

Software User's Manual

CEI-x30-SW

Copyrights

User's Manual Copyright © 2004 -2021 Abaco Systems, Inc.

This software product is copyrighted and all rights are reserved. The distribution and sale of this product are intended for the use of the original purchaser only per the terms of the License Agreement.

Confidential Information - This document contains Confidential/Proprietary Information of Abaco Systems, Inc. and/or its suppliers or vendors. Distribution or reproduction prohibited without permission.

THIS DOCUMENT AND ITS CONTENTS ARE PROVIDED "AS IS", WITH NO REPRESENTATIONS OR WARRANTIES OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO WARRANTIES OF DESIGN, