “Last Vector Word” is sent to in case of a service request handling
ApiCmdBCSrvReqVecStatus	Read BC Service Request and Vector Word Status information maintained by the BC for a specific RT
ApiCmdBCStart	Starts the execution of pre-defined BC transfers within minor/major frame structure and defines minor fame timing
ApiCmdBCStatusRead	Reads execution status of the BC
ApiCmdBCTrafficOverrideControl	Enable/disable the bus idle check for BC transmissions
ApiCmdBCTrafficOverrideGet	Check if the bus idle check for BC transmissions is enabled
ApiCmdBCXferCtrl	Enables/Disables the BC Transfer
ApiCmdBCXferDef	Defines all transfer properties including source/destination information, error insertion and interrupt generation
ApiCmdBCXferDefErr	Defines a transfer error injection on the fly.
ApiCmdBCXferDescGet	Get a transfer descriptor value
ApiCmdBCXferDescMod	Modify a transfer descriptor value
ApiCmdBCXferRead	Reads status of an individual BC transfer

7.1 Low Speed Functions

7.1.1 ApiCmdBCAcycPrep

Prototype:

```
AiReturn ApiCmdBCAcycPrep(AiUInt32 ul_ModuleHandle,
                          AiUInt8 biu,
                          TY_API_BC_ACYC *pacyc);
```

Purpose:

This function is used to define the content of the acyclic 'on-the-fly' minor frame to be inserted into the normal BC framing sequence. It is similar to the **ApiCmdBCFrameDef** function. Transmission of the specified acyclic minor frame is started using the **ApiCmdBCAcycSend** function.

Input:

TY_API_BC_ACYC *pacyc

BC Acyclic Frame description

```
typedef struct ty_api_bc_acyc {
  AiUInt8 cnt;
  AiUInt8 padding1;
  AiUInt16 instr[MAX_API_BC_XACYC];
  AiUInt16 xid[MAX_API_BC_XACYC];
}
TY_API_BC_ACYC;
```

AiUInt8 cnt

Value	Description
1..127	Amount of instructions in Acyclic Minor Frame

AiUInt8 padding1

Reserved (0)

`AiUInt16 instr[]`

Value	Constant	Description
1	**API_BC_INSTR_TRANSFER**	**BC Transfer Instruction** →Used for normal transfers
2	**API_BC_INSTR_SKIP**	**BC Skip Instruction** The skip instruction is a relative, unconditional branch forward. This means, that the instruction execution is continued n instruction after the location of the skip instruction, whereby n is equal to the value set in 'xid[]'. This instruction is also intended to be used as a 'No Operation' instruction with interrupt capability (with "Skip Count=1" and interrupt enabled in 'xid[]') **Note:** This instruction may be used to implement larger delays, if using this instruction alone in a minor frame!
4	**API_BC_INSTR_STROBE**	**BC external Strobe Instruction** If this instruction is executed, the BC trigger output of the board is pulsed. After this operation, the instruction execution continues with the following instruction.

`AiUInt16 xid[]`

'instr'	Bit-Pos	Value	Description
1	**15..0**	**1 .. n**	BC Transfer Identifier See Section 1.3.5 for the range allowed for this parameter.
2	**15**	**0**	Disable Interrupt
		1	Enable Interrupt
	7..0	**1..127**	Skip Count Number of instructions which shall be skipped, including the current instruction.
4	**-**	**0**	Reserved

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.2 ApiCmdBCAcycPrepAndSendTransferBlocking

Prototype:

```
AiReturn ApiCmdBCAcycPrepAndSendTransferBlocking
                              (AiUInt32 ul_ModuleHandle,
                               AiUInt8 biu,
                               TY_API_BC_XFER * xfer,
                               AiUInt16 data[32],
                               TY_API_BC_XFER_DSP * transfer_status);
```

Purpose:

This function is a combination of BC transfer definition, acyclic transfer preparation and transmission and a BC transfer status read. This function can be used to improve the performance of sending a single acyclic RT to BC or BC to RT transfer on a ASC1553 device.

The following functions are called:

- ApiCmdBCXferDef with “xfer” as input
- ApiCmdBufDef with “data[32]” and “xfer.hid” as input.
- ApiCmdBCAcycPrep with “xfer.xid” as a single transfer.
- ApiCmdBCXferRead is called to clear the xfer status with clr=0x7
- ApiCmdBCAcycSend with mode=API_BC_ACYC_SEND_IMMEDIATELY
- ApiCmdBCXferRead is called until BUF_STAT != API_BUF_NOT_USED or timeout reached.
- ApiCmdBufRead with “xfer.hid” as input and “data[32]” as output

The timeout for the transfer is 1ms. If the timeout is reached before the transfer status is updated the function returns API_ERR_TIMEOUT.

Prerequisites:

- ApiCmdBCBHDef must be called once before this function to create the buffer header “xfer.hid” and link it to a free buffer id. The buffer header must be created with qsize=API_QUEUE_SIZE_1.

Input:

`TY_API_BC_XFER * xfer`

BC Transfer description. Only BC to RT and RT to BC transfer are allowed.

See ApiCmdBCXferDef for details.

AiUInt8 type

Value	Constant	Description
1	**API_BC_TYPE_BCRT**	BC to RT transfer type
2	**API_BC_TYPE_RTBC**	RT to BC transfer type

AiUInt16 data[32]

This data is copied into the BC buffer before the transfer is started. 32 data words must be provided. The "xfer.wcnt" field can be used to send less data with the actual transfer.

Output:

AiUInt16 data[32]

This data is copied from the BC buffer after the transfer is finished. 32 data words are returned. Depending on the "xfer.wcnt" field not all buffer words are valid.

This data is not valid if the function returned a timeout or any other error.

TY_API_BC_XFER_DSP * transfer_status

BC transfer status at the end of the transfer. See ApiCmdBCXferRead for details.

This status is not valid if the function returned a timeout or any other error.

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.3 ApiCmdBCAcycSend

Prototype:

```
AiReturn ApiCmdBCAcycSend(AiUInt32 ul_ModuleHandle,
                          AiUInt8 biu, AiUInt8 mode,
                          AiUInt32 timetag_high,
                          AiUInt32 timetag_low);
```

Purpose:

This function is used to start insertion of the specified acyclic minor frame into the normal BC framing sequence. The acyclic minor frame content should first be defined using the **ApiCmdBCAcycPrep** function.

Input:

`AiUInt8 mode`

Acyclic send mode

Value	Constant	Description
0	**API_BC_ACYC_SEND_IMMEDIATELY**	Send acyclic frame immediately
1	**API_BC_ACYC_SEND_ON_TIMETAG**	Send acyclic frame when the board time reaches the timetag given in parameters 'timetag_high' and 'timetag_low'
2	**API_BC_ACYC_SEND_AT_END_OF_FRAME**	Send acyclic frame between end of normal minor frame and start of next normal minor frame

Note:
For mode 1 only: Mode 1 is only available on APE boards. See chapter "Board functionality overview" for details.

Note:
For mode 2 only: If normal frame execution waits for an external trigger event, the acyclic frame is sent until the trigger event gets valid.

Note:
For mode 2 only: Mode 2 is not available on multi channel boards.

`AiUInt32 timetag_high`

Timetag High

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
0 (reserved)							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0 (reserved)				DAYS			

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
DAYS					HOURS		

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HOURS		MINUTES					

DAYS

Value	Description
1..365	Days of year

HOURS

Value	Description
0..23	Hours of day

MINUTES

Value	Description
0..59	Minutes of hour

`AiUInt32 timetag_low`

Timetag Low

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
(0) reserved						SECONDS	

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
SECONDS				MICROSECONDS			

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
MICROSECONDS							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MICROSECONDS							

SECONDS

Value	Description
0..59	Seconds of minute

MICROSECONDS

Value	Description
0..999999	Microseconds of second

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.4 ApiCmdBCBHDef

Prototype:

```
AiReturn ApiCmdBCBHDef(AiUInt32 ul_ModuleHandle,
                      AiUInt8 biu, AiUInt16 hid,
                      AiUInt16 bid,
                      AiUInt16 sid,
                      AiUInt16 eid,
                      AiUInt8 qsize,
                      AiUInt8 bqm,
                      AiUInt8 bsm,
                      AiUInt8 sqm,
                      AiUInt8 eqm,
                      AiUInt8 dbm,
                      TY_API_BC_BH_INFO *pbh);
```

Purpose:

This function is used to associate a Data Buffer ID, Buffer Queue size, Queue mode, and error protocol to a BC Buffer Header ID. The BC Buffer Header specified should first be assigned to a BC Transfer using the function **ApiCmdBCXferDef**.

Input:

`AiUInt16 hid`

Buffer Header ID

> **Note:**
> See Section 1.3.5 for the range allowed for this parameter.

`AiUInt16 bid`

Assigned Data Buffer Identifier

> **Note:**
> See Section 1.3.5 for the range allowed for this parameter.

`AiUInt16 sid`

0 (Reserved for Status Queue Entry Identifier)

`AiUInt16 eid`

0 (Reserved for Event Queue Entry Identifier)

`AiUInt8 qsize`

Buffer Queue size definition (Amount of contiguous Data Buffers)

Value	Constant	Description
0	**API_QUEUE_SIZE_1**	Queue size 1
1	**API_QUEUE_SIZE_2**	Queue size 2
2	**API_QUEUE_SIZE_4**	Queue size 4
3	**API_QUEUE_SIZE_8**	Queue size 8
4	**API_QUEUE_SIZE_16**	Queue size 16
5	**API_QUEUE_SIZE_32**	Queue size 32
6	**API_QUEUE_SIZE_64**	Queue size 64
7	**API_QUEUE_SIZE_128**	Queue size 128
8	**API_QUEUE_SIZE_256**	Queue size 256

`AiUInt8 bqm`

Buffer Queue Mode

Value	Constant	Description
0	**API_BQM_CYCLIC**	Cyclic data storage
1	**API_BQM_STAY_LAST**	Store once and stop at end of queue (last index) and reuse last buffer
2	**API_BQM_HOST_CONTROLLED**	Do not advance automatically to next buffer (host controlled)
3		Reserved until status queue is supported

Note:
The Buffer Queue Mode is only processed, if the “Buffer is valid”. The Buffer Queue Modes affect the Current Buffer Index and thus (in the future) the Status- and Event Queue Operation of the BC!

`AiUInt8 bsm`

Data Buffer Store Mode

For Receive Operation the following options are valid:

Value	Constant	Description
0	**API_BSM_RX_DISCARD**	Discard error messages from the current Data Buffer
1	**API_BSM_RX_KEEP_CURRENT**	Keep error messages at the current Data Buffer

For Transmit Operation the following options are valid:

Value	Constant	Description
0	**API_BSM_TX_KEEP_SAME**	Keep the same Data Buffer at transfer error
1	**API_BSM_TX_GO_ON_NEXT**	Go on with the next Data Buffer in the Buffer Queue

Note:
The data buffer store mode is always evaluated at the ‘Transfer-End’: ‘bsm’ = 0:
If a transfer error is detected at the ‘Transfer-End’, the Buffer is not valid! ‘bsm’ = 1:
At ‘Transfer-End’ the Buffer is always valid and it does not matter, if a transfer error is detected or not. If an error occurred during a RT-BC or RT-RT transfer, the BC terminates the transfer with the error detection. Thus, the possible remainder of the transfer after the error will not be stored in the buffer.

AiUInt8 sqm

Status Queue Mode

Value	Constant	Description
0, 1	**API_SQM_ONE_ENTRY_ONLY**	Set Status Queue size to one entry only. This means that one status entry is available for the whole buffer queue.
2	**API_SQM_AS_QSIZE**	Set status queue size equal to the buffer queue size. This means, for each buffer a separate status entry is available.

AiUInt8 eqm

0 (Reserved for Event Queue mode)

AiUInt8 dbm

0 (Reserved for Double Buffer mode)

Output:

TY_API_BC_BH_INFO *pbh

BC Buffer Header information

```
typedef struct ty_api_bc_bh_info {
  AiUInt16 bid;
  AiUInt16 sid;
  AiUInt16 eid;
  AiUInt16 nbufs;
  AiUInt32 hid_addr;
  AiUInt32 bq_addr;
  AiUInt32 sq_addr;
  AiUInt32 eq_addr;
}
TY_API_BC_BH_INFO;
```

AiUInt16 bid

Value	Description
0..2047	Assigned Data Buffer Identifier

AiUInt16 sid

Status Queue Entry Identifier

AiUInt16 eid

Event Queue Entry Identifier

AiUInt16 nbufs

Value	Description
1..256	Amount of allocated contiguous Data Buffers

AiUInt32 hid_addr

26-bit BC Buffer Header Address (Byte-Address)

AiUInt32 bq_addr

26-bit BC Data Buffer Queue Base Pointer (Byte-Address)

AiUInt32 sq_addr

26-bit BC Status Queue Base Pointer (Byte-Address)

AiUInt32 eq_addr

26-bit BC Event Queue Base Pointer (Byte-Address)

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.5 ApiCmdBCBHRead

Prototype:

```
AiReturn ApiCmdBCBHRead(AiUInt32 ul_ModuleHandle,
                        AiUInt8 biu,
                        AiUInt16 uw_HeaderId,
                        TY_API_BC_BH_INFO *px_Pbh);
```

Purpose:

This function is used to read the Data Buffer ID, Buffer Queue size, Queue mode, and error protocol of a BC Buffer Header ID.

Input:

`AiUInt16 uw_HeaderId`

Buffer Header ID

Note:
See Section 1.3.5 for the range allowed for this parameter.

Output:

`TY_API_BC_BH_INFO *px_Pbh`

BC Buffer Header information

```
typedef struct ty_api_bc_bh_info {
  AiUInt16 bid;
  AiUInt16 sid;
  AiUInt16 eid;
  AiUInt16 nbufs;
  AiUInt32 hid_addr;
  AiUInt32 bq_addr;
  AiUInt32 sq_addr;
  AiUInt32 eq_addr;
}
TY_API_BC_BH_INFO;
```

`AiUInt16 bid`

Value	Description
0..2047	Assigned Data Buffer Identifier

`AiUInt16 sid`

Status Queue Entry Identifier

`AiUInt16 eid`

Event Queue Entry Identifier

`AiUInt16 nbufs`

Value	Description
1..256	Amount of allocated contiguous Data Buffers

`AiUInt32 hid_addr`

26-bit BC Buffer Header Address (Byte-Address)

`AiUInt32 bq_addr`

26-bit BC Data Buffer Queue Base Pointer (Byte-Address)

`AiUInt32 sq_addr`

26-bit BC Status Queue Base Pointer (Byte-Address)

`AiUInt32 eq_addr`

26-bit BC Event Queue Base Pointer (Byte-Address)

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.6 ApiCmdBCDytagDef

Prototype:

```
AiReturn ApiCmdBCDytagDef(AiUInt32 ul_ModuleHandle,
                          AiUInt8 biu,
                          AiUInt8 con,
                          AiUInt16 bc_hid,
                          AiUInt16 mode,
                          TY_API_BC_DYTAG bc_dytag[4]);
```

Purpose:

This function is used to define the generation of dynamic data words for BC-RT transfer type or for BC Broadcast transfer type. One to four words of the BC Transmit Buffer for up to 255 different Buffer headers can be selected for dynamic data generation. Dynamic data generation is performed in the BC Transmit Data Buffer by the internal firmware. The BC Transmit Buffer should already be defined using the functions **ApiCmdBCXferDef** and **ApiCmdBCBHDef** when dynamic data generation is enabled.

In **Function Mode** the BC Transmit Buffer is modified (for up to 2 Data Words) after the Data Word has been transmitted. The start value for the dynamic Data Word function shall be within the specified upper and lower limit.

In **Tagging Mode** the BC Transmit Buffer is modified (for up to 4 Data Words) before the Data Words are transmitted. In this mode the dynamic Data Word tagging function can be performed directly on Data Words located in the Transmit Data Buffer or on Function Words which are patched into the Transmit Data Buffer.

> **Note:**
> This function is not supported on embedded devices! (see also chapter B.1.7 "Limitations for embedded board variants")

Input:

`AiUInt8 con`

BC Dynamic Tagging Control

Value	Constant	Description
0	**API_DIS**	Disable Dynamic Data Generation
1	**API_ENA**	Enable Dynamic Data Generation

`AiUInt16 bc_hid`

BC Buffer Header ID

> **Note:**
> See Section 1.3.5 for the range allowed for this parameter.

AiUInt16 mode

BC Dynamic Tagging Mode

Value	Constant	Description
0	**API_DYTAG_STD_MODE**	Function Mode
1	**API_DYTAG_EFA_MODE**	Tagging Mode

TY_API_BC_DYTAG *bc_dytag[4]

BC Dynamic Data description

```
typedef struct ty_api_bc_dytag {
  AiUInt16 tag_fct;
  AiUInt16 min;
  AiUInt16 max;
  AiUInt16 step;
  AiUInt16 wpos;
}
TY_API_BC_DYTAG;
```

Parameter description for Function Mode ('mode' = 0)

Note:
Applicable for up to 2 dynamic words 'bc_dytag[0..1]', structure indexes bc_dytag[2..3] are reserved!

AiUInt16 tag_fct

Dynamic Data Word Generation Function

Value	Constant	Description
0	**API_DYTAG_FCT_DISABLE**	Disable
1	**API_DYTAG_FCT_POS_RAMP**	Positive Ramp Function
2	**API_DYTAG_FCT_NEG_RAMP**	Negative Ramp Function
3	**API_DYTAG_FCT_POS_TRIANGLE**	Positive Triangle Function
4	**API_DYTAG_FCT_NEG_TRIANGLE**	Negative Triangle Function
5	**API_DYTAG_FCT_XMT_WORD**	Transmit Data Word from specified Data Buffer ID
6	**API_DYTAG_FCT_SYNC_COUNTER**	Transmit Synchronisation Counter value (see also ApiCmdSyncCounterGet / Set).

AiUInt16 min

'tag_fct'	Value	Description
1..4	**0..n**	Lower Limit of the dynamic Data Word
5	**0..2047**	Data Buffer Identifier
6	**0**	Reserved

AiUInt16 max

'tag_fct'	Value	Description
1..4	**0..n**	Upper Limit of the dynamic Data Word
5	**0..31**	Word Position of the Data Buffer Word to transmit
6	**0**	Reserved

AiUInt16 step

'tag_fct'	Value	Description
1..4	**0..n**	Stepsize used to increment or decrement the dynamic Data Word
5, 6	**0**	Reserved

AiUInt16 wpos

'tag_fct'	Value	Description
1..6	**0..31**	Word Position of the Data Buffer Word

Parameter description for Tagging Mode ('mode' = 1)

> **Note:**
> Applicable for up to 4 dynamic words 'bc_dytag[0..3]'!

AiUInt16 tag_fct

Dynamic Data Word Generation Function utilizes an incrementer by 1 to modify the data word at the location specified. An initial value of the incremented data word/byte can be specified.

Bit	Value	Constant	Description
15..8	**0**		Reserved
7	**0**	**API_DYTAG_DIRECT_MODIFY**	**Direct Data Buffer Modification** The function modifies the Data Word in the Transfer Data Buffer.
	1	**API_DYTAG_FUNCTION_MODIFY**	**Function Data Word Modification** The function is applied to the Function Word 'min' which is written to the Transfer Data Buffer.
6..0	**0**	**API_DYTAG_FCT_DISABLE**	Disable
	1	**API_DYTAG_FCT_SAWTOOTH_16**	16-bit Sawtooth
	2	**API_DYTAG_FCT_SAWTOOTH_8_LOW**	8-bit Sawtooth in Data Bits 7..0
	3	**API_DYTAG_FCT_SAWTOOTH_8_HIGH**	8-bit Sawtooth in Data Bits 15..8

`AiUInt16 min`

Value	Description
0..n	Initial value of dynamic Function Word

Note:
Only applicable if 'tag_fct'-Bit 7 = 1!

`AiUInt16 max`

Value	Description
0	Reserved

`AiUInt16 step`

Value	Description
0	Reserved

`AiUInt16 wpos`

Value	Description
0..31	Word Position of the dynamic Data Word

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.7 ApiCmdBCFrameDef

Prototype:

```
AiReturn ApiCmdBCFrameDef(AiUInt32 ul_ModuleHandle,
                          AiUInt8 biu
                          TY_API_BC_FRAME *pframe);
```

Purpose:

This command is used to define the sequence of Bus Controller Transfers within a Minor Frame sequence with options for inserting delays, strobe pulse outputs, and skip transfer instructions. Transfers shall first be defined using the function **ApiCmdBCXferDef** and are identified by their Transfer Identifier (xid). Each Minor Frame is identified by a unique Frame Identifier.

Input:

TY_API_BC_FRAME *pframe

BC Minor Frame description

```
typedef struct ty_api_bc_frame {
  AiUInt8 id;
  AiUInt8 cnt;
  AiUInt16 instr[MAX_API_BC_XFRAME];
  AiUInt16 xid[MAX_API_BC_XFRAME];
}
TY_API_BC_FRAME;

#define MAX_API_BC_XFRAME 128
```

AiUInt8 id

Value	Description
1..64	Minor Frame Identifier

AiUInt8 cnt

Value	Description
1..128	Amount of instructions in Minor Frame

AiUInt16 instr[]

Value	Constant	Description
1	**API_BC_INSTR_TRANSFER**	**BC Transfer Instruction** →Used for normal transfers
2	**API_BC_INSTR_SKIP**	**BC Skip Instruction** The skip instruction is a relative, unconditional branch forward. This means, that the instruction execution is continued n instruction after the location of the skip instruction, whereby n is equal to the value set in 'xid[]'. This instruction is also intended to be used as a 'No Operation' instruction with interrupt capability (with "Skip Count=1" and interrupt enabled in 'xid[]')
3	**API_BC_INSTR_WAIT**	**BC Wait Instruction** Stops the BC operation until the delay time (set in 'xid[]') is expired. That means during this time, no instructions will be executed. After the delay time is expired, the instruction execution continues with the following instruction. **Note:** This instruction may be used to implement larger delays, if it is the only instruction within a minor frame (the delay time is then determined by the minor frame time).
4	**API_BC_INSTR_STROBE**	**BC external Strobe Instruction** If this instruction is executed, the BC trigger output of the board is pulsed. After this operation, the instruction execution continues with the following instruction.

AiUInt16 xid[]

'instr'	Bit-Pos	Value	Description
1	**15..0**	**1 .. n**	BC Transfer Identifier See Section 1.3.5 for the range allowed for this parameter.
2	**15**	**0**	Disable Interrupt
		1	Enable Interrupt
	7..0	**1..127**	Skip Count Number of instructions which shall be skipped, including the current instruction.
3	**15..0**	**0..65535**	**BC Wait Time (in 250ns steps)** The maximum delay time is appr. 16ms
4	**15..0**	**0**	Reserved

Output:

none

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.8 ApiCmdBCGetDytagDef

Prototype:

```
AiReturn ApiCmdBCGetDytagDef(AiUInt32 ul_ModuleHandle, AiUInt8 biu,
                             AiUInt16 bc_hid, AiUInt16 *mode,
                             TY_API_BC_DYTAG bc_dytag[4]);
```

Purpose:

This function is used to read Dytag settings for the generation of dynamic data words for BC-RT transfer type or for BC broadcast transfer type.

Input:

AiUInt16 bc_hid

BC Buffer Header ID

> **Note:**
> See Section 1.3.5 for the range allowed for this parameter.

Output:

AiUInt16 *mode

BC Dynamic Tagging Mode

Value	Constant	Description
0	**API_DYTAG_STD_MODE**	Function Mode
1	**API_DYTAG_EFA_MODE**	Tagging Mode

TY_API_BC_DYTAG *bc_dytag[4]

BC Dynamic Data description

```
typedef struct ty_api_bc_dytag {
  AiUInt16 tag_fct;
  AiUInt16 min;
  AiUInt16 max;
  AiUInt16 step;
  AiUInt16 wpos;
}
TY_API_BC_DYTAG;
```

Parameter description for Function Mode (`mode' = 0)

> **Note:**
> Applicable for up to 2 dynamic words ('bc_dytag[0..1]', structure indexes bc_dytag[2..3] are reserved!

AiUInt16 tag_fct

Dynamic Data Word Generation Function

Value	Constant	Description
0	**API_DYTAG_FCT_DISABLE**	Disable
1	**API_DYTAG_FCT_POS_RAMP**	Positive Ramp Function
2	**API_DYTAG_FCT_NEG_RAMP**	Negative Ramp Function
3	**API_DYTAG_FCT_POS_TRIANGLE**	Positive Triangle Function
4	**API_DYTAG_FCT_NEG_TRIANGLE**	Negative Triangle Function
5	**API_DYTAG_FCT_XMT_WORD**	Transmit Data Word from specified Data Buffer ID
6	**API_DYTAG_FCT_SYNC_COUNTER**	Transmit Synchronisation Counter value (see also ApiCmdSyncCounterGet / Set).

AiUInt16 min

'tag_fct'	Value	Description
1..4	**0..n**	Lower Limit of the dynamic Data Word
5	**0..2047**	Data Buffer Identifier
6	**0**	Reserved

AiUInt16 max

'tag_fct'	Value	Description
1..4	**0..n**	Upper Limit of the dynamic Data Word
5	**0..31**	Word Position of the Data Buffer Word to transmit
6	**0**	Reserved

AiUInt16 step

'tag_fct'	Value	Description
1..4	**0..n**	Stepsize used to increment or decrement the dynamic Data Word
5, 6	**0**	Reserved

AiUInt16 wpos

'tag_fct'	Value	Description
1..6	**0..31**	Word Position of the Data Buffer Word

Parameter description for Tagging Mode ('mode' = 1)

> **Note:**
> Applicable for up to 4 dynamic words ('bc_dytag[0..3]'!

AiUInt16 tag_fct

Dynamic Data Word Generation Function utilizes an incrementer by 1 to modify the data word at the location specified. An initial value of the incremented data word/byte can be specified.

Bit	Value	Constant	Description
15..8	**0**		Reserved
7	**0**	**API_DYTAG_DIRECT_MODIFY**	**Direct Data Buffer Modification** The function modifies the Data Word in the Transfer Data Buffer.
	1	**API_DYTAG_FUNCTION_MODIFY**	**Function Data Word Modification** The function is applied to the Function Word 'min' which is written to the Transfer Data Buffer.
6..0	**0**	**API_DYTAG_FCT_DISABLE**	Disable
	1	**API_DYTAG_FCT_SAWTOOTH_16**	16-bit Sawtooth
	2	**API_DYTAG_FCT_SAWTOOTH_8_LOW**	8-bit Sawtooth in Data Bits 7..0
	3	**API_DYTAG_FCT_SAWTOOTH_8_HIGH**	8-bit Sawtooth in Data Bits 15..8

`AiUInt16 min`

Value	Description
0..n	Initial value of dynamic Function Word

Note:
Only applicable if 'tag_fct'-Bit 7 = 1!

`AiUInt16 max`

Value	Description
0	Reserved

`AiUInt16 step`

Value	Description
0	Reserved

`AiUInt16 wpos`

Value	Description
0..31	Word Position of the dynamic Data Word

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.9 ApiCmdBCGetMajorFrameDefinition

Prototype:

```
AiReturn ApiCmdBCGetMajorFrameDefinition(AiUInt8 Module,
                                         AiUInt8 Biu,
                                         TY_API_BC_MFRAME *px_BCMajorFrame);
```

Purpose:

This function is used to read the sequence of Minor Frames within the Major Frame.

Input:

none

Output:

TY_API_BC_MFRAME * px_BCMajorFrame

BC Major Frame description

```
typedef struct ty_api_bc_mframe {
  AiUInt8 cnt;
  AiUInt8 fid[MAX_API_BC_MFRAME];
}
TY_API_BC_MFRAME;

#define MAX_API_BC_MFRAME 64
```

AiUInt8 cnt

Value	Description
1..64	Amount of Minor Frames in Major Frame

AiUInt8 fid[]

Value	Description
1..64	Sequence of Minor Frame Identifiers

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.10 ApiCmdBCGetMinorFrameDefinition

Prototype:

```
AiReturn ApiCmdBCGetMinorFrameDefinition(AiUInt8 Module,
                                         AiUInt8 Biu,
                                         AiUInt8 uc_FrameID,
                                         TY_API_BC_FRAME *px_BCMinorFrame);
```

Purpose:

This command is used to read the sequence of Bus Controller Transfers within a Minor Frame sequence with options for inserting delays, strobe pulse outputs, and skip transfer instructions.

Input:

AiUInt8 uc_FrameID

Minor Frame Identifier

Output:

TY_API_BC_FRAME *pframe

BC Minor Frame description

```
typedef struct ty_api_bc_frame {
  AiUInt8 id;
  AiUInt8 cnt;
  AiUInt16 instr[MAX_API_BC_XFRAME];
  AiUInt16 xid[MAX_API_BC_XFRAME];
}
TY_API_BC_FRAME;

#define MAX_API_BC_XFRAME 128
```

AiUInt8 id

Value	Description
1..64	Minor Frame Identifier

AiUInt8 cnt

Value	Description
1..128	Amount of instructions in Minor Frame

AiUInt16 instr[]

Value	Constant	Description
1	**API_BC_INSTR_TRANSFER**	**BC Transfer Instruction** →Used for normal transfers
2	**API_BC_INSTR_SKIP**	**BC Skip Instruction** The skip instruction is a relative, unconditional branch forward. This means, that the instruction execution is continued n instruction after the location of the skip instruction, whereby n is equal to the value set in 'xid[]'. This instruction is also intended to be used as a 'No Operation' instruction with interrupt capability (with "Skip Count=1" and interrupt enabled in 'xid[]')
3	**API_BC_INSTR_WAIT**	**BC Wait Instruction** Stops the BC operation until the delay time (set in 'xid[]') is expired. That means during this time, no instructions will be executed. After the delay time is expired, the instruction execution continues with the following instruction. **Note:** This instruction may be used to implement larger delays, if it is the only instruction within a minor frame (the delay time is then determined by the minor frame time).
4	**API_BC_INSTR_STROBE**	**BC external Strobe Instruction** If this instruction is executed, the BC trigger output of the board is pulsed. After this operation, the instruction execution continues with the following instruction.

AiUInt16 xid[]

'instr'	Bit-Pos	Value	Description
1	**15..0**	**1 .. n**	BC Transfer Identifier See Section 1.3.5 for the range allowed for this parameter.
2	**15**	**0**	Disable Interrupt
		1	Enable Interrupt
	7..0	**1..127**	Skip Count Number of instructions which shall be skipped, including the current instruction.
3	**15..0**	**0..65535**	**BC Wait Time (in 250ns steps)** The maximum delay time is appr. 16ms
4	**15..0**	**0**	Reserved

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.11 ApiCmdBCGetXferBufferHeaderInfo

Prototype:

```
AiReturn ApiCmdBCGetXferBufferHeaderInfo(AiUInt32 ul_ModuleHandle,
                                         AiUInt8 biu,
                                         AiUInt32 ul_XferId,
                                         AiUInt32 *pul_BufHeaderIndex,
                                         AiUInt32 *pul_BufHeaderAddr);
```

Purpose:

This function is used to get the buffer header id of given transfer.

Input:

`AiUInt32 ul_XferId`

BC Transfer ID

> **Note:**
> See Section 1.3.5 for the range allowed for this parameter.

Output:

`AiUInt32 *pul_BufHeaderIndex`

The buffer header index of this transfer

`AiUInt32 *pul_BufHeaderAddr`

The address of the buffer header of this transfer

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.12 ApiCmdBCGetXferDef

Prototype:

```
AiReturn ApiCmdBCGetXferDef(AiUInt32 ul_ModuleHandle,
                            AiUInt8 biu,
                            AiUInt16 uw_XferId,
                            TY_API_BC_XFER * px_BCXfer);
```

Purpose:

This function is used to get all transfer properties of a Bus Controller Transfer including source/destination information, error injection specifications, and interrupt generation.

Input:

AiUint16 uw_XferId

BC Transfer ID

Note: See Section 1.3.5 for the range allowed for this parameter.

Output:

TY_API_BC_XFER *pxfer

BC Transfer description

```
typedef struct ty_api_bc_xfer {
  AiUInt16 xid;
  AiUInt16 hid;
  AiUInt8 type;
  AiUInt8 chn;
  AiUInt8 xmt_rt;
  AiUInt8 rcv_rt;
  AiUInt8 xmt_sa;
  AiUInt8 rcv_sa;
  AiUInt8 wcnt;
  AiUInt8 tic;
  AiUInt8 hlt;
  AiUInt8 rte;
  AiUInt8 res;
  AiUInt8 sxh;
  AiUInt8 rsp;
  AiUInt8 gap_mode;
  AiUInt16 swxm;
  struct ty_api_bc_err err;
  AiUInt16 gap;
}
TY_API_BC_XFER;
```

AiUInt16 xid

BC Transfer ID

AiUInt16 hid

Buffer Header ID

AiUInt8 type

Bit	Value	Constant	Description
7..6	**0**		reserved for Service Request / Vector Word handling
5..2	**0**		reserved
1..0	**0**	**API_BC_TYPE_BCRT**	BC to RT transfer type
	1	**API_BC_TYPE_RTBC**	RT to BC transfer type
	2	**API_BC_TYPE_RTRT**	RT to RT transfer type

AiUInt8 chn

Transfer Channel

Value	Constant	Description
0	**API_BC_XFER_BUS_PRIMARY**	Primary Bus
1	**API_BC_XFER_BUS_SECONDARY**	Secondary Bus

AiUInt8 xmt_rt

Value	Description
0..31	RT Address of transmitting terminal (31 = broadcast RT address)

AiUInt8 rcv_rt

Value	Description
0..31	RT Address of receiving terminal (31 = broadcast RT address)

AiUInt8 xmt_sa

Value	Description
0..31	Subaddress of transmitting terminal (0, 31 = Mode code Subaddress)

AiUInt8 rcv_sa

Value	Description
0..31	Subaddress of receiving terminal (0, 31 = Mode code Subaddress)

AiUInt8 wcnt

Value	Description
0	Word Count / Mode code field for Command Word is 32
1..31	Word Count / Mode code field for Command Word

AiUInt8 tic

Transfer Interrupt Control

Value	Constant	Description
0	**API_BC_TIC_NO_INT**	No Interrupt
1	**API_BC_TIC_INT_ON_XFER_END**	Interrupt on End of Transfer
2	**API_BC_TIC_INT_ON_XFER_ERR**	Interrupt on Transfer Error
3	**API_BC_TIC_INT_ON_STAT_EXCEPT**	Interrupt on Status Word Exception
4..7	**API_BC_TIC_NO_INT**	Reserved

AiUInt8 hlt

BC Halt Control

Value	Constant	Description
0	**API_BC_HLT_NO_HALT**	Do not halt on exception
1	**API_BC_HLT_HALT_ON_XFER_ERR**	Reserved for Halt on Transfer Error
2	**API_BC_HLT_HALT_ON_STAT_EXCEPT**	Reserved for Halt on Status Word Exception
3	**API_BC_HLT_HALT_ON_ERR_EXCEPT**	Reserved for Halt on Transfer Error and Status Word Exception
4	**API_BC_HLT_HALT_ON_ANY_INT**	Halt on any Interrupt
5..7		Reserved

AiUInt8 rte

Retry enable (see also parameter 'retr' of command ApiCmdBCIni) and toggle bus control

Value	Constant	Description
0	**API_DIS**	Retry disabled
1	**API_ENA**	Retry enabled
2	**API_TOGGLE_BUS_KEEP**	Keep bus at next transmission of this transfer.
3	**API_TOGGLE_BUS_ALT**	Usa alternate bus at next transmission of this transfer

> **Note:**
> Values 2 and 3 are not available on devices with a multi channel firmware. Please see chapter "Limitations for specific boards" for details.

AiUInt8 res

0 (Reserved)

AiUInt8 sxh

Status Word Exception handling of Service Request control (only applicable when Service Request / Vector Word Mode Control is enabled using **ApiCmdBCIni** command). See **ApiCmdBCSrvReqVecStatus** for further information.

Value	Constant	Description
0	**API_BC_SRVW_DIS**	No Status Word Service Request handling
1	**API_BC_SRVW_ENA**	Generate automatic Transmit Vector Word mode code (Mode code 16) if Service Request Bit of the received Status Word is set Note: Automatic Mode code generation on Service Request Bit will only be processed if no error is detected and no fast Intermessage Gap mode is enabled (refer to 'gap_mode'). For Automatic Mode code generation on Service Request Bit the first received Status Word is evaluated, thus this function is only applicable for BC-RT and RT-BC transfer types (refer to 'type').
2	**API_BC_SRVW_SINGLE**	Single RX/TX transaction Only applicable for BC-RT and RT-BC transfer types. (refer to 'type'). This transfer will define the Single RX/TX transmission to be used if requested in the RT Vector Word. Since such transfer is inserted after decoding of a previous Transmit Vector Word mode code 'xid' shall not be used in the ApiCmdBCFrameDef command
3	**API_BC_SRVW_MULTIPLE**	Multiple / Delete RX/TX transaction Only applicable for BC-RT and RT-BC transfer types (refer to 'type'). This transfer will define the Multiple RX/TX transmission to be used if requested in the RT Vector Word. Since such transfer is inserted after decoding of a previous Transmit Vector Word mode code 'xid' shall not be used in the ApiCmdBCFrameDef command

Note:
Values 2 and 3 are not applicable, when using the functions ApiCmdBCInstrTblIni, ApiCmdBCInstrTblGen and ApiCmdBCInstrTblGetAddrFromLabel!!!

`AiUInt8 rsp`

Expected Response Control for Command Word

Value	Constant	Description
0	**API_BC_RSP_AUTOMATIC**	Automatic Response Control In this mode BC expects appropriate RT-Response to the defined Transfer Type, dependent on the MIL-STD 1553 protocol type(s), which is (are) defined for the RT('s) with the function ApiCmdDefMilbusProtocol
1	**API_BC_RSP_NO_SW1_EXPECTED**	No Status Word 1 expected (Note: for RT-RT transfer type Status Word 1 represents the status response for the Transmit Command word)
2	**API_BC_RSP_NO_SW2_EXPECTED**	No Status Word 2 expected (RT-RT transfer type only, Note: for RT-RT transfer type Status Word 2 represents the status response for the Receive Command word)

`AiUInt8 gap_mode`

Three different Gap Modes are available which are described after the following table:

Value	Constant	Description
0	**API_BC_GAP_MODE_DELAY**	Delay mode / Transfer Wait Time mode
1	**API_BC_GAP_MODE_STANDARD**	Standard Gap Mode
2	**API_BC_GAP_MODE_FAST**	Fast Intermessage Gap Mode

Delay mode / Transfer Wait Time Mode

In this mode '**gap**' (14bit) specifies the time from the start of a transfer to the start of the following transfer in μs.

Note:
If the specified time is shorter than the actual time needed for transfer transmission, 'gap' is ignored and the Bus Controller automatically generates a minimum intermessage gap of app. 11μs.

This gap mode guarantees deterministic framing independent from RT response times and provides excellent repeatability of timed bus events.

The nominal Transfer Wait Time is calculated as:

$$\mathbf{T_D = \text{'gap'} * 1\mu s}$$

Standard Gap Mode

In Standard Gap Mode '**gap**' (14bit) specifies the wait gap between the end of the current transfer and start of the following transmitted Command Word. The nominal Intermessage Gap Time is calculated as:

$$\mathbf{T_{IG} = \text{`gap'} * 1\mu s}$$

> **Note:**
> The idle time of the physical bus is 2.0μs less than this time due to the gap measurement definition of MIL-STD-1553B.

> **Note:**
> The minimum Intermessage gap time achieved using the Standard Gap Mode is equal to the minimum gap time achieved in the Transfer Wait Time Mode.

> **Note:**
> At broadcast transfers, the response timeout is used to check that no status word is responded. Thus, the Intermessage Gap Time in Standard Gap Mode shall be greater than the response timeout, to guarantee proper operation.

Fast Intermessage Gap Mode

In the Fast Intermessage Gap Mode, the Bus Controller generates Intermessage Gap Times down to 4 μs. In this mode, the value of '**gap**' is limited to 6bits. The nominal Intermessage Gap Time is calculated as:

$$\mathbf{T_{IG} = 4\mu s + (\text{`gap'} * 0.25\mu s) \ \ (1\ Mbit\ Transmission\ Mode)}$$

Therefore, Intermessage Gap times between 4μs up to 19.75μs can be achieved in this mode. This mode can be used for RT validation and testing, but **shall not be used** during normal operation since line busy and word count high checks are disabled on the BC. Also worst case operation is not guaranteed if this mode is combined with Multi RT, monitor and Mailbox Operation. The Fast Intermessage Gap Mode shall only be used for transfer instructions which are followed by another transfer instruction. In case of a transfer error, the Fast Intermessage Gap Mode is switched off for this transfer and a default Intermessage gap of app. 11μs is generated.

> **Note:**
> The idle time of the physical bus is 2.0μs less than this time due to the gap measurement definition of MIL-STD-1553B.

> **Note:**
> This mode is not supported on ASC1553-1

AiUInt16 swxm

Status Word Exception Mask

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
reserved (0)					MERR	INSTR	SREQ

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
reserved (0)			BRCV	BUSY	SUB	DBCA	TERM

MERR	Message Error Bit
INSTR	Instrumentation Bit
SREQ	Service Request
BRCV	Broadcast Received Bit
BUSY	Busy Bit
SUB	Subsystem Flag
DBCA	Dynamic Bus Control Acceptance
TERM	Terminal Flag

TY_API_BC_ERR err

BC Transfer Error Injection specifications

```
typedef struct ty_api_bc_err {
  AiUInt8 type;
  AiUInt8 sync;
  AiUInt8 contig;
  AiUInt8 padding1;
  AiUInt32 err_spec;
}
TY_API_BC_ERR;
```

AiUInt8 type

Error Type

Value	Constant	Description
0	**API_ERR_TYPE_NO_INJECTION**	No error injection
1	**API_ERR_TYPE_COMMAND_SYNC**	Command Sync Error with Sync Pattern in ‘sync’
2	**API_ERR_TYPE_DATA_SYNC**	Data Sync Error with Sync Pattern in ‘sync’ in Word ‘wpos’
3	**API_ERR_TYPE_PARITY**	Parity Error in Word ‘wpos’
4	**API_ERR_TYPE_MANCHESTER_HIGH**	Manchester Stuck at High Error in Word ‘wpos’ at Bit Position bpos’
5	**API_ERR_TYPE_MANCHESTER_LOW**	Manchester Stuck at Low Error in Word ‘wpos’ at Bit Position bpos’
6	**API_ERR_TYPE_GAP**	Gap Error with Gap defined in ‘contig’ in Word ‘wpos’

7	**API_ERR_TYPE_WORD_CNT_HIGH**	Word Count High (+1) Error
8	**API_ERR_TYPE_WORD_CNT_LOW**	Word Count Low (-1) Error
9	**API_ERR_TYPE_BIT_CNT_HIGH**	Bit Count High Error in Word 'wpos' (+ 'bc_bits')
10	**API_ERR_TYPE_BIT_CNT_LOW**	Bit Count Low Error in Word 'wpos' (- 'bc_bits')
11		Reserved
12	**API_ERR_TYPE_ZERO_CROSS_NEG**	Zero Crossing Low Deviation Error, in word 'WPOS' at bit position 'BPOS' with negative deviation in 'contig'
13	**API_ERR_TYPE_ZERO_CROSS_POS**	Zero Crossing High Deviation Error, in word 'WPOS' at bit position 'BPOS' with positive deviation in 'contig'
14		Reserved
15	**API_ERR_TYPE_BUS_SWITCH**	Bus Switching Support The BIU disables the physical decoder device on the alternate bus, if the current transfer is in fast gap mode (RT-Validation Test 5.2.1.8)

Note:
The error type 9 (Bit Count High) is not supported together with Modecodes without data!!!

AiUInt8 sync

Sync Field Error Half-Bit-Pattern (6 LS-Bits)
(38hex = 111000 Sync Pattern)

AiUInt8 contig

'type'	Value	Description
Gap Error	**1..15**	Low Speed Gap Error Half Bits
Zero Crossing Error	**0**	125ns deviation
	1	187.5ns deviation
	2	156.25ns deviation
	3	218.75ns deviation **Note:** Values 2 and 3 are not available for all AIM devices. Please check the chapter "Board functionality overview" and look for "Error Injection of High Resolution Zero Crossing Deviation".

Note:
The resolution is:
1Mbit: 0.5μs per Half Bit

AiUInt8 padding1

Reserved (0)

AiUInt32 err_spec

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
reserved (0)							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
WPOS							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
BPOS							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BCBITS							

WPOS

Value	Description
0..32	Error Word Position (0 = Command Word)

BPOS

Value	Description
4..20	Error Bit Position (20 =Parity Bit)

BCBITS

Value	Description
1..3	Amount of Bit Count Error Bits

`AiUInt16 gap`

'gap_mode'	Value	Description
0 or 1	**0..16383**	Transfer Wait Time in μs
2	**0..63**	Transfer Wait Time It is calculated as follows: $\mathbf{Time = 4\mu s + (Value * 0.25\mu s)}$

See the description of 'gap_mode' for more detailed information.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.13 ApiCmdBCHalt

Prototype:

```
AiReturn ApiCmdBCHalt(AiUInt32 ul_ModuleHandle,
                      AiUInt8 biu);
```

Purpose:

This function is used to stop execution of BC Transfers on the AIM board.

Input:

none

Output:

none

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.14 ApiCmdBCIni

Prototype:

```
AiReturn ApiCmdBCIni(AiUInt32 ul_ModuleHandle,
                     AiUInt8 biu,
                     AiUInt8 retr,
                     AiUInt8 svrq,
                     AiUInt8 tbm,
                     AiUInt8 gsb);
```

Purpose:

This function is used to initialize the Bus Controller mode of the AIM board. This must be the first BC function called.

Input:

AiUInt8 retr

Defines the number of retries which shall be performed by the BC, if a transfer is erroneous. The retry mechanism and bus switching method works as follows: Retry on the same bus until the retry count (internally used value) is decremented to zero, then switch the bus and try once more again on this bus.

Value	Constant	Description
0	**API_DIS**	Retry disabled
1	**API_RETRY_1ALT**	One retry on the alternate bus
2	**API_RETRY_1SAME_1ALT**	One retry on the same bus, one retry on the alternate bus
3	**API_RETRY_2SAME_1ALT**	Two retries on the same bus, one retry on the alternate bus

The switching remains permanent for all transfers in “Global Transfer Bus Mode” (see parameter ‘tbm’), or for the single transfer in “Transfer Specific Bus Mode”, respectively. Afterwards, the bus remains in this state until it is switched again by the application software or as a result of another retry sequence. The different bus modes can be defined with the parameter ‘tbm’ of this function.

Note:
When set to retry disabled, the retry of individual BC transfers cannot be enabled using ApiCmdBCXferDef.

AiUInt8 svrq

Service Request / Vector Word Mode Control

Value	Constant	Description
0	**API_DIS**	Disable
1	**API_ENA**	Enable

Note:
If RT capability of the AIM board is used for simulatiuon of Mode code 16, the function ApiCmdRTSACon shall set ‘smod’=3 and ‘swm’=0xFEFF in order to reset the Service Request Bit of the Status Word response automatically!

AiUInt8 tbm

BC Transfer Bus Mode

Value	Constant	Description
0	**API_TBM_TRANSFER**	**Transfer Specific Bus Mode** The initial default MILBus (primary or secondary) for a **specific** MILBus transfer is defined in the corresponding Transfer Descriptor (see relating setting of the command “ApiCmdBCXferDef”). If a retry is successful on a MILBus, this MILBus becomes the default MILBus for the **specific** MILBus transfer only until another retry switches the bus back.
1	**API_TBM_GLOBAL**	**Global Bus Mode** The initial default MILBus (primary or secondary) for all MILBus transfers is defined with the parameter ‘gsb’ which is described below. If a retry is successful on the alternate MILBus, the MILBus is permanently switched for all transfers until another retry switches the bus back.

Note:
Value 1 is not available on devices with a multi channel or embedded firmware. Please see chapter “Limitations for specific boards” for details.

AiUInt8 gsb

Global Start Bus

TBM	Value	Constant	Description
1	**0**	**API_BC_XFER_BUS_PRIMARY**	Primary Bus
1	**1**	**API_BC_XFER_BUS_SECONDARY**	Secondary Bus

Output:

none

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.15 ApiCmdBCInstrTblGen

Prototype:

```
AiReturn ApiCmdBCInstrTblGen(AiUInt32 ul_ModuleHandle,
                             AiUInt8 biu,
                             AiUInt8 mode,
                             AiUInt32 cnt,
                             AiUInt32 dest_cnt,
                             AiUInt32 dest_offset,
                             TY_API_BC_FW_INSTR *tbl,
                             AiUInt32 *ctbl,
                             AiUInt32 *err_line,
                             AiUInt8 *status);
```

Purpose:

This function is used to convert the BC Instruction Table structure (tbl) into an array of instruction long-words (ctbl). Labels, Transfer IDs, Jumps, Calls and other firmware instructions contained within the BC Instruction Table structure (tbl) are calculated to absolute addresses during the conversion. The converted output represents an image of Firmware instructions which is then written into the BC Instruction list area of the Global memory.

Using this approach is an alternate method of defining Minor and Major Frame sequences besides the standard library functions **ApiCmdBCFrameDef** and **ApiCmdBCMFrameDef**.

A cyclic frame can be setup more efficiently and with more flexibility within the Global memory. Using this function provides no limitation to the number of Transfers allowed within a Minor Frame and the number of Minor Frames in a Major Frame (only limitation is the available size of the BC Instruction list within the Global memory layout, refer to **ApiCmdBCInstrTblIni**). Using the BC Instruction Table structure (tbl) allows the user to prepare additional Transfer sequences (Minor Frames) within the Global memory for later use (cyclic or acyclic). For each Instruction within the tbl structure a label can be defined. Using function **ApiCmdBCInstrTblGetAddrFromLabel** returns the address for such label which can then be used as an input parameter for **ApiCmdBCStart** (parameter saddr). To start the BC operation / Transfer execution of the specified BC Instruction Table (tbl) function **ApiCmdBCStart** shall be called with smod = API_BC_START_INSTR_TABLE.

Using parameters dest_cnt and dest_offset it is possible to write specific parts or entire ctbl to the Instruction list area of the Global memory.

Input:

AiUInt8 mode

Value	Description
0	Initalize all parameters of Instruction Table tbl to 0 (CLEAR)
1	Convert Instruction Table tbl to array ctbl (CONVERT)
2	Write array ctbl to the Instruction list memory area (WRITE)
3	CONVERT and immediately WRITE
4	Convert Instruction Table tbl to array ctbl using “dest_offset” for calculating the addresses with an offset (CONVERT_OFFSET)
5	CONVERT_OFFSET and immediately WRITE

AiUInt32 cnt

Amount of entries to convert from BC Instruction Table structure (tbl).

AiUInt32 dest_cnt

Amount of ctbl entries to write to the Instruction list area of the Global memory.

AiUInt32 dest_offset

Byte offset for writing ctbl entries to Instruction list area. An Offset value of 0 is identical to the Start address of the Instruction list area within the Global memory layout (refer to to **ApiCmdBCInstrTblIni**).

‘mode’	Value	Description
0, 1	**0**	Reserved
2, 3	**byte offset**	The instruction list (ctbl) is written to the Global Memory Instruction list area at offset ‘byte offset’ starting with instruction at offset ‘byte offset’ in ctbl Ctbl[byte offset] →GRAM Instruction List Area [byte offset]
4, 5	**byte offset**	The instruction list (ctbl) is written to the Global Memory Instruction list area at offset ‘byte offset’ starting with first instruction in ctbl Ctbl →GRAM Instruction List Area [byte offset] **Note:** This is typically used to compile a partial list only and write it to the board!

TY_API_BC_FW_INSTR *tbl

BC Instruction Table

```
typedef struct ty_api_bc_fw_instr {
  AiUInt16 label; /* Label */
  AiUInt8 op; /* Firmware Opcode */
  AiUInt8 res; /* Reserved */
  AiUInt32 par1;
  AiUInt32 par2;
  AiUInt32 laddr; /* Instruction Address for Label */
}
TY_API_BC_FW_INSTR;
```

AiUInt16 label

Instruction Label (eg. for idenification of jump destination, start of a transfer sequence, etc)

AiUInt8 op

Firmware operation (opcode)

Value	Constant	Description
0x10	**API_BC_FWI_XFER**	Execute BC Transfer (defined using ApiCmdBCXferDef)
0x11	**API_BC_FWI_EFEX_XFER**	Execute BC EFEX Transfer (defined using Api3910CmdEfBCXferDef)
0x11 – 0x1F	**Reserved**	-
0x20	**API_BC_FWI_CALL**	Call Subtable – allows you to jump to a subtable/minor frame definition identified with a label.
0x21	**API_BC_FWI_RET**	Return from Subtable – used to return to the main BC Instruction Table entry following the related API_BC_FWI_CALL opcode.
0x22	**API_BC_FWI_JUMP**	Jump to Subtable / Instruction – absolute jump to an Instruction Table entry identified by a label.
0x24	**API_BC_FWI_SKIP**	Relative Branch (Skip) – skip a user-specified number of instructions
0x25	**API_BC_FWI_WTRG**	Wait for BC Trigger input – ties the external pulse to start of minor frames, or starts execution of the major frame based on the external pulse
0x26	**API_BC_FWI_STRB**	Strobe BC Trigger output – instruction to output a strobe signal
0x27	**API_BC_FWI_DJZ**	Decrement and Jump on Zero – When using non-cyclic major frame (as specified in ApiCmdBCStart) you can jump to a label in the BC Instruction Table when the major frame counter is decremented to zero.
0x28	**API_BC_FWI_WMFT**	Wait for next Minor Frame Time slot – instruction to wait until the next minor frame time slot begins, then continue with the following entry in the BC Instruction Table
0x29	**API_BC_FWI_HALT**	BC Operation Halt – Halt the BC

0x2A	**API_BC_FWI_DELAY**	Delay – delay the execution of the next entry in the BC Instruction Table by a user-specified time
0x2B	**Reserved**	-
0x2C	**API_BC_FWI_CMFT**	Change Minor Frame Time – instruction to change the minor frame time “on-the-fly” to a user-specified value.
0x2D	**API_BC_FWI_RESMF**	Reset Major Frame Used to implement swapping between several different major frames in one BC setup. This instruction shall be used as pre-instruction before each major frame, which is addressed / started via a BC jump instruction. The instruction itself resets the BC High and Low Instruction List stacks and re-read the major frame count from the Global RAM, if RMFC is set to 1. To achieve an equidistant minor framing after a major frame change, please insert a change minor frame time instruction to re-synchronize the minor frame timing of the BC.
0x2E	**Reserved**	-
0x2F	**API_BC_FWI_RET _ON_XFER_ERR**	Return on Error from Subtable - Used to return to the main BC Instruction Table entry in case of an faulty transfer following the related API_BC_FWI_CALL opcode.

`AiUInt32 par1 / par2`

Instruction parameters

Op	par1	par2	Description
API_BC_FWI_XFER	<xid>	-	Execute BC Transfer identified by BC Transfer ID <xid>(refer to function **ApiCmdBCXferDef**)

	<xid>	<hs_xf_type>	For HS RT-RT type transfers two contiguous entries are required within the BC Intruction Table (corresponds to first and second LS Action Word transfer). In this case hs_xf_type shall be set to one of the following values for the second entry: API_HS_BC_TYPE_RTRT API_HS_BC_TYPE_RTBR APIEF_BC_TYPE_EERTRT APIEF_BC_TYPE_E_RTBR APIEF_BC_TYPE_E_RTBRMIXED
API_BC_FWI_EFEX_XFER	<xid>	<efex_xf_type>	Execute pure EFEX BC Transfer identified by HS BC Transfer ID <xid>(refer to function **Api3910CmdEfBCXferDef**)
API_BC_FWI_CALL	<label>!= 0	-	Call subtable / jump to Instruction Table entry which is identified by <label>. Subtable must be terminated with Return opcode.
API_BC_FWI_RET	-	-	Return from subtable, execution of instructions is continued with instruction which is following the related Call opcode.
API_BC_FWI_JUMP	<label>!= 0	-	Absolute Jump to Instruction Table entry which is identified by <label>.
API_BC_FWI_SKIP	<offset>	DIR, INT	Relative branch Par1 / Bit 15-0 (OFFSET): Number of instructions which shall be skipped including current instruction Par2 / Bit 0 (DIR) : 0 = branch forward 1 = branch backward Par2 / Bit 1 (INT) : 0 = no interrupt 1 = assert interrupt on instruction execution All other bits are reserved Note: A ‘No Operation’ instruction (NOP) with interrupt can be implemented with OFFSET=1, DIR=0, INT=1

API_BC_FWI_WTRG	-	RES, SYN	Wait for external BC Trigger input Par2 / Bit 0 (RES) : reserved bit Par2 / Bit 1 (SYN) : 0 = BC Minor Framing is not synchronized 1 = BC Minor Framing is synchronized to external trigger, such that all Minor Frames after the trigger are equally spaced All other bits are reserved
API_BC_FWI_STRB	-	-	Strobe external BC Trigger output
API_BC_FWI_DJZ	<label>!= 0	-	Decrement and Jump on zero, if Major Frame Counter (MFC register in Firmware) is not zero then instruction execution continues with the following instruction, if zero then jump to Instruction Table entry which is identified by <label>. The DJZ instruction shall only be used once within the Instruction Table to guarantee proper counting of Major Frame cycles.
API_BC_FWI_WMFT	-	-	Wait for next Minor Frame Time slot – then continue with the following entry in the BC Instruction Table.
API_BC_FWI_HALT	-	-	BC Halt
API_BC_FWI_DELAY	<time>	-	Delay time time in 250ns steps, up to 16ms delay can be defined (only Bits 15-0 are relevant) **Note:** Delay instructions are not allowed for Acyclic Frame Instruction Table definitions.
API_BC_FWI__CMFT	<time>	-	Change Minor Frame Time on-the-fly to new Minor Frame Time, time is in 1us steps (only Bits 19-0 are relevant)
API_BC_FWI__RESMF	RMFC	-	Re-Read major frame count 0 = Major Frame Count is not re-read 1 = Major Frame Count is re-read

API_BC_FWI_RET _ON_XFER_ERR	-	-	Return from subtable after error, execution of instructions is continued with instruction which is following the related Call opcode.

AiUInt32 laddr

This parameter returns the instruction address for the label thus operates as output parameter.

Output:

AiUInt32 *ctbl

Converted instruction array (image) from BC Instruction Table (tbl). Each entry is stored as a 32-bit word, the format is according to the AIM 1553/3910 Firmware Specification.

AiUInt32 *err_line

Entry of BC Instruction Table in which a conversion error occurred (e.g duplicate Labels, zero passed for par1 where opcode requires label, or any other invalid input parameters for par1 or par2).

AiUInt8 *status

API_OK or API_ERROR (eg. NULL pointer passed for tbl or ctbl)

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.16 ApiCmdBCInstrTblGetAddrFromLabel

Prototype:

```
AiReturn ApiCmdBCInstrTblGetAddrFromLabel(AiUInt32 ul_ModuleHandle,
                                          AiUInt8 biu,
                                          AiUInt32 label,
                                          AiUInt32 cnt,
                                          TY_API_BC_FW_INSTR *tbl,
                                          AiUInt32 *raddr,
                                          AiUInt32 *line);
```

Purpose:

This function is used to return the Global memory address of the specified label within the BC Instruction Table structure.

Input:

AiUInt32 label

Label ID to search for within the BC Instruction Table structure (tbl).

AiUInt32 cnt

Amount of entries in the BC Instruction Table structure (tbl).

TY_API_BC_FW_INSTR *tbl

BC Instruction Table

Output:

AiUInt32 *raddr

Returning Global memory address (zero if label was not found)

AiUInt32 *line

Returning Line (entry number) where the label was found. (0 = first line)

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.17 ApiCmdBCInstrTblIni

Prototype:

```
AiReturn ApiCmdBCInstrTblIni(AiUInt32 ul_ModuleHandle,
                             AiUInt8 biu)
```

Purpose:

This function is used to initialize the BC Instruction Table mode. For this function, **ApiCmdSysMemLayout** is called internally in order to get the size and start address of the BC Instruction List from the Global memory layout. The Start Address and size of the BC Instruction list area are derived from parameters *mem_info.biu_addr[biu].bc_hip_instr* and *mem_info.biu_size[biu].bc_hip_instr* of structure *TY_API_MEM_INFO*, refer to function **ApiCmdSysMemLayout**).

Input:

none

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.18 ApiCmdBCMFrameDef

Prototype:

```
AiReturn ApiCmdBCMFrameDef(AiUInt32 ul_ModuleHandle,
                           AiUInt8 biu,
                           TY_API_BC_MFRAME *pmframe);
```

Purpose:

This function is used to define the sequence of Minor Frames within the Major Frame. The BC Minor Frames shall be defined previously using the **ApiCmdBCFrameDef** function.

Input:

`TY_API_BC_MFRAME *pmframe`

BC Major Frame description

```
typedef struct ty_api_bc_mframe {
  AiUInt8 cnt;
  AiUInt8 fid[MAX_API_BC_MFRAME];
}
TY_API_BC_MFRAME;

#define MAX_API_BC_MFRAME 64
```

`AiUInt8 cnt`

Value	Description
1..64	Amount of Minor Frames in Major Frame

`AiUInt8 fid[]`

Value	Description
1..64	Sequence of Minor Frame Identifiers

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.19 ApiCmdBCMFrameDefEx

Prototype:

```
AiReturn ApiCmdBCMFrameDefEx(AiUInt32 ul_ModuleHandle,
                             AiUInt8 biu,
                             TY_API_BC_MFRAME_EX *pmframe);
```

Purpose:

This function is used to define the sequence of Minor Frames within the Major Frame. The BC Minor Frames shall be defined previously using the **ApiCmdBCFrameDef** function.

Input:

TY_API_BC_MFRAME_EX *pmframe

BC Major Frame description

```
typedef struct ty_api_bc_mframe_ex {
  AiUInt16 cnt;
  AiUInt16 padding1;
  AiUInt8  fid[MAX_API_BC_MFRAME_EX];
}
TY_API_BC_MFRAME_EX;

#define MAX_API_BC_MFRAME_EX 512
```

AiUInt16 cnt

Value	Description
1..512	Amount of Minor Frames in Major Frame

AiUInt16 padding1

0 (reserved)

AiUInt8 fid[]

Value	Description
1..512	Sequence of Minor Frame Identifiers

Output:

none

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.20 ApiCmdBCModeCtrl

Prototype:

```
AiReturn ApiCmdBCModeCtrl(AiUInt32 ul_ModuleHandle,
                          AiUInt8 uc_Biu,
                          TY_API_BC_MODE_CTRL *px_BcModeCtrl);
```

Purpose:

This function is used to enable / disable various BC functionality on-the-fly.

Input:

TY_API_BC_MODE_CTRL *px_BcModeCtrl

BC Mode Control description

```
typedef struct ty_api_bc_mode_ctrl {
  AiUInt32 ul_BcMode;
  AiUInt32 ul_Ctrl;
  AiUInt32 ul_Param1;
  AiUInt32 ul_Param2;
  AiUInt32 ul_Param3;
}
TY_API_BC_MODE_CTRL;
```

AiUInt32 ul_BcMode

Value	Constant	Description
1	**API_BC_MODE _INSERT_MC17 _SYNC_CNT**	If enabled this mode makes the BC inserting the synchronization counter in the data word of a mode code 17 (Synchronize with Data). The mode code transfer to be handled is given with the transfer id in parameter 'ul_Param1'. →see also the commands : **ApiCmdSyncCounterSet** **ApiCmdSyncCounterGet** **Note:** This mode is not available on devices with a multi channel firmware. Please see chapter "Limitations for specific boards" for details.
2	**API_BC_MODE _STOP_ON_MC0 _DBCA_FLAG**	If enabled this mode makes the BC stopping if an RT sets the DBCA flag in its status word in reaction of a previously sent mode code 0 (Dynamic Bus Control). The mode code transfer to be handled is given with the transfer id in parameter 'ul_Param1'.

3	**API_BC_MODE_CONFIGURED_DYTAGS**	This mode is used to enable / disable all configured BC dytags **Note:** This mode is not available on embedded devices! (see also chapter B.1.7 "Limitations for embedded board variants")

AiUInt32 ul_Ctrl

Value	Constant	Description
0	**API_DIS**	Disable the functionality referenced with parameter 'ul_BcMode'
1	**API_ENA**	Enable the functionality referenced with parameter 'ul_BcMode'

AiUInt32 ul_Param1

ul_BcMode	Value	Description
1, 2	**n**	BC Transfer ID **Note:** See Section 1.3.5 for the range allowed for this parameter.
3	**0**	Reserved

AiUInt32 ul_Param2

0 (reserved)

AiUInt32 ul_Param3

0 (reserved)

Output:

none

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.21 ApiCmdBCSrvReqVecCon

Prototype:

```
AiReturn ApiCmdBCSrvReqVecCon(AiUInt32 ul_ModuleHandle,
                              AiUInt8 uc_Biu,
                              AiUInt8 uc_RtAddr,
                              TY_API_BC_SRVW_CON *px_SrvReqVecCon);
```

Purpose:

This function is used to set the sub address where the modecode “Last Vector Word” is sent to in case of a service request handling. This command should only be applied if the BC is enabled to perform Service Request / Vector Word handling (refer to ApiCmdBCIni command).

Input:

AiUInt8 uc_RtAddr

Value	Description
0..31	Remote Terminal Address

TY_API_BC_SRVW_CON *px_SrvReqVecCon

BC Service Request Control description

```
typedef struct ty_api_bc_srvw_con {
  AiUInt8 uc_SubAddress;
  AiUInt8 uc_Padding1;
  AiUInt16 uw_Padding2;
  AiUInt32 ul_Reserved1;
  AiUInt32 ul_Reserved2;
  AiUInt32 ul_Reserved3;
}
TY_API_BC_SRVW;
```

AiUInt8 uc_SubAddress

Value	Constant	Description
0	**API_SEND_SRVW_ON_SA0**	Send modecode “Last Vector Word” on RT sub address 0
1	**API_SEND_SRVW_ON_SA31**	Send modecode “Last Vector Word” on RT sub address 31

AiUInt8 uc_Padding1

Reserved (0)

AiUInt16 uw_Padding2

Reserved (0)

`AiUInt32 ul_Reserved1`

Reserved (0)

`AiUInt32 ul_Reserved2`

Reserved (0)

`AiUInt32 ul_Reserved3`

Reserved (0)

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.22 ApiCmdBCSrvReqVecStatus

Prototype:

```
AiReturn ApiCmdBCSrvReqVecStatus(AiUInt32 ul_ModuleHandle,
                                  AiUInt8 uc_Biu,
                                  AiUInt8 uc_RtAddr,
                                  TY_API_BC_SRVW *px_SrvReqVecStatus);
```

Purpose:

This function is used to read the Service Request and Vector Word Status information the BC has maintained for a specific Remote Terminal. This command should only be applied if the BC is enabled to perform Service Request / Vector Word handling (refer to **ApiCmdBCIni** command).

Input:

AiUInt8 uc_RtAddr

Value	Description
0..31	Remote Terminal Address

Output:

TY_API_BC_SRVW *psrvw

BC Service Request and Vector Word Status description

```
typedef struct ty_api_bc_srvw {
  AiUInt16 uw_XferId;
  AiUInt16 uw_LastVecWord;
  AiUInt32 ul_SrvReqCnt;
}
TY_API_BC_SRVW;
```

Note:
'uw_XferId', 'uw_LastVecQord' and 'ul_SrvReqCnt' are set to zero whenever the ApiCmdBCStart command is executed!

AiUInt16 uw_XferIid

Last BC Transfer ID

Note:
See Section 1.3.5 for the range allowed for this parameter.

(value 0 means: no transaction)

Note:
this value does not change if 'normal' Service Request Handling is used and the parameter sxh of the command ApiCmdBcXferDef was set to 'API_BC_SRVW_ENABLE'. This value is for rare special cases.

AiUInt16 uw_LastVecWord

Last Vector Word received from the specified RT

AiUInt32 ul_SrvReqCnt

Amount of Service Requests issued by the specified RT

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

Notice: The Target Software implements the following Service Request / Vector Word handling in BC mode: When the BC detects a service request from a RT (Service Request Bit set in RT Status response) a Transmit Vector Word Mode Code command is transferred to the requesting RT. As response on the mode code the RT transmits a Vector Word to the BC, which is interpreted and handled by the BC following the description given in the 'AVS Databus Usage Report R-J-403-V-1209 Par. 7.2.'.

The Vector Word comprises a 4-bit identification code informing the BC about the service request reason and specifying the action the BC has to perform. Besides the identification code, the vector word defines the RT address and the Subaddress / HS Message Identifier of the service requesting RT.

Bit 15..12	Bit 11..7	Bit 6..0
Id – Code	RT-Address	Subaddress / MID

The Vector Word Id – Codes requiring BC transactions are handled by the Target Software as follows:

Request Single RX/TX (1000/1001):
The related BC transaction is executed once immediately (acyclic) after receipt and decoding of the vector word.

Request Multiple RX/TX (1010/1011):
The related BC transaction is enabled by appending it to the end of the current BC minor frame (Cyclic execution until disabled).

Delete RX/TX (1100/1101):
The related BC transaction is disabled by deleting it from the end of the related BC minor frame. Re-enabling requires receipt of the corresponding vector word code (Request Multiple RX/TX 1010/1011). Only BC transactions which have been enabled by a previous 'Request Multiple RX/TX' vector word can be disabled.

BC transfers which shall be used as 'vector word driven' transactions have to be marked with the **ApiCmdBCXferDef** command (refer to parameter 'sxh') accordingly. Only BC-RT and RT-BC transfer types can be defined to be 'vector word driven transactions'.

7.1.23 ApiCmdBCStart

Prototype:

```
AiReturn ApiCmdBCStart(AiUInt32 ul_ModuleHandle,
                       AiUInt8 biu,
                       AiUInt8 smod,
                       AiUInt32 cnt,
                       AiFloat frame_time,
                       AiUInt32 saddr,
                       AiUInt32 *maj_frame_addr,
                       AiUInt32 min_frame_addr[64]);
```

Purpose:

This function is used to start execution of pre-defined BC Transfers within the minor/major frame structure and to define the minor frame timing.

Input:

AiUInt8 smod

Bus Controller Starting Mode

For Standard Framing Mode:		
Value	Constant	Description
1	**API_BC_START_IMMEDIATELY**	Start BC operation / Transfer execution immediately
2	**API_BC_START_EXT_TRIGGER**	Start BC operation / Transfer execution by external BC trigger input
3	**API_BC_START_RT_MODECODE**	Start BC operation / Transfer execution by RT Modecode Dynamic Bus Control
4	**API_BC_START_EXTERNAL_PULSE**	Drive minor framing from external pulse (BC trigger input)
5	**API_BC_START_SETUP**	Prepare BC Instruction sequence (= Minor and Major Frame sequence) in the Board Global memory for a fast start with API_BC_START_FAST.

Value	Constant	Description
6	**API_BC_START_FAST**	Start BC operation / Transfer execution immediately fast. Note that API_BC_START_SETUP and API_BC_START_FAST shall be used in conjunction. API_BC_START_FAST after running BC was stopped with **ApiCmdBCHalt** would also be possible. For this mode all other **ApiCmdBCStart** parameters are not applicable.

For BC Instruction Table Mode:

Value	Constant	Description
7	**API_BC_START_INSTR_TABLE**	Start BC operation / Transfer execution with predefined BC Instruction Table (refer to **ApiCmdBCInstrGen**)
8	**API_BC_START_CONTINUE_ON_BC_HALT**	Continue BC operation (only applicable with predefined BC Instruction Table, BC operation must be already started with API_BC_START_INSTR_TABLE)

Note:
Value 8 is not available on devices with a multi channel firmware. Please see chapter “Limitations for specific boards” for details.

`AiUInt32 cnt`

Value	Description
0	Cyclic execution of the transfers within the Major Frame
>0	cnt-times execution of the transfers within the Major Frame

`AiFloat frame_time`

‘smod’	Value	Description
1..3,7..8	**0.001..1048.576**	Minor Frame Time in 0.001 ms steps (Range: 0.001..1048.576 ms)
4..6	**0**	-

`AiUInt32 saddr`

‘smod’	Value	Description
1..6	**0**	Not applicable
7	**0**	Start from beginning of BC Instruction List area
	>0	Start from specified address (returned by function ApiCmdBCInstrTblGetAddrFromLabel)
8	**0**	n/a
	>0	Continue BC operation at specified address

Output:

`AiUInt32 *maj_frame_addr`

Start address of the major frame relative to the start of the Global Memory

`AiUInt32 min_frame_addr[]`

Start addresses of the minor frames relative to the start of the Global Memory

> **Note:**
> This Address values are only returned for API_BC_START_SETUP. The Minor Frame sequence is dependent from the call of the function ApiCmdBCMFrameDef. If less Minor Frames are used than maximum is possible in the Major Frame, then zero is returned.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.24 ApiCmdBCStatusRead

Prototype:

```
AiReturn ApiCmdBCStatusRead(AiUInt32 ul_ModuleHandle,
                            AiUInt8 biu,
                            TY_API_BC_STATUS_DSP *pdsp);
```

Purpose:

This function is used to read the actual execution status of the Bus Controller.

Input:

none

Output:

`TY_API_BC_STATUS_DSP *pdsp`

BC Status Information

```
typedef struct ty_api_bc_status_dsp {
  AiUInt8 status;
  AiUInt8 padding1;
  AiUInt16 hxfer;
  AiUInt32 glb_msg_cnt;
  AiUInt32 glb_err_cnt;
  AiUInt32 hip;
  AiUInt32 mfc;
}
TY_API_BC_STATUS_DSP;
```

`AiUInt8 status`

Bus Controller execution status

Value	Constant	Description
1	**API_BC_STATUS_HALTED**	BC halted
2	**API_BC_STATUS_BUSY**	BC busy

`AiUInt8 padding1`

Reserved (0)

`AiUInt16 hxfer`

BC HALT on Interrupt Transfer Identifier

Value	Description
0	BC halted normally
1.. n	BC halted by Interrupt at Transfer Identifier (hxfer) See Section 1.3.5 for the range allowed for this parameter

`AiUInt32 glb_msg_cnt`

Global BC Message Counter

`AiUInt32 glb_err_cnt`

Global BC Error Counter

Note:
'glb_msg_cnt' and 'glb_err_cnt' are cleared on restart of the AIM board Bus Controller when the ApiCmdBCStart' function is called.

`AiUInt32 hip`

BC Instruction List Pointer (address value relative to the start of the Global Memory)

`AiUInt32 mfc`

Actual BC Major Frame Counter (= decrement value, only applicable when BC operation was started non-cyclic, e.g. ApiCmdBCStart with parameter 'cnt' >0)

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.25 ApiCmdBCTrafficOverrideControl

Prototype:

```
AiReturn ApiCmdBCTrafficOverrideControl(AiUInt32 ul_ModuleHandle,
                                        AiUInt8 biu, AiUInt32 con);
```

Purpose:

This function can be used to enable or disable the bus idle check of the BC. If enabled the BC can send transfers regardless of the idle state of the bus. Be aware that enabling this feature can produce bus collisions.

The default state is disabled. The functions ApiCmdReset and ApiCmdBCInit set this feature to the default state disabled. This function must be called after ApiCmdBCIni and before ApiCmdBCStart.

The function ApiCmdSysGetBoardInfo can be used to check if this feature is available for your board

Note:
This function is not available on all devices. See chapter "Limitations for specific boards".

Input:

AiUInt8 con

Traffic Override Control

Value	Constant	Description
0	**API_DIS**	Disabled
1	**API_ENA**	Enabled

Output:

none

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.26 ApiCmdBCTrafficOverrideGet

Prototype:

```
AiReturn ApiCmdBCTrafficOverrideGet(AiUInt32 ul_ModuleHandle,
                                    AiUInt8 biu, AiUInt32 * con);
```

Purpose:

This function can be used to read the enable or disable state of the BC bus idle check.

Note:
This function is not available on all devices. See chapter "Limitations for specific boards".

Input:

none

Output:

`AiUIn32 * con`

Traffic Override Control

Value	Constant	Description
0	**API_DIS**	Disabled
1	**API_ENA**	Enabled

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.27 ApiCmdBCXferCtrl

Prototype:

```
AiReturn ApiCmdBCXferCtrl(AiUInt32 ul_ModuleHandle,
                          AiUInt8 biu,
                          AiUInt16 xid, AiUInt8 mode);
```

Purpose:

This function is used to enable/disable the specified BC Transfer.

Input:

`AiUInt16 xid`

BC Transfer ID

Note:
See Section 1.3.5 for the range allowed for this parameter.

`AiUInt8 mode`

BC Transfer Control

Value	Constant	Description
0	**API_DIS**	Disable BC Transfer (insert NOP in BC Transfer descriptor)
1	**API_ENA**	Enable BC Transfer (restore original BC Transfer type)

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.28 ApiCmdBCXferDef

Prototype:

```
AiReturn ApiCmdBCXferDef(AiUInt32 ul_ModuleHandle,
                         AiUInt8 biu,
                         TY_API_BC_XFER *pxfer,
                         AiUInt32 *desc_addr);
```

Purpose:

This function is used to define all transfer properties of a Bus Controller Transfer including source / destination information, error injection specifications, and interrupt generation.

Note:
The Status Word Exception Mask swxm is used to mask (AND mask) the Status Word response of the terminal (both terminals for RT-RT transfer type). If the result of the mask process is not zero an Interrupt on Status Word Exception is asserted if enabled. Unexpected responses are not counted as errors and will not cause an error interrupt. The terminal address field of the received Status Word is checked automatically and causes a Transfer Error on mismatch. In this case error counting and Interrupt on Transfer Error assertion is enabled.

Input:

TY_API_BC_XFER *pxfer

BC Transfer description

```
typedef struct ty_api_bc_xfer {
  AiUInt16 xid;
  AiUInt16 hid;
  AiUInt8 type;
  AiUInt8 chn;
  AiUInt8 xmt_rt;
  AiUInt8 rcv_rt;
  AiUInt8 xmt_sa;
  AiUInt8 rcv_sa;
  AiUInt8 wcnt;
  AiUInt8 tic;
  AiUInt8 hlt;
  AiUInt8 rte;
  AiUInt8 res;
  AiUInt8 sxh;
  AiUInt8 rsp;
  AiUInt8 gap_mode;
  AiUInt16 swxm;
  struct ty_api_bc_err err;
  AiUInt16 gap;
}
TY_API_BC_XFER;
```

AiUInt16 xid

BC Transfer ID

> **Note:**
> See Section 1.3.5 for the range allowed for this parameter.

AiUInt16 hid

Buffer Header ID

> **Note:**
> See Section 1.3.5 for the range allowed for this parameter.

AiUInt8 type

Bit	Value	Constant	Description
7..6	**0**		reserved for Service Request / Vector Word handling
5..2	**0**		reserved
1..0	**0**	**API_BC_TYPE_BCRT**	BC to RT transfer type
	1	**API_BC_TYPE_RTBC**	RT to BC transfer type
	2	**API_BC_TYPE_RTRT**	RT to RT transfer type

AiUInt8 chn

Transfer Channel

Value	Constant	Description
0	**API_BC_XFER_BUS_PRIMARY**	Primary Bus
1	**API_BC_XFER_BUS_SECONDARY**	Secondary Bus

AiUInt8 xmt_rt

Value	Description
0..31	RT Address of transmitting terminal (31 = broadcast RT address)

AiUInt8 rcv_rt

Value	Description
0..31	RT Address of receiving terminal (31 = broadcast RT address)

AiUInt8 xmt_sa

Value	Description
0..31	Subaddress of transmitting terminal (0, 31 = Mode code Subaddress)

AiUInt8 rcv_sa

Value	Description
0..31	Subaddress of receiving terminal (0, 31 = Mode code Subaddress)

AiUInt8 wcnt

Value	Description
0	Word Count / Mode code field for Command Word is 32
1..31	Word Count / Mode code field for Command Word

AiUInt8 tic

Transfer Interrupt Control

Value	Constant	Description
0	**API_BC_TIC_NO_INT**	No Interrupt
1	**API_BC_TIC_INT_ON_XFER_END**	Interrupt on End of Transfer
2	**API_BC_TIC_INT_ON_XFER_ERR**	Interrupt on Transfer Error
3	**API_BC_TIC_INT_ON_STAT_EXCEPT**	Interrupt on Status Word Exception
4..7	**API_BC_TIC_NO_INT**	Reserved

AiUInt8 hlt

BC Halt Control

Value	Constant	Description
0	**API_BC_HLT_NO_HALT**	Do not halt on exception
1	**API_BC_HLT_HALT_ON_XFER_ERR**	Reserved for Halt on Transfer Error
2	**API_BC_HLT_HALT_ON_STAT_EXCEPT**	Reserved for Halt on Status Word Exception
3	**API_BC_HLT_HALT_ON_ERR_EXCEPT**	Reserved for Halt on Transfer Error and Status Word Exception
4	**API_BC_HLT_HALT_ON_ANY_INT**	Halt on any Interrupt
5..7		Reserved

AiUInt8 rte

Retry enable (see also parameter 'retr' of command ApiCmdBCIni) and toggle bus control

Value	Constant	Description
0	**API_DIS**	Retry disabled
1	**API_ENA**	Retry enabled
2	**API_TOGGLE_BUS_KEEP**	Keep bus at next transmission of this transfer.
3	**API_TOGGLE_BUS_ALT**	Usa alternate bus at next transmission of this transfer

Note:
An individual transfer retry (value 0 or 1) can only be enabled/disabled if global BC retry is enabled first with ApiCmdBCIni. The retry protocol followed will be the protocol defined with ApiCmdBCIni.

AiUInt8 res

0 (Reserved)

AiUInt8 sxh

Status Word Exception handling of Service Request control (only applicable when Service Request / Vector Word Mode Control is enabled using **ApiCmdBCIni** command). See **ApiCmdBCSrvReqVecStatus** for further information.

Value	Constant	Description
0	**API_BC_SRVW_DIS**	No Status Word Service Request handling
1	**API_BC_SRVW_ENA**	Generate automatic Transmit Vector Word mode code (Mode code 16) if Service Request Bit of the received Status Word is set Note: Automatic Mode code generation on Service Request Bit will only be processed if no error is detected and no fast Intermessage Gap mode is enabled (refer to 'gap_mode'). For Automatic Mode code generation on Service Request Bit the first received Status Word is evaluated, thus this function is only applicable for BC-RT and RT-BC transfer types (refer to 'type').
2	**API_BC_SRVW_SINGLE**	Single RX/TX transaction Only applicable for BC-RT and RT-BC transfer types. (refer to 'type'). This transfer will define the Single RX/TX transmission to be used if requested in the RT Vector Word. Since such transfer is inserted after decoding of a previous Transmit Vector Word mode code 'xid' shall not be used in the ApiCmdBCFrameDef command
3	**API_BC_SRVW_MULTIPLE**	Multiple / Delete RX/TX transaction Only applicable for BC-RT and RT-BC transfer types (refer to 'type'). This transfer will define the Multiple RX/TX transmission to be used if requested in the RT Vector Word. Since such transfer is inserted after decoding of a previous Transmit Vector Word mode code 'xid' shall not be used in the ApiCmdBCFrameDef command

Note:
Values 2 and 3 are not applicable, when using the functions ApiCmdBCInstrTblIni, ApiCmdBCInstrTblGen and ApiCmdBCInstrTblGetAddrFromLabel!!!

AiUInt8 rsp

Expected Response Control for Command Word

Value	Constant	Description
0	**API_BC_RSP_AUTOMATIC**	Automatic Response Control In this mode BC expects appropriate RT-Response to the defined Transfer Type, dependent on the MIL-STD 1553 protocol type(s), which is (are) defined for the RT('s) with the function ApiCmdDefMilbusProtocol
1	**API_BC_RSP_NO_SW1_EXPECTED**	No Status Word 1 expected (Note: for RT-RT transfer type Status Word 1 represents the status response for the Transmit Command word)
2	**API_BC_RSP_NO_SW2_EXPECTED**	No Status Word 2 expected (RT-RT transfer type only, Note: for RT-RT transfer type Status Word 2 represents the status response for the Receive Command word)

`AiUInt8 gap_mode`

Three different Gap Modes are available which are described after the following table:

Value	Constant	Description
0	**API_BC_GAP_MODE_DELAY**	Delay mode / Transfer Wait Time mode
1	**API_BC_GAP_MODE_STANDARD**	Standard Gap Mode
2	**API_BC_GAP_MODE_FAST**	Fast Intermessage Gap Mode

Delay mode / Transfer Wait Time Mode

In this mode '**gap**' (14bit) specifies the time from the start of a transfer to the start of the following transfer in μs.

> **Note:**
> If the specified time is shorter than the actual time needed for transfer transmission, 'gap' is ignored and the Bus Controller automatically generates a minimum intermessage gap of app. 11μs.

This gap mode guarantees deterministic framing independent from RT response times and provides excellent repeatability of timed bus events.

The nominal Transfer Wait Time is calculated as:

$$\mathbf{T_D = \text{'gap'} * 1\mu s}$$

Standard Gap Mode

In Standard Gap Mode '**gap**' (14bit) specifies the wait gap between the end of the current transfer and start of the following transmitted Command Word. The nominal Intermessage Gap Time is calculated as:

$$\mathbf{T_{IG} = \text{`gap'} * 1\mu s}$$

Note:
The idle time of the physical bus is 2.0μs less than this time due to the gap measurement definition of MIL-STD-1553B.

Note:
The minimum Intermessage gap time achieved using the Standard Gap Mode is equal to the minimum gap time achieved in the Transfer Wait Time Mode.

Note:
At broadcast transfers, the response timeout is used to check that no status word is responded. Thus, the Intermessage Gap Time in Standard Gap Mode shall be greater than the response timeout, to guarantee proper operation.

Fast Intermessage Gap Mode

In the Fast Intermessage Gap Mode, the Bus Controller generates Intermessage Gap Times down to 4 μs. In this mode, the value of '**gap**' is limited to 6bits. The nominal Intermessage Gap Time is calculated as:

$$\mathbf{T_{IG} = 4\mu s + (\text{`gap'} * 0.25\mu s) \quad (1\ Mbit\ Transmission\ Mode)}$$

Therefore, Intermessage Gap times between 4μs up to 19.75μs can be achieved in this mode. This mode can be used for RT validation and testing, but shall not be used during normal operation since line busy and word count high checks are disabled on the BC. Also worst case operation is not guaranteed if this mode is combined with Multi RT, monitor and Mailbox Operation. The Fast Intermessage Gap Mode shall only be used for transfer instructions which are followed by another transfer instruction. In case of a transfer error, the Fast Intermessage Gap Mode is switched off for this transfer and a default Intermessage gap of app. 11μs is generated.

Note:
The idle time of the physical bus is 2.0μs less than this time due to the gap measurement definition of MIL-STD-1553B.

Note:
This mode is not supported on ASC1553-1

`AiUInt16 swxm`

Status Word Exception Mask

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
reserved (0)					MERR	INSTR	SREQ

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
reserved (0)			BRCV	BUSY	SUB	DBCA	TERM

MERR Message Error Bit
INSTR Instrumentation Bit
SREQ Service Request
BRCV Broadcast Received Bit
BUSY Busy Bit
SUB Subsystem Flag
DBCA Dynamic Bus Control Acceptance
TERM Terminal Flag

TY_API_BC_ERR err

BC Transfer Error Injection specifications

```
typedef struct ty_api_bc_err {
  AiUInt8 type;
  AiUInt8 sync;
  AiUInt8 contig;
  AiUInt8 padding1;
  AiUInt32 err_spec;
}
TY_API_BC_ERR;
```

AiUInt8 type
Error Type

Value	Constant	Description
0	**API_ERR_TYPE_NO_INJECTION**	No error injection
1	**API_ERR_TYPE_COMMAND_SYNC**	Command Sync Error with Sync Pattern in 'sync'
2	**API_ERR_TYPE_DATA_SYNC**	Data Sync Error with Sync Pattern in 'sync' in Word 'wpos'
3	**API_ERR_TYPE_PARITY**	Parity Error in Word 'wpos'
4	**API_ERR_TYPE_MANCHESTER_HIGH**	Manchester Stuck at High Error in Word 'wpos' at Bit Position bpos'
5	**API_ERR_TYPE_MANCHESTER_LOW**	Manchester Stuck at Low Error in Word 'wpos' at Bit Position bpos'
6	**API_ERR_TYPE_GAP**	Gap Error with Gap defined in 'contig' in Word 'wpos'
7	**API_ERR_TYPE_WORD_CNT_HIGH**	Word Count High with count in 'wpos'
8	**API_ERR_TYPE_WORD_CNT_LOW**	Word Count Low with count in 'wpos'
9	**API_ERR_TYPE_BIT_CNT_HIGH**	Bit Count High Error in Word 'wpos' (+ 'bc_bits')

10	**API_ERR_TYPE_BIT_CNT_LOW**	Bit Count Low Error in Word 'wpos' (- 'bc_bits')
11		Reserved
12	**API_ERR_TYPE_ZERO_CROSS_NEG**	Zero Crossing Low Deviation Error in word 'WPOS' at bit position 'BPOS' with negative deviation in 'contig'
13	**API_ERR_TYPE_ZERO_CROSS_POS**	Zero Crossing High Deviation Error in word 'WPOS' at bit position 'BPOS' with positive deviation in 'contig'
14		Reserved
15	**API_ERR_TYPE_BUS_SWITCH**	Bus Switching Support The BIU disables the physical decoder device on the alternate bus, if the current transfer is in fast gap mode (RT-Validation Test 5.2.1.8)

> **Note:**
> The error type 9 (Bit Count High) is not supported together with Modecodes without data.

> **Note:**
> Limitation for error type 1, 2, 4, 5, on some boards. Please see chapter B.1.3.

> **Note:**
> This function is not available on all devices. See chapter "Limitations for specific boards".

AiUInt8 sync

Sync Field Error Half-Bit-Pattern (6 LS-Bits)
(38hex = 111000 Sync Pattern)

AiUInt8 contig

'type'	Value	Description
Gap Error	**1..15**	Low Speed Gap Error Half Bits
Zero Crossing Error	**0**	125ns deviation
	1	187.5ns deviation
	2	156.25ns deviation
	3	218.75ns deviation **Note:** Values 2 and 3 are not available for all AIM devices. Please check the chapter "Board functionality overview" and look for "Error Injection of High Resolution Zero Crossing Deviation".

> **Note:**
> The resolution is:
> 1Mbit: 0.5μs per Half Bit

AiUInt8 padding1

Reserved (0)

AiUInt32 err_spec

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
reserved (0)							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
WPOS							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
BPOS							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BCBITS							

WPOS

Value	Description
0..32	Error Word Position (0 = Command Word)

BPOS

Value	Description
4..20	Error Bit Position (20 =Parity Bit)

BCBITS

Value	Description
1..3	Amount of Bit Count Error Bits

`AiUInt16 gap`

'gap_mode'	Value	Description
0 or 1	**0..16383**	Transfer Wait Time in μs
2	**0..63**	Transfer Wait Time It is calculated as follows: $\mathbf{Time} = \mathbf{4}\boldsymbol{\mu}\mathbf{s} + (\mathbf{Value} * \mathbf{0.25}\boldsymbol{\mu}\mathbf{s})$

See the description of 'gap_mode' for more detailed information.

Output:

`AiUInt32 *desc_addr`

26-bit Transfer Descriptor Address (Byte-Address relative to the start of the Global RAM)

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

RT-Validation Test Plan Support

The BC provides some special transfer error conditions, due to the further BC support of the MIL-HDBK-1553A/B 'RT Validation TestPlan'. Therefore there are provided two special error modes for dedicated validation test support:

5.2.1.4 Superceding CMD, Test D Support

At test D, after the last data word of the previous BC-RT transfer a valiud legal transmit command shall follow contiguously. This test can only be stimulated by the BC, it two error conditions are inserted simultaneously at the BC-RT transfer. Set the following parameters:

- err.type = API_ERR_TYPE_DATA_SYNC
- err.err_spec.WPOS = one entry greater than the word count of the transfer
- err.sync = 0x38 (111000b Sync Pattern)
- err.config = 0

Thus, the BC inserts automatically a word count high error and the sync error is inserted at the additional word, which shall be appropriate set in the buffer, to generate the expected command word.

Attention for transfers with 32 data words:

Since the maximum buffer size for AIM boards is 32 words, the following data buffer id (relating to this transfer) has to be used to define the additional word. This data buffer id then should not be used for other transfers.

5.2.1.8 Bus Switching Test Support

At this test, the BC sends a second command on the alternate bus, while the first message is not completed. Due to the single encoder/decoder hardware implementation on the AIM product family, this test can only be handled by the firmware, if the decoder device on the bus where the first command is sent will be disabled for the duration of the second message.

Therefore the first command (RT-BC with 32 words) shall be setup as follows:

- rsp = API_BC_RSP_NO_SW1_EXPECTED
- gap_mode = API_BC_GAP_MODE_FAST

Then, the second command shall be initialized with :

- err.type = API_ERR_TYPE_BUS_SWITCH

7.1.29 ApiCmdBCXferDefErr

Prototype:

```
AiReturn ApiCmdBCXferDefErr(AiUInt32 ul_ModuleHandle,
                            AiUInt8 biu,
                            AiUint16 xid,
                            TY_API_BC_ERR * pxError);
```

Purpose:

This function can be used to modify a BC transfer descriptor error injection on the fly.

Note: This function is not available on all devices. See chapter “Limitations for specific boards”.

Input:

`AiUInt16 xid`

BC Transfer ID

`TY_API_BC_ERR * pxError`

See description of ApiCmdBCXferDef for a detailed description of supported error modes.

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.30 ApiCmdBCXferDescGet

Prototype:

```
AiReturn ApiCmdBCXferDescGet(AiUInt32 ul_ModuleHandle,
                             AiUInt8 biu,
                             AiUInt16 xid,
                             AiUInt32 mode,
                             AiUInt32 value[4]);
```

Purpose:

This function can be used to read several BC transfer descriptor values on the fly.

Input:

`AiUInt16 xid`

BC Transfer ID

`AiUInt32 mode`

See description of ApiCmdBCXferDescMod for supported modes.

Output:

`AiUInt32 value[4]`

See description of ApiCmdBCXferDescMod for supported values.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.31 ApiCmdBCXferDescMod

Prototype:

```
AiReturn ApiCmdBCXferDescMod(AiUInt32 ul_ModuleHandle,
                             AiUInt8 biu,
                             AiUInt16 xid,
                             AiUInt32 mode,
                             AiUInt32 value[4]);
```

Purpose:

This function can be used to modify several BC transfer descriptor values on the fly.

Input:

`AiUInt16 xid`

BC Transfer ID

`AiUInt32 mode`

Define	Description
XFER_DESC_FIELD_CHN	Modify channel
XFER_DESC_FIELD_CW1	Modify command word 1
XFER_DESC_FIELD_CW2	Modify command word 2
XFER_DESC_FIELD_TRTYPE	Modify transfer type
XFER_DESC_FIELD_CW1_CW2	Modify command word 1 and 2

`AiUInt32 value[4]`

Define	Index	Value
XFER_DESC_FIELD_CHN	0	Channel 0=pri; 1=sec
XFER_DESC_FIELD_CW1	0	Command word 1
XFER_DESC_FIELD_CW2	0	Command word 2
XFER_DESC_FIELD_TRTYPE	0	0=BC2RT; 1=RT2BC; 2=RT2RT; 3=NOP
	1	Command word 1
	2	Command word 2
XFER_DESC_FIELD_CW1_CW2	0	Command word 1
	1	Command word 2

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

7.1.32 ApiCmdBCXferRead

Prototype:

```
AiReturn ApiCmdBCXferRead(AiUInt32 ul_ModuleHandle,
                          AiUInt8 biu,
                          AiUInt16 xid,
                          AiUInt16 clr,
                          TY_API_BC_XFER_DSP *pxfer_dsp);
```

Purpose:

This function is used to read the status of an individual Bus Controller Transfer.

Input:

AiUInt16 xid

BC Transfer ID

Note:
See Section 1.3.5 for the range allowed for this parameter.

AiUInt16 clr

Buffer Status Flag Control

Reset Flags

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
0 (reserved)							

Entry Type to find

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 (reserved)					SR	ERR	STAT

STAT If set to 1: Reset Buffer Status bits to ‘Buffer not used’ state
ERR If set to 1: Reset Buffer Error bits to ‘No error’ state
SR If set to 1: Reset ‘Last Vector Word’ and ‘Service Request Counter’

Output:

TY_API_BC_XFER_DSP *pxfer_dsp

BC Transfer Status information

```
typedef struct ty_api_bc_xfer_dsp {
  AiUInt16 cw1;
  AiUInt16 st1;
  AiUInt16 cw2;
  AiUInt16 st2;
  AiUInt16 bid;
```

```
  AiUInt16 brw;
  AiUInt32 bufp;
  AiUInt32 ttag;
  AiUInt32 msg_cnt;
  AiUInt32 err_cnt;
  AiUInt32 lvw;
  AiUInt32 srvreq_cnt;
}
TY_API_BC_XFER_DSP;
```

AiUInt16 cw1

Command Word 1

AiUInt16 st1

Last received Status Word on Command Word 1

AiUInt16 cw2

Command Word 2 (RT-RT Transfers)

AiUInt16 st2

Last received Status Word on Command Word 2 (RT-RT Transfers)

AiUInt16 bid

Value	Description
0..2047	Current Buffer Index

AiUInt16 brw

Buffer Report Word

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
0 (reserved)		BUF_STAT		0 (reserved)			

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 (reserved)			RBUS	ERR_REPORT			

BUF_STAT

Buffer Status

Value	Constant	Description
0	**API_BUF_NOT_USED**	Buffer not used
1	**API_BUF_FULL**	Buffer full
2	**API_BUF_EMPTY**	Buffer empty
3		reserved

RBUS

Received Bus Flag

Value	Constant	Description
0	**API_RCV_BUS_SECONDARY**	Secondary Bus
1	**API_RCV_BUS_PRIMARY**	Primary Bus

ERR_REPORT

Error Report Field

Value	Constant	Description
0	**API_BC_REPORT_NO_ERR**	No error
1	**API_BC_REPORT_ERR_FRAME_COD**	Coding of Framing error
2	**API_BC_REPORT_ERR_SW1_NOT_RCV**	Status Word 1 not received
3	**API_BC_REPORT_ERR_SW2_NOT_RCV**	Status Word 2 not received
4	**API_BC_REPORT_ERR_SW_EXCEPTION**	Status Word Exception detected

`AiUInt32 bufp`

Current Data Buffer Pointer (Byte-Address)

`AiUInt32 ttag`

32-bit Time Tag Word

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
MINUTES_OF_HOUR (0..59)						SEC_OF_MIN	

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
SEC_OF_MINUTE (0..59)				MICROSEC_OF_SEC			

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
MICROSEC_OF_SEC (0..999999)							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MICROSEC_OF_SEC (0..999999)							

`AiUInt32 msg_cnt`

Number of Transfers executed

`AiUInt32 err_cnt`

Number of Transfer errors detected

`AiUInt32 lvw`

Last vector word that has been received for this transfer This value is only available, if Service Request / Vector Word mechanism is enabled with the functions ApiCmdBCIni() and ApiCmdBCXferDef().

`AiUInt32 srvreq_cnt`

Number of service requests detected for this transfer This value is only available, if Service Request / Vector Word mechanism is enabled with the functions ApiCmdBCIni() and ApiCmdBCXferDef().

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

Note:
'msg_cnt', 'err_cnt', 'lvw' and 'srvreq_cnt' are cleared on restart of the AIM board Bus Controller when the ApiCmdBCStart' function is called.

7.1.33 ApiCmdBCXferReadEx

Prototype:

```
AiReturn ApiCmdBCXferReadEx(AiUInt32 ul_ModuleHandle,
                            TY_API_BC_XFER_READ_IN *px_XferReadIn,
                            TY_API_BC_XFER_STATUS_EX *px_XferStat);
```

Purpose:

This function is used to read the status of an individual Bus Controller Transfer, including status information

Input:

TY_API_BC_XFER_READ_IN *px_XferReadIn

BC Transfer Status input information

```
typedef struct ty_api_hs_bc_xfer_read_in {
  AiUInt32 ul_XferId;
  AiUInt32 ul_Clear;
  AiUInt32 ul_Biu;
}
TY_API_BC_XFER_READ_IN;
```

AiUInt32 ul_XferId

BC Transfer ID

Note:
See Section 1.3.5 for the range allowed for this parameter.

AiUInt32 ul_Clear

Buffer Status Flag Control

Reset Flags

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
0 (reserved)							

Entry Type to find

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 (reserved)					SR	ERR	STAT

STAT If set to 1: Reset Buffer Status bits to ‘Buffer not used’ state
ERR If set to 1: Reset Buffer Error bits to ‘No error’ state
SR If set to 1: Reset ‘Last Vector Word’ and ‘Service Request Counter’

AiUInt32 ul_Biu

BIU number, if board was opened with function ApiOpen(). Otherwise reserved (0).

Output:

TY_API_BC_XFER_STATUS_EX *px_XferStat

BC Transfer Status information

```
typedef struct ty_api_bc_xfer_status_queue {
  AiUInt32 ul_SqCtrlWord;
  AiUInt32 ul_SqStatusWords;
  AiUInt32 ul_SqActBufPtr;
  AiUInt32 ul_SqTimeTag;
}
TY_API_BC_XFER_STATUS_QUEUE;

typedef struct ty_api_bc_xfer_event_queue {
  AiUInt32 ul_EqEntryWord1;
  AiUInt32 ul_EqEntryWord2;
  AiUInt32 ul_EqEntryWord3;
  AiUInt32 ul_EqEntryWord4;
}
TY_API_BC_XFER_EVENT_QUEUE;

typedef struct ty_api_bc_xfer_status_info {
  AiUInt32 ul_ActBufferId;
  AiUInt32 ul_ActionWord;
  AiUInt32 ul_XferCnt;
  AiUInt32 ul_ErrCnt;
  AiUInt32 ul_LastVectorWord;
  AiUInt32 ul_ServiceRequestCnt;
}
TY_API_BC_XFER_STATUS_INFO;

typedef struct ty_api_bc_xfer_status_ex {
  TY_API_BC_XFER_STATUS_QUEUE x_XferSQueue[256];
  TY_API_BC_XFER_EVENT_QUEUE x_XferEQueue[1];
  TY_API_BC_XFER_STATUS_INFO x_XferInfo;
  AiUInt32 ul_BufferQueueSize;
}
TY_API_BC_XFER_STATUS_EX;
```

AiUInt32 ul_BufferQueueSize

Indicates the data buffer queue size set with ApiCmdBCBHDef(). This queue size also indicates how many status queue entries are valid

AiUInt32 X_XferInfo.ul_ActBufferId

Actual Buffer/Status Queue Identifier of the transfer that is currently processed

AiUInt32 X_XferInfo.ul_ActionWord

Reserved (0)

AiUInt32 X_XferInfo.ul_XferCnt

Number of Transfers executed

AiUInt32 X_XferInfo.ul_ErrCnt

Number of Transfer errors

AiUInt32 X_XferInfo.ul_LastVectorWord

Last vector word that has been received
This value is only available, if Service Request / Vector Word mechanism is enabled with the functions ApiCmdBCIni() and ApiCmdBCXferDef().

AiUInt32 X_XferInfo.ul_ServiceRequestCnt

Number of service requests detected
This value is only available, if Service Request / Vector Word mechanism is enabled with the functions ApiCmdBCIni() and ApiCmdBCXferDef().

AiUInt32 x_XferEQueue[].ul_EqEntryWord1

Reserved (0)

AiUInt32 x_XferEQueue[].ul_EqEntryWord2

Reserved (0)

AiUInt32 x_XferEQueue[].ul_EqEntryWord3

Reserved (0)

AiUInt32 x_XferEQueue[].ul_EqEntryWord4

Reserved (0)

AiUInt32 x_XferSQueue[].ul_SqCtrlWord

Status Queue control word

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
reserved		BUF_STAT		reserved			

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
reserved			RBUS	ERR			

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
CMD							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CMD							

BUF_STAT

Buffer Status

Value	Constant	Description
0	**API_BUF_NOT_USED**	Buffer not used
1	**API_BUF_FULL**	Buffer full
2	**API_BUF_EMPTY**	Buffer empty
3		reserved

RBUS

Received Bus Flag

Value	Constant	Description
0	**API_RCV_BUS_SECONDARY**	Secondary Bus
1	**API_RCV_BUS_PRIMARY**	Primary Bus

ERR_REPORT

Error Report Field

Value	Constant	Description
0	**API_BC_REPORT_NO_ERR**	No error
1	**API_BC_REPORT_ERR_FRAME_COD**	Coding or Framing error
2	**API_BC_REPORT_ERR_SW1_NOT_RCV**	Status Word 1 not received
3	**API_BC_REPORT_ERR_SW2_NOT_RCV**	Status Word 2 not received
4	**API_BC_REPORT_ERR_SW_EXCEPTION**	Status Word exception detected
5..15		reserved

CMD

Command Word

`AiUInt32 x_XferSQueue[].ul_ SqStatusWords`

Last received Status Word on Command Word

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
STATUS_WORD_2							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
STATUS_WORD_2							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
STATUS_WORD_1							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STATUS_WORD_1							

`AiUInt32 x_XferSQueue[].ul_ SqActBufPtr`

Actual data buffer address relative to the start of the Global RAM

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
						BUF_PTR	

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
BUF_PTR							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
BUF_PTR							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BUF_PTR							

`AiUInt32 x_XferSQueue[].ul_ SqTimeTag`

32-bit Time Tag Word

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
MINUTES_OF_HOUR (0..59)						SEC_OF_MIN	

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
SEC_OF_MINUTE (0..59)				MICROSEC_OF_SEC			

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
MICROSEC_OF_SEC (0..999999)							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MICROSEC_OF_SEC (0..999999)							

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

Note:
'msg_cnt', 'err_cnt', 'lvw' and 'srvreq_cnt' are cleared on restart of the AIM board Bus Controller when the ApiCmdBCStart' function is called.

8 REMOTE TERMINAL FUNCTIONS

Chapter 8 defines the Remote Terminal function calls of the API S/W Library. The RT functions provide configuration, status and error insertion for RT transfers. The function calls in this table are listed in a functional order, however, the detailed descriptions of the RT function calls in the following sections are in alphabetical order. .

Table 8.1: Remote Terminal Function Descriptions

Function	Description
ApiCmdRTBHDef	Defines an RT Buffer Header to be assigned to an RT SA/Mode code
ApiCmdRTBHRead	Read the RT-SA buffer header structure
ApiCmdRTDytagDef	Defines dynamic data to be inserted into the RT transmit Data words
ApiCmdRTEnaDis	Enables/Disables a selected RT on the fly
ApiCmdRTGetDytagDef	Read the Dytag settings for the generation of dynamic data words for a RT transmit SA
ApiCmdRTGetSABufferHeaderInfo	Get the buffer header id of a certain RT/SA combination
ApiCmdRTGetSAConErr	Read the error injection settings of the specified RT Sub-address/Mode code
ApiCmdRTGetSimulationInfo	Read the simulation and monitoring status of an RT
ApiCmdRTGlobalCon	Initializes multiple RTs at one time (combination of ApiCmdRTIni and ApiCmdRTSACon)
ApiCmdRTHalt	Stops the RT operation for all assigned RTs
ApiCmdRTIni	Initializes a select RT including configuration for simulation/mailbox mode, Response time and Next Status word
ApiCmdRTLCW	Redefines the Last Command word associated with the RT
ApiCmdRTLSW	Redefines the Last Status word associated with the RT
ApiCmdRTMsgRead	Reads the individual RT's Next/Last Status word, Last Command word and message and error counter
ApiCmdRTMsgReadAll	Reads the RT message and error counter for all 32 RTs
ApiCmdRTNXW	Redefines the Next Status word associated with the RT
ApiCmdRTRespTime	Redefines the Response time associated with the RT
ApiCmdRTRespTimeGet	Gets the Response time associated with the RT
ApiCmdRTSACon	Defines the properties of the RT SA/Mode code such as interrupt control, and unique Next Status word setup
ApiCmdRTSAConErr	Defines the error injection of the RT SA/Mode code
ApiCmdRTSADWCGet	Reads the defined word count of the RT SA
ApiCmdRTSADWCSet	Defines the defined word count of the RT SA
ApiCmdRTSAMsgRead	Reads the execution status for an RT SA/Mode code
ApiCmdRTStart	Starts the RT operation for all assigned RTs
ApiCmdRTStatusRead	Reads the execution status of the general RT operation and the RT global message and error counters

This section has multiple references to the Status Word, Next Status Word and Last Status Word. All Status words have the same format as shown in Figure 8.1. Command words have the format as shown in Figure 8.2.

Figure 8.1: Status Word

Status Word									
15...............11	10	9	8	7....5	4	3	2	1	0
5	1	1	1	3	1	1	1	1	1
Remote Terminal Address	Message Error	Instrumentation	Service request	Reserved	Broadcast Command received	Busy	Sub System Flag	Dynamic Bus Control Acceptance	Terminal Flag

Figure 8.2: Command Word

Command Word			
15 11	10	9 5	4 0
Remote Terminal Address	T/R	Subaddress or Mode	Data Word Count or Mode Code

8.1 Low Speed Functions

8.1.1 ApiCmdRTBHDef

Prototype:

```
AiReturn ApiCmdRTBHDef(AiUInt32 ul_ModuleHandle,
                       AiUInt8 biu, AiUInt16 hid
                       AiUInt16 bid,
                       AiUInt16 sid,
                       AiUInt16 eid,
                       AiUInt8 qsize,
                       AiUInt8 bqm,
                       AiUInt8 bsm,
                       AiUInt8 sqm,
                       AiUInt8 eqm,
                       AiUInt8 dbm,
                       TY_API_RT_BH_INFO *pbh);
```

Purpose:

This function is used to associate a Data Buffer ID, Buffer Queue size, Queue mode, and error protocol to an RT Buffer Header ID. The RT Buffer Header specified should first be assigned to an RT Transfer using the function **ApiCmdRTSACon**.

Input:

`AiUInt16 hid`

Buffer Header ID

Note:
See Section 1.3.5 for the range allowed for this parameter.

`AiUInt16 bid`

Assigned Data Buffer Identifier

Note:
See Section 1.3.5 for the range allowed for this parameter.

`AiUInt16 sid`

0 (Reserved for Status Queue Entry Identifier)

`AiUInt16 eid`

0 (Reserved for Event Queue Entry Identifier)

`AiUInt8 qsize`

Buffer Queue size definition (Amount of contiguous Data Buffers)

Value	Constant	Description
0	**API_QUEUE_SIZE_1**	Queue size 1
1	**API_QUEUE_SIZE_2**	Queue size 2
2	**API_QUEUE_SIZE_4**	Queue size 4
3	**API_QUEUE_SIZE_8**	Queue size 8
4	**API_QUEUE_SIZE_16**	Queue size 16
5	**API_QUEUE_SIZE_32**	Queue size 32
6	**API_QUEUE_SIZE_64**	Queue size 64
7	**API_QUEUE_SIZE_128**	Queue size 128
8	**API_QUEUE_SIZE_256**	Queue size 256

`AiUInt8 bqm`

Buffer Queue Mode

Value	Constant	Description
0	**API_BQM_CYCLIC**	Cyclic data storage
1	**API_BQM_STAY_LAST**	Store once and stop at end of queue (last index) and reuse last buffer
2	**API_BQM_HOST_CONTROLLED**	Do not advance automatically to next buffer (host controlled)
3		Reserved until status queue is supported

Note:
The Buffer Queue Mode is only processed, if the “Buffer is valid”. The Buffer Queue Modes affect the Current Buffer Index and thus (in future) the Status- and Event Queue Operation of the RT!

`AiUInt8 bsm`

Data Buffer Store Mode

For Receive RT Operation the following options are valid:

Value	Constant	Description
0	**API_BSM_RX_DISCARD**	Discard error messages from the current Data Buffer
1	**API_BSM_RX_KEEP_CURRENT**	Keep error messages at the current Data Buffer

For Transmit RT Operation the following options are valid:

Value	Constant	Description
0	**API_BSM_TX_KEEP_SAME**	Keep the same Data Buffer at transfer error
1		Reserved

Note:
The data buffer store mode is always evaluated at the ‘Transfer-End’: ‘bsm’ = 0:
If a transfer error is detected at the ‘Transfer-End’, the Buffer is not valid! ‘bsm’ = 1:
At ‘Transfer-End’ the Buffer is always valid and it does not matter, if a transfer error is detected or not. If an error occurred during a RT-Receive transfer, the RT terminates the transfer with the error detection. Thus, the possible remainder of the transfer after the error will not be stored in the buffer.

AiUInt8 sqm

Status Queue Mode

Value	Constant	Description
0, 1	**API_SQM_ONE_ENTRY_ONLY**	Set Status Queue size to one entry only. This means that one status entry is available for the whole buffer queue.
2	**API_SQM_AS_QSIZE**	Set status queue size equal to the buffer queue size. This means, for each buffer a separate status entry is available.

AiUInt8 eqm

0 (Reserved for Event Queue mode)

AiUInt8 dbm

0 (Reserved for Double Buffer mode)

Output:

TY_API_RT_BH_INFO *pbh

RT Buffer Header information

```
typedef struct ty_api_rt_bh_info {
  AiUInt16 bid;
  AiUInt16 sid;
  AiUInt16 eid;
  AiUInt16 nbufs;
  AiUInt32 hid_addr;
  AiUInt32 bq_addr;
  AiUInt32 sq_addr;
  AiUInt32 eq_addr;
}
TY_API_RT_BH_INFO
```

AiUInt16 bid

Value	Description
0..2047	Assigned Data Buffer Identifier

AiUInt16 sid

Status Queue Entry Identifier

AiUInt16 eid

Event Queue Entry Identifier

AiUInt16 nbufs

Value	Description
1..256	Amount of allocated contiguous Data Buffers

AiUInt32 hid_addr

26-bit RT Buffer Header Address (Byte-Address)

AiUInt32 bq_addr

26-bit RT Data Buffer Queue Base Pointer (Byte-Address)

AiUInt32 sq_addr

26-bit RT Status Queue Base Pointer (Byte-Address)

AiUInt32 eq_addr

26-bit RT Event Queue Base Pointer (Byte-Address)

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.2 ApiCmdRTBHRead

Prototype:

```
AiReturn ApiCmdRTBHRead(AiUInt32 ul_ModuleHandle,
                        AiUInt8 biu,
                        AiUInt16 uw_HeaderId,
                        TY_API_RT_BH_INFO *px_Pbh);
```

Purpose:

This function is used to read the RT-SA buffer header structure (containing data buffer base pointer, status queue base pointer and event queue base pointer).

Input:

AiUInt16 uw_HeaderId

Buffer Header ID

Note:
See Section 1.3.5 for the range allowed for this parameter.

Output:

TY_API_RT_BH_INFO *px_Pbh

RT Buffer Header information

```
typedef struct ty_api_rt_bh_info {
  AiUInt16 bid;
  AiUInt16 sid;
  AiUInt16 eid;
  AiUInt16 nbufs;
  AiUInt32 hid_addr;
  AiUInt32 bq_addr;
  AiUInt32 sq_addr;
  AiUInt32 eq_addr;
}
TY_API_RT_BH_INFO
```

AiUInt16 bid

Value	Description
0..2047	Assigned Data Buffer Identifier

AiUInt16 sid

Status Queue Entry Identifier

`AiUInt16 eid`

Event Queue Entry Identifier

`AiUInt16 nbufs`

Value	Description
1..256	Amount of allocated contiguous Data Buffers

`AiUInt32 hid_addr`

26-bit RT Buffer Header Address (Byte-Address)

`AiUInt32 bq_addr`

26-bit RT Data Buffer Queue Base Pointer (Byte-Address)

`AiUInt32 sq_addr`

26-bit RT Status Queue Base Pointer (Byte-Address)

`AiUInt32 eq_addr`

26-bit RT Event Queue Base Pointer (Byte-Address)

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.3 ApiCmdRTDytagDef

Prototype:

```
AiReturn ApiCmdRTDytagDef(AiUInt32 ul_ModuleHandle,
                          AiUInt8 biu,
                          AiUInt8 con,
                          AiUInt16 rt_hid,
                          AiUInt16 mode,
                          TY_API_RT_DYTAG rt_dytag[4]);
```

Purpose:

This function is used to define the generation of dynamic data words for RT transmit SA's. One to four words of the RT Transfmit SA Buffer for up to 255 different Buffer headers can be selected for dynamic data generation. Dynamic data generation is performed in the RT Transmit SA Data Buffer by the internal firmware. The RT Transmit SA Buffer already has to be defined using the functions **ApiCmdRTSACon** and **ApiCmdRTBHDef** when dynamic data generation is enabled.

In **Function Mode** the RT Transmit SA Buffer is modified (for up to 2 Data Words) after the Data Word has been transmitted. The start value for the dynamic Data Word function shall be within the specified upper and lower limit.

In **Tagging Mode** the RT Transmit SA Buffer is modified (for up to 4 Data Words) before the Data Words are transmitted. In this mode the dynamic Data Word tagging function can be performed directly on Data Words located in the Transmit Data Buffer or on Function Words which are patched into the Transmit Data Buffer.

Note:
This function is not supported on embedded devices! (see also chapter B.1.7 "Limitations for embedded board variants")

Input:

`AiUInt8 con`

RT Dynamic Data Generation Control

Value	Constant	Description
0	**API_DIS**	Disable Dynamic Data Generation
1	**API_ENA**	Enable Dynamic Data Generation

`AiUInt16 rt_hid`

RT Buffer Header ID

Note:
See Section 1.3.5 for the range allowed for this parameter.

AiUInt16 mode

RT Dynamic Data Generation Mode

Value	Constant	Description
0	**API_DYTAG_STD_MODE**	Function Mode
1	**API_DYTAG_EFA_MODE**	Tagging Mode

TY_API_RT_DYTAG *rt_dytag[4]

RT Dynamic Data description

```
typedef struct ty_api_rt_dytag {
  AiUInt16 tag_fct;
  AiUInt16 min;
  AiUInt16 max;
  AiUInt16 step;
  AiUInt16 wpos;
}
TY_API_RT_DYTAG;
```

Parameter description for Function Mode (‘mode’ = 0)

Note:
Applicable for up to 2 dynamic words ‘bc_dytag[0..1]’, structure indexes bc_dytag[2..3] are reserved!

AiUInt16 tag_fct

Dynamic Data Word Generation Function

Value	Constant	Description
0	**API_DYTAG_FCT_DISABLE**	Disable
1	**API_DYTAG_FCT_POS_RAMP**	Positive Ramp Function
2	**API_DYTAG_FCT_NEG_RAMP**	Negative Ramp Function
3	**API_DYTAG_FCT_POS_TRIANGLE**	Positive Triangle Function
4	**API_DYTAG_FCT_NEG_TRIANGLE**	Negative Triangle Function
5	**API_DYTAG_FCT_XMT_WORD**	Transmit Data Word from specified Data Buffer ID
6	**API_DYTAG_FCT_SYNC_COUNTER**	Transmit Synchronisation Counter value (see also ApiCmdSyncCounterGet / Set).

AiUInt16 min

‘tag_fct’	Value	Description
1..4	**0..n**	Lower Limit of the dynamic Data Word
5	**0..2047**	Data Buffer Identifier
6	**0**	Reserved

AiUInt16 max

'tag_fct'	Value	Description
1..4	**0..n**	Upper Limit of the dynamic Data Word
5	**0..31**	Word Position of the Data Buffer Word to transmit
6	**0**	Reserved

AiUInt16 step

'tag_fct'	Value	Description
1..4	**0..n**	Stepsize used to increment or decrement the dynamic Data Word
5, 6	**0**	Reserved

AiUInt16 wpos

'tag_fct'	Value	Description
1..6	**0..31**	Word Position of the Data Buffer Word

Parameter description for Tagging Mode ('mode' = 1)

> **Note:**
> Applicable for up to 4 dynamic words 'bc_dytag[0..3]'!

AiUInt16 tag_fct

Dynamic Data Word Generation Function

Bit	Value	Constant	Description
15..8	**0**		Reserved
7	**0**	**API_DYTAG_DIRECT_MODIFY**	**Direct Data Buffer Modification** The function modifies the Data Word in the Transfer Data Buffer.
	1	**API_DYTAG_FUNCTION_MODIFY**	**Data Word Modification** The incrementing function (increment by 1) is applied to the Function Word 'min' which is written to the Transfer Data Buffer.
6..0	**0**	**API_DYTAG_FCT_DISABLE**	Disable
	1	**API_DYTAG_FCT_SAWTOOTH_16**	16-bit Sawtooth
	2	**API_DYTAG_FCT_SAWTOOTH_8_LOW**	8-bit Sawtooth in Data Bits 7..0
	3	**API_DYTAG_FCT_SAWTOOTH_8_HIGH**	8-bit Sawtooth in Data Bits 15..8

`AiUInt16 min`

Value	Description
0..n	Initial value of dynamic Function Word

> **Note:**
> Only applicable if ‘tag_fct’-Bit 7 = 1!

`AiUInt16 max`

Value	Description
0	Reserved

`AiUInt16 step`

Value	Description
0	Reserved

`AiUInt16 wpos`

Value	Description
0..31	Word Position of the dynamic Data Word

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.4 ApiCmdRTEnaDis

Prototype:

```
AiReturn ApiCmdRTEnaDis(AiUInt32 ul_ModuleHandle,
                        AiUInt8 biu,
                        AiUInt8 rt_addr,
                        AiUInt8 con)
```

Purpose:

This function is used to enable / disable the selected Remote Terminal (identified by its RT Address) on the fly.

Input:

`AiUInt8 rt_addr`

Value	Description
0..31	Remote Terminal Address

`AiUInt8 con`

Value	Constant	Description
0	**API_DIS**	Disable RT
1	**API_ENA**	Enable RT

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.5 ApiCmdRTGetDytagDef

Prototype:

```
AiReturn ApiCmdRTGetDytagDef(AiUInt32 ul_ModuleHandle,
                             AiUInt8 biu,
                             AiUInt16 rt_hid,
                             AiUInt16 *mode,
                             TY_API_RT_DYTAG rt_dytag[4]);
```

Purpose:

This function is used to read the Dytag settings for the generation of dynamic data words for a RT transmit SA.

Input:

AiUInt16 rt_hid

RT Buffer Header ID

Note:
See Section 1.3.5 for the range allowed for this parameter.

Output:

AiUInt16 *mode

RT Dynamic Data Generation Mode

Value	Constant	Description
0	**API_DYTAG_STD_MODE**	Function Mode
1	**API_DYTAG_EFA_MODE**	Tagging Mode

TY_API_RT_DYTAG *rt_dytag[4]

RT Dynamic Data description

```
typedef struct ty_api_rt_dytag {
  AiUInt16 tag_fct;
  AiUInt16 min;
  AiUInt16 max;
  AiUInt16 step;
  AiUInt16 wpos;
}
TY_API_RT_DYTAG;
```

`Parameter description for Function Mode ('mode' = 0)`

Note:
Applicable for up to 2 dynamic words ('bc_dytag[0..1]', structure indexes bc_dytag[2..3] are reserved!

`AiUInt16 tag_fct`

Dynamic Data Word Generation Function

Value	Constant	Description
0	**API_DYTAG_FCT_DISABLE**	Disable
1	**API_DYTAG_FCT_POS_RAMP**	Positive Ramp Function
2	**API_DYTAG_FCT_NEG_RAMP**	Negative Ramp Function
3	**API_DYTAG_FCT_POS_TRIANGLE**	Positive Triangle Function
4	**API_DYTAG_FCT_NEG_TRIANGLE**	Negative Triangle Function
5	**API_DYTAG_FCT_XMT_WORD**	Transmit Data Word from specified Data Buffer ID
6	**API_DYTAG_FCT_SYNC_COUNTER**	Transmit Synchronisation Counter value (see also ApiCmdSyncCounterGet / Set).

`AiUInt16 min`

'tag_fct'	Value	Description
1..4	**0..n**	Lower Limit of the dynamic Data Word
5	**0..2047**	Data Buffer Identifier
6	**0**	Reserved

`AiUInt16 max`

'tag_fct'	Value	Description
1..4	**0..n**	Upper Limit of the dynamic Data Word
5	**0..31**	Word Position of the Data Buffer Word to transmit
6	**0**	Reserved

`AiUInt16 step`

'tag_fct'	Value	Description
1..4	**0..n**	Stepsize used to increment or decrement the dynamic Data Word
5, 6	**0**	Reserved

`AiUInt16 wpos`

'tag_fct'	Value	Description
1..6	**0..31**	Word Position of the Data Buffer Word

`Parameter description for Tagging Mode ('mode' = 1)`

Note:
Applicable for up to 4 dynamic words 'bc_dytag[0..3]'!

`AiUInt16 tag_fct`

Dynamic Data Word Generation Function

Bit	Value	Constant	Description
15..8	**0**		Reserved
7	**0**	**API_DYTAG_DIRECT_MODIFY**	**Direct Data Buffer Modification** The function modifies the Data Word in the Transfer Data Buffer.
	1	**API_DYTAG_FUNCTION_MODIFY**	**Data Word Modification** The incrementing function (increment by 1) is applied to the Function Word 'min' which is written to the Transfer Data Buffer.
6..0	**0**	**API_DYTAG_FCT_DISABLE**	Disable
	1	**API_DYTAG_FCT_SAWTOOTH_16**	16-bit Sawtooth
	2	**API_DYTAG_FCT_SAWTOOTH_8_LOW**	8-bit Sawtooth in Data Bits 7..0
	3	**API_DYTAG_FCT_SAWTOOTH_8_HIGH**	8-bit Sawtooth in Data Bits 15..8

`AiUInt16 min`

Value	Description
0..n	Initial value of dynamic Function Word

Note:
Only applicable if 'tag_fct'-Bit 7 = 1!

`AiUInt16 max`

Value	Description
0	Reserved

`AiUInt16 step`

Value	Description
0	Reserved

`AiUInt16 wpos`

Value	Description
0..31	Word Position of the dynamic Data Word

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.6 ApiCmdRTGetSABufferHeaderInfo

Prototype:

```
AiReturn ApiCmdRTGetSABufferHeaderInfo(AiUInt8 Module,
                                       AiUInt8 biu, AiUInt8 rt_addr,
                                       AiUInt8 sa_type, AiUInt8 sa,
                                       AiUInt32 *pul_BufHeaderIndex,
                                       AiUInt32 *pul_BufHeaderAddr);
```

Purpose:

This function is used to get the buffer header id of a certain RT/SA combination.

Input:

AiUInt8 rt_addr

Value	Description
0..31	Remote Terminal Address

AiUInt8 sa

Mode	Value	Description
Subaddress	**1..30**	RT Subaddress
ModeCode	**0..31**	Mode code Number

AiUInt8 sa_type

Subaddress Type

Value	Constant	Description
0	**API_RT_TYPE_RECEIVE_SA**	Receive Subaddress
1	**API_RT_TYPE_TRANSMIT_SA**	Transmit Subaddress
2	**API_RT_TYPE_RECEIVE_MODECODE**	Receive Mode code
3	**API_RT_TYPE_TRANSMIT_MODECODE**	Transmit Mode code

Output:

AiUInt32 *pul_BufHeaderIndex

The buffer header index of this RT/SA combination

AiUInt32 *pul_BufHeaderAddr

The address of the buffer header of this RT/SA combination

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.7 ApiCmdRTGetSAConErr

Prototype:

```
AiReturn ApiCmdRTGetSAConErr(AiUInt32 ul_ModuleHandle,
                             AiUInt8 biu
                             AiUInt8 rt_addr,
                             AiUInt8 sa,
                             AiUInt8 sa_type,
                             TY_API_RT_ERR *perr);
```

Purpose:

This command is used to read the error injection settings of the specified RT Sub-address/Mode code of the selected Remote Terminal (identified by its RT Address).

Note:
This function is not vailable on all devices. See chapter “Limitations for specific boards”.

Input:

`AiUInt8 rt_addr`

Value	Description
0..31	Remote Terminal Address

`AiUInt8 sa`

Mode	Value	Description
Subaddress	**1..30**	RT Subaddress
ModeCode	**0..31**	Mode code Number

`AiUInt8 sa_type`

Subaddress Type

Value	Constant	Description
0	**API_RT_TYPE_RECEIVE_SA**	Receive Subaddress
1	**API_RT_TYPE_TRANSMIT_SA**	Transmit Subaddress
2	**API_RT_TYPE_RECEIVE_MODECODE**	Receive Mode code
3	**API_RT_TYPE_TRANSMIT_MODECODE**	Transmit Mode code

Output:

`TY_API_RT_ERR *perr`

RT SA Error Injection specifications

```
typedef struct ty_api_rt_err {
  AiUInt8 type;
  AiUInt8 sync;
```

```
  AiUInt8 contig;
  AiUInt8 padding1;
  AiUInt32 err_spec;
}
TY_API_RT_ERR;
```

AiUInt8 type

Error Type

Value	Constant	Description
0	**API_ERR_TYPE_NO_INJECTION**	No error injection
1	**API_ERR_TYPE_COMMAND_SYNC**	Status Sync Error with Sync Pattern in ‘sync’
2	**API_ERR_TYPE_DATA_SYNC**	Data Sync Error with Sync Pattern in ‘sync’ in Word ‘wpos’
3	**API_ERR_TYPE_PARITY**	Parity Error in Word ‘wpos’
4	**API_ERR_TYPE_MANCHESTER_HIGH**	Manchester Stuck at High Error in Word ‘wpos’ at Bit Position bpos’
5	**API_ERR_TYPE_MANCHESTER_LOW**	Manchester Stuck at Low Error in Word ‘wpos’ at Bit Position bpos’
6	**API_ERR_TYPE_GAP**	Gap Error with Gap defined in ‘contig’ in Word ‘wpos’
7	**API_ERR_TYPE_WORD_CNT_HIGH**	Word Count High (+1) Error
8	**API_ERR_TYPE_WORD_CNT_LOW**	Word Count Low (-1) Error
9	**API_ERR_TYPE_BIT_CNT_HIGH**	Bit Count High Error in Word ‘wpos’ (+ ‘bc_bits’)
10	**API_ERR_TYPE_BIT_CNT_LOW**	Bit Count Low Error in Word ‘wpos’ (- ‘bc_bits’)
11	**API_ERR_TYPE_ALTERNATE_BUS**	Alternate Bus Error
12	**API_ERR_TYPE_ZERO_CROSS_NEG**	Zero Crossing Low Deviation Error, in word ‘wpos’ at bit position ‘bpos’ with negative deviation in ‘contig’
13	**API_ERR_TYPE_ZERO_CROSS_POS**	Zero Crossing High Deviation Error, in word ‘wpos’ at bit position ‘bpos’ with positive deviation in ‘contig’

AiUInt8 sync

Sync Field Error Half-Bit-Pattern (6 LS-Bits)
(38hex = 111000 Sync Pattern)

AiUInt8 contig

Error Type	Value	Description
Gap Error	**1..15**	Gap Error Half Bits (0.5μs per Half Bit)
Zero Crossing Error	**0**	125ns deviation
	1	187.5ns deviation
	2	156.25ns deviation
	3	218.75ns deviation
		Note: Values 2 and 3 are not available for all AIM devices. Please check the chapter "Board functionality overview" and look for "Error Injection of High Resolution Zero Crossing Deviation".

AiUInt8 padding1

Reserved (0)

AiUInt32 err_spec

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
reserved (0)							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
WPOS							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
BPOS							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BCBITS							

WPOS

Value	Description
0	Error Word Position (Status Word)
1..32	Error Word Position

BPOS

Value	Description
4..19	Error Bit Position
20	Error Bit Position (Parity Bit)

BCBITS

Value	Description
1..3	Amount of Bit Count Error Bits

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.8 ApiCmdRTGetSimulationInfo

Prototype:

```
AiReturn ApiCmdRTGetSimulationInfo(AiUInt32 ul_ModuleHandle,
                                   AiUInt8 biu,
                                   AiUInt8 rt_addr,
                                   TY_RT_INFO *px_RTInfo);
```

Purpose:

This function is used to read the simulation and monitoring status of an RT including sub addresses and MID as bit field.

Input:

AiUInt8 rt_addr

Value	Description
0..31	Remote Terminal Address

Output:

TY_RT_INFO *px_RTInfo

RT Simulation description

```
typedef struct rt_info_tag {
  AiUInt8 uc_Mode;
  AiUInt32 ul_RxSa;
  AiUInt32 ul_TxSa;
  AiUInt32 ul_RxMC;
  AiUInt32 ul_TxMC;
  AiUInt32 ul_HSRxMID[8];
  AiUInt32 ul_HSTxMID[8];
  AiUInt32 ul_HSMC;
}
TY_RT_INFO;
```

AiUInt8 uc_Mode

RT Operation Mode

Value	Constant	Description
0	**API_RT_DISABLE_OPERATION**	RT Operation disabled
1	**API_RT_ENABLE_SIMULATION**	RT Simulation enabled
2	**API_RT_ENABLE_MONITORING**	RT Mailbox Monitoring enabled

AiUInt32 ul_RxSa

Rx Status SA 1. . . .30 as bitfield, whereas bit 1 represents SA 1 and bit 30 represents SA 30

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
0	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STAT	STAT	STAT	STAT	STAT	STAT	STAT	0

STAT

Value	Description
0	RX Sub Address disabled
1	RX Sub Address enabled

AiUInt32 ul_TxSa

Tx Status SA 1. . . .30 as bitfield, whereas bit 1 represents SA 1 and bit 30 represents SA 30

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
0	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STAT	STAT	STAT	STAT	STAT	STAT	STAT	0

STAT

Value	Description
0	TX Sub Address disabled
1	TX Sub Address enabled

AiUInt32 ul_RxMC

Rx Modecode Status MC 0. . . .31 as bitfield, whereas bit 0 represents Modecode 0 and bit 31 represents Modecode 31

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

STAT

Value	Description
0	RX Modecode disabled
1	RX Modecode enabled

`AiUInt32 ul_TxMC`

Tx Modecode Status MC 0. . . .31 as bitfield, whereas bit 0 represents Modecode 0 and bit 31 represents Modecode 31

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

STAT

Value	Description
0	TX Modecode disabled
1	TX Modecode enabled

`AiUInt32 ul_HSRxMID[8]`

Bitfield representing HS Rx MID Status (8x32Bit) , whereas bit 0 of 'ul_HSRxMID[0]' represents Message ID 0 and bit 31 of 'ul_HSRxMID[7]' represents Message ID 127

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

STAT

Value	Description
0	RX Message ID disabled
1	RX Message ID enabled

`AiUInt32 ul_HSTxMID[8]`

Bitfield representing HS Tx MID Status (8x32Bit) , whereas bit 0 of 'ul_HSTxMID[0]' represents Message ID 0 and bit 31 of 'ul_HSTxMID[7]' represents Message ID 127

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

STAT

Value	Description
0	TX Message ID disabled
1	TX Message ID enabled

`AiUInt32 ul_HSMC`

Bitfield representing HS Modecode Status, whereas bit 0 HS Modecode 0 and bit 31 represents HS Modecode 31

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STAT	STAT	STAT	STAT	STAT	STAT	STAT	STAT

STAT

Value	Description
0	HS Modecode disabled
1	HS Modecode enabled

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.9 ApiCmdRTGlobalCon

Prototype:

```
AiReturn ApiCmdRTGlobalCon(AiUInt32 ul_ModuleHandle,
                           AiUInt8 biu,
                           AiUInt16 count,
                           TY_API_RT_SA *rt_glob);
```

Purpose:

This function is used to initialize all RTs, initialize the Subaddress/Mode code of the selected RTs, initialize the RT buffers and update the RT transmit buffers.

Input:

AiUInt16 count

Value	Description
1..511	Number of TY_API_RT_SA structures to process

TY_API_RT_SA *rt_glob

Pointer to count contiguous TY_API_RT_SA structures used as an array.

```
Typedef struct ty_api_rt_sa {
  AiUInt16 buffer[32];
  AiUInt8 mode;
  AiUInt8 rt;
  AiUInt8 rt_con;
  AiUInt8 sa_mc;
  AiUInt8 sa_type;
  AiUInt8 sa_con;
  AiUInt8 resp_time;
  AiUInt8 smod;
  AiUInt16 nxw;
  AiUInt16 swm;
  AiUInt16 hid;
  AiUInt16 bid;
}
TY_API_RT_SA;
```

AiUInt16 buffer[]

Data buffer. In update mode the buffer contents will only be used for transmit data.

AiUInt8 mode

TY_API_RT_SA usage Control

Value	Constant	Description
0	**API_RT_SA_USAGE_INIT**	RT Initialization
1	**API_RT_SA_USAGE_BUF_INIT**	Subaddress and buffer initialization for the selected RT
2	**API_RT_SA_USAGE_UPDATE**	Update transmit buffer

AiUInt8 rt

Value	Description
0..31	Remote Terminal Address

AiUInt8 rt_con

Remote Terminal Operation Control

Value	Constant	Description
0	**API_DIS**	Disable RT Operation
1	**API_ENA**	Enable RT Simulation

AiUInt8 sa_mc

Value	Description
1..30	RT Subaddress
0..31	Mode code

AiUInt8 sa_type

Subaddress Type

Value	Constant	Description
0	**API_RT_TYPE_RECEIVE_SA**	Receive Subaddress
1	**API_RT_TYPE_TRANSMIT_SA**	Transmit Subaddress
2	**API_RT_TYPE_RECEIVE_MODECODE**	Receive Mode code
3	**API_RT_TYPE_TRANSMIT_MODECODE**	Transmit Mode code

AiUInt8 sa_con

Subaddress Control

Value	Constant	Description
0	**API_RT_DISABLE_SA**	Subaddress disabled
1	**API_RT_ENABLE_SA**	Subaddress enabled, no interrupt

AiUInt8 resp_time

Remote Terminal Response Time

Value	Description
16..255	Range: 4..63.75μs in steps of 0.25μs (16=4μs, .., 255=63.75μs

AiUInt16 nxw

Next RT Status Word – See Figure 8.1.

`AiUInt8 smod`

Status Word Mask Control

Value	Constant	Description
0	**API_RT_SWM_OR**	Use 'SWM' as OR mask for Status Word response
1	**API_RT_SWM_AND**	Use 'SWM' as AND mask for Status Word response

`AiUInt16 swm`

Status Word Modification Mask This mask can raise or supress specific bits of the Status Word response dependent on the selected Status Word Mask Control mode.

`AiUInt16 hid`

Header ID

Note:
See Section 1.3.5 for the range allowed for this parameter.

`AiUInt16 bid`

Value	Description
1..2047	Assigned Data Buffer Identifier

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.10 ApiCmdRTHalt

Prototype:

```
AiReturn ApiCmdRTHalt(AiUInt32 ul_ModuleHandle,
                      AiUInt8 biu);
```

Purpose:

This command is used to stop the Remote Terminal operation for all assigned RTs.

Input:

none

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.11 ApiCmdRTIni

Prototype:

```
AiReturn ApiCmdRTIni(AiUInt32 ul_ModuleHandle,
                     AiUInt8 biu,
                     AiUInt8 rt_addr,
                     AiUInt8 con,
                     AiUInt8 bus,
                     AiFloat resp_time,
                     AiUInt16 nxw);
```

Purpose:

This function is used to initialize the selected Remote Terminal (identified by its RT Address) and is used to define the default for the Next RT Status Word and the RT Response Time in steps of 0.25μs.

Input:

AiUInt8 rt_addr

Value	Description
0..31	Remote Terminal Address

AiUInt8 con

Remote Terminal Operation Control

Value	Constant	Description
0	**API_RT_DISABLE_OPERATION**	Disable RT Operation
1	**API_RT_ENABLE_SIMULATION**	Enable RT Simulation
2	**API_RT_ENABLE_MONITORING**	Enable RT Mailbox Monitoring
3	**API_RT_ENABLE_DUAL_REDUNDANT**	Enable Single RT / Real Dual Redundant Operation → see also detailed description of this mode on next page **Note:** The real dual redundant / single RT mode is not available on all devices. See chapter "Limitations for non embedded boards".

Note:
All subaddresses will be enabled per default and the status word mask is set to enable the message error bit!
Therefore when disabling an RT and re-enabling it, all wanted subaddresses have to be initialized again with the command ApiCmdRTSACon()!

> **Note:**
> The dual redundant mode (con = 3) can only be set for one RT address at a time. However, other RT addresses can be run in normal operation mode (con = 1). To change the RT, that is used for dual redundant operation just disable the previous RT and then enable the new RT in dual redundant mode (both can be done calling this function).

Single RT / Real Dual Redundant mode
The Single RT / Real Dual Redundant mode must be used to be able to execute the test "5.2.2.1 Dual Redundant Operation" of the SAE AS4112A RT Production Test Plan.

In this mode the RT is able to:

1. Accept a valid command received on the alternate bus while responding to a command on the original bus.
2. Respond to the valid command occurring later in time when overlapping valid commands are received on both buses.
3. When point 1 or 2 occurs, the RT resets and responds to the new command on the alternate bus.

`AiUInt8 bus`

Remote Terminal Bus Respond Control

Value	Constant	Description
0	**API_RT_RSP_BOTH_BUSSES**	Respond to command words from both busses
1	**API_RT_RSP_PRI_BUS**	Respond to command words from primary bus only
2	**API_RT_RSP_SEC_BUS**	Respond to command words from secondary bus only

> **Note:**
> If a "No Response" condition occurs due to the bus respond control, the current RT will discard the transfer without any operation.

`AiFloat resp_time`

Value	Description
4..63.75	Remote Terminal Response Time Value (Range: 4..63.72μs in steps of 0.25μs

`AiUInt16 nxw`

Next RT Status Word - The Status word response the RT will transmit in response to a BC Command. See Figure 8.1.

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.12 ApiCmdRTLCW

Prototype:

```
AiReturn ApiCmdRTLCW(AiUInt32 ul_ModuleHandle,
                     AiUInt8 biu,
                     AiUInt8 rt_addr,
                     AiUInt16 lcw);
```

Purpose:

This function is used to redefine the Last Command Word associated with the selected Remote Terminal (identified by its RT Address).

Input:

`AiUInt8 rt_addr`

Value	Description
0..31	Remote Terminal Address

`AiUInt16 lcw`

RT Last Command Word – See Figure 8.2.

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.13 ApiCmdRTLSW

Prototype:

```
AiReturn ApiCmdRTLSW(AiUInt32 ul_ModuleHandle,
                     AiUInt8 biu,
                     AiUInt8 rt_addr,
                     AiUInt16 lsw);
```

Purpose:

This function is used to redefine the Last Status Word associated with the selected Remote Terminal (identified by its RT Address).

Input:

`AiUInt8 rt_addr`

Value	Description
0..31	Remote Terminal Address

`AiUInt16 lsw`

RT Last Status Word – See Figure 8.1.

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.14 ApiCmdRTModeCtrl

Prototype:

```
AiReturn ApiCmdRTModeCtrl(AiUInt32 ul_ModuleHandle,
                          AiUInt8 uc_Biu,
                          TY_API_RT_MODE_CTRL *px_RtModeCtrl);
```

Purpose:

This function is used to enable / disable various RT functionality on-the-fly.

Input:

TY_API_RT_MODE_CTRL *px_RtModeCtrl

RT Mode Control description

```
typedef struct ty_api_rt_mode_ctrl {
  AiUInt32 ul_RtMode;
  AiUInt32 ul_Ctrl;
  AiUInt32 ul_Param1;
  AiUInt32 ul_Param2;
  AiUInt32 ul_Param3;
}
TY_API_RT_MODE_CTRL;
```

AiUInt32 ul_RtMode

Value	Constant	Description
3	**API_RT_MODE_CONFIGURED_DYTAGS**	This mode is used to enable / disable all configured RT dytags **Note:** This mode is not available on embedded devices! (see also chapter B.1.7 “Limitations for embedded board variants”)
4	**API_RT_MODE_LANE_CTRL**	This mode is used to control the lane A and/or lane B bus response
6	**API_RT_MODE_OPERATION_CTRL**	This mode is used to control the RT mode

`AiUInt32 ul_Ctrl`

ul_RtMode	Constant	Description
3	**API_DIS**	Disable the functionality referenced with parameter 'ul_RtMode'
	API_ENA	Enable the functionality referenced with parameter 'ul_RtMode'
4, 6		0 (reserved)

`AiUInt32 ul_Param1`

ul_RtMode	Value	Description
3	**0..31**	Remote Terminal Address that is affected
	0xFFFFFFFF	Affect all RTs
4, 6	**0..31**	Remote Terminal Address that is affected

`AiUInt32 ul_Param2`

ul_RtMode	Value	Description
3	**0**	Reserved
4	**API_RT_RSP_BOTH_BUSSES**	Respond to both busses
	API_RT_RSP_PRI_BUS	Respond only to bus A
	API_RT_RSP_SEC_BUS	Respond only to bus B
6	**API_RT_DISABLE_OPERATION**	Disable RT operation
	API_RT_ENABLE_SIMULATION	RT simulation enabled
	API_RT_ENABLE_MONITORING	RT Mailbox Monitoring enabled

`AiUInt32 ul_Param3`

0 (reserved)

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.15 ApiCmdRTMsgRead

Prototype:

```
AiReturn ApiCmdRTMsgRead(AiUInt32 ul_ModuleHandle,
                         AiUInt8 biu,
                         AiUInt8 rt_addr,
                         TY_API_RT_MSG_DSP *pmsg_dsp);
```

Purpose:

This function is used to read the current execution status of the specified Remote Terminal number including the RT's Next/Last Status word, Last Command word and message and error counter.

Input:

AiUInt8 rt_addr

Value	Description
0..31	Remote Terminal Address

Output:

TY_API_RT_MSG_DSP *pmsg_dsp

RT Message Status information

```
typedef struct ty_api_rt_msg_dsp {
  AiUInt16 nxw;
  AiUInt16 lsw;
  AiUInt16 lcw;
  AiUInt32 msg_cnt;
  AiUInt32 err_cnt;
}
TY_API_RT_MSG_DSP;
```

AiUInt16 nxw

Next Status Word – See Figure 8.1.

AiUInt16 lsw

Last Status Word transmitted – See Figure 8.1.

AiUInt16 lcw

Last Command Word received – See Figure 8.2.

AiUInt32 msg_cnt

Number of messages transferred –

AiUInt32 err_cnt

Number of message errors detected .

> **Note:**
> msg_cnt and err_cnt are cleared on restart of the AIM board Remote Terminals when the ApiCmdRTStart' function is called.

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.16 ApiCmdRTMsgReadAll

Prototype:

```
AiReturn ApiCmdRTMsgReadAll(AiUInt32 ul_ModuleHandle,
                            AiUInt8 biu,
                            TY_API_RT_MSG_ALL_DSP *pall_dsp);
```

Purpose:

This function is used to read the Transfer-/Error counter values of all Remote Terminals on the AIM board.

Input:

none

Output:

TY_API_RT_MSG_ALL_DSP *pall_dsp

Entire RT Status information

```
typedef struct ty_api_rt_msg_all_dsp_rt {
  AiUInt32 rt_msg;
  AiUInt32 rt_err;
}
TY_API_RT_MSG_ALL_DSP_RT;

typedef struct ty_api_rt_msg_all_dsp {
  TY_API_RT_MSG_ALL_DSP_RT rt[32];
}
TY_API_RT_MSG_ALL_DSP;
```

AiUInt32 rt_msg

Number of Messages I to/from RT0 ... RT31

AiUInt32 rt_err

Number of Message Errors detected for RT0 ... RT31

Note:
rt_msg and rt_err are cleared on restart of the AIM board Remote Terminals when the ApiCmdRTStart' function is called.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.17 ApiCmdRTNXW

Prototype:

```
AiReturn ApiCmdRTNXW(AiUInt32 ul_ModuleHandle,
                     AiUInt8 biu,
                     AiUInt8 rt_addr,
                     AiUInt16 nxw);
```

Purpose:

This function is used to redefine the Next RT Status Word for the selected Remote Terminal (identified by its RT Address).

Input:

AiUInt8 rt_addr

Value	Description
0..31	Remote Terminal Address

AiUInt16 nxw

Next RT 1553 Status Word – See Figure 8.1.

Output:

none

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.18 ApiCmdRTRespTime

Prototype:

```
AiReturn ApiCmdRTRespTime(AiUInt32 ul_ModuleHandle, AiUInt8 biu,
                          AiUInt8 rt_addr,  AiFloat resp_time);
```

Purpose:

This function is used to redefine the Response Time of the selected Remote Terminal (identified by its RT Address).

Input:

AiUInt8 rt_addr

Value	Description
0..31	Remote Terminal Address

AiFloat resp_time

Response Time in μs.

Value	Description
4..63.75	Defines the RT response time in 0.25μs steps. Response time values less than 4μs are not allowed. If the response time is programmed to less than 4.5μs, the terminal does not check, if the lines are busy before it transmits the status word. In this case, the word count high check is disabled. At 3910 transfers, this method is also used by the LS-RT and LS-Monitor transfer evaluation or HS transfer preparation, respectively.

Note:
Due to the extended operations, which have to be performed on Mode Codes, the response time may be up to 7μs / 9μs. For Mailbox monitoring this function is not provided. Due to the gap measurement definition of the MIL-STD-1553B, a min. physical bus idle time of 2μs can be achieved in 1 Mbit Transmission Mode (min. 4μs Response Time).

Output:

none

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.19 ApiCmdRTRespTimeGet

Prototype:

```
AiReturn ApiCmdRTRespTimeGet(AiUInt32 ul_ModuleHandle,
                             AiUInt8 biu,
                             AiUInt8 rt_addr,
                             AiFloat *pResp_time);
```

Purpose:

This function is used to read the Response Time of the selected Remote Terminal (identified by its RT Address). For RT in simulation mode this is the value specified with ApiCmdRTRespTime, for RT in mailbox mode this is the measured response time.

Input:

`AiUInt8 rt_addr`

Value	Description
0..31	Remote Terminal Address

Output:

`AiFloat *pResp_time`

Response Time in μs.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.20 ApiCmdRTSACon

Prototype:

```
AiReturn ApiCmdRTSACon(AiUInt32 ul_ModuleHandle,
                       AiUInt8 biu,
                       AiUInt8 rt_addr,
                       AiUInt8 sa,
                       AiUInt16 hid,
                       AiUInt8 sa_type,
                       AiUInt8 con,
                       AiUInt8 rmod,
                       AiUInt8 smod,
                       AiUInt16 swm);
```

Purpose:

This function is used to define the properties of the specified RT Subaddress/Mode code (identified by its RT Address) such as interrupt control and unique Next Status word setup. The RT Buffer Header Identifier shall already be defined using the function **ApiCmdRTBHDef** before applying this function.

Input:

`AiUInt8 rt_addr`

Value	Description
0..31	Remote Terminal Address

`AiUInt8 sa`

Mode	Value	Description
Subaddress	**1..30**	RT Subaddress
ModeCode	**0..31**	Mode code Number

`AiUInt16 hid`

Buffer Header ID

> **Note:**
> See Section 1.3.5 for the range allowed for this parameter.

`AiUInt8 sa_type`

Subaddress Type

Value	Constant	Description
0	**API_RT_TYPE_RECEIVE_SA**	Receive Subaddress
1	**API_RT_TYPE_TRANSMIT_SA**	Transmit Subaddress
2	**API_RT_TYPE_RECEIVE_MODECODE**	Receive Mode code
3	**API_RT_TYPE_TRANSMIT_MODECODE**	Transmit Mode code

AiUInt8 con

Subaddress Control

Value	Constant	Description
0	**API_RT_DISABLE_SA**	Subaddress disabled
1	**API_RT_ENABLE_SA**	Subaddress enabled, no interrupt
2	**API_RT_ENABLE_SA_INT_XFER**	Subaddress enabled, Interrupt on any Transfer
3	**API_RT_ENABLE_SA_INT_ERR**	Subaddress enabled, Interrupt on any Transfer Error

AiUInt8 rmod

0 (Reserved for RT SA Response Mode Control)

AiUInt8 smod

Status Word Mask Control

Value	Constant	Description
0	**API_RT_SWM_OR**	Use 'SWM' as OR mask for Status Word response
1	**API_RT_SWM_AND**	Use 'SWM' as AND mask for Status Word response
2	**API_RT_SWM_NXW_OR**	Use 'SWM' as OR mask for Status Word response and to modify "Next Status Word"
3	**API_RT_SWM_NXW_AND**	Use 'SWM' as AND mask for Status Word response and to modify "Next Status Word"

AiUInt16 swm

Status Word Modification Mask This mask can raise specific bits of the Status Word response dependent on the selected Status Word Mask Control mode. See Figure 8.1 for the Status word format.

Note:
On embedded board variants with disabled BC the Dynamic Bus Control modecode can only be set to either DISABLED or ENABLED with smod API_RT_SWM_OR and swm 0x400 (Message Error).

Output:

none

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.21 ApiCmdRTSAConErr

Prototype:

```
AiReturn ApiCmdRTSAConErr(AiUInt32 ul_ModuleHandle,
                          AiUInt8 biu,
                          AiUInt8 rt_addr,
                          AiUInt8 sa,
                          AiUInt8 sa_type,
                          TY_API_RT_ERR *perr);
```

Purpose:

This command is used to control the error injection capability of the specified RT Sub-address/Mode code of the selected Remote Terminal (identified by its RT Address).

Note:
This function is not available on all devices. See chapter "Limitations for specific boards".

Input:

`AiUInt8 rt_addr`

Value	Description
0..31	Remote Terminal Address

`AiUInt8 sa`

Mode	Value	Description
Subaddress	**1..30**	RT Subaddress
ModeCode	**0..31**	Mode code Number

`AiUInt8 sa_type`

Subaddress Type

Value	Constant	Description
0	**API_RT_TYPE_RECEIVE_SA**	Receive Subaddress
1	**API_RT_TYPE_TRANSMIT_SA**	Transmit Subaddress
2	**API_RT_TYPE_RECEIVE_MODECODE**	Receive Mode code
3	**API_RT_TYPE_TRANSMIT_MODECODE**	Transmit Mode code

`TY_API_RT_ERR *perr`

```
RT SA Error Injection specifications
typedef struct ty_api_rt_err {
  AiUInt8 type;
  AiUInt8 sync;
  AiUInt8 contig;
  AiUInt8 padding1;
  AiUInt32 err_spec;
}
TY_API_RT_ERR;
```

`AiUInt8 type`

Error Type

Value	Constant	Description
0	**API_ERR_TYPE_NO_INJECTION**	No error injection
1	**API_ERR_TYPE_COMMAND_SYNC**	Status Sync Error with Sync Pattern in 'sync'
2	**API_ERR_TYPE_DATA_SYNC**	Data Sync Error with Sync Pattern in 'sync' in Word 'wpos'
3	**API_ERR_TYPE_PARITY**	Parity Error in Word 'wpos'
4	**API_ERR_TYPE_MANCHESTER_HIGH**	Manchester Stuck at High Error in Word 'wpos' at Bit Position bpos'
5	**API_ERR_TYPE_MANCHESTER_LOW**	Manchester Stuck at Low Error in Word 'wpos' at Bit Position bpos'
6	**API_ERR_TYPE_GAP**	Gap Error with Gap defined in 'config' in Word 'wpos'
7	**API_ERR_TYPE_WORD_CNT_HIGH**	Word Count High (+1) Error
8	**API_ERR_TYPE_WORD_CNT_LOW**	Word Count Low (-1) Error
9	**API_ERR_TYPE_BIT_CNT_HIGH**	Bit Count High Error in Word 'wpos' (+ 'bc_bits')
10	**API_ERR_TYPE_BIT_CNT_LOW**	Bit Count Low Error in Word 'wpos' (- 'bc_bits')
11	**API_ERR_TYPE_ALTERNATE_BUS**	Alternate Bus Error
12	**API_ERR_TYPE_ZERO_CROSS_NEG**	Zero Crossing Low Deviation Error, in word 'wpos' at bit position 'bpos' with negative deviation in 'contig'
13	**API_ERR_TYPE_ZERO_CROSS_POS**	Zero Crossing High Deviation Error, in word 'wpos' at bit position 'bpos' with positive deviation in 'contig'
16	**API_ERR_TYPE_PHYSICAL_ERROR _SUPPRESSION**	The RX-RT suppressesthe Physical Error termination in order to send a Status Word to an erroneous received message. **(not available on all boards see table 15.2 Board functionality overview, no Multichannel Support)**

17	**API_ERR_TYPE_WCLO_EXT**	Word Count Low (-'wpos') Error The BIU Processor transmits the number of Data Words defined via the Command Word minus the value set 'wpos' in. **(not available on all boards see table 15.2 Board functionality overview, no Multichannel Support)**
18	**API_ERR_TYPE_MSG_LENGTH_HI_IGNORE**	Message Length High Ignore Support The RX-RT suppresses the Error termination in Word Count +1 case, in order to send a Status Word directly after the Last Data Word of the lengthened message. **(not available on all boards see table 15.2 Board functionality overview, no Multichannel Support)**
19	**API_ERR_TYPE_MSG_LENGTH_LO_IGNORE**	The RX-RT shortens the expected Data Words by the number defined in the 'wpos', in order to send a Status Word directly after the Last Data Word of the shortened message. **(not available on all boards see table 15.2 Board functionality overview, no Multichannel Support)**

Note:
Limitation for error type 1, 2, 4, 5, on some boards. Please see chapter B.1.3.

AiUInt8 sync

Sync Field Error Half-Bit-Pattern (6 LS-Bits) (38hex = 111000 Sync Pattern)

AiUInt8 config

Error Type	Value	Description
Gap Error	**1..15**	Gap Error Half Bits (0.5μs per Half Bit)
Zero Crossing Error	**0**	125ns deviation
	1	187.5ns deviation
	2	156.25ns deviation
	3	218.75ns deviation

Note:
Values 2 and 3 are not available for all AIM devices. Please check the chapter "Board functionality overview" and look for "Error Injection of High Resolution Zero Crossing Deviation".

AiUInt8 padding1

Reserved (0)

AiUInt32 err_spec

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
0 (reserved)							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
WPOS							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
BPOS							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BCBITS							

WPOS

Value	Description
0	Error Word Position (Status Word)
1..32	Error Word Position

BPOS

Value	Description
4..19	Error Bit Position
20	Error Bit Position (Parity Bit)

BCBITS

Value	Description
1..3	Amount of Bit Count Error Bits

Output:

none

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.22 ApiCmdRTSADWCGet

Prototype:

```
AiReturn ApiCmdRTSADWCGet(AiUInt32 ul_ModuleHandle,
                           AiUInt8 uc_Biu,
                           AiUInt8 uc_RtAddr,
                           AiUInt8 uc_SA,
                           AiUInt8 uc_SAType,
                           AiUInt32 *pul_WordCnt);
```

Purpose:

This function is used to read out the current defined Word Count of the specified RT Subaddress (identified by its RT Address). The defined word count value is used during processing for Illegal Command detection purposes.

If a word count check is defined, the RT does only respond to a command, when its word cnt is equal to the previously set SA defined word count (see parameter ul_WordCnt). If it is not equal, the RT interpretes the command as illegal command and only sets the message error bit in the last RT status word. The function ApiCmdRTMsgRead() can be used to read the last status word.

Note:
This function is not available on all devices. See chapter "Limitations for specific boards".

Input:

`AiUInt8 uc_RtAddr`

Value	Description
0..31	Remote Terminal Address

`AiUInt8 uc_SA`

Mode	Value	Description
Subaddress	**1..30**	RT Subaddress

Note:
The defined word count is not supported for Modecodes!

`AiUInt8 uc_SAType`

Subaddress Type

Value	Constant	Description
0	**API_RT_TYPE_RECEIVE_SA**	Receive Subaddress
1	**API_RT_TYPE_TRANSMIT_SA**	Transmit Subaddress

Note:
The defined word count is not supported for Modecodes!

Output:

`AiUInt32 *pul_WordCnt`

RT SA Defined Word Count that has been set previously with the function ApiCmdRTSADWCSet().

Value	Description
0	Defined word count check is not used
1..32	Word Count to be checked

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.23 ApiCmdRTSADWCSet

Prototype:

```
AiReturn ApiCmdRTSADWCSet(AiUInt32 ul_ModuleHandle,
                          AiUInt8 uc_Biu,
                          AiUInt8 uc_RtAddr,
                          AiUInt8 uc_SA,
                          AiUInt8 uc_SAType,
                          AiUInt32 ul_WordCnt);
```

Purpose:

This function is used to define a defined Word Count of the specified RT Subaddress (identified by its RT Address). The defined word count value is used during processing for Illegal Command detection purposes.

If a word count check is defined, the RT does only respond to a command, when its word cnt is equal to the previously set SA defined word count (see parameter ul_WordCnt). If it is not equal, the RT interpretes the command as illegal command and only sets the message error bit in the last RT status word. The function ApiCmdRTMsgRead() can be used to read the last status word.

This function can be used to prepare the RT to be able to execute the test “5.2.2.6 Illegal Commands” of the SAE AS4112A RT Production Test Plan.

Note:
This function is not available on all devices. See chapter “Limitations for specific boards”.

Input:

`AiUInt8 uc_RtAddr`

Value	Description
0..31	Remote Terminal Address

`AiUInt8 uc_SA`

Mode	Value	Description
Subaddress	**1..30**	RT Subaddress

Note:
The defined word count is not supported for Modecodes!

`AiUInt8 uc_SAType`

Subaddress Type

Value	Constant	Description
0	**API_RT_TYPE_RECEIVE_SA**	Receive Subaddress
1	**API_RT_TYPE_TRANSMIT_SA**	Transmit Subaddress

> **Note:**
> The defined word count is not supported for Modecodes!

`AiUInt32 ul_WordCnt`

RT SA Defined Word Count Specification

Value	Description
0	Do not use the defined word count check
1..32	Word Count to be checked

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.24 ApiCmdRTSAMsgRead

Prototype:

```
AiReturn ApiCmdRTSAMsgRead(AiUInt32 ul_ModuleHandle,
                           AiUInt8 biu,
                           AiUInt8 rt_addr,
                           AiUInt8 sa,
                           AiUInt8 sa_type,
                           AiUInt8 clr,
                           TY_API_RT_SA_MSG_DSP *psa_dsp);
```

Purpose:

This function is used to read the execution status of the specified RT Subaddress/Mode code of the selected AIM board Remote Terminal (identified by its RT Address).

Input:

AiUInt8 rt_addr

Value	Description
0..31	Remote Terminal Address

AiUInt8 sa

Mode	Value	Description
Subaddress	**1..30**	RT Subaddress
ModeCode	**0..31**	Mode code Number

AiUInt8 sa_type

Subaddress Type

Value	Constant	Description
0	**API_RT_TYPE_RECEIVE_SA**	Receive Subaddress
1	**API_RT_TYPE_TRANSMIT_SA**	Transmit Subaddress
2	**API_RT_TYPE_RECEIVE_MODECODE**	Receive Mode code
3	**API_RT_TYPE_TRANSMIT_MODECODE**	Transmit Mode code

AiUInt8 clr

Buffer Status Flag Control

Value	Constant	Description
0	**API_DONT_MODIFY_STATUS_BITS**	Do not modify Buffer Status bits
1	**API_RESET_STATUS_BITS**	Reset Buffer Status bits to ‘Buffer not used’ state
2	**API_RESET_ERROR_BITS**	Reset Buffer Error bits to ‘No error’ state
3	**API_RESET_ERR_STAT_BITS**	Reset Buffer Status bits to ‘Buffer not used’ state and Buffer Error bits to ‘No error’ state

Output:

TY_API_RT_SA_MSG_DSP *psa_dsp

RT SA Message Status information

```
typedef struct ty_api_rt_sa_msg_dsp {
  AiUInt16 bid;
  AiUInt16 trw;
  AiUInt16 lcw;
  AiUInt16 lsw;
  AiUInt32 bufp;
  AiUInt32 ttag;
}
TY_API_RT_SA_MSG_DSP;
```

AiUInt16 bid

Indicates the absolute buffer id that has been updated by the last transfer. If no transfer has been received yet the bid returned will be the last buffer in the queue. The BUF_STAT value of the trw word below can be used to check if the data buffer contains data or not.

Value	Description
0..2047	Current Buffer Index

AiUInt16 trw

Transfer Report Word

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
0 (reserved)		BUF_STAT		0 (reserved)			

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 (reserved)			RBF	ERR_REPORT			

BUF_STAT

Buffer Status

Value	Constant	Description
0	**API_BUF_NOT_USED**	Buffer not used
1	**API_BUF_FULL**	Buffer full
2	**API_BUF_EMPTY**	Buffer empty
3		reserved

RBF

Received Bus Flag

Value	Constant	Description
0	**API_RCV_BUS_SECONDARY**	Secondary Bus
1	**API_RCV_BUS_PRIMARY**	Primary Bus

ERR_REPORT

Error Report Field

Value	Constant	Description
0	**API_RT_REPORT_NO_ERR**	No error
1	**API_RT_REPORT_ERR_FRAME_COD**	Coding or Framing error
2	**API_RT_REPORT_ERR_RSP_TIMEOUT**	Response Timeout error

`AiUInt16 lsw`

Last Status Word – See Figure 8.1.

`AiUInt16 lcw`

Last Command Word – See Figure 8.2.

`AiUInt32 bufp`

Current Data Buffer Pointer (Byte-Address)

`AiUInt32 ttag`

32-bit Time Tag Word

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
MINUTES						SECONDS	

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
SECONDS				MICROSECONDS			

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
MICROSECONDS							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MICROSECONDS							

MINUTES

Value	Description
0..59	Minutes of hour

SECONDS

Value	Description
0..59	Seconds of minute

MICROSECONDS

Value	Description
0..999999	Microseconds of second

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.25 ApiCmdRTSAMsgReadEx

Prototype:

```
AiReturn ApiCmdRTSAMsgReadEx(AiUInt32 ul_ModuleHandle,
                             TY_API_RT_SA_MSG_READ_IN *px_MsgReadIn,
                             TY_API_RT_SA_STATUS_EX *px_RtSaStatus);
```

Purpose:

This function is used to read the execution status of the specified RT Subaddress/Mode code of the selected AIM board Remote Terminal (identified by its RT Address), including status queue information

Input:

TY_API_RT_SA_MSG_READ_IN *px_MsgReadIn

RT Status input information

```
typedef struct ty_api_rt_sa_msg_read_in
{
  AiUInt32 ul_RtAddr;
  AiUInt32 ul_SA;
  AiUInt32 ul_SaType;
  AiUInt32 ul_Biu;
}
TY_API_RT_SA_MSG_READ_IN;
```

AiUInt32 ul_RtAddr

Value	Description
0..31	Remote Terminal Address

AiUInt32 ul_SA

Value	Description
1..30	RT Subaddress
0..31	Mode code number

AiUInt32 ul_SaType

Subaddress Type

Value	Constant	Description
0	**API_RT_TYPE_RECEIVE_SA**	Receive Subaddress
1	**API_RT_TYPE_TRANSMIT_SA**	Transmit Subaddress
2	**API_RT_TYPE_RECEIVE_MODECODE**	Receive Mode code
3	**API_RT_TYPE_TRANSMIT_MODECODE**	Transmit Mode code

AiUInt32 ul_Biu

The same as the biu parameter of other functions.

Output:

TY_API_RT_SA_STATUS_EX *px_RtSaStatus

RT SA Message Status information

```
typedef struct ty_api_rt_sa_status_queue {
  AiUInt32 ul_SqCtrlWord;
  AiUInt32 ul_SqStatusWords;
  AiUInt32 ul_SqActBufPtr;
  AiUInt32 ul_SqTimeTag;
}
TY_API_RT_SA_STATUS_QUEUE;

typedef struct ty_api_rt_sa_event_queue {
  AiUInt32 ul_EqEntryWord1;
  AiUInt32 ul_EqEntryWord2;
  AiUInt32 ul_EqEntryWord3;
  AiUInt32 ul_EqEntryWord4;
}
TY_API_RT_SA_EVENT_QUEUE;

typedef struct ty_api_rt_sa_status_info {
  AiUInt32 ul_ActBufferId;
  AiUInt32 ul_ActionWord;
  AiUInt32 ul_XferCnt;
  AiUInt32 ul_ErrCnt;
}
TY_API_RT_SA_STATUS_INFO;

typedef struct ty_api_rt_sa_status_ex {
  TY_API_RT_SA_STATUS_QUEUE x_XferSQueue[256];
  TY_API_RT_SA_EVENT_QUEUE x_XferEQueue[1];
  TY_API_RT_SA_STATUS_INFO x_XferInfo;
  AiUInt32 ul_BufferQueueSize;
}
TY_API_RT_SA_STATUS_EX;
```

AiUInt32 ul_BufferQueueSize

Indicates the data buffer queue size set with ApiCmdRTBHDef(). This queue size also indicates how many status queue entries are valid

AiUInt32 X_XferInfo.ul_ActBufferId

Actual Buffer/Status Queue Identifier of the transfer that is currently processed

AiUInt32 X_XferInfo.ul_ActionWord

Reserved (0)

AiUInt32 X_XferInfo.ul_XferCnt

Number of Transfers executed

AiUInt32 X_XferInfo.ul_ErrCnt

Number of Transfer errors

Note:
ul_XferCnt and ul_ErrCnt are RT global. Dedicated SA counters are only available in the BM activity page.

AiUInt32 x_XferEQueue[].ul_EqEntryWord1

Reserved (0)

AiUInt32 x_XferEQueue[].ul_EqEntryWord2

Reserved (0)

AiUInt32 x_XferEQueue[].ul_EqEntryWord3

Reserved (0)

AiUInt32 x_XferEQueue[].ul_EqEntryWord4

Reserved (0)

AiUInt32 x_XferSQueue[].ul_SqCtrlWord

Status Queue control word

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
reserved		BUF_STAT		reserved			

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
reserved		RBUS		ERR			

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
CMD							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CMD							

BUF_STAT

Buffer Status

Value	Constant	Description
0	**API_BUF_NOT_USED**	Buffer not used
1	**API_BUF_FULL**	Buffer full
2	**API_BUF_EMPTY**	Buffer empty
3		reserved

RBUS

Received Bus Flag

Value	Constant	Description
0	**API_RCV_BUS_SECONDARY**	Secondary Bus
1	**API_RCV_BUS_PRIMARY**	Primary Bus

ERR_REPORT

Error Report Field

Value	Constant	Description
0	**API_RT_REPORT_NO_ERR**	No error
1	**API_RT_REPORT_ERR_FRAME_COD**	Coding or Framing error
2	**API_RT_REPORT_ERR_RSP_TIMEOUT**	Response Timeout error

CMD

Command Word

`AiUInt32 x_XferSQueue[].ul_ SqStatusWords`

Last received Status Word on Command Word

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
reserved							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
reserved							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
STATUS_WORD							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STATUS_WORD							

`AiUInt32 x_XferSQueue[].ul_ SqActBufPtr`

Actual data buffer address relative to the start of the Global RAM

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
						BUF_PTR	

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
BUF_PTR							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
BUF_PTR							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BUF_PTR							

`AiUInt32 x_XferSQueue[].ul_ SqTimeTag`

32-bit Time Tag Word

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
MINUTES_OF_HOUR (0..59)						SEC_OF_MIN	

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
SEC_OF_MINUTE (0..59)				MICROSEC_OF_SEC			

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
MICROSEC_OF_SEC (0..999999)							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MICROSEC_OF_SEC (0..999999)							

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.26 ApiCmdRTStart

Prototype:

```
AiReturn ApiCmdRTStart(AiUInt32 ul_ModuleHandle,
                       AiUInt8 biu);
```

Purpose:

This command is used to start the Remote Terminal operation for all assigned RTs.

Input:

none

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

8.1.27 ApiCmdRTStatusRead

Prototype:

```
AiReturn ApiCmdRTStatusRead(AiUInt32 ul_ModuleHandle,
                            AiUInt8 biu,
                            TY_API_RT_STATUS_DSP *pdsp);
```

Purpose:

This function is used to read the execution status of the RT operation and the global RT message and error counters.

Input:

none

Output:

`TY_API_RT_STATUS_DSP *pdsp`

RT Status information

```
typedef struct ty_api_rt_status_dsp {
  AiUInt8 status;
  AiUInt8 padding1;
  AiUInt16 padding2;
  AiUInt32 glb_msg_cnt;
  AiUInt32 glb_err_cnt;
}
TY_API_RT_STATUS_DSP;
```

`AiUInt8 status`

Remote Terminal execution status

Value	Constant	Description
1	**API_RT_STATUS_HALTED**	RT halted
2	**API_RT_STATUS_BUSY**	RT busy

`AiUInt8 padding1`

0 (reserved)

`AiUInt16 padding2`

0 (reserved)

`AiUInt32 glb_msg_cnt`

Global RT Message Counter

`AiUInt32 glb_err_cnt`

Global RT Error Counter

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9 BUS MONITOR FUNCTIONS

Chapter 9 defines the Bus Monitor function calls of the API S/W Library. The BM functions provide configuration of the Bus Monitor for Chronological recording of all or filtered data streams. Single or multiple sequenced triggers can be programmed to trigger on error, word and/or data word in limits. The function calls in this table are listed in a functional order, however, the detailed descriptions of the BM function calls in the following sections are in alphabetical order

Table 9.1: Bus Monitor Function Descriptions

Function	Description
ApiCmdBMActRead	Reads BM Bus Activity transfer/error counters
ApiCmdBMCapMode	Configures the Capture/Recording mode of the BM
ApiCmdBMDytagMonDef	Define a dytag monitor id
ApiCmdBMDytagMonRead	Read the actual dytag monitor status
ApiCmdBMFilterIni	Disables the monitoring of specific RT SA/Mode codes
ApiCmdBMFTWIni	Defines the bit pattern to be used by the BM to initiate a Start Trigger Event and/or Stop Trigger Event used for Start/Stop of the “Data Capture”
ApiCmdBMHalt	Starts the chronological BM operation
ApiCmdBMIllegalIni	Sets up the BM to tag/not tag illegal command transfers to specific RT SA/Mode codes
ApiCmdBMIni	Initializes the Bus Monitor
ApiCmdBMIniMsgFltRec	Defines the command words used for filtering 1553 transfers to determine what data the BM will record when in Message Filter Recording Mode
ApiCmdBMIntrMode	Enables/disables the generation of interrupt and strobe outputs for various BM conditions
ApiCmdBMReadMsgFltRec	Retrieves multiple 1553 message transfers from the Monitor Buffer in one of four special formats (for data recorded in Message Filter Recording Mode)
ApiCmdBMRTActRead	Reads the BM transfer/error counters for a specified RT
ApiCmdBMRTSAActRead	Reads the BM transfer/error counters for a specified RT Subaddress
ApiCmdBMStackEntryFind	Finds a specific BM entry in the Monitor buffer
ApiCmdBMStackEntryRead	Obtains information about a specific BM entry in the BM buffer
ApiCmdBMStackpRead	Obtains the BM buffer pointers to be used to index into the Monitor Buffer to read entries
ApiCmdBMStart	Starts the chronological BM operation
ApiCmdBMStatusRead	Reads the status of the BM
ApiCmdBMSWXMIni	Enables the bits in the BM Status Word Exception Mask to be used by the BM to check the status word for errors/exceptions

Function	Description
ApiCmdBMTCBIni	Sets up the Trigger Control Block which defines the conditions evaluated by the BM to generate a Start/Stop Trigger Event
ApiCmdBMTCIIni	Defines the next Trigger Control Block to be evaluated for the next trigger
ApiCmdBMTIWIni	Arms the BM with the Triggers to be evaluated
ApiCmdDataQueueOpen	Creates a Data Queue on the ASP
ApiCmdDataQueueClose	Closes the Data Queue
ApiCmdDataQueueRead	Reads from the Data Queue
ApiCmdDataQueueControl	Starts/stops/resumes/flushes the Data Queue
ApiCmdQueueFlush	Flush the messages recorded while in Record with Queuing mode
ApiCmdQueueHalt	Stops queueing bus data to the Monitor buffer
ApiCmdQueueIni	Initializes the Record with Queueing process
ApiCmdQueueRead	Read a queued 1553 transfer message in the Monitor buffer
ApiCmdQueueStart	Starts queueing bus data to the Monitor buffer
ApiCmdScopeSetup	Setup and initialize the MIL-Scope
ApiCmdScopeStart	Start the APE MIL-Scope
ApiCmdScopeStatus	Status of the APE MIL-Scope
ApiCmdScopeStop	Stop the APE-Mil-Scope
ApiCmdScopeReset	Reset the APE MIL-Scope
ApiCmdScopeCalibrate	Calibrate a APX, ACX MIL-Scope
ApiCmdScopeOffsetCompensation	Perform an offset compensation on a APE MIL-Scope
ApiCmdScopeTriggerDef	Define a trigger condition for the APX, ACX MIL-Scope
ApiCmdScopeTriggerDefEx	Define a trigger condition for the APE MIL-Scope
ApiCreateScopeBuffer	Allocate a buffer to receive APE MIL-Scope data.
ApiCreateScopeBufferList	Allocate a list of buffers to receive APE MIL-Scope data.
ApiFreeScopeBuffer	Free a buffer scope data buffer.
ApiProvideScopeBuffers	Provide a list of scope buffers to the system driver.
ApiWaitForScopeFinished	Block the current execution thread until the data is available.

9.1 Low Speed Functions

9.1.1 ApiCmdBMActRead

Prototype:

```
AiReturn ApiCmdBMActRead(AiUInt32 ul_ModuleHandle,
                         AiUInt8 biu,
                         AiUInt16 entry_no,
                         TY_API_BM_ACT_DSP *pact);
```

Purpose:

This function is used to find and read the Bus Monitor Bus Activity Transfer/Error counters and returns the corresponding RT Subaddress/Mode code on which activity occurs. Starting with the entry number 0 only active RT Subaddress/Mode codes are reported. In the last active entry number+1, ‘fnd’ is set to zero. Note that activity is recorded by the AIM board whenever the Bus Monitor is enabled, independent from the occurrence of the trigger event.

Input:

AiUInt16 entry_no

Value	Description
0..4095	Activity List Entry Number to identify active Activity List entries 0: First active entry 1: Second active entry 2..n: 3rd to nth active entry **Note:** An entry is <u>active</u> when its Transfer or Error Counter is not equal to zero.

Output:

TY_API_BM_ACT_DSP *pact

BM Activity information

```
typedef struct ty_api_bm_act_dsp {
  AiUInt8 fnd;
  AiUInt8 rt;
  AiUInt8 tr;
  AiUInt8 sa;
  AiUInt32 cc;
  AiUInt32 ec;
  AiUInt32 et;
}
TY_API_BM_ACT_DSP;
```

AiUInt8 fnd

Value	Constant	Description
0	**API_BM_ENTRY_NOT_FOUND**	Entry not found
1	**API_BM_ENTRY_FOUND**	Entry found

AiUInt8 rt

Remote Terminal Address (if ‘fnd’ = 1)

AiUInt8 tr

Subaddress Type (if ‘fnd’ = 1)

Value	Constant	Description
0	**API_RT_TYPE_RECEIVE_SA**	Receive Subaddress
1	**API_RT_TYPE_TRANSMIT_SA**	Transmit Subaddress
2	**API_RT_TYPE_RECEIVE_MODECODE**	Receive Mode code
3	**API_RT_TYPE_TRANSMIT_MODECODE**	Transmit Mode code

AiUInt8 sa

Subaddress / Mode code (if fnd = 1)

AiUInt32 cc

Number of commands detected for <RT>_<TR>_<SA>_<XX>
(if fnd = 1) (32 Bit, MS-Byte first)

AiUInt32 ec

Number of errors detected for <RT>_<TR>_<SA>_<XX>
(if Efnd = 1) (32 Bit, MS-Byte first)

AiUInt32 et

Error Type Word Indicates the type of faults which are detected. All fault bits are Ored together so that this word stores all detected faults for this RT Subaddress/Mode code since the last Bus Monitor Start.

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
0 (reserved)							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0 (reserved)							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
ERR	ALTER	LCNT	HCNT	STAT	TADDR	GAP	ILLEGL

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TX	IWGAP	ISYNC	PAR	LBIT	HBIT	MANCH	NRESP

ERR	If Bit = 1: Any Error
ALTER	If Bit = 1: Alternate Bus Response Error
LCNT	If Bit = 1: Low Wordcount Error
HCNT	If Bit = 1: High Wordcount Error
STAT	If Bit = 1: Status Word Exception Error
TADDR	If Bit = 1: Terminal Address Error
GAP	If Bit = 1: Early Response or Gap too AiInt16
ILLEGL	If Bit = 1: Illegal Command Word
TX	If Bit = 1: Transmission on both MILbus channels
IWGAP	If Bit = 1: Interword Gap Error
ISYNC	If Bit = 1: Inverted Sync Error
PAR	If Bit = 1: Parity Error
LBIT	If Bit = 1: Low Bit Count Error
HBIT	If Bit = 1: High Bit Count Error
MANCH	If Bit = 1: Manchester Coding Error
NRESP	If Bit = 1: Terminal No Response Error

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.2 ApiCmdBMCapMode

Prototype:

```
AiReturn ApiCmdBMCapMode(AiUInt32 ul_ModuleHandle,
                         AiUInt8 biu,
                         TY_API_BM_CAP_SETUP *pcap);
```

Purpose:

This function is used to configure the Bus Monitor Capture/Recording mode on the AIM board according to the specified input parameters.

Input:

TY_API_BM_CAP_SETUP *pcap

BM Capture Mode description

```
typedef struct ty_api_bm_cap_setup {
  AiUInt8 cap_mode;
  AiUInt32 cap_tat;
  AiUInt16 cap_mcc;
  AiUInt16 cap_fsize;
}
TY_API_BM_CAP_SETUP;
```

AiUInt8 cap_mode

Capture Mode selection

Value	Constant	Description
0	**API_BM_CAPMODE_ALL**	**Capture 'ALL' (Standard Capture Mode)** The 'Trace After Trigger Count' cap_tat defines the number of entries to be stored in the Monitor Buffer after the occurrence of the Trigger Start event.

1	**API_BM_CAPMODE_ONLY**	**Capture 'ONLY' (Capture Only Mode)** The 'Message Capture Count' cap_mcc defines the number of messages, which shall be captured after the Trigger Start event. When the defined number of messages is stored in the Monitor buffer, capturing of data is stopped and restarted with the next occurrence of the Trigger Start event. This process is stopped when Bus Monitor operation is disabled or when the the number of entries defined by the 'Trace After Trigger Count' cap_tat has been stored in the Monitor Buffer. **Note:** This mode is not available on embedded devices! (see also chapter B.1.7 "Limitations for embedded board variants")
2	**API_BM_CAPMODE_RECORDING**	**Recording Mode** The Bus Monitor is setup in Continuous Capture mode. The 'Trace After Trigger Count' cap_tat is implicitely set to zero in order to enable Continuous Capturing (Recording). In Recording Mode the Buffer Full Interrupt is enabled, so that every time the AIM board Bus Monitor has captured a certain amount of data an interrupt is asserted (Half Buffer Full Interrupt).
3	**API_BM_CAPMODE_FILTER**	**Message Filter Recording Mode** The Bus Monitor is setup in Continuous Capture mode. The 'Trace After Trigger Count' cap_tat is implicitely set to zero in order to enable Continuous Capturing (Recording). No interrupts are enabled.

AiUInt32 cap_tat

Trace After Trigger Count Used in Standard Capture mode (cap_mode = 0) and Capture Only mode (cap_mode = 1). Amount of entries to be stored within the BM Buffer after occurrence of the Trigger Start event. If set to zero Continuous Capturing (Recording mode) is enabled.

AiUInt16 cap_mcc

Message Capture Count Number of messages stored after each Trigger Start event in Capture Only mode (**cap_mode = 1**).

AiUInt16 cap_fsize

0 (Reserved)

Output:

None

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.3 ApiCmdBMDytagMonDef (obsolete)

Prototype:

```
AiReturn ApiCmdBMDytagMonDef(AiUInt32 ul_ModuleHandle,
                             AiUInt8 biu,
                             TY_API_BM_DYTAG_MON_DEF *px_DytagMon);
```

Purpose:

This function is used to define the Dytag Monitor ID.

Input:

TY_API_BM_DYTAG_MON_DEF *px_DytagMon

BM Dytag Monitor Control information

```
typedef struct ty_api_bm_dytag_mon_def {
  AiUInt8 uc_Id;
  AiUInt8 uc_Con;
  AiUInt8 uc_RtAddr;
  AiUInt8 uc_SubAddrMsgId;
  AiUInt8 uc_SubAddrMsgId;
  AiUInt8 uc_Padding1;
  AiUInt16 uc_DytagType;
  AiUInt16 uc_DytagWPos;
  AiUInt16 uc_DytagBPos;
  AiUInt16 uc_DytagBLen;
  AiUInt16 uc_DytagStep;
}
TY_API_BM_DYTAG_MON_DEF;
```

AiUInt8 uc_Id

Value	Description
1..64	Dytag Monitor Identifier

AiUInt8 uc_Con

Dytag Monitor Control

Value	Constant	Description
0	**API_DIS**	Disable Dytag Monitor
1	**API_ENA**	Enable Dytag Monitor

AiUInt8 uc_RtAddr

Value	Description
0..31	Remote Terminal Address

AiUInt8 uc_SubAddrMsgId

Value	Description
1..30	RT Subaddress
0..31	Modecode

AiUInt8 uc_SubAddrMsgIdType

Subaddress Type

Value	Constant	Description
0	**API_RT_TYPE_RECEIVE_SA**	Receive Subaddress
1	**API_RT_TYPE_TRANSMIT_SA**	Transmit Subaddress
2	**API_RT_TYPE_RECEIVE_MODECODE**	Receive Modecode
3	**API_RT_TYPE_TRANSMIT_MODECODE**	Transmit Modecode

AiUInt8 uc_Padding1

Reserved (0)

AiUInt16 uw_DytagType

Value	Constant	Description
0	**API_DYTAG_FCT_DISABLE**	
1	**API_DYTAG_FCT_SAWTOOTH_16**	
2	**API_DYTAG_FCT_SAWTOOTH_8_LOW**	
3	**API_DYTAG_FCT_SAWTOOTH_8_HIGH**	

AiUInt16 uw_DytagWPos

Value	Description
0..31	Dytag Data Word position

AiUInt16 uw_DytagBPos

0 (Reserved)

AiUInt16 uw_DytagBLen

0 (Reserved)

AiUInt16 uw_DytagStep

0 (Reserved)

Output:

None

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.4 ApiCmdBMDytagMonRead (obsolete)

Prototype:

```
AiReturn ApiCmdBMDytagMonRead(AiUInt32 ul_ModuleHandle,
                              AiUInt8 biu,
                              TY_API_BM_DYTAG_MON_READ_CTRL *px_DytagMonCtrl,
                              TY_API_BM_DYTAG_MON_ACT *px_DytagMonAct);
```

Purpose:

This function is used to read the actual Dytag Monitor status.

Input:

TY_API_BM_DYTAG_MON_READ_CTRL * px_DytagMonCtrl

BM Dytag Monitor Control information

```
typedef struct ty_api_bm_dytag_mon_read_ctrl {
  AiUInt32 uc_Id;
}
TY_API_BM_DYTAG_MON_READ_CTRL;
```

AiUInt8 uc_Id

Value	Description
1..64	Dytag Monitor Identifier

Output:

TY_API_BM_DYTAG_MON_ACT * px_DytagMonAct

BM Dytag Monitor Activity information

```
typedef struct ty_api_bm_dytag_mon_act {
  AiUInt32 ul_Stale;
  AiUInt32 ul_Bad;
  AiUInt32 ul_Good;
}
TY_API_BM_DYTAG_MON_ACT;
```

AiUInt32 ul_Stale

Stale Dytag Counter (32-bit)

AiUInt32 ul_Bad

Bad Dytag Counter (32-bit)

`AiUInt32 ul_Good`

Good Dytag Counter (32-bit)

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.5 ApiCmdBMFilterIni

Prototype:

```
AiReturn ApiCmdBMFilterIni(AiUInt32 ul_ModuleHandle,
                           AiUInt8 biu,
                           AiUInt8 rt_addr,
                           AiUInt32 rx_sa,
                           AiUInt32 tx_sa,
                           AiUInt32 rx_mc,
                           AiUInt32 tx_mc);
```

Purpose:

This function is used to provide a filter capability on RT Subaddress and Mode code level for the specified RT Address. As a default, all Message Capture Enable / Filter bits are set with the **ApiCmdBMIni** function.

Input:

AiUInt8 rt_addr

Value	Description
0..31	Remote Terminal Address

AiUInt32 rx_sa

Message Capture Enable / Filter bits for RT Receive Subaddresses
Bit31...0 correspond to SA31...SA0.

AiUInt32 tx_sa

Message Capture Enable / Filter bits for RT Transmit Subaddresses
Bit31...0 correspond to SA31...SA0.

AiUInt32 rx_mc

Message Capture Enable / Filter bits for RT Receive Mode codes
Bit31...0 correspond to MC31...MC0.

AiUInt32 tx_mc

Message Capture Enable / Filter bits for RT Transmit Mode codes
Bit31...0 correspond to MC31...MC0.

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.6 ApiCmdBMFTWIni

Prototype:

```
AiReturn ApiCmdBMFTWIni(AiUInt32 ul_ModuleHandle,
                        AiUInt8 biu,
                        AiUInt8 con,
                        AiUInt8 htm,
                        AiUInt8 htc,
                        AiUInt8 stm,
                        AiUInt8 stc);
```

Purpose:

This function is used to define the BM Function Trigger Word bit patterns to be used by the BM to identify a Start Trigger Event and/or Stop Trigger Event. The Start/Stop Trigger Events are used by the BM for Start/Stop of the "Data Capture". The Function Trigger pattern specified is compared to the BM's Monitor Status Trigger pattern which is set/reset as specified for each trigger with the **ApiBMTCBIni** function.

Input:

AiUInt8 con

Value	Constant	Description
0	**API_BM_WRITE_ALL**	Write all bytes
1	**API_BM_WRITE_STC**	Write 'stc' bytes
2	**API_BM_WRITE_STM**	Write 'stm' bytes
3	**API_BM_WRITE_HTC**	Write 'htc' bytes
4	**API_BM_WRITE_HTM**	Write 'htm' bytes

AiUInt8 htm

Stop Trigger Mask pattern

AiUInt8 htc

Stop Trigger Compare pattern

AiUInt8 stm

Start Trigger Mask pattern

AiUInt8 stc

Start Trigger Compare pattern

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.7 ApiCmdBMHalt

Prototype:

```
AiReturn ApiCmdBMHalt(AiUInt32 ul_ModuleHandle,
                      AiUInt8 biu);
```

Purpose:

This command is used to stop the AIM board Chronological Bus Monitor operation.

Input:

None

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.8 ApiCmdBMIllegalIni

Prototype:

```
AiReturn ApiCmdBMIllegalIni(AiUInt32 ul_ModuleHandle,
                            AiUInt8 biu,
                            AiUInt8 rt_addr,
                            AiUInt32 rx_sa,
                            AiUInt32 tx_sa,
                            AiUInt32 rx_mc,
                            AiUInt32 tx_mc);
```

Purpose:

This function is used to setup the chronological Bus Monitor to indicate illegal Command Errors within the Monitor Buffer. This function is useful to customize Mode code usage or to trap illegal Command transfers on an RT Subaddress and Mode code level for the specified RT Address. As a default, all illegalize bits are cleared with the **ApiCmdBMIni** function.

Input:

`AiUInt8 rt_addr`

Value	Description
0..31	Remote Terminal Address

`AiUInt32 rx_sa`

Illegalize bits for RT Receive Subaddresses
Bit31...0 correspond to SA31...SA0.

`AiUInt32 tx_sa`

Illegalize bits for RT Transmit Subaddresses
Bit31...0 correspond to SA31...SA0.

`AiUInt32 rx_mc`

Illegalize bits for RT Receive Mode codes
Bit31...0 correspond to MC31...MC0.

`AiUInt32 tx_mc`

Illegalize bits for RT Transmit Mode codes
Bit31...0 correspond to MC31...MC0.

Output:

None

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.9 ApiCmdBMIni

Prototype:

```
AiReturn ApiCmdBMIni(AiUInt32 ul_ModuleHandle,
                     AiUInt8 biu);
```

Purpose:

This function is used to initialize the chronological Bus Monitor of the AIM board.

Execution of this function will:

(1) Enable the capturing of all RT Transmit and Receive Subaddresses and Mode codes. (This can later be modified using the **ApiCmdBMFilterIni function**.)

(2) Set the BM Status Word Exception Mask to 0x07FF such that all error/status bits in the Status word will be evaluated by the Bus Monitor for triggering and/or BM Error Word (Status word exception) indication. (This can later be modified using the **ApiCmdBMSWXMIni** function.)

(3) Disable the Illegal Command/Mode code tagging feature of the Bus Monitor for all RT Transmit and Receive Subaddresses and Mode codes. (This can later be modified using the **ApiCmdBMIllegalIni** function.)

Input:

None

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.10 ApiCmdBMIniMsgFltRec (obsolete)

Prototype:

```
AiReturn ApiCmdBMIniMsgFltRec(AiUInt32 ul_ModuleHandle,
                              AiUInt8 biu,
                              AiUInt8 cnt,
                              TY_API_BM_INI_MSG_FLT_REC *pmrec);
```

Purpose:

This function is used to setup the BM Message Filter Recording according to the specified Filter Command Words and data word positions when the Bus Monitor is setup in Message Filter Recording mode (see **ApiCmdBMCapMode** library function). Together with the ApiCmdBMReadMsgFltRec instruction this function can be used to implement a Message Filter Recording with output formats which are described in the following section.

Note:
For RT-RT transfers the first Command Word shall be specified as the Filter Command Word. If a single data word is required from the message specified in ‘cw’ the value of ‘pos1’ shall be set to the value of ‘pos2’. Values of ‘pos1’ or ‘pos2’ which do not match the Word Count range of the corresponding Command Word are returning a negative acknowledge character (NAK). Note that this function will override the initial settings from library function ApiCmdBMFilterIni’.

Input:

AiUInt8 cnt

Value	Description
1..255	Amount of Filter Command Words

TY_API_BM_INI_MSG_FLT_REC *pmrec

BM Message Filter Recording setup structure

```
typedef struct ty_api_bm_ini_msg_flt_rec {
  AiUInt16 cw;
  AiUInt8 pos1;
  AiUInt8 pos2;
}
TY_API_BM_INI_MSG_FLT_REC;
```

AiUInt16 cw

Filter Command Word (MS-Byte first)

AiUInt8 pos1

Value	Description
1..Word Count Field of CW	First Data Word position in message ‘cw’
32	If Word Count Field is 0

AiUInt8 pos2

Value	Description
1..Word Count Field of CW	Last Data Word position in message 'cw'
32	If Word Count Field is 0

Note:
Be sure that pos1 <= pos2!

Output:

None

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.11 ApiCmdBMIntrMode

Prototype:

```
AiReturn ApiCmdBMIntrMode(AiUInt32 ul_ModuleHandle,
                          AiUInt8 biu,
                          AiUInt8 imod,
                          AiUInt8 smod,
                          AiUInt8 res);
```

Purpose:

This function is used to enable/disable the Interrupts and Output Strobes related to the Bus Monitor mode of the AIM board.

Input:

AiUInt8 imod

Interrupt Mode

Value	Constant	Description
0	**API_BM_MODE_NO_INT**	No interrupt
1	**API_BM_MODE_HFI_INT**	Enable Monitor Buffer Full or Half Buffer Full Interrupt (Recording)
2	**API_BM_MODE_START_EVENT_INT**	Enable Interrupt on Capture Start Event
3	**API_BM_MODE_STOP_EVENT_INT**	Enable Interrupt on Capture Stop or End of Selective Capture Event

AiUInt8 smod

Output Strobe Mode

Value	Constant	Description
0	**API_BM_NO_STROBE**	No strobe
1	**API_BM_STROBE_ON_HFI**	Enable Strobe on Monitor Buffer Full or Half Buffer Full Recording)
2	**API_BM_STROBE_ON_START_EVENT**	Enable Strobe on Capture Start Event
3	**API_BM_STROBE_ON_STOP_EVENT**	Enable Strobe on Capture Stop or End of Selective Capture Event

AiUInt8 res

0 (Reserved)

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.12 ApiCmdBMReadMsgFltRec (obsolete)

Prototype:

```
AiReturn ApiCmdBMReadMsgFltRec(AiUInt32 ul_ModuleHandle,
                               AiUInt8 biu,
                               AiUInt8 mode,
                               AiUInt8 con,
                               AiUInt32 max_size,
                               AiUInt16 max_msg,
                               void *lpBuf,
                               AiUInt8 *ovfl,
                               AiUInt32 *lWordsRead)
```

Purpose:

This function is used to retrieve multiple 1553 message transfers from the Monitor Buffer in one of four fomats. The data is required to have been recorded by the BM while in Message Filter Recording mode (see **ApiCmdBMCapMode**).

Note:
This function will copy the recorded data to the output buffer specified by the user in 'lpBuf' input parameter. The amount of valid data in Words copied to the 'lpbuf' location is returned in 'rsize' (MS-Byte first). The format of the data copied to the 'lpbuf' location is based on the 'con'- parameter.

Input:

`AiUInt8 mode`

Value	Constant	Description
0	**API_BM_FLT_MODE_INDEPENDENT**	Returning 'lWordsRead' independend from Monitor Trigger event
1	**API_BM_FLT_MODE_DEPENDENT**	Calculation of 'lWordsRead' depending on Monitor Trigger event

`AiUInt8 con`

Message Filter Recording format – (See chapter 9.1.12.1)

Value	Constant	Description
0	**API_BM_MSG_FLT_FORMAT_0**	Format 0
1	**API_BM_MSG_FLT_FORMAT_1**	Format 1
2	**API_BM_MSG_FLT_FORMAT_2**	Format 2
3	**API_BM_MSG_FLT_FORMAT_3**	Format 3
4	**API_BM_MSG_FLT_FORMAT_4**	Format 4

`AiUInt32 max_size`

Size of the allocated buffer in Words

`AiUInt16 max_msg`

Number of messages to read

'con'	Value	Description
0..2	**0**	Read all messages
3	**>0**	Read 'max_msg' messages

`void *lpBuf`

Pointer to application buffer area (used to store recording data)

Output:

`AiUInt8 *ovfl`

> **Note:**
> If 'rsize' would be greater than 'max_size' or data would be overwritten within the Bus Monitor Stack, an overflow is indicated through 'ovfl'.

Overflow indication flag

Value	Constant	Description
0	**API_BM_NO_OVERFLOW**	No overflow
1	**API_BM_OVERFLOW**	Overflow indicated

`AiUInt32 *lWordsRead`

Returning size in Words / amount of Words read by this function

'mode'	Value	Description
0	**n**	Amount of valid formated data in Words
1	**0**	Monitor Trigger event was not detected
	>0	Amount of valid formated message filter recording data in Words if Monitor Trigger event was detected. The Message Filter Recording starts with the Monitor Trigger message.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.12.1 Message Filter Recording Formats

Each recorded message transfer recorded contains:

a. IRIG time or (milliseconds & microseconds)
b. Message Error information
c. Status word value
d. Command word value
e. Data words

Using the Message Filter Recording mode provides for retrieval of the recorded data in one of three available output formats (Format 0 – 2). The recorded data retrieved from the Monitor Buffer can be formatted into one of four different formats as follows:

Format 0 – LS-byte first with milli- and microseconds

Format 1 – MS-byte first with milli- and microseconds

Format 2 – LS-byte first with IRIG time

When not using the Message Filter Recording mode (i.e., the BM is in either Standard Data Cpature mode, Selective Data Capture mode, or Recording mode), Format 3 provides for retrieval of the recorded data:

Format 3 - Same as Format 2. To be used when not in Message Filter Recording mode.

The basic structure of all for formats includes a Message Header and Message Data:

Message Header
Message Data
Next Message Header
Next Message Data
...

The specific content for each Format is defined in the following sections.

9.1.12.1.1 Format 0

Message Header (LS-Byte always first)

7	0 15	8
	Message Status	
	LEN	
	Low Milli-Time	
	High Milli-Time	
	Micro-Time	
	Command Word	

Message Status

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
ERROR_INFO				0 (res)	STATUS_WORD		

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STATUS_WORD							

ERROR_INFO

Message Error Information

Value	Constant	Description
0	**API_BM_MSG_NO_ERR**	No error
1	**API_BM_MSG_ERR_NO_RESP**	Terminal no response error
2	**API_BM_MSG_ERR_MANCH**	Manchester coding error
3	**API_BM_MSG_ERR_HBIT**	High bit count error
4	**API_BM_MSG_ERR_LBIT**	Low bit count error
5	**API_BM_MSG_ERR_PARITY**	Parity error
6	**API_BM_MSG_ERR_ISYNC**	Inverted sync error
7	**API_BM_MSG_ERR_GAP**	Interword gap error
8	**API_BM_MSG_ERR_BOTH_CHN**	Transmission on both MILbus channels
9	**API_BM_MSG_ERR_MODECODE**	Reserved mode code error
10	**API_BM_MSG_ERR_EARLY_RESP**	Early response or gap too AiInt16
11	**API_BM_MSG_ERR_BIT_SET**	Message error bit set
12	**API_BM_MSG_ERR_SW_EXCEPT**	Status word exception error
13	**API_BM_MSG_ERR_HCNT**	High wordcount error
14	**API_BM_MSG_ERR_LCNT**	Low wordcount error
15	**API_BM_MSG_ERR_ILLEGL**	Illegal command word

STATUS_WORD

Bit10 to Bit0 from Status Word of MILbus

LEN

Value	Description
1..32	Amount of Data Words following the Message Header

Low Milli-Time – Message Time in milli-seconds (Low Word)

High Milli-Time – Time in milli-seconds (High Word)

Micro-Time

Value	Description
0..999	Message Time in micro-seconds

Command Word - Command Word

Message Data (LS-Byte always first)

9.1.12.1.2 Format 1

Message Header (MS-Byte always first)

15	8 7	0
	Message Status	
	LEN	
	Low Milli-Time	
	High Milli-Time	
	Micro-Time	
	Command Word	

Message Status

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
GERR	0	0	1	PERR	0 (reserved)		

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 (reserved)			TBUSY	0 (reserved)			

GERR	Global Error
PERR	Parity Error
TBUSY	Terminal Busy

LEN

Value	Description
1..32	Amount of Data Words following the Message Header

Low Milli-Time – Message Time in milli-seconds (Low Word)

High Milli-Time – Time in milli-seconds (High Word)

Micro-Time

Value	Description
0..999	Message Time in micro-seconds

Command Word - Command Word

Message Data (LS-Byte always first)

9.1.12.1.3 Format 2

Message Header (LS-Byte always first)

7	0 15	8
Message Status		
LEN		
TimeTag High (LOW)		
TimeTag High (HIGH)		
TimeTag Low (LOW)		
TimeTag Low (HIGH)		
Command Word		

Message Status

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
ERROR_INFO				BUS	STATUS_WORD		

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STATUS_WORD							

ERROR_INFO

Value	Constant	Description
0	**API_BM_MSG_NO_ERR**	No error
1	**API_BM_MSG_ERR_NO_RESP**	Terminal no response error
2	**API_BM_MSG_ERR_MANCH**	Manchester coding error
3	**API_BM_MSG_ERR_HBIT**	High bit count error
4	**API_BM_MSG_ERR_LBIT**	Low bit count error
5	**API_BM_MSG_ERR_PARITY**	Parity error
6	**API_BM_MSG_ERR_ISYNC**	Inverted sync error
7	**API_BM_MSG_ERR_GAP**	Interword gap error
8	**API_BM_MSG_ERR_BOTH_CHN**	Transmission on both MILbus channels
9	**API_BM_MSG_ERR_MODECODE**	Reserved mode code error
10	**API_BM_MSG_ERR_EARLY_RESP**	Early response or gap too AiInt16
11	**API_BM_MSG_ERR_BIT_SET**	Message error bit set
12	**API_BM_MSG_ERR_SW_EXCEPT**	Status word exception error
13	**API_BM_MSG_ERR_HCNT**	High wordcount error
14	**API_BM_MSG_ERR_LCNT**	Low wordcount error
15	**API_BM_MSG_ERR_ILLEGL**	Illegal command word

BUS

Value	Description
1	Primary Bus
0	Secondary Bus

STATUS_WORD

Bit10 to Bit0 from Status Word of MILbus

LEN

Value	Description
1..32	Amount of Data Words following the Message Header

TimeTag High (LOW) – Time Tag High Entry (Low Word)

TimeTag High (HIGH) – Time Tag High Entry (High Word)

High Word (High Entry)							
MSB							
Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
0 (reserved)							

High Word (High Entry)							
LSB							
Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0 (reserved)				DAY_OF_YEAR (1..365)			

Low Word (High Entry)							
MSB							
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
DAY_OF_YEAR (1..365)					HOURS_OF_DAY (0..23)		

Low Word (High Entry)							
LSB							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HOURS (0..23)		MINUTES_OF_HOUR (0..59)					

TimeTag Low (LOW) – Time Tag Low Entry (Low Word)

TimeTag Low (HIGH) – Time Tag Low Entry (High Word)

High Word (Low Entry)							
MSB							
Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
0 (reserved)						SECS (0..59)	

High Word (Low Entry)							
LSB							
Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
SECONDS_OF_MINUTE (0..59)				MICROSECONDS (0..999999)			

Low Word (Low Entry)							
MSB							
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
MICROSECONDS_OF_SECOND (0..999999)							

Low Word (Low Entry)							
LSB							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MICROSECONDS_OF_SECOND (0..999999)							

Command Word - Command Word

Message Data (LS-Byte always first)

9.1.12.1.4 Format 3

Format 3 is the same as Format 2. Format 3 output is provided when the user is recording BM data while in either the Standard Capture mode, Selective Capture Mode or Recording Mode. Format 3 output is to be used when not in Message Filter Recording mode (**ApiCmdBMCapMode** 'cap_mode' not equal to 3 (API_BM_CAPMODE_FILTER)) In addition the **ApiCmdBMIniMsgFltRec** should not be used.

9.1.12.1.5 Format 4

Format 4 output is provided when the user is recording BM data while in either the Standard Capture mode, Selective Capture Mode or Recording Mode. Format 4 output is to be used when not in Message Filter Recording mode (**ApiCmdBMCapMode** 'cap_mode' not equal to 3 (API_BM_CAPMODE_FILTER)) In addition the **ApiCmdBMIniMsgFltRec** should not be used. **Message Header (LS-Byte always first)**

7	0 15	8
	Message Status	
	LEN	
	TimeTag High (LOW)	
	TimeTag High (HIGH)	
	TimeTag Low (LOW)	
	TimeTag Low (HIGH)	
	First Command Word	
	Second Command Word (for RTRT transfer)	
	First Status Word	
	Second Status Word (for RTRT Transfer)	

Message Status

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
ERROR_INFO				BUS	reserved		

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
reserved							

ERROR_INFO

Value	Constant	Description
0	**API_BM_MSG_NO_ERR**	No error
1	**API_BM_MSG_ERR_NO_RESP**	Terminal no response error
2	**API_BM_MSG_ERR_MANCH**	Manchester coding error
3	**API_BM_MSG_ERR_HBIT**	High bit count error
4	**API_BM_MSG_ERR_LBIT**	Low bit count error
5	**API_BM_MSG_ERR_PARITY**	Parity error
6	**API_BM_MSG_ERR_ISYNC**	Inverted sync error
7	**API_BM_MSG_ERR_GAP**	Interword gap error
8	**API_BM_MSG_ERR_BOTH_CHN**	Transmission on both MILbus channels
9	**API_BM_MSG_ERR_MODECODE**	Reserved mode code error
10	**API_BM_MSG_ERR_EARLY_RESP**	Early response or gap too AiInt16
11	**API_BM_MSG_ERR_BIT_SET**	Message error bit set
12	**API_BM_MSG_ERR_SW_EXCEPT**	Status word exception error
13	**API_BM_MSG_ERR_HCNT**	High wordcount error
14	**API_BM_MSG_ERR_LCNT**	Low wordcount error
15	**API_BM_MSG_ERR_ILLEGL**	Illegal command word

BUS

Value	Description
1	Primary Bus
0	Secondary Bus

LEN

Value	Description
1..32	Amount of Data Words following the Message Header

TimeTag High (LOW) – Time Tag High Entry (Low Word)

TimeTag High (HIGH) – Time Tag High Entry (High Word)

High Word (High Entry)							
MSB							
Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
0 (reserved)							

High Word (High Entry)							
LSB							
Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0 (reserved)				DAY_OF_YEAR (1..365)			

Low Word (High Entry)							
MSB							
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
DAY_OF_YEAR (1..365)					HOURS_OF_DAY (0..23)		

Low Word (High Entry)							
LSB							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HOURS (0..23)		MINUTES_OF_HOUR (0..59)					

TimeTag Low (LOW) – Time Tag Low Entry (Low Word)

TimeTag Low (HIGH) – Time Tag Low Entry (High Word)

High Word (Low Entry)							
MSB							
Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
0 (reserved)						SECS (0..59)	

High Word (Low Entry)							
LSB							
Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
SECONDS_OF_MINUTE (0..59)				MICROSECONDS (0..999999)			

Low Word (Low Entry)							
MSB							
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
MICROSECONDS_OF_SECOND (0..999999)							

Low Word (Low Entry)							
LSB							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MICROSECONDS_OF_SECOND (0..999999)							

First Command Word – Command Word

Second Command Word – Command Word (only for RTRT transfers)

First Status Word

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
STATUS_WORD							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STATUS_WORD							

STATUS_WORD

For RTRT transfers this reflects the transmit RT status word

Second Status Word

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
STATUS_WORD							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STATUS_WORD							

STATUS_WORD (only for RTRT transfers)

For RTRT transfers this reflects the receive RT status word

Message Data (LS-Byte always first)

9.1.13 ApiCmdBMRTActRead

Prototype:

```
AiReturn ApiCmdBMRTActRead(AiUInt32 ul_ModuleHandle,
                           AiUInt8 biu,
                           AiUInt8 rt,
                           TY_API_BM_RT_ACT *pact)
```

Purpose:

This function is used to read the actual Transfer/Error counter and error type information for the specified RT Address.

Input:

`AiUInt8 rt`

Value	Description
0..31	Remote Terminal Address

Output:

`TY_API_BM_RT_ACT *pact`

BM RT Activity information

```
typedef struct ty_api_bm_rt_act {
  AiUInt32 mc;
  AiUInt32 ec;
  AiUInt32 et;
}
TY_API_BM_RT_ACT;
```

`AiUInt32 mc`

Message Counter (32-bit)

`AiUInt32 ec`

Error Counter (32-bit)

`AiUInt32 et`

Error Type Word: Indicates the type of faults which are detected. All fault bits are Ored together so that this word stores all detected faults for the specified RT.

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
0 (reserved)							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0 (reserved)							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
ERR	ALTER	LCNT	HCNT	STAT	TADDR	GAP	ILLEGL

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TX	IWGAP	ISYNC	PAR	LBIT	HBIT	MANCH	NRESP

ERR If Bit = 1: Any Error
ALTER If Bit = 1: Alternate Bus Response Error
LCNT If Bit = 1: Low Wordcount Error
HCNT If Bit = 1: High Wordcount Error
STAT If Bit = 1: Status Word Exception Error
TADDR If Bit = 1: Terminal Address Error
GAP If Bit = 1: Early Response or Gap too AiInt16
ILLEGL If Bit = 1: Illegal Command Word
TX If Bit = 1: Transmission on both MILbus channels
IWGAP If Bit = 1: Interword Gap Error
ISYNC If Bit = 1: Inverted Sync Error
PAR If Bit = 1: Parity Error
LBIT If Bit = 1: Low Bit Count Error
HBIT If Bit = 1: High Bit Count Error
MANCH If Bit = 1: Manchester Coding Error
NRESP If Bit = 1: Terminal No Response Error

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.14 ApiCmdBMRTSAActRead

Prototype:

```
AiReturn ApiCmdBMRTSAActRead(AiUInt32 ul_ModuleHandle,
                             AiUInt8 biu,
                             AiUInt8 rt,
                             AiUInt8 sa,
                             AiUInt8 sa_type,
                             TY_API_BM_RT_ACT *pact);
```

Purpose:

This function is used to read the actual transfer/error counts and error type information for the specified RT Subaddress.

Input:

AiUInt8 rt

Value	Description
0..31	Remote Terminal Address

AiUInt8 sa

Value	Description
1..30	RT Subaddress
0..31	Mode code

AiUInt8 sa_type

Subaddress Type

Value	Constant	Description
0	**API_RT_TYPE_RECEIVE_SA**	Receive Subaddress
1	**API_RT_TYPE_TRANSMIT_SA**	Transmit Subaddress
2	**API_RT_TYPE_RECEIVE_MODECODE**	Receive Mode code
3	**API_RT_TYPE_TRANSMIT_MODECODE**	Transmit Mode code
4	**API_RT_TYPE_ALL**	Overall counts

Output:

TY_API_BM_RT_ACT *pact

BM RT SA Activity information

```
typedef struct ty_api_bm_rt_act {
  AiUInt32 mc;
  AiUInt32 ec;
  AiUInt32 et;
}
TY_API_BM_RT_ACT;
```

AiUInt32 mc

Message Counter (32-bit)

AiUInt32 ec

Error Counter (32-bit)

AiUInt32 et

Error Type Word: Indicates the type of faults which are detected. All fault bits are Ored together so that this word stores all detected faults for the specified RT Subaddress.

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
0 (reserved)							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0 (reserved)							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
ERR	ALTER	LCNT	HCNT	STAT	TADDR	GAP	ILLEGL

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TX	IWGAP	ISYNC	PAR	LBIT	HBIT	MANCH	NRESP

ERR	If Bit = 1: Any Error
ALTER	If Bit = 1: Alternate Bus Response Error
LCNT	If Bit = 1: Low Wordcount Error
HCNT	If Bit = 1: High Wordcount Error
STAT	If Bit = 1: Status Word Exception Error
TADDR	If Bit = 1: Terminal Address Error
GAP	If Bit = 1: Early Response or Gap too AiInt16
ILLEGL	If Bit = 1: Illegal Command Word
TX	If Bit = 1: Transmission on both MILbus channels
IWGAP	If Bit = 1: Interword Gap Error
ISYNC	If Bit = 1: Inverted Sync Error
PAR	If Bit = 1: Parity Error
LBIT	If Bit = 1: Low Bit Count Error
HBIT	If Bit = 1: High Bit Count Error
MANCH	If Bit = 1: Manchester Coding Error
NRESP	If Bit = 1: Terminal No Response Error

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.15 ApiCmdBMStackEntryFind (obsolete)

Prototype:

```
AiReturn ApiCmdBMStackEntryFind(AiUInt32 ul_ModuleHandle,
                                AiUInt8 biu,
                                AiUInt8 ptype,
                                AiUInt8 sgn,
                                AiUInt32 offset,
                                AiUInt16 fspec,
                                AiUInt16 fdata,
                                AiUInt16 fmask,
                                TY_API_BM_STACK_FND *pfnd);
```

Purpose:

This function is used to find a specific entry in the Bus Monitor buffer. The entry type and the buffer location to start searching from is defined by the Buffer Pointer Read Mode plus offset.

Input:

AiUInt8 ptype

Buffer Pointer Read Mode

Value	Constant	Description
0	**API_BM_READ_STP**	Read from Monitor Start Entry Pointer (stp)
1	**API_BM_READ_CTP**	Read from Monitor Capture Start / Trigger Pointer (ctp)
2	**API_BM_READ_ETP**	Read from Monitor Buffer Fill / End Pointer (etp)

AiUInt8 sgn

Entry Offset Sign

Value	Constant	Description
0	**API_BM_SIGN_POS**	Positive entry offset (ptype=0 and ptype=1)
1	**API_BM_SIGN_NEG**	Negative entry offset (ptype=1 and ptype=2)

AiUInt32 offset

Number of entries (positive or negative offset)

Value	Description
0..20000h	Number of entries relative to the selected Buffer Pointer Read Mode

AiUInt16 fspec

BM Search Specification

Word Entry Specification (Bus Word Entry only)

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
0 (reserved)		CMD	DATA	STAT	CMD2	WORDB	WORDA

Entry Type to find

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 (reserved)						ENTRY_TYPE	

CMD	Command Word
DATA	Data Word
STAT	Status Word
CMD2	Second Command Word of RT-RT Transfer
WORDB	Word on MIL-Bus B
WORDA	Word on MIL-Bus A

ENTRY_TYPE

Entry Type to find

Value	Constant	Description
0	**API_BM_ENTRY_BUS_WORD**	Bus Word Entry
1	**API_BM_ENTRY_ERROR_WORD**	Error Word Entry
2	**API_BM_ENTRY_TIMETAG_LOW**	Reserved for Time Tag Low Entry
3	**API_BM_ENTRY_TIMETAG_HIGH**	Reserved for Time Tag High Entry

AiUInt16 fdata

Entry Word to find: For Bus Word Entry: Received Bus Word For Error Word Entry: Error specification as described for ApiCmdBMStackEntryRead'

AiUInt16 fmask

Entry Mask Word (Bus Word Entry and Error Word Entry) Defines which bits are relevant for the search operation.

Output:

TY_API_BM_STACK_FND *pfnd

BM Entry Find information

```
typedef struct ty_api_bm_stack_fnd {
  AiUInt8 efnd;
  AiUInt8 padding1;
  AiUInt16 padding2;
  AiUInt32 eptr;
  AiUInt32 entry;
}
TY_API_BM_STACK_FND;
```

AiUInt8 efnd

Value	Constant	Description
0	**API_BM_ENTRY_NOT_FOUND**	Entry not found
1	**API_BM_ENTRY_FOUND**	Entry found

AiUInt8 padding1

0 (reserved)

AiUInt16 padding2

0 (reserved)

AiUInt32 eptr

For efnd = 1: Pointer to Monitor Buffer Entry (Byte-Address)

AiUInt32 entry

For efnd = 1: Monitor Buffer Entry as described for ApiCmdBMStackEntryRead'

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.16 ApiCmdBMStackEntryRead (obsolete)

Prototype:

```
AiReturn ApiCmdBMStackEntryRead(AiUInt32 ul_ModuleHandle,
                                AiUInt8 biu,
                                AiUInt8 ptype,
                                AiUInt8 sgn,
                                AiUInt32 offset,
                                TY_API_BM_STACK_DSP *pentry);
```

Purpose:

This function is used to read a Monitor Buffer Entry specified by the Buffer Pointer Read Mode plus offset. Each Monitor Buffer Entry comprises a 32-bit Word.

Input:

AiUInt8 ptype

Buffer Pointer Read Mode

Value	Constant	Description
0	**API_BM_READ_STP**	Read from Monitor Start Entry Pointer (stp)
1	**API_BM_READ_CTP**	Read from Monitor Capture Start / Trigger Pointer (ctp)
2	**API_BM_READ_ETP**	Read from Monitor Buffer Fill / End Pointer (etp)
3	**API_BM_READ_ABS**	Read from address given in the parameter 'offset' relative to the start of the monitor memory area

AiUInt8 sgn

Entry Offset Sign

'ptype'	Value	Constant	Description
0,1	**0**	**API_BM_SIGN_POS**	Positive entry offset
1,2	**1**	**API_BM_SIGN_NEG**	Negative entry offset
3	**0**	**-**	reserved

AiUInt32 offset

'ptype'	Value	Description
0..2	**0..m**	Number of entries relative to the selected Buffer Pointer Read Mode (where m is the bus monitor size / 4)
3	**0..n**	Long word address offset to read from, relative to the start of the monitor memory area

Output:

TY_API_BM_STACK_DSP *pentry

BM Stack Entry Data

```
typedef struct ty_api_bm_stack_dsp {
  AiUInt32 entry;
}
TY_API_BM_STACK_DSP;
```

AiUInt32 entry

Monitor Buffer Entry

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
ENTRY_TYPE				EC	ENTRY_DATA see below		

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
ENTRY_DATA see below							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
ENTRY_DATA see below							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ENTRY_DATA see below							

ENTRY_TYPE

Entry Type Definition

Value	Constant	Description
0	**API_BM_ENTRY_NOT_UPDATED**	Entry not updated
1	**API_BM_ENTRY_ERROR_WORD**	Error Word Entry
2	**API_BM_ENTRY_TIMETAG_LOW**	Timetag Low Word Entry
3	**API_BM_ENTRY_TIMETAG_HIGH**	Timetag High Word Entry
4..7		Reserved
8	**API_BM_ENTRY_CW_PRIMARY**	Command Word on Primary Bus
9	**API_BM_ENTRY_CW2_PRIMARY**	Command Word 2 on Primary Bus
10	**API_BM_ENTRY_DW_PRIMARY**	Data Word on Primary Bus
11	**API_BM_ENTRY_SW_PRIMARY**	Status Word on Primary Bus
12	**API_BM_ENTRY_CW_SECONDARY**	Command Word on Secondary Bus
13	**API_BM_ENTRY_CW2_SECONDARY**	Command Word 2 on Secondary Bus
14	**API_BM_ENTRY_DW_SECONDARY**	Data Word on Secondary Bus
15	**API_BM_ENTRY_SW_SECONDARY**	Status Word on Secondary Bus

EC

Entry Connection Flag

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
see above					START	0 (reserved)	

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0 (reserved)							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
ERR	ALTER	LCNT	HCNT	STAT	TADDR	GAP	ILLEGL

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TX	IWGAP	ISYNC	PAR	LBIT	HBIT	MANCH	NRESP

ERR	If Bit = 1: Any Error
ALTER	If Bit = 1: Alternate Bus Response Error
LCNT	If Bit = 1: Low Wordcount Error
HCNT	If Bit = 1: High Wordcount Error
STAT	If Bit = 1: Status Word Exception Error
TADDR	If Bit = 1: Terminal Address Error
GAP	If Bit = 1: Early Response or Gap too AiInt16
ILLEGL	If Bit = 1: Illegal Command Word
TX	If Bit = 1: Transmission on both MILbus channels
IWGAP	If Bit = 1: Interword Gap Error
ISYNC	If Bit = 1: Inverted Sync Error
PAR	If Bit = 1: Parity Error
LBIT	If Bit = 1: Low Bit Count Error
HBIT	If Bit = 1: High Bit Count Error
MANCH	If Bit = 1: Manchester Coding Error
NRESP	If Bit = 1: Terminal No Response Error

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
see above					0 (res)	SECONDS	

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
SECONDS				MICROSECONDS			

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
MICROSECONDS							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MICROSECONDS							

SECONDS	Seconds of minute (0..59)
MICROSECONDS	Microseconds of second (0..999999)

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
see above					0 (reserved)		

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
0 (reserved)				DAYS			

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
DAYS				HOURS			

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HOURS		MINUTES					

DAYS Days of year (1..365)
HOURS Hours of day (0..23)
MINUTES Minutes of hour(0..59)

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
see above					START	0 (res)	GAP

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
GAP							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
BUS_WORD							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BUS_WORD							

START Start Trigger Flag
GAP Gap Time in 0.25μs steps
BUS_WORD Bus Word / Receive MILBus Word

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.17 ApiCmdBMStackpRead

Prototype:

```
AiReturn ApiCmdBMStackpRead(AiUInt32 ul_ModuleHandle,
                            AiUInt8 biu,
                            TY_API_BM_STACKP_DSP *pstackp);
```

Purpose:

This command can be used to read the current Bus Monitor Buffer Pointer values.

Note:
This function is only available for portability of old applications. For new developments please use the Data Queue functions instead.

Input:

None

Output:

`TY_API_BM_STACKP_DSP *pstackp`

BM Stack Pointer information

```
typedef struct ty_api_bm_stackp_dsp {
  AiUInt8 status;
  AiUInt8 padding1;
  AiUInt16 padding2;
  AiUInt32 stp;
  AiUInt32 ctp;
  AiUInt32 etp;
}
TY_API_BM_STACKP_DSP;
```

`AiUInt8 status`

Status Indication for Monitor Pointer

Value	Constant	Description
0	**API_BM_TRG_NOT_OCCURED**	Trigger not I (STP and CTP not valid)
1	**API_BM_START_EVENT_OCCURED**	Capture Start event (Trigger) I (STP, CTP and ETP valid)
2	**API_BM_STOP_EVENT_OCCURED**	Capture Stop event (Trigger) I (STP, CTP and ETP valid)

`AiUInt8 padding1`

0 (reserved)

`AiUInt16 padding2`

0 (reserved)

`AiUInt32 stp`

Monitor Start Entry Pointer

`AiUInt32 ctp`

Monitor Capture Start/Trigger Pointer

`AiUInt32 etp`

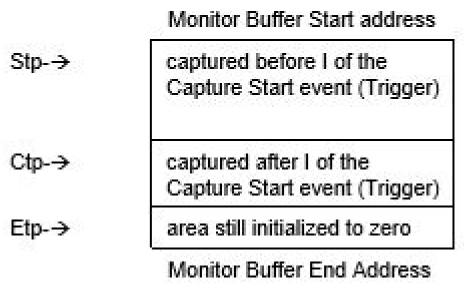
Monitor Buffer Fill/End Pointer

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

Figure 9.1: Monitor Buffer Pointer



Note:
If Etp >Monitor Buffer End Address, then the Monitor Buffer Fill Pointer of the AIM board Bus Monitor wraps around to the Monitor Buffer Start Address.

9.1.18 ApiCmdBMStart

Prototype:

```
AiReturn ApiCmdBMStart(AiUInt32 ul_ModuleHandle,
                       AiUInt8 biu);
```

Purpose:

This command is used to start the AIM board Chronological Bus Monitor operation.

Input:

None

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.19 ApiCmdBMStatusRead

Prototype:

```
AiReturn ApiCmdBMStatusRead(AiUInt32 ul_ModuleHandle,
                            AiUInt8 biu,
                            TY_API_BM_STATUS_DSP *pdsp);
```

Purpose:

This command is used to read the current execution status of the AIM board Chronological Bus Monitor.

Input:

None

Output:

`TY_API_BM_STATUS_DSP *pdsp`

BM Status information

```
typedef struct ty_api_bm_status_dsp {
  AiUInt8 men;
  AiUInt8 msw;
  AiUInt8 msp;
  AiUInt8 padding1;
  AiUInt32 glb_msg_cnt;
  AiUInt32 glb_err_cnt;
  AiUInt32 glb_word_cnt_sec;
  AiUInt32 glb_word_cnt_pri;
  AiUInt32 glb_msg_cnt_sec;
  AiUInt32 glb_msg_cnt_pri;
  AiUInt32 glb_err_cnt_sec;
  AiUInt32 glb_err_cnt_pri;
  AiUInt32 bus_load_sec;
  AiUInt32 bus_load_pri;
  AiUInt32 bus_load_sec_avg;
  AiUInt32 bus_load_pri_avg;
}
TY_API_BM_STATUS_DSP;
```

`AiUInt8 men`

Monitor Enable Status Word

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 (reserved)				TRIG	STOP	START	MEN

TRIG 0 (Reserved for Monitor Trigger Interrupt asserted)
STOP 0 (Reserved for Monitor Stop Interrupt asserted)
START 0 (Reserved for Monitor Start Interrupt asserted)
MEN

Value	Constant	Description
0	**API_DIS**	Monitor disabled
1	**API_ENA**	Monitor enabled

`AiUInt8 msw`

Monitor Status Word

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 (res)	SYNC	EXTRN	ERR	0 (res)	STOP	ACT	TRIG

SYNC Monitor in Sync if set to 1
EXTRN Monitor External Event I if set to 1
ERR Monitor Any Error Trigger I if set to 1
STOP Buffer full. Monitor stopped data capturing if set to 1
ACT Monitor data capturing active if set to 1
TRIG Any Trigger occurred if set to 1

`AiUInt8 msp`

Monitor Status Trigger Pattern (determines the BM Capture Start and Capture Stop), refer to ApiCmdBMFTWIni' function.

`AiUInt8 padding1`

Reserved (0)

`AiUInt32 glb_msg_cnt`

Global BM Message Counter

The Global BM Message Counter is incremented after each monitored transfer on the MILBus regardless of whether the transfer is errorneous or not. At RT-RT transfer, the Global BM Message Counter is incremented by two.

`AiUInt32 glb_err_cnt`

Global BM Error Counter

The Global BM Error Counter is incremented after each errorneous monitored transfer on the MILBus. If multiple errors occur at one transfer, only one error will be counted. At an errorneous RT-RT transfer, the Global BM Error Counter is incremented by two, if at both parts of transfer errors are detected. Otherwise, if only one part of the RT-RT transfer includes an error, the Global BM Error Counter is incremented by one.

`AiUInt32 glb_word_cnt_sec`

Global BM Word Counter (Secondary Bus)

The Global BM Word Counter is incremented after each received word on the secondary bus.

Note: This counter is not available on embedded devices and therefore returned as 0. Please see chapter “Limitations for specific boards” for details.

`AiUInt32 glb_word_cnt_pri`

Global BM Word Counter (Primary Bus)

The Global BM Word Counter is incremented after each received word on the primary bus.

Note: This counter is not available on embedded devices and therefore returned as 0. Please see chapter “Limitations for specific boards” for details.

`AiUInt32 glb_msg_cnt_sec`

Global BM Message Counter (Secondary Bus)

The Global BM Message Counter returns the same information than the parameter ‘**glb_msg_cnt**’ except that the the count is restricted to the secondary bus traffic.

Note: This counter is not available on embedded devices and therefore returned as 0. Please see chapter “Limitations for specific boards” for details.

`AiUInt32 glb_msg_cnt_pri`

Global BM Message Counter (Primary Bus)

The Global BM Message Counter returns the same information than the parameter ‘**glb_msg_cnt**’ except that the the count is restricted to the primary bus traffic.

Note: This counter is not available on embedded devices and therefore returned as 0. Please see chapter “Limitations for specific boards” for details.

`AiUInt32 glb_err_cnt_sec`

Global BM Error Counter (Secondary Bus)

The Global BM Error Counter returns the same information than the parameter ‘**glb_err_cnt**’ except that the the count is restricted to the secondary bus traffic.

Note: This counter is not available on embedded devices and therefore returned as 0. Please see chapter “Limitations for specific boards” for details.

`AiUInt32 glb_err_cnt_pri`

Global BM Error Counter (Primary Bus)

The Global BM Error Counter returns the same information than the parameter ‘**glb_err_cnt**’ except that the the count is restricted to the primary bus traffic.

Note: This counter is not available on embedded devices and therefore returned as 0. Please see chapter “Limitations for specific boards” for details.

`AiUInt32 bus_load_sec`

BM Bus Load (Secondary Bus)

Indicates the current bus load on the secondary bus in 0.01% steps, thus a value of 1000 means 10.00% bus load. The bus load is calculated from the traffic of the last 500ms.

Note:
This value is not available on embedded devices and therefore returned as 0. Please see chapter "Limitations for specific boards" for details.

`AiUInt32 bus_load_pri`

BM Bus Load (Primary Bus)

Indicates the current bus load on the primary bus in 0.01% steps, thus a value of 1000 means 10.00% bus load. The bus load is calculated from the traffic of the last 500ms.

Note:
This value is not available on embedded devices and therefore returned as 0. Please see chapter "Limitations for specific boards" for details.

`AiUInt32 bus_load_sec_avg`

BM Average Bus Load (Secondary Bus)

Indicates the average bus load on the secondary bus in 0.01% steps, thus a value of 1000 means 10.00% bus load. The bus load is calculated from the traffic beginning with the first received word until now.

Note:
This value is not available on embedded devices and therefore returned as 0. Please see chapter "Limitations for specific boards" for details.

`AiUInt32 bus_load_pri_avg`

BM Average Bus Load (Primary Bus)

Indicates the average bus load on the primary bus in 0.01% steps, thus a value of 1000 means 10.00% bus load. The bus load is calculated from the traffic beginning with the first received word until now.

Note:
This value is not available on embedded devices and therefore returned as 0. Please see chapter "Limitations for specific boards" for details.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

Note:
All count parameters and bus load parameters are cleared on restart of the AIM board Bus Monitor when the ApiCmdBMStart' function is called.

9.1.20 ApiCmdBMSWXMIni

Prototype:

```
AiReturn ApiCmdBMSWXMIni(AiUInt32 ul_ModuleHandle,
                         AiUInt8 biu,
                         AiUInt16 swxm);
```

Purpose:

This function is used to initialize the BM 1553 Status Word Exception Mask according to the specified input parameter.

> **Note:**
> As a default swxm is set to 0x07FF with the ApiCmdBMIni function.

Input:

`AiUInt16 swxm`

BM 1553 Status Word Exception Mask

Bit 10 – 0 are used to enable (Bit = 1) or disable (Bit = 0) the valuation of the corresponding bits in the RT 1553 Status Word response.

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.21 ApiCmdBMTCBIni

Prototype:

```
AiReturn ApiCmdBMTCBIni(AiUInt32 ul_ModuleHandle,
                        AiUInt8 biu,
                        AiUInt8 tid,
                        AiUInt8 ten,
                        TY_API_BM_TCB *ptcb);
```

Purpose:

This function set up a complete Trigger Control Block structure which defines the conditions evaluated by the Bus Monitor to generate a Start/Stop Trigger Event.

Input:

AiUInt8 tid

Value	Description
0..255	Trigger Control Block Index

AiUInt8 ten

Trigger Control Block enable

Value	Constant	Description
0	**API_DIS**	Disable Trigger Control Block
1	**API_ENA**	Enable Trigger Control Block

TY_API_BM_TCB *ptcb

Trigger Control Block description

```
typedef struct ty_api_bm_tcb {
  AiUInt8 tt;
  AiUInt8 sot;
  AiUInt8 tri;
  AiUInt8 inv;
  AiUInt8 tres;
  AiUInt8 tset;
  AiUInt8 tsp;
  AiUInt8 next;
  AiUInt8 eom;
  AiUInt8 padding1;
  AiUInt16 tdw;
  AiUInt16 tmw;
  AiUInt16 tuli;
  AiUInt16 tlli;
}
TY_API_BM_TCB;
```

AiUInt8 tt

Trigger Type

Value	Constant	Description
0	**API_BM_TRG_ERR_CONDITION**	Trigger on Error Condition (Static)
1	**API_BM_TRG_EXTERNAL_EVENT**	Trigger on External event (Static)
2	**API_BM_TRG_RECEIVED_WORD**	Trigger on Received Word (Dynamic)
3	**API_BM_TRG_DATA_VALUE**	Trigger on Data Value (Dynamic) **Note:** This mode is not available on embedded devices! (see also chapter B.1.7 “Limitations for embedded board variants”)
4	**API_BM_TRG_HS_EVENT**	Trigger on High Speed Event (only valid for STANAG3910 boards)

AiUInt8 sot

Generate External Strobe on Trigger

Value	Constant	Description
0	**API_DIS**	Disable
1	**API_BM_PULSE_STROBE_ON_TRG**	Pulse Monitor Output Strobe on Trigger

AiUInt8 tri

Interrupt on Trigger

Value	Constant	Description
0	**API_DIS**	Disable
1	**API_BM_ASSERT_TRG_INT**	Assert Trigger Interrupt if Trigger is true

AiUInt8 inv

Inversion of Limit Check

Value	Constant	Description
0	**API_DIS**	Disable
1	**API_BM_INVERT_RESULT**	Invert result of Limit Check

AiUInt8 tres (trigger reset)

8-bit field indicating which bit(s) of the Monitor Status Trigger pattern shall be cleared.

AiUInt8 tset (trigger set)

8-bit field indicating which bit(s) of the Monitor Status Trigger pattern shall be set.

AiUInt8 tsp (Trigger Specification

The usage of this byte depends on the selected Trigger Type (tt).

'tt'	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	**0**							
1	**0**							
2	**0**		**RXA**	**RXB**	**CW2**	**ST**	**DW**	**CW**
3			**WPOS**					

RXA If Bit=1: Trigger if Word received on Primary Bus

RXB If Bit=1: Trigger if Word received on Secondary Bus

CW2 If Bit=1: Trigger if Word is second Command Word for RT-RT transfer

ST If Bit=1: Trigger if Word is Status Word

DW If Bit=1: Trigger if Word is Data Word

CW If Bit=1: Trigger if Word is Command Word

WPOS Specifies the Word Position within the message to be checked for a specific data value or a range.

`AiUInt8 next`

Next Trigger Control Block Index if trigger condition becomes true for this Trigger Control Block.

'tt'	Value	Description
0	**0xFF**	disabled for Static Trigger
1	**0xFF**	disabled for Static Trigger
2	**0..254**	Sequence Trigger 0/1 modified
	255	do not modify Sequence Trigger 0/1 **Note:** This funcionality is not available on embedded devices and therefore must be set to 255! (see also chapter B.1.7 "Limitations for embedded board variants")
3	**0..254**	Sequence Trigger 0/1 modified
	255	do not modify Sequence Trigger 0/1 **Note:** This funcionality is not available on embedded devices and therefore must be set to 255! (see also chapter B.1.7 "Limitations for embedded board variants")

`AiUInt8 eom`

Next Trigger Control Block Index on EOM (End of Message) (if this trigger control block condition is not met for this transfer.)

'tt'	Value	Description
0	**0xFF**	disabled for Static Trigger
1	**0xFF**	disabled for Static Trigger
2	**0..254**	Sequence Trigger 0/1 modified
	255	do not modify Sequence Trigger 0/1
3	**0..254**	Sequence Trigger 0/1 modified
	255	do not modify Sequence Trigger 0/1

`AiUInt8 padding1`

0 (reserved)

`AiUInt16 tdw`

Trigger Data Word The usage of this Word depends on the selected Trigger Type (tt)

‘tt’	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
0	**ERR**	**ALTER**	**LWCNT**	**HWCNT**	**STAT**	**TADDR**	**GAP**	**ILLEGL**
1	**0**							
2	**WORD**							
3	**0**							

‘tt’	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	**TX**	**IGAP**	**ISYNC**	**PAR**	**LBIT**	**HBIT**	**MANCH**	**NRESP**
1	**0**							
2	**WORD**							
3	**0**							

ERR If Bit = 1: Any Error
ALTER If Bit = 1: Alternate Bus Response Error
LCNT If Bit = 1: Low Wordcount Error
HCNT If Bit = 1: High Wordcount Error
STAT If Bit = 1: Status Word Exception Error
TADDR If Bit = 1: Terminal Address Error
GAP If Bit = 1: Early Response or Gap too AiInt16
ILLEGL If Bit = 1: Illegal Command Word
TX If Bit = 1: Transmission on both MILbus channels
IWGAP If Bit = 1: Interword Gap Error
ISYNC If Bit = 1: Inverted Sync Error
PAR If Bit = 1: Parity Error
LBIT If Bit = 1: Low Bit Count Error
HBIT If Bit = 1: High Bit Count Error
MANCH If Bit = 1: Manchester Coding Error
NRESP If Bit = 1: Terminal No Response Error
WORD Data / Command / Status Word to trigger on

`AiUInt16 tmw`

Trigger Mask Word The usage of this Word depends on the selected Trigger Type (tt).

‘tt’	Value	Description
0	**0**	-
1	**0**	-
2	**n**	Used to define which bits of the received data/Command/Status Word are relevant for the Trigger condition check
3	**n**	Used to define which bits of the received Data Word are relevant for the Trigger condition check

AiUInt16 tuli

Trigger Upper Limit Value The usage of this Word depends on the selected Trigger Type (tt)

'tt'	Value	Description
0	**0**	-
1	**0**	-
2	**0**	-
3	**n**	Specifies the upper limit value of the range to be checked

AiUInt16 tlli

Trigger Lower Limit Value The usage of this Word depends on the selected Trigger Type (tt)

'tt'	Value	Description
0	**0**	-
1	**0**	-
2	**0**	-
3	**n**	Specifies the lower limit value of the range to be checked

Output:

None

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.22 ApiCmdBMTCIIni

Prototype:

```
AiReturn ApiCmdBMTCIIni(AiUInt32 ul_ModuleHandle,
                        AiUInt8 biu,
                        AiUInt8 rt,
                        AiUInt8 sa,
                        AiUInt8 sa_type,
                        AiUInt8 tagm,
                        AiUInt8 tfct,
                        AiUInt8 tid);
```

Purpose:

This function is used to define the next Trigger Control Block index to be evaluated for the selected Trigger Control Block. This function allows the user to setup a predefined trigger sequence for a specific Command Word on an RT Subaddress and Mode code level.

Input:

AiUInt8 rt

Value	Description
0..31	Remote Terminal Address

AiUInt8 sa

Value	Description
1..30	RT Subaddress
0..31	Mode code

AiUInt8 sa_type

Subaddress Type

Value	Constant	Description
0	**API_RT_TYPE_RECEIVE_SA**	Receive Subaddress
1	**API_RT_TYPE_TRANSMIT_SA**	Transmit Subaddress
2	**API_RT_TYPE_RECEIVE_MODECODE**	Receive Mode code
3	**API_RT_TYPE_TRANSMIT_MODECODE**	Transmit Mode code

AiUInt8 tagm

Time Tag Mode

Value	Constant	Description
0	**API_BM_TTMODE_LOW_HIGH**	Store Low and High Timetag

AiUInt8 tfct

0 (Reserved for Trigger Block Function)

`AiUInt8 tid`

Trigger Control Block Index

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.23 ApiCmdBMTIWIni

Prototype:

```
AiRetung ApiCmdBMTIWIni(AiUInt32 ul_ModuleHandle,
                        AiUInt8 biu,
                        AiUInt8 con,
                        AiUInt8 eti,
                        AiUInt8 aei,
                        AiUInt8 ti0,
                        AiUInt8 ti1);
```

Purpose:

This function is used to arm the BM with the Triggers to be evaluated. The triggers to be evaluated are stored into the Trigger Index Word according to the specified input parameters.

Input:

AiUInt8 con

Value	Constant	Description
0	**API_BM_WRITE_ALL**	Write all bytes
1	**API_BM_WRITE_TI0**	Write 'ti0' bytes
2	**API_BM_WRITE_TI1**	Write 'ti1' bytes **Note:** This mode is not available on embedded devices! (see also chapter B.1.7 "Limitations for embedded board variants")
3	**API_BM_WRITE_AEI**	Write 'aei' bytes
4	**API_BM_WRITE_ETI**	Write 'eti' bytes

AiUInt8 eti

External Trigger Index (= static Trigger)

Value	Constant	Description
0	**API_DIS**	Disable
>0		Trigger Control Block Index

AiUInt8 aei

Any Error Index (= static Trigger)

Value	Constant	Description
0	**API_DIS**	Disable
>0		Trigger Control Block Index

AiUInt8 ti0

Sequence Trigger 0 Control Block Index (= dynamic Trigger)

Value	Constant	Description
0	**API_DIS**	Disable
>0		Trigger Control Block Index

AiUInt8 ti1

Sequence Trigger 1 Control Block Index (= dynamic Trigger)

Value	Constant	Description
0	**API_DIS**	Disable
>0		Trigger Control Block Index

Output:

None

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.24 ApiCmdDataQueueClose

Prototype:

```
AiReturn ApiCmdDataQueueClose(AiUInt32 ul_ModuleHandle,
                              AiUInt32 id);
```

Purpose:

This function is used to close a previously created data queue.

Input:

`AiUInt32 id`

Unique Queue Id representing the data queue to be closed
See description of the parameter px_Queue->uc_Id in the function ApiCmdDataQueueOpen.

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.25 ApiCmdDataQueueControl

Prototype:

```
AiReturn ApiCmdDataQueueControl(AiUInt32 ul_ModuleHandle,
                                AiUInt32 id, AiUInt32 mode);
```

Purpose:

This function is used to control a data queue.

Input:

`AiUInt32 id`

Unique Queue Id representing the data queue to be controlled
See description of the parameter px_Queue->uc_Id in the function ApiCmdDataQueueOpen.

`AiUInt32 mode`

Control Mode

Value	Constant	Description
0	**API_DATA_QUEUE_CTRL_MODE_START**	Start the data capturing of the data queue **Note:** The data queue will be flushed before starting data capturing!
1	**API_DATA_QUEUE_CTRL_MODE_STOP**	Stop the data queue from data capturing **Note:** Data which is already in the Queue can still be read by the application!
2	**API_DATA_QUEUE_CTRL_MODE_RESUME**	Continue data capturing of a previously stopped data queue **Note:** The data queue will not be flushed before starting data capturing. Therefore the data may be inconsistent!
3	**API_DATA_QUEUE_CTRL_MODE_FLUSH**	Flush the data queue **Note:** All stored data within the data queue will be lost!

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.26 ApiCmdDataQueueOpen

Prototype:

```
AiReturn ApiCmdDataQueueOpen(AiUInt32 ul_ModuleHandle,
                             AiUInt32 id,
                             AiUInt32 * size);
```

Purpose:

This function is used to create a data queue on the Application Support Processor (ASP).

Input:

`AiUInt32 id`

Unique Queue Id representing the data queue to be opened

Constant	Description
API_DATA_QUEUE_ID_BM_REC_BIU1	BIU 1 recording
API_DATA_QUEUE_ID_BM_REC_BIU2	BIU 2 recording
API_DATA_QUEUE_ID_BM_REC_BIU3	BIU 3 recording
API_DATA_QUEUE_ID_BM_REC_BIU4	BIU 4 recording
API_DATA_QUEUE_ID_BM_REC_BIU5	BIU 5 recording
API_DATA_QUEUE_ID_BM_REC_BIU6	BIU 6 recording
API_DATA_QUEUE_ID_BM_REC_BIU7	BIU 7 recording
API_DATA_QUEUE_ID_BM_REC_BIU8	BIU 8 recording
API_DATA_QUEUE_ID_MIL_SCOPE	Reserved
API_DATA_QUEUE_GEN_ASP	Reserved

Output:

`AiUInt32 * size`

The size of the data queue in bytes allocated by the onboard ASP.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.27 ApiCmdDataQueueRead

Prototype:

```
AiReturn ApiCmdDataQueueRead(AiUInt32 ul_ModuleHandle,
                             TY_API_DATA_QUEUE_READ * data_queue_read,
                             TY_API_DATA_QUEUE_STATUS * info);
```

Purpose:

This function is used to read data from a previously created data queue. See Programmers Guide for a description of the Bus Monitor data format.

Input:

TY_API_DATA_QUEUE_READ * data_queue_read

Pointer to the TY_API_DATA_QUEUE_READ structure

```
typedef struct ty_api_data_queue_read {
  AiUInt32 id;
  void * buffer;
  AiUInt32 bytes_to_read;
}
TY_API_DATA_QUEUE_READ;
```

AiUInt32 data_queue_read->id

Unique Queue Id representing the data queue to read from.
See description of the parameter px_Queue->uc_Id in the function ApiCmdDataQueueOpen.

Void *data_queue_read->buffer

Pointer to the application memory where the data that was read is stored.

AiUInt32 data_queue_read->bytes_to_read

Amount of bytes to be read from the data queue

Note:
A value of 0 in ‘bytes_to_read’ can be used to read the data queue status !

Output:

TY_API_DATA_QUEUE_STATUS * info

Pointer to the TY_API_DATA_QUEUE_STATUS structure

```
typedef struct ty_api_data_queue_status {
```

```
  AiUInt32 status;
  AiUInt32 bytes_transfered;
  AiUInt32 bytes_in_queue;
  AiUInt64 total_bytes_transfered;
}
TY_API_DATA_QUEUE_STATUS;
```

AiUInt32 info ->status

Information about the data queue status

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
ENA		RESUM					

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
CAPS	CAPC	CHN_OPR					

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
OVF_A							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
				RB_ERR	LB_ERR	OVF_R	OVF_L

ENA	Queue is enabled if set to 1
RESUM	Queue has been resumed (see function ApiCmdDataQueueControl) if set to 1
CAPS	**MilScope only** indicates single shot capture mode
CAPC	**MilScope only** indicates continuous capture mode
CHN_OPR	**MilScope only** 00 = both channels A and B powered down 01 = channel B powered down 10 = normal operation (data align disabled) 11 = data align enabled (data from both channels available on rising edge of clock A, channel B data is delayed by ½ clock cycle) only available if one ADC is coupled to the other one!
OVF_A	Overflow detected on AIM board (ASP) if set to 1
RB_ERR	Remote buffer error if set to 1
LB_ERR	Local buffer error if set to 1
OVF_R	Overflow detected in remote buffer if set to 1
OVF_L	Overflow detected in local buffer if set to 1

AiUInt32 info ->bytes_transfered

Amount of bytes that have been read from the data queue

`AiUInt32 info ->bytes_in_queue`

Amount of bytes that is currently stored in the data queue

`AiUInt32 info->total_bytes_transfered`

Amount of bytes that have been read from the data queue since it was created.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.28 ApiCmdQueueFlush

Prototype:

```
AiReturn ApiCmdQueueFlush(AiUInt32 ul_ModuleHandle,
                          AiUInt8 biu);
```

Purpose:

This function is used to flush all 1553 messages from the indicated queue while in the Record with Queuing mode.

Input:

None

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.29 ApiCmdQueueHalt

Prototype:

```
AiReturn ApiCmdQueueHalt(AiUInt32 ul_ModuleHandle,
                         AiUInt8 biu);
```

Purpose:

This function is used to stop the queueing of 1553 messages while in the Record with Queuing mode.

Input:

None

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.30 ApiCmdQueueIni

Prototype:

```
AiReturn ApiCmdQueueIni(AiUInt32 ul_ModuleHandle,
                        AiUInt8 biu);
```

Purpose:

This function is used to configure the BM for Record with Queuing mode. (When programming in Record with Queueing mode, this function will be used to setup the BM record mode instead of the **ApiCmdBMCapMode** function.) In Record with Queuing mode, the BM will capture all 1553 traffic and utilize the Moniter Buffer as a queue by storing the messages as a circular queue of entries. The 1553 message(s) can then be retrieved from the queue, one at a time, using the **ApiCmdQueueRead** function.

Note:
ApiCmdBMReadMsgFltRec function for retrieving data from the Bus Monitor Buffer is not allowed in the Record with Queuing mode.

Input:

None

Output:

None

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.31 ApiCmdQueueRead

Prototype:

```
AiReturn ApiCmdQueueRead(AiUInt32 ul_ModuleHandle,
                         AiUInt8 biu,
                         TY_API_QUEUE_BUF *qmesg);
```

Purpose:

This function is used to read a single 1553 message structure (TY_API_QUEUE_BUF structure) from the head of the BM Monitor Buffer queue. If the queue is empty the function returns API_ERR_QUEUE_EMPTY. If the queue has overflowed the function returns API_ERR_QUEUE_OVERFLOW. After an overflow has l the queueing system must be stopped and restarted again with the **ApiCmdQueueHalt** and **ApiCmdQueueIni** / **ApiCmdQueueFlush** functions. The internal queue pointer is incremented automatically such that subsequent **ApiCmdQueueRead** function calls can be made immedately thereafter.

Input:

None

Output:

TY_API_QUEUE_BUF *qmesg

Pointer to the TY_API_QUEUE_BUF structure

```
typedef struct ty_api_queue_buf {
  AiUInt8 rt_addr;
  AiUInt8 sa_mc;
  AiUInt8 sa_type;
  AiUInt8 rt_addr2;
  AiUInt8 sa_mc2;
  AiUInt8 sa_type2;
  AiUInt8 word_cnt;
  AiUInt8 rbf_trw;
  AiUInt16 msg_trw;
  AiUInt16 status_word1;
  AiUInt16 status_word2;
  AiUInt16 buffer[32];
  AiUInt32 ttag;
}
TY_API_QUEUE_BUF;
```

AiUInt8 rt_addr

Value	Description
0..31	Remote Terminal Address (Receive RT for RT-RT transfers)

AiUInt8 sa_mc

Value	Description
1..30	RT Subaddress (SA of Receive RT in RT-RT transfers)
0..31	Mode code (MC of Receive RT in RT-RT transfers)

AiUInt8 sa_type

Subaddress Type

Value	Constant	Description
0	**API_RT_TYPE_RECEIVE_SA**	Receive Subaddress
1	**API_RT_TYPE_TRANSMIT_SA**	Transmit Subaddress
2	**API_RT_TYPE_RECEIVE_MODECODE**	Receive Mode code
3	**API_RT_TYPE_TRANSMIT_MODECODE**	Transmit Mode code

AiUInt8 rt_addr2

Only valid for RTRT transfers

Value	Description
0..31	Remote Terminal Address of Transmit RT

AiUInt8 sa_mc2

Only valid for RTRT transfers

Value	Description
1..30	RT Subaddress of Transmit RT
0..31	Mode code of Transmit RT

AiUInt8 sa_type2

Only valid for RTRT transfers
Subaddress Type of Transmit RT

Value	Constant	Description
0	**API_RT_TYPE_RECEIVE_SA**	Receive Subaddress
1	**API_RT_TYPE_TRANSMIT_SA**	Transmit Subaddress
2	**API_RT_TYPE_RECEIVE_MODECODE**	Receive Mode code
3	**API_RT_TYPE_TRANSMIT_MODECODE**	Transmit Mode code

AiUInt8 word_cnt

Value	Description
1..32	Amount fo Data Words associated with this message

AiUInt8 rbf_trw

Received Bus Flag

Value	Constant	Description
1	**API_RCV_BUS_PRIMARY**	Primary Bus
0	**API_RCV_BUS_SECONDARY**	Secondary Bus

AiUInt8 padding1

Reserved (0)

AiUInt16 msg_trw

Message Status (16-bit Word, LS-Byte first)

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
ERROR_INFO				BUS	STATUS_WORD		

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STATUS_WORD							

ERROR_INFO

Message Error Information

Value	Constant	Description
0	**API_BM_MSG_NO_ERR**	No error
1	**API_BM_MSG_ERR_NO_RESP**	Terminal no response error
2	**API_BM_MSG_ERR_MANCH**	Manchester coding error
3	**API_BM_MSG_ERR_HBIT**	High bit count error
4	**API_BM_MSG_ERR_LBIT**	Low bit count error
5	**API_BM_MSG_ERR_PARITY**	Parity error
6	**API_BM_MSG_ERR_ISYNC**	Inverted sync error
7	**API_BM_MSG_ERR_GAP**	Interword gap error
8	**API_BM_MSG_ERR_BOTH_CHN**	Transmission on both MILbus channels
9	**API_BM_MSG_ERR_MODECODE**	Reserved mode code error
10	**API_BM_MSG_ERR_EARLY_RESP**	Early response or gap too short
11	**API_BM_MSG_ERR_BIT_SET**	Message error bit set in status word (e.g. on RT address mismatch)
12	**API_BM_MSG_ERR_SW_EXCEPT**	Status word exception error
13	**API_BM_MSG_ERR_HCNT**	High wordcount error
14	**API_BM_MSG_ERR_LCNT**	Low wordcount error
15	**API_BM_MSG_ERR_ILLEGL**	Illegal command word

BUS

Bus Flag

Value	Constant	Description
1	**API_RCV_BUS_PRIMARY**	Primary Bus
0	**API_RCV_BUS_SECONDARY**	Secondary Bus

STATUS_WORD
Bit 10 to Bit 0 from Status Word of MILbus

AiUInt16 status_word1

Status Word (16-bit Word, LS-Byte first)

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
STATUS_WORD							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STATUS_WORD							

STATUS_WORD
Status Word of MILbus
For RTRT-Transfers this reflects the status word of the Transmit RT

AiUInt16 status_word2

Only valid for RTRT transfers
Status Word (16-bit Word, LS-Byte first)

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
STATUS_WORD							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
STATUS_WORD							

STATUS_WORD
Status Word of MILbus of the Receive RT

AiUInt16 buffer[32]

Message Data

AiUInt32 ttag

32-Bit Time Tag

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
0 (reserved)						SECS (0..59)	

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
SECONDS_OF_MINUTE (0..59)				MICROSECONDS (0..999999)			

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
MICROSECONDS_OF_SECOND (0..999999)							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MICROSECONDS_OF_SECOND (0..999999)							

Return Value:

AiReturn

This function returns API_Ok or API_ERR_QUEUE_EMPTY if no error occurred. If the return value is not equal to API_OK or API_ERR_QUEUE_EMPTY the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.32 ApiCmdQueueStart

Prototype:

```
AiReturn ApiCmdQueueStart(AiUInt32 ul_ModuleHandle,
                          AiUInt8 biu);
```

Purpose:

This function is used to start the queueing of 1553 messages from the stream of 1553 traffic.

Input:

None

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.33 ApiCmdScopeSetup

Prototype:

```
AiReturn ApiCmdScopeSetup(AiUInt32 ulModuleHandle,
                          TY_API_SCOPE_SETUP *px_ScopeSetup);
```

Purpose:

Note:
This function is only usable with AIM's MIL-Scope module.

This function is used to setup and initialize the MIL-Scope. There are two ADC channels available onboard. Both ADCs can be programmed separately. After calling this function with a different coupling than before, the application has to wait about 10 milli seconds for the Relais to settle before starting the scope.

9.1.33.1 Using Scope Functionality On APE Cards

With APE cards, the API functions ApiCreateScopeBuffer and ApiCreateScopeBufferList must be used to create memory buffers which will receive the recorded scope data.
The API function ApiProvideScopeBuffers marks a number of these buffers ready for data reception from Hardware.
Attached to each of these buffers is a user definable handler function, which is called once the corresponding buffer is filled with scope data.
Scope samples in the buffer are grouped to 3 samples per 32Bit data value, further on called **sample package**.
Packages in one buffer are consecutively numbered with Ids from 0 to n.
Equally, all samples in **one** buffer are consecutively numbered with Ids from 0 to n.
This is the sample layout in one of these sample packages, where samples are sorted chronologically from ID 0 to 2:

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
SAMPLE0							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
SAMPLE0		SAMPLE1					

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
SAMPLE1				SAMPLE2			

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SAMPLE2						Not Used	

The API provides several Macros to ease the sample extraction from raw data for the user. Parameter **pBuffer** of these Macros specifies the pointer to the raw data buffer (member pvBuffer of TY_API_SCOPE_BUFFER structure), while **data_size** parameter shall be set to actual number of bytes in the buffer (member ulDataSize of TY_API_SCOPE_BUFFER structure).

#define SCOPE_SAMPLE_PACKET_BY_PACKET_ID(pBuffer, packet_id)
Gets whole 32Bit value of the package with given ID.

#define SCOPE_SAMPLE_PACKET_BY_SAMPLE_ID(pBuffer, sample_id)
Gets whole 32Bit value of the package which contains the sample with given ID.

#define NUM_SCOPE_SAMPLE_PACKETS(data_size)
Gets number of sample packets in a buffer according to its size.

#define NUM_SCOPE_SAMPLES(data_size)
Gets number of scope samples in a buffer according to its size.

#define SCOPE_SAMPLE_BY_SAMPLE_ID(pBuffer, sample_id)
Gets value of the sample with the given ID.

Scope has to be started with ApiCmdScopeStart and can be stopped with ApiCmdScopeStop.
In single shot mode the API function ApiWaitForScopeFinished can be used to wait until all buffers are filled with scope data, or a user definable time out is reached.

Input:

TY_API_SCOPE_SETUP *px_ScopeSetup

Pointer to the list of TY_API_SCOPE_SETUP structure elements

```
typedef struct ty_api_scope_setup {
  AiUInt32 ul_CouplingPri;
  AiUInt32 ul_CouplingSec;
  AiUInt32 ul_SamplingRate;
  AiUInt32 ul_CaptureMode;
  AiUInt32 ul_OperationMode;
  AiUInt32 ul_SingleShotBuf;
}
TY_API_SCOPE_SETUP;
```

AiUInt32 px_ScopeTrg->ul_CouplingPri

Coupling mode of the primary ADC channel (MilScope PBI 1.x and PBI 2.x)

Value	Constant	Description
0	API_SCOPE_COUPLING_SECONDARY	Use signal from secondary channel.
1 – 3		Reserved
4	API_SCOPE_COUPLING_EXTERN	The differential inputs of the primary ADC channel are coupled to the general purpose connector on the front plate (e.g. for trigger from BIU). The expected signal level is 30V. **Note:** To prevent damages on the ADC, the input should be protected, e.g. with suppressor diodes.
5	API_SCOPE_COUPLING_STUB	The differential inputs of the primary ADC channel are coupled to the MILBus stub. The expected signal level is 30V.
6	API_SCOPE_COUPLING_STUB _3V	The differential inputs of the primary ADC channel are coupled to network emulation of the MIL-STD 1553 front end. **Note:** This coupling is only possible, if the front end is network coupled (see ApiCmdCalCplCon)

Additional coupling mode of the primary ADC channel (MilScope PBI 2.x and newer)

Value	Constant	Description
7	API_SCOPE_COUPLING_EXTERN_3V	The differential inputs of the primary ADC channel are coupled to the general purpose connector on the front plate (e.g. for trigger from BIU). The expected signal level is 3V. **Note:** To prevent damages on the ADC, the input should be protected, e.g. with suppressor diodes.

Note:
The define API_SCOPE_COUPLING_NETWORK value 6 is obsolete please use API_SCOPE_COUPLING_STUB_3V instead.

AiUInt32 px_ScopeTrg->ul_CouplingSec

Coupling mode of the secondary ADC channel (MilScope PBI 1.x and PBI 2.x)
See above description of px_ScopeTrg->ul_CouplingPri.

AiUInt32 px_ScopeTrg->ul_SamplingRate

Sampling Rate
The ADC has a fixed sampling rate of 50Msamples per sec. In some cases this resolution is not necessary, therefore the sampling rate is adjustable. E.g. if the sampling rate is 4, every 4th sample from the ADC will be stored, the others will be lost. The sampling rate can be calculated as follows:

Real Sampling-Rate in MHz = 50 MHz / (ul_SamplingRate+1)

Value	Constant	Description
0	API_SCOPE_RATE_EVERY	Every sample will be stored
1..255	-	Every (n+1)-th sample will be stored (e.g. 255 = every 256th sample will be stored)

AiUInt32 px_ScopeTrg->ul_CaptureMode

Capture Mode This parameter defines the capture mode of the MILScope. The MILScope is actually started and stopped with the function ApiCmdDataQueueControl(), ApiCmdScopeStart() on APE cards respectively.

Value	Constant	Description
1	API_SCOPE_START_CONTINUOUS	On APE cards, scope will record data until it is explicitly stopped, or no more buffers are provided.
2	API_SCOPE_START_SINGLE	On APE cards, the scope will record data until all buffers, that were provided before scope start, are full.

AiUInt32 px_ScopeTrg->ul_OperationMode

Operation Mode This parameter defines the operation mode of both channels of the MILScope.

Value	Constant	Description
0	API_SCOPE_OPR_CHANNEL_DISABLED	Both channels are disabled
1	API_SCOPE_OPR_CHANNEL_A	Only channel A is enabled
2	API_SCOPE_OPR_CHANNEL_B	Only channel B is enabled
3	API_SCOPE_OPR_CHANNEL_AB	Channel A and channel B are enabled

Additional operation modes (MilScope PBI 2.x and newer)

Value	Constant	Description
4	API_SCOPE_OPR_CHANNEL_A100	No longer supported
5	API_SCOPE_OPR_CHANNEL_B100	No longer supported

`AiUInt32 px_ScopeTrg->ul_SingleShotBuf`

Reserved (0)

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.34 ApiCmdScopeStart

Prototype:

```
AiReturn ApiCmdScopeStart(AiUInt32 ul_ModuleHandle);
```

Purpose:

This function is used to start APE milscope.

Note: This function is only usable with AIM's APE MIL-Scope module.

Input:

None

Output:

None

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.35 ApiCmdScopeStatus

Prototype:

```
AiReturn ApiCmdScopeStatus(AiUInt32 ul_ModuleHandle,
                           TY_API_SCOPE_STATUS* p_ScopeStatus,
                           AiUInt32* pulNumBuffersLeft);
```

Purpose:

This function returns the current statue of the APE-MIL-Scope. The function of not yet filled buffers is returned as well.

Note:
This function is only usable with AIM's APE MIL-Scope module.

Input:

None

Output:

TY_API_SCOPE_STATUS *p_ScopeStatus

Current MIL-Scope status.

Enum Value	Description
SCOPE_STATUS_WAIT_FOR_TRIGGER	Scope is started and waiting for trigger
SCOPE_STATUS_TRIGGERED	Scope was triggered and is now recording scope data
SCOPE_STATUS_STOPPED	Scope was stopped, either through call of ApiCmdScopeStop or due to all buffers being filled in single-shot mode
SCOPE_STATUS_OVERFLOW	Scope buffers ran out while recording scope data. Scope is stopped now

AiUInt32* pulNumBuffersLeft

Number of buffers that are still free for Scope data.

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.36 ApiCmdScopeStop

Prototype:

```
AiReturn ApiCmdScopeStop(AiUInt32 ul_ModuleHandle);
```

Purpose:

This function is used to stop the APE MIL-Scope and the MIL-Scope thread. This function should be called when the MIL-Scope is no longer used.

Note:
This function is only usable with AIM's APE MIL-Scope module.

Input:

None

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.37 ApiCmdScopeReset

Prototype:

```
AiReturn ApiCmdScopeReset(AiUInt32 ul_ModuleHandle);
```

Purpose:

This function is used to reset all APE-MIL-Scope settings to the default state.

Note: This function is only usable with AIM's APE MIL-Scope module.

Input:

None

Output:

None

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.38 ApiCmdScopeTriggerDef (obsolete)

Prototype:

```
AiReturn ApiCmdScopeTriggerDef(AiUInt32 ulModuleHandle,
                               TY_API_SCOPE_TRG *px_ScopeTrg);
```

Purpose:

This function could be used to define a trigger condition for the legacy APX, ACX MIL-Scope. This function is obsolete. Use the function ApiCmdScopeTriggerDefEx instead.

Note:
This function is only usable with AIM's APX, ACX MIL-Scope module.

Note:
Every 3rd valid sample is checked for the trigger condition. This means, that spikes who are smaller than 3*20ns*sampling rate may not be triggered!

Input:

TY_API_SCOPE_TRG *px_ScopeTrg

Pointer to the list of TY_API_SCOPE_TRG structure elements

```
typedef struct ty_api_scope_trg {
  AiUInt32 ul_TrgMode;
  AiUInt32 ul_TrgSrc;
  AiUInt32 ul_TrgValue;
  AiUInt32 ul_TrgNbSamples;
  AiUInt32 ul_TrgFlags;
  AiUInt32 ul_TrgDelay;
  AiUInt32 ul_TrgTbt;
}
TY_API_SCOPE_TRG;
```

AiUInt32 px_ScopeTrg->ul_TrgMode

Trigger mode on which condition the trigger is activated

Value	Constant	Description
0	API_SCOPE_MODE_GREATER_THAN	Higher than the value given in parameter **ul_TrgValue** (rising edge)
1	API_SCOPE_MODE_LESS_THAN	Lower than the value given in parameter **ul_TrgValue** (falling edge)
2	API_SCOPE_MODE_GREATER_OR_LESS_THAN	Higher than the positive value given in parameter **ul_TrgValue** or Lower than the negative value given in parameter **ul_TrgValue**
3	-	Reserved
4	API_SCOPE_MODE_GREATER_THAN_SAMPLES	Higher than the value given in parameter **ul_TrgValue** for a given number of samples (parameter **ul_TrgNbSamples**) (high pulse)
5	API_SCOPE_MODE_LESS_THAN_SAMPLES	Lower than the value given in parameter **ul_TrgValue** for a given number of samples (parameter **ul_TrgNbSamples**) (low pulse)
7	API_SCOPE_MODE_BIU	The MilScope is triggered by the BIU trigger, which has to be setup with the function ApiCmdBMTCBIni()

AiUInt32 px_ScopeTrg->ul_TrgSrc

Trigger source Primary or secondary bus on which the trigger is activated

Value	Constant	Description
0	API_SCOPE_SRC_PRIMARY	Primary Bus
1	API_SCOPE_SRC_SECONDARY	Secondary Bus

AiUInt32 px_ScopeTrg->ul_TrgValue

Trigger value, that is used to compare against. The voltage range that this value is representing is dependent on the coupling that was set with the function **ApiCmdScopeSetup()**.

Coupling	Value	Description
Extern 30V	**0..1023**	Representing the voltage range of -15V..+15V (steps of 29.3mV)
Extern 3V	**0..1023**	Representing the voltage range of -1.5V..+1.5V (steps of 2.9mV)
Stub 30V	**0..1023**	Representing the voltage range of -15V..+15V (steps of 29.3mV)
Stub 3V	**0..1023**	Representing the voltage range of -1.5V..+1.5V (steps of 2.9mV)

AiUInt32 px_ScopeTrg->ul_TrgNbSamples

Number of samples

'ul_TrgMode'	Value	Description
0, 1	**0**	Reserved
4, 5	**1..255**	Number of samples

`AiUInt32 px_ScopeTrg->ul_Flags`

Trigger flags

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
reserved (0)							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
reserved (0)							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
reserved (0)							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
reserved (0)							DIS_FDBT

DIS_FDBT

0 : The Trigger is enabled immediately

1 : The Trigger is enabled after the DMA Buffer chain is filled. This mode will always produce valid trace before trigger data. In this mode trigger conditions might be lost.

`AiUInt32 px_ScopeTrg->ul_TrgDelay`

Trigger Delay Time

This value defines the delay time after a trigger has occurred until the capturing is started. The time resolution is calculated from the sampling rate, which is setup with parameter **px_ScopeSetup->ul_SamplingRate** in the function **ApiCmdScopeSetup()**.

Value	Description
0..262143	Delay Time in steps of 1μs

`AiUInt32 px_ScopeTrg->ul_TrgTbt`

Trace Before Trigger

This value defines the time the data is captured before a trigger occurres.

Ul_TrgDelay	Value	Description
0	**0..262143**	Time in steps of 20ns before physical trigger event
>0	**0..262143**	Time in steps of 20ns before virtual trigger event

Note:
The trace before trigger time can only be set in single shot mode. In the continuous capturing mode, this value is ignored.

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.39 ApiCmdScopeTriggerDefEx

Prototype:

```
AiReturn ApiCmdScopeTriggerDefEx(AiUInt32 ulModuleHandle,
                                 TY_API_SCOPE_TRG_EX * px_ScopeTrg);
```

Purpose:

This function is used to define a trigger condition for the APE MIL-Scope.

Note:
This function is only usable with AIM's APE MIL-Scope module.

Input:

```
typedef struct ty_api_scope_trg {
  AiUInt32 ul_TrgMode;
  AiUInt32 ul_TrgSrc;
  AiUInt32 ul_TrgValue;
  AiUInt32 ul_TrgValue2;
  AiUInt32 ul_TrgNbSamples;
  AiUInt32 ul_TrgNbSamples2;
  AiUInt32 ul_TrgFlags;
  AiUInt32 ul_TrgDelay;
  AiUInt32 ul_TrgTbt;
  AiUInt32 ul_TrgDiscrete;
}
TY_API_SCOPE_TRG_EX;
```

AiUInt32 px_ScopeTrg->ul_TrgMode

Trigger mode on which condition the trigger is activated

Value	Constant	Description
0	API_SCOPE_MODE_GREATER_THAN	Higher than the value given in parameter **ul_TrgValue** (rising edge)
1	API_SCOPE_MODE_LESS_THAN	Lower than the value given in parameter **ul_TrgValue** (falling edge)

2	API_SCOPE_MODE_GREATER_OR_LESS_THAN	Higher than the positive value given in parameter **ul_TrgValue** or lower than the negative value given in parameter **ul_TrgValue**. On APE cards, ul_TrgValue2 must be used to specify lower boundary value.
3	API_SCOPE_MODE_COMMAND_SYNC	Reserved for future use only.
4	API_SCOPE_MODE_GREATER_THAN_SAMPLES	Higher than the value given in parameter **ul_TrgValue** for a given number of samples (parameter **ul_TrgNbSamples**) (high pulse)
5	API_SCOPE_MODE_LESS_THAN_SAMPLES	Lower than the value given in parameter **ul_TrgValue** for a given number of samples (parameter **ul_TrgNbSamples**) (low pulse)
7	API_SCOPE_MODE_BIU_BM	Reserved for future use only.
8	API_SCOPE_MODE_BIU_BC	Reserved for future use only.
9	API_SCOPE_MODE_BIU2_BM	Reserved for future use only.
10	API_SCOPE_MODE_BIU2_BC	Reserved for future use only.
11	API_SCOPE_MODE_DISCRETES	Reserved for future use only.

> **Note:**
> The modes 3, 7, 8, 9, 10 and 11 are only defined for the APE MIL-Scope module.

AiUInt32 px_ScopeTrg->ul_TrgSrc

Trigger source
Primary or secondary bus on which the trigger is activated

Value	Constant	Description
0	API_SCOPE_SRC_PRIMARY	Primary Bus
1	API_SCOPE_SRC_SECONDARY	Secondary Bus

AiUInt32 px_ScopeTrg->ul_TrgValue

Trigger value that is used to compare against. The voltage range that this value is representing is dependent on the coupling that was set with the function **ApiCmdScopeSetup()**.

Coupling	Value	Description
Extern 30V	**0..1023**	Representing the voltage range of -15V..+15V (steps of 29.3mV)
Extern 3V	**0..1023**	Representing the voltage range of -1.5V..+1.5V (steps of 2.9mV)
Stub 30V	**0..1023**	Representing the voltage range of -15V..+15V (steps of 29.3mV)
Stub 3V	**0..1023**	Representing the voltage range of -1.5V..+1.5V (steps of 2.9mV)

AiUInt32 px_ScopeTrg->ul_TrgValue2

Boundary for 'Less than" trigger in Trigger mode API_SCOPE_MODE_GREATER_OR_LESS_THAN on APE cards. The voltage range that this value is representing is dependent on the coupling that was set with the function **ApiCmdScopeSetup()**.

Coupling	Value	Description
Extern 30V	**0..1023**	Representing the voltage range of -15V..+15V (steps of 29.3mV)
Extern 3V	**0..1023**	Representing the voltage range of -1.5V..+1.5V (steps of 2.9mV)
Stub 30V	**0..1023**	Representing the voltage range of -15V..+15V (steps of 29.3mV)
Stub 3V	**0..1023**	Representing the voltage range of -1.5V..+1.5V (steps of 2.9mV)

`AiUInt32 px_ScopeTrg->ul_TrgNbSamples`

Number of samples

'ul_TrgMode'	Value	Description
0, 1	**0**	Reserved
3,4, 5	**1..255**	Number of samples

`AiUInt32 px_ScopeTrg->ul_TrgNbSamples2`

Reserved for future use only.

`AiUInt32 px_ScopeTrg->ul_TrgFlags`

Trigger flags

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
reserved (0)							

Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
reserved (0)							

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
reserved (0)							

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
reserved (0)							DIS_FDBT

DIS_FDBT

0: The Trigger is enabled immediately

1: The Trigger is enabled after the first data buffer is completely filled once. This mode will always produce valid trace before trigger data. In this mode trigger conditions might be lost.

`AiUInt32 px_ScopeTrg->ul_TrgDelay`

Trigger Delay Time

This value defines the delay time after a trigger has occurred until the capturing is started. The time resolution is calculated from the sampling rate, which is setup with parameter **px_ScopeSetup->ul_SamplingRate** in the function **ApiCmdScopeSetup()**.

Value	Description
0..262143	Delay Time in steps of 1μs

AiUInt32 px_ScopeTrg->ul_TrgTbt

Obsolete parameter must be set to 0.

AiUInt32 px_ScopeTrg->ul_TrgDiscrete

Reserved for future use only.

Output:

None

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.40 ApiCmdScopeCalibrate (obsolete)

Prototype:

```
AiReturn ApiCmdScopeCalibrate(AiUInt32 ulModuleHandle,
                              AiUInt8 ucMode,
                              TY_API_SCOPE_CALIBRATION_INFO * pxInfo);
```

Purpose:

This function could be used to calibrate a MIL-Scope. It was designed for the legacy AYX board design. This function is obsolete. Use ApiCmdScopeOffsetCompensation instead.

Note:
This function is only usable with AIM's MIL-Scope module.

Note:
The hardware needs to be set into a defined state before calibrating the MIL-Scope. The coupling of channel A and B must be set to internal. It is not allowed to have any BC or RT simulation running during the calibration process. The AIM board must be physically disconnected from the external bus.

Input:

ucMode

Calibration mode

Value	Constant	Description
0		Reset calibration offset
1		Calibrate MIL-Scope

Output:

TY_API_SCOPE_CALIBRATION_INFO * pxInfo

Pointer to the list of TY_API_CALIBRATION_INFO structure elements

Note:
These values are only valid on APX/ACX cards. Don't use them with APE cards at this time.

```
Typedef struct ty_api_calibration_info {
  AiUInt32 lOffsetA;
  AiUInt32 lOffsetB;
  AiUInt32 lOffsetA100;
  AiUInt32 lOffsetB100;
  AiUInt32 lOffsetA100Sec;
  AiUInt32 lOffsetB100Sec;
}
```

`TY_API_SCOPE_CALIBRATION_INFO;`

`AiUInt32 pxInfo->ulOffsetA`

The offset which will be added to all channel A values.

`AiUInt32 pxInfo->ulOffsetB`

The offset which will be added to all channel B values.

`AiUInt32 pxInfo->ulOffsetA100`

obsolete

`AiUInt32 pxInfo->ulOffsetB100`

obolsete

`AiUInt32 pxInfo->ulOffsetA100Sec`

obsolete

`AiUInt32 pxInfo->ulOffsetB100Sec`

obsolete

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.41 ApiCmdScopeOffsetCompensation

Prototype:

```
AiReturn ApiCmdScopeOffsetCompensation(AiUInt32 ulModuleHandle,
                                       AiUInt8 ucMode,
                                       TY_API_SCOPE_OFFSETS* pxOffsets);
```

Purpose:

Like on a normal Osziloscope, where the probes are connected to GND to find the zero-line, this function is used to find and set the zero-line for the MIL-Scope. Use this function each time before a precise measurement should be started, for compensation of el. Components tolerances and temperature drifts. It will perform some measurement probes to determine the pitch of the scope data from the zero line of an idle bus. An offset from the sampled values will be determined by calculating the mean value of the noise and DC drift on the idle bus in each of the available scope modes. For all further scope recordings, the ascertained offsets will be used to do an automated correction of the scope sample values.

Note:
This function is only usable with AIM's APE MIL-Scope module.

Note:
The hardware needs to be set into a defined state before this function to determine the offset compensation is run.
The coupling of channel A and B must NOT be set to internal. If you're using API_CAL_CPL_ISOLATED as coupling mode, please call function ApiCmdCalCplCon in order to set any of the other possible coupling modes.
It is also not allowed to have any BC or RT simulation running during the measurement process and the bus must be idle.
The zero line pitch is temperature dependent. Deviations of about 500mV can be observed in a temperature range from -40 to 80℃. So it is recommended to re-adjust the zero line from time to time, or if it is necessary to make sure to have the max. accuracy.
The offset values for the “API_SCOPE_COUPLING_EXTERN” modes are adopted from the measured values of the “API_SCOPE_COUPLING_STUB” modes instead of being calculated separately.

Input:

ucMode

Offset compensation mode

Value	Description
API_SCOPE_OFFSET_COMP_RESET	Reset compensation offsets to a value of 0
API_SCOPE_OFFSET_COMP_MEASURE	Measure zero line offsets, set the values for further scope recordings and return the ascertained offset values
API_SCOPE_OFFSET_COMP_GET	Just returns the active offsets without performing any measurements

Output:

`TY_API_SCOPE_OFFSETS* pxOffsets`

Optional pointer to TY_API_SCOPE_OFFSETS structure where the active zero line offset values will be stored to. Set this pointer to NULL, if you don't need the values.

```
Typedef struct api_offsets {
  AiInt32 lOffsetA_3V_stub;
  AiInt32 lOffsetB_3V_stub;
  AiInt32 lOffsetA_30V_stub;
  AiInt32 lOffsetB_30V_stub;
  AiInt32 lOffsetA_3V_ext;
  AiInt32 lOffsetB_3V_ext;
  AiInt32 lOffsetA_30V_ext;
  AiInt32 lOffsetB_30V_ext;
}
TY_API_SCOPE_OFFSETS;
```

`AiInt32 pxOffsets-> lOffsetA_3V_stub`

The offset which will be added to all channel A values when scope is used in mode API_SCOPE_COUPLING_STUB and 3V measurement range.

`AiInt32 pxOffsets-> lOffsetB_3V_stub`

The offset which will be added to all channel B values when scope is used in mode API_SCOPE_COUPLING_STUB and 3V measurement range.

`AiInt32 pxOffsets-> lOffsetA_30V_stub`

The offset which will be added to all channel A values when scope is used in mode API_SCOPE_COUPLING_STUB and 30V measurement range.

`AiInt32 pxOffsets-> lOffsetB_30V_stub`

The offset which will be added to all channel B values when scope is used in mode API_SCOPE_COUPLING_STUB and 30V measurement range.

`AiInt32 pxOffsets-> lOffsetA_3V_ext`

The offset which will be added to all channel A values when scope is used in mode API_SCOPE_COUPLING_EXTERN and 3V measurement range.

`AiInt32 pxOffsets->lOffsetB_3V_ext`

The offset which will be added to all channel B values when scope is used in mode API_SCOPE_COUPLING_EXTERN and 3V measurement range.

`AiInt32 pxOffsets->lOffsetA_30V_ext`

The offset which will be added to all channel A values when scope is used in mode API_SCOPE_COUPLING_EXTERN and 30V measurement range.

`AiInt32 pxOffsets->lOffsetB_30V_ext`

The offset which will be added to all channel B values when scope is used in mode API_SCOPE_COUPLING_EXTERN and 30V measurement range.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.42 ApiCreateScopeBuffer

Prototype:

```
TY_API_SCOPE_BUFFER* ApiCreateScopeBuffer
                     (AiUInt32 ulModuleHandle,
                      AiUInt32 ulID,
                      TY_API_SCOPE_BUFFER_TYPE eBufferType,
                      TY_API_SCOPE_BUFFER_HANDLER pBufferHandler,
                      void* pvUserData);
```

Purpose:

This function allocates a scope buffer which can be used as input to the function ApiProvideScopeBuffers. The MILScope data will be transferred directly into this buffer with a DMA transfer. The buffer will be allocated with the required size according to the MILScope setup. Before this function is called the function ApiCmdScopeSetup must be called.

Note:
This function is only usable with AIM's APE MIL-Scope module.

Input:

`ulId`

User-definable ID of the buffer.

`TY_API_SCOPE_BUFFER_TYPE eBufferType`

The MIL channel the buffer is determined for.

Enum Value	Description
SCOPE_BUFFER_PRIMARY	Buffer is for data on primary channel
SCOPE_BUFFER_SECONDARY	Buffer is for data on secondary channel

`TY_API_SCOPE_BUFFER_HANDLER pBufferHandler`

Handler function, which is called when the buffer is full. Parameters are the module handle of the board and a pointer to the buffer that is ready. See description of ApiProvideScopeBuffer.

`Void* pvUserData`

Pointer to user definable data which is attached to the scope buffer. This data can be accessed in the scope buffer handler when buffer is completed or canceled.

Output:

None

Return Value:

TY_API_SCOPE_BUFFER*

Pointer to the created scope buffer structure. On failure, NULL is returned. See description of ApiProvideScopeBuffer.

9.1.43 ApiCreateScopeBufferList

Prototype:

```
AiReturn ApiCreateScopeBufferList(
        AiUInt32 ulModuleHandle,
        AiUInt32 ulNumBuffers,
        TY_API_SCOPE_BUFFER* paxBufferList[],
        TY_API_SCOPE_BUFFER_HANDLER pBufferHandler,
        void* pvUserData);
```

Purpose:

This function allocates a list of scope buffers which can be used as input to the function ApiProvideScopeBuffers. The MILScope data will be transferred directly into this buffer with a DMA transfer. The buffer will be allocated with the required size, channel flag and order according to the MILScope setup. Before this function is called the function ApiCmdScopeSetup must be called.

Note:
This function is only usable with AIM's APE MIL-Scope module.

Input:

`AiUInt32 ulNumBuffers`

Number of scope buffers to create for the list. In single channel mode, this has to be a multiple of 2, while in dual channel mode this has to be a multiple of 4.

`TY_API_SCOPE_BUFFER_HANDLER pBufferHandler`

Function pointer of type TY_API_SCOPE_BUFFER_HANDLER. This handler function is set for all of the buffers in the list.

See description of ApiProvideScopeBuffer.

`Void* pvUserData`

Pointer to user definable data which is attached to the scope buffers. This data can be accessed in the scope buffer handler when buffer is completed or canceled.

Output:

`TY_API_SCOPE_BUFFER * paxBufferList[]`

Array of pointers to TY_API_SCOPE_BUFFER structures which forms the actual scope buffer list. You have to ensure this array has at least ulNumBuffers entries. See description of ApiProvideScopeBuffer.

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.44 ApiFreeScopeBuffer

Prototype:

```
AiReturn ApiFreeScopeBuffer(AiUInt32 ulModuleHandle,
                            TY_API_SCOPE_BUFFER* px_Buffer);
```

Purpose:

This function frees a scope buffer which was allocated with ApiCreateScopeBuffer or ApiCreateScopeBufferList. The corresponding buffer must not be accessed after calling this function. In particular, it must not be provided for scope data reception via ApiProvideScopeBuffers. Also, if the buffer has been provided before, it has to be completed or canceled by calling ApiCmdScopeStop.

Note:
This function is only usable with AIM's APE MIL-Scope module.

Input:

`TY_API_SCOPE_BUFFER* pxBuffer`

A pointer to the scope buffer to free.

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.45 ApiProvideScopeBuffers

Prototype:

```
AiReturn ApiProvideScopeBuffers(AiUInt32 ulModuleHandle,
                                AiUInt32 ulNumBuffers,
                                TY_API_SCOPE_BUFFER* paxBuffers);
```

Purpose:

This function provides one or more buffers created with ApiCreateScopeBuffer or ApiCreateScopeBufferList to the system driver. The buffers will be directly filled with data through a DMA transfer. Each buffer can have his own handler function installed which is called as soon as the buffer is full. The handler calls are automatically searialized on board level so that only one handler is called at a time.

Note:
This function is only usable with AIM's APE MIL-Scope module.

Input:

`AiUInt32 ulNumBuffers`

Number of buffers in the array paxBuffers.

`TY_API_SCOPE_BUFFER * paxBuffers[]`

Array of pointers to TY_API_SCOPE_BUFFER structure elements, each of them containing information about one provided buffer.

```
Typedef enum API_SCOPE_BUFFER_TYPE {
  SCOPE_BUFFER_PRIMARY = 0;
  SCOPE_BUFFER_SECONDARY;
}
TY_API_SCOPE_BUFFER_TYPE;

typedef struct ty_api_scope_buffer {
  AiUInt32 ulID;
  TY_API_SCOPE_BUFFER_TYPE eBufferType;
  void * pvBuffer;
  AiUInt32 ulBufferSize;
  TY_API_SCOPE_BUFFER_FLAGS ulFlags;
  AiUInt32 ulDataSize;
  AiUInt32 ulTriggerId;
  TY_API_SCOPE_BUFFER_HANLDER pBufferHandler,
  void * pvUserData);
}
TY_API_SCOPE_BUFFER;
```

AiUInt32 ulID

ID of the buffer that can be defined by user for his own use.

TY_API_SCOPE_BUFFER_TYPE eBufferType

The MIL channel the buffer is determined for.

Enum Value	Description
SCOPE_BUFFER_PRIMARY	Buffer is for data on primary channel
SCOPE_BUFFER_SECONDARY	Buffer is for data on secondary channel

void* pvBuffer

Pointer to the actual buffer. This buffer is allocated with ApiCreateScopeBuffer and freed with ApiFreeScopeBuffer.

AiUInt32 ulBufferSize

Size of the buffer in bytes. On APE cards, this value has to be 64K in single-channel mode and 32K in dual-channel mode.

TY_API_SCOPE_BUFFER_FLAGS ulFlags

Specific flags for the buffer.

Flag	Description
SCOPE_BUFFER_FLAG_FILLED	Buffer is filled with data
SCOPE_BUFFER_FLAG_CANCELED	Receiving of Scope Data for this buffer was canceled.
SCOPE_BUFFER_FLAG_TRIGGER	The buffer contains the trigger sample

AiUInt32 ulDataSize

Size of the available MIL-Scope data in bytes. Only valid if SCOPE_BUFFER_FLAG_FILLED flag is set for the buffer.

AiUInt32 ulTriggerId

ID of the trigger sample in the buffer starting from 0. This value is only valid if SCOPE_BUFFER_FLAG_TRIGGER flag is set for the buffer.

TY_API_SCOPE_BUFFER_HANDLER pBufferHandler

Handler function, which is called when the buffer is full. Parameters are the module handle of the board and a pointer to the buffer that is ready.

```
Typedef void(_cdecl * TY_API_SCOPE_BUFFER_HANDLER)(
  AiUInt32 ulModuleHandle,
  TY_API_SCOPE_BUFFER * pBuffer);
```

void* pvUserData

Pointer to user definable data which is attached to the buffer. This data can be accessed in the scope buffer handler when buffer is completed or canceled.

Output:

None

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

9.1.46 ApiWaitForScopeFinished

Prototype:

```
AiReturn ApiWaitForScopeFinished(AiUInt32 ulModuleHandle,
                                 AiInt32 lTimeOut,
                                 TY_API_SCOPE_WAIT_STATE* pWaitResultFlags);
```

Purpose:

In single shot mode this function blocks the current execution thread until all the buffers are filled or the specified timeout occurred.

Note:
This function is only usable with AIM's APE MIL-Scope module.

Input:

lTimeOut

Time-out in milliseconds.

Value	Description
-1	Infinite blocking
0	Function returns at once.
>0	Time-out in milliseconds

Output:

TY_API_SCOPE_WAIT_STATE * pWaitResultFlags

Pointer to value where flags according to wait result are set

```
typedef AiUInt32 TY_API_SCOPE_WAIT_STATE
```

Flag	Description
SCOPE_WAIT_FINISHED	Scope has finished, all scope buffers have been processed
SCOPE_WAIT_TIMEOUT	Scope is still running. The specified time out elapsed.
SCOPE_WAIT_OVERFLOW	Scope is idle but Overflow in scope data capturing occurred.

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

10 REPLAY FUNCTIONS

Chapter 10 defines the Replay function calls of the API S/W Library. The Replay functions provide configuration of the Replay process to replay pre-recorded Bus Monitor data entries in entirety or filtered by specified RTs. The function calls in this table are listed in a functional order, however, the detailed descriptions of the BM function calls in the following sections are in alphabetical order

Table 10.1: Replay Function Descriptions

Function	Description
ApiCmdReplayIni	Initializes Replay interrupts, Time tag replay and defines the size of the data to be replayed
ApiCmdReplayStart	Starts the Replay of data in the Replay buffer
ApiCmdReplayStop	Stops the Replay of data in the Replay buffer
ApiCmdReplayStatus	Reads the status of the Replay activity
ApiCmdReplayRT	Disables replay of one or more specific RT(s)

10.1 Low Speed Functions

10.1.1 ApiCmdReplayIni

Prototype:

```
AiReturn ApiCmdReplayIni(AiUInt32 handle,
                         AiUInt8 biu,
                         AiUInt8 cet,
                         AiUInt8 nct,
                         AiUInt8 cyc,
                         AiUInt8 nimg,
                         AiUInt8 alt,
                         AiUInt8 rlt,
                         AiUInt8 rint,
                         AiUInt32 min,
                         AiUInt32 msec,
                         AiUInt8 sign,
                         AiUInt32 fsize);
```

Purpose:

This function is used to initialize replay interrupts and replay time tag and also defines the size of the data to be replayed.

Note:
This function is not supported on embedded devices! (see also chapter B.1.7 "Limitations for embedded board variants")

Input:

`AiUInt8 cet`

1 (Reserved)

`AiUInt8 nct`

0 (Reserved)

`AiUInt8 cyc`

0 (Reserved)

`AiUInt8 nmig`

0 (Reserved)

`AiUInt8 alt`

Value	Constant	Description
0	**API_REP_NO_ABS_TIME**	No absolute time replay (used for LS replay only)
1	**API_REP_ABS_TIME**	Replay based on absolute Long Timetag (used for synchronous LS and HS Replay)

`AiUInt8 rlt`

Relative Long Tag

Value	Constant	Description
0	**API_REP_RLT_LOW_TT**	Relative Replay mode uses Low Time Tags only
1	**API_REP_RLT_ALL**	Relative Replay mode uses High and Low Time Tags entries

Note:
This flag has no function if the parameter “alt” is set to 1, which implicitly sets the “rlt” to 1. If set to 1, the replay file must contain the full IRIG time tag. These entries are performed from the monitor in tag mode 0 only.
If this flag is not set, the replay module uses only the “Low Time Tag” entries. As the “Low Time Tag” covers 60 seconds, any gap greater than half this range (30 seconds) will not be properly reproduced. Usage of this flag extends this period to almost one year.
No special handling is provided for the “End of Year” overrun!

`AiUInt8 rint`

Replay Interrupt Control

Value	Constant	Description
0	**API_REP_NO_INT**	No Interrupt
1	**API_REP_HFI_INT**	Half Buffer Transmitted Interrupt
2		Reserved

`AiUInt32 min`

0 (Reserved)

`AiUInt32 msec`

0 (Reserved)

`AiUInt8 sign`

1 (Reserved)

`AiUInt32 fsize`

Replay file size in Bytes

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

10.1.2 ApiCmdReplayRT

Prototype:

```
AiReturn ApiCmdReplayRT(AiUInt32 ul_ModuleHandle,
                        AiUInt8 biu, AiUInt8 con,
                        AiUInt8 mode, AiUInt8 rt);
```

Purpose:

This function is used to enable or disable the Replay for the specified RT- Address. If the selected RT is disabled for Replay an external RT shall be connected to the bus. After initialization of the Replay mode (**ApiCmdReplayIni**) all RT's are enabled.

Note:
This function is not supported on embedded devices! (see also chapter B.1.7 "Limitations for embedded board variants")

Input:

AiUInt8 con

Value	Constant	Description
0	**API_MODE_ALL_RT**	All RTs
1	**API_MODE_SINGLE_RT**	Single RT

AiUInt8 mode

Value	Constant	Description
0	**API_DIS**	Disable RT for Replay (external RT)
1	**API_ENA**	Enable RT for Replay

AiUInt8 rt

'con'	Value	Description
0	**0**	-
1	**0..31**	RT

Output:

none

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

10.1.3 ApiCmdReplayStart

Prototype:

```
AiReturn ApiCmdReplayStart(AiUInt32 ul_ModuleHandle,
                           AiUInt8 biu);
```

Purpose:

This command is used to start the AIM board Replay operation.

Note:
When using an AIM 3910 board, the HS replay mechanism will also be started, if it was initialized with the corresponding HS replay initializing command (only available for AIM 3910 boards). This is to avoid synchronization problems between LS and HS replay!

Note:
This function is not supported on embedded devices! (see also chapter B.1.7 “Limitations for embedded board variants”)

Input:

none

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

10.1.4 ApiCmdReplayStatus

Prototype:

```
AiReturn ApiCmdReplayStatus(AiUInt32 ul_ModuleHandle,
                            AiUInt8 biu,
                            TY_API_REP_STATUS *prep);
```

Purpose:

This function is used to read the Replay Status Information. The initial value of parameter 'rpi_cnt' indicates the amount of Half Buffer Transmitted Interrupts required for writing the requested Replay file size (refer to parameter 'fsize' of library function **ApiCmdReplayIni**).

Note:
This function is not supported on embedded devices! (see also chapter B.1.7 "Limitations for embedded board variants")

Input:

none

Output:

TY_API_REP_STATUS *prep

Replay Status parameter

```
typedef struct ty_api_rep_status {
  AiUInt8 status;
  AiUInt8 padding1;
  AiUInt16 padding2;
  AiUInt32 rpi_cnt;
  AiUInt32 saddr;
  AiUInt32 size;
  AiUInt32 entry_cnt;
}
TY_API_REP_STATUS;
```

AiUInt8 status

Replay Status

Value	Constant	Description
0	**API_REP_HALTED**	Replay halted
1	**API_REP_BUSY**	Replay busy

AiUInt8 padding1

0 (Reserved)

AiUInt16 padding2

0 (Reserved)

AiUInt32 rpi_cnt

Actual value of the Half Buffer Transmitted Interrupt counter (incremented each time a Half Buffer Transmitted Interrupt occurs)

When the **rpi_cnt** value is incremented new Replay datas should be reloaded to the Shared RAM and should be copied from the Shared RAM to the Global RAM of the AIM board area using the library function ApiWriteRepData'.

AiUInt32 saddr

Start Address of the AIM board Replay buffer in the Global RAM area to copy the replay buffer entries from the Shared RAM area to the Global RAM area of the AIM board.

AiUInt32 size

- (not used)

AiUInt32 entry_cnt

Initial value calculated from the file size (refer to ApiCmdReplayIni, parameter 'fsize') and decremented when Replay is started.

Return Value:

AiReturn

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

10.1.5 ApiCmdReplayStop

Prototype:

```
AiReturn ApiCmdReplayStop(AiUInt32 ul_ModuleHandle,
                          AiUInt8 biu);
```

Purpose:

This command is used to stop the AIM board Replay operation.

Note:
When using an AIM 3910 board, the HS replay mechanism will also be stopped, if it was started with the ApiReplayStart function. This is to avoid synchronization problems between LS and HS replay!

Note:
This function is not supported on embedded devices! (see also chapter B.1.7 “Limitations for embedded board variants”)

Input:

none

Output:

none

Return Value:

`AiReturn`

All API functions return API_OK if no error occurred. If the return value is not equal to API_OK the function **ApiGetErrorMessage** can be used to obtain an error description.

11 GENERAL I/O (GENIO) FUNCTIONS

Chapter 11 defines the GenIo function calls of the API S/W Library. The GenIo functions are used to handle acquisition functions. Data acquisition is performed direct access from the ASP processor on the legacy APX board. Since this functions are only available on no longer supported legacy boards this functions are marked as obsolete.

Table 11.1: GenIo Function Descriptions

Function	Description
ApiCmdGenIoAddrInit	Initialise IO module access address for ASP
ApiCmdGenIoOutChnWrite	Write data to output channel of IO module
ApiCmdGenIoOutChnRead	Read data from output channel of IO module
ApiCmdGenIoInChnRead	Read data from input channel of IO module
ApiCmdGenIoSysSTaskCtrl	Enable /Disable Sampling Task
ApiCmdGenIoTestSeq	Handle sequences of discrete signal changes

11.1 Low Speed Functions

11.1.1 ApiCmdGenIoAddrInit (obsolete)

Prototype:

```
AiReturn ApiCmdGenIoAddrInit(AiUInt32 ul_Module,
                             AiUInt8 genio_type,
                             AiUInt8 io_type,
                             AiUInt8 chn,
                             AiUInt8 res,
                             AiUInt32 addr);
```

Purpose:

This function is used to initialise ASP access to the specified module.

Input:

AiUInt8 genio_type

7........4	3.......0
SlotNumber	BoardType

Slotnumber: 0

BoardType:

Value	Define	Description
0	**API_GEN_IO_INIT_TYPE**	Init ASP Driver structures and IO access
1	**API_GEN_IO_APX_TYPE**	APX Discrete I/O type

AiUInt8 io_type

I/O channel type

Value	Description
1	Setup HW access address for Input channel type in 'addr'
2	Setup HW access address for Output channel type in 'addr'
3	Init HW with value defined in parameter 'addr'

AiUInt8 chn

Channel / IO Model selection

For 'BoardType'=	Value	Description
API_GEN_IO_INIT_TYPE	**-**	(n/a)
API_GEN_IO_APX_TYPE	**1**	APX Onboard Discrete model

AiUInt8 res

Reserved

AiUInt32 addr

For 'io_type' = 1, 2:

N/A

For 'io_type' = 3:

Init value to be used for IO initialization

For 'BoardType'=	Value	Description
API_GEN_IO_APX_TYPE	**0x0F**	Program APX Discretes to Input or Output Bits 7..4 = Inputs, Bits 3..0 = Output
API_GEN_IO_DIG_TYPE	**0x0000**	Output Init value (set all Output Discretes)

Output:

none

Return Value:

AiReturn

API_OK in case of success.

11.1.2 ApiCmdGenIoOutChnWrite (obsolete)

Prototype:

```
AiReturn ApiCmdGenIoOutChnWrite(AiUInt32 ul_Module,
                                AiUInt8 genio_type,
                                AiUInt8 chn,
                                AiUInt32 res,
                                AiUInt32 val);
```

Purpose:

This function is used to write data to the specified output channel.

Input:

AiUInt8 genio_type

7........4	3.......0
SlotNumber	BoardType

Slotnumber: 0

BoardType:

Value	Define	Description
0	**API_GEN_IO_INIT_TYPE**	Init ASP Driver structures and IO access
1	**API_GEN_IO_APX_TYPE**	APX Discrete I/O type

AiUInt8 chn

Channel selection

	For 'BoardType' = 1
Value	Description
0	write all discrete output channels at once
1..32	write to single discrete output channel

AiUInt32 res

Spare Parameter

		For 'BoardType' = 1
'chn'	Value/ Range	Description
0	**0**	write all discrete output channels at once
	1	write inverted 'val' to all output channels
1..32	**0**	write 'val' to the selected output channel (set/reset bit)
	1..1000	toggle the selected output channel for 'res' msec

`AiUInt32 val`

Output data value Init value to be used for IO initialization

For 'genio_type' = 1		
'chn'	Value/ Range	Description
0	**0x00000000** **0xFFFFFFFF**	32-bit value (set all discrete channels)
1..32	**0** / **1**	reset bit / set bit

Output:

none

Return Value:

`AiReturn`

API_OK in case of success.

11.1.3 ApiCmdGenIoOutChnRead (obsolete)

Prototype:

```
AiReturn ApiCmdGenIoOutChnRead(AiUInt32 ul_Module,
                               AiUInt8 genio_type,
                               AiUInt8 chn,
                               AiUInt32 ctl,
                               AiUInt32 * val);
```

Purpose:

This function is used to read data from the specified IO output channel.

Input:

`AiUInt8 genio_type`

7........4	3.......0
SlotNumber	BoardType

Slotnumber: 0

BoardType:

Value	Define	Description
0	**API_GEN_IO_INIT_TYPE**	Init ASP Driver structures and IO access
1	**API_GEN_IO_APX_TYPE**	APX Discrete I/O type

`AiUInt8 chn`

Channel selection

	For 'BoardType' = 1
Value	Description
0	Read all discrete output channels at once
1..32	read from single discrete output channel

`AiUInt32 ctl`

Reserved

Output:

`AiUInt32 val`

Read Input data value

For 'genio_type' = 1		
'chn'	Value/ Range	Description
0	**0x00000000.. 0xFFFFFFFF**	32-bit register value
1..32	**0 / 1**	Discrete input / bit value

Return Value:

`AiReturn`

API_OK in case of success.

11.1.4 ApiCmdGenIoInChnRead (obsolete)

Prototype:

```
AiReturn ApiCmdGenIoInChnRead(AiUInt32 ul_Module,
                              AiUInt8 genio_type,
                              iUInt8 chn,
                              AiUInt32 ctl,
                              AiUInt32 * val);
```

Purpose:

This function is used to read data from the specified IO input channel.

Input:

AiUInt8 genio_type

7........4	3.......0
SlotNumber	BoardType

Slotnumber: 0

BoardType:

Value	Define	Description
0	**API_GEN_IO_INIT_TYPE**	Init ASP Driver structures and IO access
1	**API_GEN_IO_APX_TYPE**	APX Discrete I/O type

AiUInt8 chn

Channel selection

For 'BoardType' = 1	
Value	Description
0	Read all discrete input channels at once
1..32	read from single discrete input channel

AiUInt32 ctl

Reserved

Output:

`AiUInt32 * val`

Read Input data value

For 'genio_type' = 1		
'chn'	Value/ Range	Description
0	**0x00000000.. 0xFFFFFFFF**	32-bit register value
1..32	**0 / 1**	Discrete input / bit value

Return Value:

`AiReturn`

API_OK in case of success.

11.1.5 ApiCmdGenIoSysSTaskCtrl (obsolete)

Prototype:

```
AiReturn ApiCmdGenIoSysSTaskCtrl (AiUInt32 ul_Module,
                                   AiUInt8 genio_type,
                                   AiUInt8 con,
                                   AiUInt32 rate,
                                   AiUInt32 rsync,
                                   AiUInt32 ustep);
```

Purpose:

This function is used to control the ASP sampling task.

Input:

AiUInt8 genio_type

7.......4	3.......0
SlotNumber	BoardType

Slotnumber: 0

BoardType:

Value	Define	Description
0	**API_GEN_IO_INIT_TYPE**	Init ASP Driver structures and IO access
1	**API_GEN_IO_APX_TYPE**	APX Discrete I/O type

AiUInt8 con

Sampling Task control

Value	Define	Description
0	API_OFF	disable
1	API_ON	enable, record each sample
2	-	(reserved)
3	(API_ON+2)	enable/disable function defined in 'rsync' (can only be applied when Sampling Task has been activated before 'con'=API_ON)

For 'BoardType' = 1	
Value	Description
0	Read all discrete input channels at once
1..32	read from single discrete input channel

`AiUInt32 rate`

ASP Sampling rate in 0.5 msec steps (5 .. 2000), Example:

Value	Time	Sampling Frequency
1	0,5 ms	2000 Hz *
		(* for internal test only)
2	1 ms	1000 Hz *
3	1.5 ms	666 Hz *
4	2 ms	500 Hz *
5	2.5 ms	400 Hz
10	5 ms	200 Hz
20	10 ms	100 Hz
100	50 ms	20 Hz
200	100 ms	10 Hz
1000	500 ms	2 Hz
2000	1000 ms	1 Hz

`AiUInt32 rsync`

Control of test functions:

Only for **BoardType = 1** and **con = (API_ON+2)**

Value	Description
1..16	Enable Test signal (with pulse frequency = 'rate' / 2)
0	Disable Test signal

`AiUInt32 ustep`

Reserved.

Output:

none

Return Value:

`AiReturn`

API_OK in case of success.

11.1.6 ApiCmdGenIoTestSeq (obsolete)

Prototype:

```
AiReturn ApiCmdGenIoTestSeq (AiUInt32 ul_Module,
                             AiUInt32 ul_Con,
                             TY_API_GENIO_TEST_SEQ_TBL *px_SeqTbl,
                             AiUInt8 auc_SeqStatus[4]);
```

Purpose:

This function is used to initialize and control Sequences of signal state changes or pulses, acyclic messages can be included into the sequences.

Input:

AiUInt32 ul_Con

ul_Con : Test Sequence control

Value	Description
0	Setup all values as specified in structure px_SeqTbl (TSW clears internally entire structure before Setup)
1	Start operation
2	Stop operation
3	Return Sequence status

TY_API_GENIO_TEST_SEQ_TBL px_SeqTbl

Test sequence table

```
#define MAX_GENIO_TEST_SEQ_ARRAY_SIZE 32

typedef struct {
  /* Start mode */
  AiUInt32 mode;

  /* Start condition */
  AiUInt32 cond;

  /* Reserved */
  AiUInt32 res1;
  AiUInt32 res2;

  /* Timeline 0..10.000 in steps of ms */
  AiUInt32 tm_ms[MAX_GENIO_TEST_SEQ_ARRAY_SIZE];

  /* Identify IP slot number and Board Type, define required operation */
  AiUInt32 genio_type_op[MAX_GENIO_TEST_SEQ_ARRAY_SIZE];
```

```
  /* Define the toggle Bits (set to 1) */
  AiUInt32 chn_val[MAX_GENIO_TEST_SEQ_ARRAY_SIZE];

}
TY_API_GENIO_TEST_SEQ_ARRAY;

typedef struct {
  TY_API_GENIO_TEST_SEQ_ARRAY ax_Seq[4];
}
TY_API_GENIO_TEST_SEQ_TBL;
```

AiUInt32 mode

Start mode of specified sequence

Value	Description
0	Start immediately
1	Start depending on specified condition (parameter 'cond')

AiUInt32 cond

Start condition

Bits	Value	Description
3..0	**1**	Board Type
7..4	**0..2**	IP slot number
15..8	**0..31**	Discrete signal number (0 = LS bit 31 = MS bit)
23..16	**0**	Signal state changes from 1 →0
	1	Signal state changes from 0 →1
	2	Signal state change
24	**0**	Trigger on input discrete
	1	Trigger on output discrete
31..25	**0** Reserved	

AiUInt32 res1

Reserved (0)

AiUInt32 res2

Reserved (0)

AiUInt32 tm_ms[ ]

Array of Time marks for sequence events

Value	Description
0 .. 10.000	Time marks in steps of ms in ascending order

AiUInt32 genio_type_op[ ]

Signal Source (IP or Message)

Bits	Value	Description
3..0	**1**	Board Type
7..4	**0..2**	IP slot number
11..8	**0**	Toggle Discrete Out as marked in ‘chn_val’
	1	Set 0 Discrete Out as marked in ‘chn_val’
	2	Set 1 Discrete Out as marked in ‘chn_val’
	3..15	Reserved
30..12	**0**	Reserved
31	**0**	Discrete Event, as defined by bits 8..11
	1	Execute Acylcilc Transer as per address in ‘chn_val’

`AiUInt32 chn_val[ ]`

Sequence event If genio_type_op Bit31 = 0:

Bits	Value	Description
31..0	**0/1**	Modify Discrete Out bit(s) marked by ‘1’

If genio_type_op Bit31 = 1: acyclic start address pointer

Output:

`AiUInt8 auc_SeqStatus[]`

Status of the four sequences

Value	Description
0	Not yet started
1	In progress
2	Finished

Return Value:

`AiReturn`

API_OK in case of success.

12 TROUBLESHOOTING

Specific questions regarding troubleshooting the AIM API S/W Library should be directed to our Technical Support form at www.aim-online.com.

12.1 Error Reporting Design

Error messages can be generated by various levels of the API S/W Library interfaces. For most functions it is possible to call ApiGetErrorMessage with the returned value to get a formatted Error message string.

13 NOTES

13.1 Definition of Terms

Application Interface	Refers to software interface between the API S/W Library function calls and the API Target S/W.
Big Endian	a system of memory addressing in which numbers that occupy more than one byte in memory are stored “big end first” with the uppermost 8 bits at the lowest address.
Broadcast	commands sent to multiple RTs at once. The RTs are responsible for distinguishing between broadcast and non-broadcast command messages. An RT address of 11111 (31) indicates a broadcast message.
Buffer Header	information detailing the location of the data buffer(s) used for the LS and HS transfer(s), and the status and event information associated with the transfer. A buffer header is to be associated with the data buffer(s) by the programmer for any transfer to/from the BC or RT
Buffer Header ID	an ID number associated with the Buffer Header structure
Data Buffer	an area of memory on the AIM device (global RAM) assigned by the programmer to accommodate transfer(s) to/from the BC or RT
Driver Command	command used by the AIM Target s/w to control the Target device
Dual Stream	indicates the AIM 1553 board supports two dual redundant MIL-STD-1553 data stream
FLASH	page oriented electrical erasable and programmable memory
function	a self-contained block of code with a specific purpose that returns a single value.
Header File	file containing C++ code consisting of definitions which are used by the executable code
intermessage gap	the time between LS message transmissions with a minimum gap time, as specified in MIL-STD-1553, of 4.0 microseconds
interrupt	a signal from a device attached to a computer or from a program within the computer that causes the main program that operates the computer (the operating system) to stop and figure out what to do next
Little Endian	a system of memory addressing in which numbers that occupy more than one byte in memory are stored “little end first” with the lowest 8 bits at the lowest address.
Major Frame	sequence of minor frames defined for transfer (max 64 minor frames in a major frame)
MIL-STD-1553	military specification defining a digital time division command/response multiplexed databus
MIL-STD-1760	based on MIL-STD-1553B, augmented with requirements to support the aircraft/store electrical interconnection system between aircraft and stores (any external device attached to the aircraft (such as bombs, missiles, etc.)
Minor Frame	sequence of LS transfers (max 128 transfers defined in a minor frame)

Mode code	Unique five bit codes that are sent to specific RTs to check their status, control their operation and manage the bus.
Monitor Status Trigger pattern	8 bits in the Monitor Status Word that reflect the results of the Monitor Trigger Block execution of the BIU Processor.
Monitor Status Word	reflects the current status of Bus Monitor operation
Prototype	a prototype of a function provides the basic information that the compiler need to check that a function is used correctly
Response Time	The time between the BC Command/Data word and the RT Status word
Response Timeout Value	The maximum time the Bus Controller will wait for a Status word response from the RT before indicating a "Response Timeout".
Single Stream	indicates the AIM 1553 board supports one dual redundant MIL-STD-1553 data stream
S-Record	An S-record file consists of a sequence of specially formatted ASCII character strings. An S-record will be less than or equal to 78 bytes in length. The general format of an S-record follows: + - - - - - - - - - -// - - - - - - // - - - - - - - - - + | type | count | address | data | checksum | + - - - - - - - - - -// - - - - - - // - - - - - - - - - +
STANAG3910	based on MIL-STD-1553B, a 1 Mbit/sec dual redundant low speed bus, augmented by a high speed fiber optic dual redundant bus operating at 20 Mbits/sec
spg	sample program
Strobe	a strobe is a signal that is sent that validates data or other signals on adjacent parallel lines
Target	Refers to the software/communication active on the AIM device
Transfer Descriptor (BC)	an area of memory assigned by the Target s/w to store status of the transfer
Transfer Descriptor (RT)	an area of memory assigned by the Target s/w to store status of the transfer
Transfer ID	an ID number associated with the transfer structure that defines the characteristics of the LS or HS transfer
Transfer Type	BC-to-RT, RT-to-BC, RT-to-RT
Vector Word	Transmitted by the RT when requested by the BC with the Mode code command "Transmit Vector Word" which is Mode code 16. The vector word will contain information indicating the next action to be taken by the BC.

LIST OF FIGURES

LIST OF TABLES

APPENDIX A
DOCUMENT/SOFTWARE HISTORY

A APPENDIX A DOCUMENT/SOFTWARE HISTORY

Appendix A provides information regarding the software and documentation changes that have occurred for each version of the API S/W Library. This information is provided in two forms:

1. **Table A.1 Summary of Changes** – This table includes a definition of changes for each s/w library function and a general summary of the documentation changes outside of the function calls.
2. **Table A.2 Summary of Version Changes for each S/W Library Function** – This table includes notations for both s/w and documentation changes.

Table A.1: Summary of Changes

Version	General or S/W Function Change	Description
V24.0.x Rev. A	ApiCmdIni	Incompatible mode parameter
	ApiCmdReset	Removed API_RESET_ONLY_IF_NOT_ALREADY_RESETTED
	ApiInstIntHandler ApiProvideScopeBuffers	Changed callback prototype from _stdcall to _cdecl
	ApiCmdGetIrigStatus ApiCmdGetIrigTime ApiCmdSetIrigStatus ApiCmdSetIrigTime	New functions to set/get the irig status. Modified get/set time to have time only.
	ApiGetTcomStatus ApiSetTgEmul ApiGetTgEmul ApiPrintfOnServer ApiGetOpenErr ApiCmdCalTransCon ApiCmdTimerIntrCheck ApiCmdBiuIntrCheck ApiCmdRamRead ApiCmdRamReadByte ApiCmdRamReadLWord ApiCmdRamReadWord ApiCmdRamWrite ApiCmdRamWriteByte ApiCmdRamWriteLWord ApiCmdRamWriteWord	Removed
	ApiReadAllVersions	New
	ApiOpen ApiReadBSPVersion ApiReadBSPVersionEx ApiReadRecData ApiCmdBMIniMsgFltRec ApiCmdBMReadMsgFltRec ApiCmdBMStackEntryFind ApiCmdBMStackEntryRead ApiCmdBMStackpRead	Declared as obsolete
	All	Return type change to AiReturn Removed obsolete driver call documentation
	ApiCmdRTSAMsgReadEx	Added note to transfer and error count which count on RT level.
	ApiCmdSysPXICon	Clarified description of time tag source and clear.
	ApiInstIntHandler	Interrupts might be subject to increased latency, because of the extended processing from the additional USB-/Ethernet layers.

Version	General or S/W Function Change	Description
	ApiCmdDataQueueClose ApiCmdDataQueueControl ApiCmdDataQueueOpen ApiCmdDataQueueRead ApiCmdQueueFlush	Removed obsolete modes. Simplified input/output parameters.
	ApiInstIntHandler	Changed callback to pass struct as pointer instead of argument.
	ApiCmdSysGetBoardInfo	Additional board information. Return count of valid array entries.
	ApiGetServerInfo	Update struct of server information.
	Appendix B	Update Tables of Board Limitations
	ApiCmdGetIrigTime	Change accuracy from milliseconds to microseconds.
V24.1.x Rev. A		No interface changes.
V24.2.x Rev. A	ApiCmdCalXmtCon	Added node for variable amplitude limitations. Added high resolution mode for PBIs with new HOLD transceiver.
	ApiCmdSysGetBoardInfo	New value TY_BOARD_INFO_CHANGE_AMPL_HIGH_RES
	ApiGetBoardInfo	Added missing values for ul_DeviceType
	ApiCmdRTModeCtrl	Add new mode API_RT_MODE_OPERATION_CTRL
V24.2.x Rev. B	ApiCmdBCAcycSend	API_BC_ACYC_SEND_ON_TIMETAG is only available for APX,AVX and ACX.
	Board functionality overview	Table Functionality Overview for boards with ASP and Table Functionality Overview for boards without ASP New column “Acyclic on Time Tag”
V24.2.x Rev. C	ApiCmdBCModeCtrl	ApiCmdBCModeCtrl/ API_BC_MODE_INSERT_MC17_SYNC_CNT Is not available on boards with multi channel firmware.
V24.3.x Rev A	ApiCmdReadDiscretesConfig	New function to read back discrete configuration set up by ApiCmdInitDiscretes
	ApiReadVersion	New function the version number of a software package component
	ApiGetErrorMessage	New function to describe an error code
	ApiCmdDataQueueRead	Added Note to check Programmers Guide about data format returned by this function.
V24.4.x Rev A	ApiCmdBCAcycSend	Added support for APE boards
	ApiInstIntHandler	Added support for ANET1553-1/2 boards
V24.4.x Rev B	ApiGetDriverInfo, ApiReadBSPVersionEx	Corrected names of some struct members
V24.5.x Rev A	ApiGetBoardInfo	Added APXX3910, APEX3910, AME1553 to ul_DeviceType
	ApiGetDriverInfo	Revised defines for uc_DeviceGroup.

Version	General or S/W Function Change	Description
	ApiCmdIni	Revised description of board_config / Platform parameter.
	Board functionality overview	Added APXE1553, AME1553
V24.5.x Rev B	Board Name	Renamed APXE1553 to APXX1553
V24.5.x Rev C	ApiCmdBMStackpRead	Removed obsolete flag from ApiCmdBMStackpRead.
V24.6.x Rev A		AME1553 support. No interface changes.
V24.7.x Rev A	ApiCmdCalTransCon	Added function ApiCmdCalTransCon
	ApiCmdSysTriggerEdgeInputSet ApiCmdSysTriggerEdgeInputGet	Added new functions.
V24.7.x Rev B	Applicable Documents	Improved Programmers Guide name and description. Added AME1553 Hardware Manual
V24.8.x Rev A	Applicable Documents	Added AME1553-1-AP Hardware Manual
	ApiCmdReadDiscretesInfo	New function to read back discrete capability (Input only, Output only, In /Out
	Limitations for embedded board variants	New chapter about embedded board variants and its limitations.
	ApiCmdBCXferDef ApiCmdBCXferDefErr ApiCmdRTSAConErr ApiCmdReplayStart	Implemented functional limitations for embedded board variants.
	ApiCmdIni ApiCmdSysGetBoardInfo	Return embedded flag in addition to simulator only and single function variants.
V24.9.x Rev A	ApiGetBoardInfo	Added ACE3910 to ul_DeviceType
	ApiCmdIni	Added new platform parameter
V24.10.x Rev A	ApiCmdDefMilbusProtocol ApiCmdSetMemPartition ApiCmdBCAcycSend ApiCmdBCIni ApiCmdBCModeCtrl ApiCmdBCXferDef ApiCmdBCXferDefErr ApiCmdRTDytagDef ApiCmdRTModeCtrl ApiCmdRTSAConErr ApiCmdBMCapMode ApiCmdBMStatusRead ApiCmdBMTCBIni ApiCmdBMTIWIni	Added notes for embedded restriction

Version	General or S/W Function Change	Description
	ApiCmdReplayIni ApiCmdReplayRT ApiCmdReplayStart ApiCmdReplayStatus ApiCmdReplayStop	Added notes for embedded restriction
V24.11.x Rev A	ApiGetBoardInfo	Extended table for parameter “ul_DeviceType” with AXE, AMCE and ASE board variants
	ApiCmdRTIni	Added new mode API_ENABLE_DUAL_REDUNDANT
	ApiCmdRTSADWCGet ApiCmdRTSADWCSet	Added new functions
	Limitations for non embedded board variants	New chapter about non embedded board variants and its limitations.
	Limitations for ASE1553	New chapter for ASE1553 boards.
	Limitations for boards with Multichannel Firmware	Added AMCE and AXE boards
	Table B.2 – Functionality Overview for boards without ASP	Added column for AMCE / AXE boards Added column for ASE boards
	Table B.4 – Function Support By Boards Without ASP	Added column for AMCE / AXE boards Added ASE to column AXC
	Table 1.3.5-I AIM Board Type Restrictions	Corrected description of -M
		Remove description for AMC, APU, AP104 and APM
V24.11.x Rev B	ApiCmdBCXferDef	Added note for parameter gapmode (limitations for ASC and AME boards)
	Table B.5 – Functions With Performance Limitations By Plattform	Added performance limitation for ApiCmdBCXferDef (fast gap mode) on AME board
V24.12.x Rev A	ApiCmdScopeSetup ApiCmdScopeCalibrate	100 MHz Mode and 2x50 MHz are obsolete
V24.13.x Rev A	ApiGetBoardInfo	Added ASE1553 and AXC3910 to ul_DeviceType
	ApiCmdIni	Added new platform parameter
	ApiCmdBCAcycPrepAndSend TransferBlocking	Added new function.
	ApiCmdReplayRT	Corrected description of input parameter ‘rt’
V24.14.x Rev A	ApiCmdSysPXIGeographical AddressGet	Added new function
	Appendix B Functionality Overview	Added AMEE1553-2
V24.14.x Rev B		Added AMEE1553 Hardware Manual to “AIM Document Family”
V24.14.x Rev. C	ApiCmdSyncCounterGet ApiCmdSyncCounterSet ApiCmdBCModeCtrl	MC17 synchronize with timetag feature is now available on non multi channel embedded board variants.
	ApiGetBoardInfo	Corrected bit fields

Version	General or S/W Function Change	Description
	ApiCmdBCSrvReqVecStatus	Corrected name of struct member
	ApiCmdBCXferRead ApiCmdBCXferReadEx	Added missing value 4 for ERR filed of buffer report word
	ApiCmdBCXferReadEx	Added missing status word 2 information
	ApiCmdRTSACon	Corrected description of smod and/or masks
	Table B.4 – Function Support By Boards Without ASP	Added missing discretes for AMCE/AXE modules
V24.15.x Rev. A	ApiCmdRTSAMsgRead	Corrected return value of BID and updated the description.
	ApiCmdSysGetBoardInfo Limitations for specific boards ApiCmdBCTrafficOverrideControl ApiCmdBCTrafficOverrideGet	Added BC traffic override feature.
V24.16.x		No interface changes
V24.17.x Rev. A	Table B.5 – Functions With Performance Limitations By Plattform	Corrected fast gap value for AME1553
	ApiCmdSysGetBoardInfo	Added option to read fast gap support Added option to read low priority list support Added option to read return on error instruction support
	ALL	Corrected invalid status word figure references. Corrected invalid command word figure references.
	ApiCmdBMDytagMonDef ApiCmdBMDytagMonRead Tables of supported functions	Declared as obsolete. Only available on AYX platform.
	ApiCmdDataQueueOpen	Corrected order of ids for GEN_ASP and MIL_SCOPE.
	ApiCmdReplayIni	Corrected default value of reserved parameter sign.
	ApiCmdBCAcycSend	Changed note for multichannel boards
	ApiCmdBCInstrTblGen	Added API_BC_FWI_RET_ON_XFER_ERR instruction
	Appendix B Functionality Overview	Added AMEE1553-1 and unified device names
	ApiCmdIni	Added new board type (bt) values
	ApiCmdRTSACon	Added note limitations on board variants with disabled BC
	ApiCmdReset	Updated function description to describe the stream based reset and the shared simulation buffer handling.
	ApiCmdTrackDef ApiCmdTrackDefEx ApiCmdTrackPreAlloc	The use of uw_MultiplexedTrackNb zero to let the TSW allocate the buffer upon reception is now obsolete. Please do always allocate the required memory beforehand with ApiCmdTrackPreAlloc in case of uw_MultiplexedTrackNb zero.

Version	General or S/W Function Change	Description
	ApiCmdBCAcycSend ApiCmdBCInstrTblGen	Reverted changes above since the required firmware for the return on error instruction has not yet been released.
	ApiGetBoardInfo	Added values returned for AMEE1553-1/-2
V24.17.x Rev. B	ApiCmdSysGetBoardInfo	Added new define TY_BOARD_INFO_USB_AIM_POWER_CLASS
	Function Overview	New device ASC1553-2 If only ASC1553 is mention without a the number of channels -1/-2 the limitation is valid for both variants.
V24.18.x Rev. A	ApiConnectToServer ApiDisconnectFromServer ApiExit ApiInit ApiSetDllDbgLevel	Added note to non-thread-safe functions
	ApiOpenEx ApiClose ApiCmdIni ApiSetDeviceConfig	Added mutex to make functions thread-safe
	All	converted reference manuals to latex
V24.19.x Rev. A	APX1553, ACX1553, AVX1553	Removed. This devices are no longer supported.
	ApiCmdGenIoAddrInit ApiCmdGenIoOutChnWrite ApiCmdGenIoOutChnRead ApiCmdGenIoInChnRead ApiCmdGenIoSysSTaskCtrl ApiCmdGenIoTestSeq ApiCmdLoadSRec ApiCmdProgFlash ApiCmdScopeTriggerDef ApiCmdScopeCalibrate ApiSetDllDbgLevel	Marked functions as obsolete since AYX devices are no longer supported.
	ApiCmdBCXferDef ApiCmdRTSAConErr	Add Limitation for Manchester- and Sync-Error on ASC1553-2 (see description B.1.3).
	ApiGetBoardInfo ApiCmdSysGetBoardInfo Function Overview	New devices AXCX1553-1T/2T Corrected values for default amount of memory.
	ApiDelIntHandler ApiInstIntHandler	Added note for unused uc_Type under VxWorks.
	ApiReadBSPVersion ApiReadBSPVersionEx	Removed description.
	ApiFlushIntHandler	Documented existing function.
	ApiCmdBufWriteBlock	Documented new function.
	ApiCmdBufHostControl	Documented new function.
	ApiGetBoardName	Documented existing function.

Version	General or S/W Function Change	Description
V24.21.x Rev. A	Hardware Manuals ApiGetBoardInfo Limitations for embedded board variants Functionality Overview	Added new boards AM2E1553-1/2
	All	Improve copy and paste behavior.
	Functionality Overview ApiGetBoardInfo ApiCmdIni	New devices AXCX1553-1, -2, -4
	ApiCmdBCInstrTblGen	Added description of instruction API_BC_FWI_RET_ON_XFER_ERR
24.22.x Rev. A	ApiGetBoardInfo ApiSysGetBoardInfo	Updated TY_IRIG_CAPABILITIES description.
	ApiCmdSetIrigTime	Updated description.

Table A.2: Summary of Version Changes for each S/W Library Function

Legend:	C	–	Changes (including prototype, Dynamic Link Library (DLL) and/or documentation changes)
	I	–	Incompatibility change
	D	–	Only documentation was changed
	O	–	Obsolete
	R	–	Removed

	V24.22	V24.21	V24.19	V24.18	V24.17	V24.15	V24.14
Library Administration Functions							
ApiClose				C			
ApiConnectToServer				D			
ApiDelIntHandler			D				
ApiDisconnectFromServer				D			
ApiExit				D			
ApiFlushIntHandler			D				
ApiGetBoardInfo	C	D	C		C		D
ApiGetBoardName			D				
ApiGetDeviceConfig							
ApiGetDriverInfo							
ApiGetErrorDescription							
ApiGetErrorMessage							
ApiGetLibraryInfo							
ApiGetServerInfo							
ApiInit				D			
ApiInstIntHandler			D				
ApiOpen							
ApiOpenEx				C			
ApiSetDeviceConfig				C			
ApiSetDllDbgLevel				D			
System Functions							
ApiCmdBite							
ApiCmdDefMilbusProtocol							
ApiCmdDefRespTout							
ApiCmdExecSys							
ApiCmdGetIrigStatus							
ApiCmdGetIrigTime							
ApiCmdIni				C	C		
ApiCmdInitDiscretes							
ApiCmdLoadSRec			O				
ApiCmdProgFlash			O				
ApiCmdReadDiscretes							
ApiCmdReadDiscretesConfig							
ApiCmdReadDiscretesInfo							
ApiCmdReadSWVersion							
ApiCmdReset					D		

	V24.22	V24.21	V24.19	V24.18	V24.17	V24.15	V24.14
ApiCmdSetIrigTime	C						
ApiCmdSyncCounterGet							D
ApiCmdSyncCounterSet							D
ApiCmdSysFree							
ApiCmdSysGetMemPartition							
ApiCmdSysMalloc							
ApiCmdSysPXICon							
ApiCmdSysPXIGeographical AddressGet							!NEW!
ApiCmdSysSetMemPartition							
ApiCmdSystagCon							
ApiCmdSystagDef							
ApiCmdSysTriggerEdgeInputSet							
ApiCmdSysTriggerEdgeInputGet							
ApiCmdSysGetBoardInfo	C		C		C	C	
ApiCmdTrackDef					D		
ApiCmdTrackDefEx					D		
ApiCmdTrackPreAlloc					D		
ApiCmdTrackRead							
ApiCmdTrackReadEx							
ApiCmdTrackScan							
ApiCmdWriteDiscretes							
ApiReadAllVersions							
ApiReadBSPVersion							
ApiReadBSPVersionEx							
ApiReadRecData							
ApiReadVersion							
ApiWriteRepData							
Buffer Functions							
ApiBHModify							
ApiCmdBufC1760Con							
ApiCmdBufDef							
ApiCmdBufHostControl			!NEW!				
ApiCmdBufRead							
ApiCmdBufWrite							
ApiCmdBufWriteBlock			!NEW!				
ApiCmdRamReadDataset							
ApiCmdRamWriteDataset							
ApiReadBlockMemData							
ApiReadMemData							
ApiWriteBlockMemData							
ApiWriteMemData							
FIFO Functions							
ApiCmdBCAssignFifo							
ApiCmdFifoIni							
ApiCmdFifoReadStatus							

	V24.22	V24.21	V24.19	V24.18	V24.17	V24.15	V24.14
ApiCmdFifoWrite							
ApiCmdRTSAAssignFifo							
Calibration Functions							
ApiCmdCalCplCon							
ApiCmdCalSigCon							
ApiCmdCalTransCon							
ApiCmdCalXmtCon							
BC Functions							
ApiCmdBCAcycPrep							
ApiCmdBCAcycPrepAndSend TransferBlocking							
ApiCmdBCAcycSend							
ApiCmdBCBHDef							
ApiCmdBCBHRead							
ApiCmdBCDytagDef							
ApiCmdBCFrameDef							
ApiCmdBCGetDytagDef							
ApiCmdBCGetMajorFrame Definition							
ApiCmdBCGetMinorFrame Definition							
ApiCmdBCGetXferBuffer HeaderInfo							
ApiCmdBCGetXferDef							
ApiCmdBCHalt							
ApiCmdBCIni							
ApiCmdBCInstrTblGen		C					
ApiCmdBCInstrTblGetAddrFrom Label							
ApiCmdBCInstrTblIni							
ApiCmdBCModeCtrl							D
ApiCmdBCMFrameDef							
ApiCmdBCMFrameDefEx							
ApiCmdBCSrvReqVecCon							
ApiCmdBCSrvReqVecStatus							D
ApiCmdBCStart							
ApiCmdBCStatusRead							
ApiCmdBCTrafficOverrideControl						!NEW!	
ApiCmdBCTrafficOverrideGet						!NEW!	
ApiCmdBCXferCtrl							
ApiCmdBCXferDef							
ApiCmdBCXferDefErr							
ApiCmdBCXferDescGet							
ApiCmdBCXferDescMod							
ApiCmdBCXferRead						D	
ApiCmdBCXferReadEx						D	

	V24.22	V24.21	V24.19	V24.18	V24.17	V24.15	V24.14
RT Function							
ApiCmdRTBHDef							
ApiCmdRTBHRead							
ApiCmdRTDytagDef							
ApiCmdRTEnaDis							
ApiCmdRTGetDytagDef							
ApiCmdRTGetSABuffer HeaderInfo							
ApiCmdRTGetSAConErr							
ApiCmdRTGetSimulationInfo							
ApiCmdRTGlobalCon							
ApiCmdRTHalt							
ApiCmdRTIni							
ApiCmdRTLCW							
ApiCmdRTLSW							
ApiCmdRTModeCtrl							
ApiCmdRTMsgRead						C	
ApiCmdRTMsgReadAll							
ApiCmdRTNXW							
ApiCmdRTRespTime							
ApiCmdRTRespTimeGet							
ApiCmdRTSACon					C		D
ApiCmdRTSAConErr							
ApiCmdRTSADWCGet							
ApiCmdRTSADWCSet							
ApiCmdRTSAMsgRead							
ApiCmdRTSAMsgReadEx							
ApiCmdRTStart							
ApiCmdRTStatusRead							
BM Functions							
ApiCmdBMActRead							
ApiCmdBMCapMode							
ApiCmdBMDytagMonDef					O		
ApiCmdBMDytagMonRead					O		
ApiCmdBMFilterIni							
ApiCmdBMFTWIni							
ApiCmdBMHalt							
ApiCmdBMIllegalIni							
ApiCmdBMIni							
ApiCmdBMIniMsgFltRec							
ApiCmdBMIntrMode							
ApiCmdBMReadMsgFltRec							
ApiCmdBMRTActRead							
ApiCmdBMRTSAActRead							
ApiCmdBMStackEntryFind							
ApiCmdBMStackEntryRead							

	V24.22	V24.21	V24.19	V24.18	V24.17	V24.15	V24.14
ApiCmdBMStackpRead							
ApiCmdBMStart							
ApiCmdBMStatusRead							
ApiCmdBMSWXMIni							
ApiCmdBMTCBIni							
ApiCmdBMTCIIni							
ApiCmdBMTIWIni							
ApiCmdDataQueueClose							
ApiCmdDataQueueControl							
ApiCmdDataQueueOpen					D		
ApiCmdDataQueueRead							
ApiCmdScopeSetup							
ApiCmdScopeTriggerDef			O				
ApiCmdScopeStart							
ApiCmdScopeStatus							
ApiCmdScopeStop							
ApiCmdScopeReset							
ApiCmdScopeTriggerDefEx							
ApiCmdScopeCalibrate			O				
ApiCmdScopeOffset Compensation							
ApiCmdQueueFlush							
ApiCmdQueueHalt							
ApiCmdQueueIni							
ApiCmdQueueRead							
ApiCmdQueueStart							
ApiCreateScopeBuffer							
ApiCreateScopeBufferList							
ApiFreeScopeBuffer							
ApiProvideScopeBuffers							
ApiWaitForScopeFinished							
Replay Functions							
ApiCmdReplayIni					D		
ApiCmdReplayRT							D
ApiCmdReplayStart							
ApiCmdReplayStatus							
ApiCmdReplayStop							
GenIo Functions							
ApiCmdGenIoAddrInit			O				
ApiCmdGenIoOutChnWrite			O				
ApiCmdGenIoOutChnRead			O				
ApiCmdGenIoInChnRead			O				
ApiCmdGenIoSysSTaskCtrl			O				
ApiCmdGenIoTestSeq			O				

APPENDIX B

FUNCTIONALITY OVERVIEW

B APPENDIX B FUNCTIONALITY OVERVIEW

Appendix B provides information regarding the software functionality for each board type. This information is provided in a comparison table.

B.1 Limitations for specific boards

B.1.1 ASC1553-1/2-A

Due to the architecture and technology of the USB link the performance of the USB devices is different from other PCI based AIM devices. This needs careful thought and consideration when migrating or developing applications, which make use of an explicit interrupt handling. The “good” interrupt latencies (with respect to real time Behavior), for the other boards, are not to be expected when working with USB devices !

B.1.2 ASC1553-1

Fast Intermessage Gap Mode (ApiCmdBcXferDef/ API_BC_GAP_MODE_FAST) is not supported by the on board 1553-Firmware, setting up a Fast Intermessage Gap will result in a Standard Gap on the MILBus.

B.1.3 ASC1553-2

The used MILBus transceiver on the ASC1553-2 provides a safety mechanism and doesn’t allow to send longer high or low pulses than 2.5us. If the transceiver should send a longer high or low pulse than 2.5us the signal goes back to the 0V-level after 2.5us. This leads to certain limitations for Manchester- and Sync-Errors. E.g. if a DW with Machester-High error on the first bit after the DW-Sync and a "1" on second databit should be send, this should result in a high pulse of 3 us (1,5us + 1 us + 0,5 us) but the transceiver sends only 2.5us.

B.1.4 ANET1553-1/2

Due to the architecture and technology of the Ethernet link the performance of the network devices is different from other PCI based AIM devices. This needs careful thought and consideration when migrating or developing applications, which make use of an explicit interrupt handling.

B.1.5 ASE1553M-1/2/4

Single Function Board (BC+BM or RT(s)+BM), Replay and Error Injection is not supported.

B.1.6 Limitations for boards with Multichannel Firmware

On devices with a multichannel firmware one BIU processor handles two milbus streams. Because of this some rearly used functionality had to be removed for performance reasons.

Boards with multichannel firmware are:

- AEC1553-2
- AXC1553-4
- AMCX1553-4
- APE1553-4
- APXX1553-4
- ACE1553-4
- ACXX1553-4
- ANET1553-2
- AMCE1553-4
- AXE1553-4
- ASE1553M-4

B.1.7 Limitations for embedded board variants

Embedded board variants can be detected by interpreting the bit field of TY_BOARD_INFO_APPLICATION_TYPE returned by ApiCmdSysGetBoardInfo.

On embedded devices, the following functionality is not available:

- Replay
- Error Injection
- Dytags
- BC "Global Bus Mode" (parameter "tbm" of function ApiCmdBcIni())
- BC Fast Gap Mode
- BC Traffic Override
- 1553-A Protocol Support
- Acyclic Transfers with Start on TimeTag
- BM Capture Mode "API_BM_CAPMODE_ONLY"
- Several Trigger Control Blocks (only one is available)
- Trigger Type API_BM_TRG_DATA_VALUE in Trigger Control Block
- Enhanced BM Activity Page
- BM Word/Transfer/Error Counters for Primary/Secondary Channel
- BM Busload Counters

Available embedded board variants are:

- AME1553-1-AP
- AME1553-1-E
- AMEE1553-1/2
- AM2E1553-1/2
- AMCE1553-1/2/4
- AXE1553-1/2/4

B.1.8 Limitations for non embedded board variants

The dual redundant RT operation mode (see ApiCmdRTIni) is only available for embedded boards (see previous chapter). The functions ApiCmdRTSADWCGet() and ApiCmdRTSADWCSet() are only available for embedded boards.

B.2 Board functionality overview

Table B.1: Functionality Overview for boards with ASP

	ANET1553-1	ANET1553-2	ASC1553-A	ASC1553
Coupling Modes				
Isolated	✓	✓		
Transformer	✓	✓	(2)	(2)
Direct	✓	✓	(2)	(2)
Network	✓	✓		
Digital Wrap-Around	✓	✓	✓	✓
IRIG				
IRIG Switch Intern/Extern	✓	✓	✓	
General functionality				
Acyclic on Time Tag	✓			
Discretes available	✓	✓	✓	
MilScope (optional)				
Variable Output Amplitude	✓	✓		
500Kbit Transmittion Mode				
Number of Global Memory Banks	1	1	1	1
Default amount of memory (MB)	$4^{(1)}$	$4^{(1)}$	$1^{(1)}$	$1^{(1)}$
Maximum global memory (MB)	16	16	16	16
Error Injection of High Resolution Zero Crossing Deviation			✓	✓
RT Error Injection no. ApiCmdRTSAConErr 16 – 19	✓			
Support of PXI Trigger and PXI TT Clear Functionality				

(1) MB per channel
(2) Depending on the hardware configuration

Table B.2: Functionality Overview for boards without ASP

	AEC / AXC / AMCX/ AXCX1553	ASE1553	AMCE / AXE1553	AXCX1553-1T/2T	AMCX1553-T	APE / APXX1553-1/2	APE / APXX1553-4	ACE / ACXX1553-1/2	ACE / ACXX1553-4	AME1553	AME1553-AP	AMEE / AM2E1553
Coupling Modes												
Isolated				✓	✓	✓	✓	✓	✓			
Transformer	(2)	(2)	(2)	✓	✓	✓	✓	✓	✓	(2)	(2)	(2)
Direct	(2)	(2)	(2)	✓	✓	✓	✓	✓	✓	(2)	(2)	(2)
External				✓	✓	✓	✓	✓	✓			
Digital Wrap-Around	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
IRIG												
IRIG Switch Intern/Extern	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
General functionality												
Acyclic on Time Tag						✓		✓				
Discretes available	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MilScope (optional)						✓		✓				
Variable Output Amplitude				✓	✓	✓	✓	✓	✓			
500Kbit Transmittion Mode						✓		✓				
Input trigger edge configuration.						✓		✓				
Number of Global Memory Banks	1	1	1	1	1	1	1	1	1	1	1	1
Default amount of memory (MB)	$4^{(1)}$	$4^{(1)}$	$4^{(1)}$	$4^{(1)}$	$8^{(1)}$	$4^{(1)}$	$4^{(1)}$	$4^{(1)}$	$4^{(1)}$	$8^{(1)}$	$8^{(1)}$	$1^{(1)}$
Maximum global memory (MB)	$16^{(1)}$	$16^{(1)}$	$16^{(1)}$	$8^{(1)}$	$16^{(1)}$	16	16	16	16	8	8	2
Error Injection of High Resolution Zero Crossing Deviation	✓		✓			✓		✓		✓		
RT Error Injection no. ApiCmdRTSAConErr 16-19						✓		✓				
Support of PXI Trigger and PXI TT Clear Functionality								✓	✓			

(1) MB per channel
(2) Depending on the hardware configuration
(3) Depending on jumper settings

Table B.3: Function Support By Boards With ASP

	ANET1553	ASC1553-A	ASC1553
BC functions			
ApiCmdBCTrafficOverrideControl	✓	✓	✓
ApiCmdBCTrafficOverrideGet	✓	✓	✓
System functions			
ApiCmdInitDiscretes	✓	✓	
ApiCmdReadDiscretes	✓	✓	
ApiCmdReadDiscretesConfig	✓	✓	
ApiCmdReadDiscretesInfo	✓	✓	
ApiCmdWriteDiscretes	✓	✓	
ApiCmdSysPXICon			
ApiCmdSysPXIGeographicalAddressGet			
ApiCmdSysTriggerEdgeInputSet			
ApiCmdSysTriggerEdgeInputGet			
ApiCmdScopeStart			
ApiCmdScopeStatus			
ApiCmdScopeStop			
ApiCmdScopeReset			
ApiCmdScopeSetup			
ApiCmdScopeTriggerDef			
ApiCmdScopeTriggerDefEx			
ApiCmdScopeCalibrate			
ApiCmdScopeOffsetCompensation			
ApiCreateScopeBuffer			
ApiCreateScopeBufferList			
ApiFreeScopeBuffer			
ApiProvideScopeBuffer			
ApiWaitForScopeFinished			
ApiCmdCalTransCon			
GenIo functions			
ApiCmdGenIoAddrInit			
ApiCmdGenIoOutChnWrite			
ApiCmdGenIoOutChnRead			
ApiCmdGenIoInChnRead			
ApiCmdGenIoSysTaskCtrl			
ApiCmdGenIoTestSeq			
RT functions			
ApiCmdRTSADWCGet			
ApiCmdRTSADWCSet			
BM functions			
ApiCmdBMDytagMonDef			
ApiCmdBMDytagMonRead			

Functions that are not listed in this table are supported on all devices.
✓: Supported

Table B.4: Function Support By Boards Without ASP

	AME1553	AME1553-1-AP	AMEE / AM2E1553	AEC / AMCX / AXC / ASE/ AXCX1553	AMCE / AXE1553	AMCX-T / AXC1553-T / AXCX1553-T	APE / APXX1553	ACE / ACXX1553
BC functions								
ApiCmdBCTrafficOverrideControl	✓			✓		✓	✓	✓
ApiCmdBCTrafficOverrideGet	✓			✓		✓	✓	✓
System functions								
ApiCmdInitDiscretes	✓	✓	✓	✓	✓	✓	✓	✓
ApiCmdReadDiscretes	✓	✓	✓	✓	✓	✓	✓	✓
ApiCmdReadDiscretesConfig	✓	✓	✓	✓	✓	✓	✓	✓
ApiCmdReadDiscretesInfo	✓	✓	✓	✓	✓	✓	✓	✓
ApiCmdWriteDiscretes	✓	✓	✓	✓	✓	✓	✓	✓
ApiCmdSysPXICon								✓
ApiCmdSysPXIGeographicalAddressGet								✓
ApiCmdSysTriggerEdgeInputSet							✓	✓
ApiCmdSysTriggerEdgeInputGet							✓	✓
ApiCmdScopeStart							✓	✓
ApiCmdScopeStatus							✓	✓
ApiCmdScopeStop							✓	✓
ApiCmdScopeReset							✓	✓
ApiCmdScopeSetup							✓	✓
ApiCmdScopeTriggerDef							✓	✓
ApiCmdScopeTriggerDefEx							✓	✓
ApiCmdScopeCalibrate							✓	✓
ApiCmdScopeOffsetCompensation							✓	✓
ApiCreateScopeBuffer							✓	✓
ApiCreateScopeBufferList							✓	✓
ApiFreeScopeBuffer							✓	✓
ApiProvideScopeBuffer							✓	✓
ApiWaitForScopeFinished							✓	✓
ApiCmdCalTransCon							✓	✓
GenIo functions								
ApiCmdGenIoAddrInit								
ApiCmdGenIoOutChnWrite								
ApiCmdGenIoOutChnRead								
ApiCmdGenIoInChnRead								
ApiCmdGenIoSysSTaskCtrl								
RT functions								
ApiCmdRTSADWCGet	✓	✓	✓		✓			
ApiCmdRTSADWCSet	✓	✓	✓		✓			
BM functions								
ApiCmdBMDytagMonDef								
ApiCmdBMDytagMonRead								

Functions that are not listed in this table are supported on all devices.

✓: Supported

Table B.5: Functions With Performance Limitations By Plattform

	ASC1553-1	ASC1553-2	AME1553	AMEE / AM2E1553	AEC / AMCX / AXC / APE / ACE APXX / ACXX / ASE1553	ANET1553-1/2
BC functions						
ApiCmdBCXferDef API_BC_GAP_MODE_FAST		✓			✓	✓
Fifo functions						
ApiCmdBCAssignFifo	✓	✓	✓	✓	✓	✓
ApiCmdFifoIni	✓	✓	✓	✓	✓	✓
ApiCmdFifoReadStatus	✓	✓	✓	✓	✓	✓
ApiCmdFifoWrite	✓	✓	✓	✓	✓	✓
ApiCmdRTSAAssignFifo	✓	✓	✓	✓	✓	✓
Track functions						
ApiCmdTrackDef	✓	✓	✓	✓	✓	✓
ApiCmdTrackDefEx	✓	✓	✓	✓	✓	✓
ApiCmdTrackPreAlloc	✓	✓	✓	✓	✓	✓
ApiCmdTrackRead	✓	✓	✓	✓	✓	✓
ApiCmdTrackReadEx	✓	✓	✓	✓	✓	✓
ApiCmdTrackScan	✓	✓	✓	✓	✓	✓
Systag						
ApiCmdSystagCon	✓	✓	✓	✓	✓	✓
ApiCmdSystagDef	✓	✓	✓	✓	✓	✓
User interrupts						
ApiInstIntHandler	✗	✗	✓	✓	✓	✗
ApiDelIntHandler	✗	✗	✓	✓	✓	✗

Functions that are not listed in this table are supported on all devices.

✓: Supported

✗: Supported with performance limitations.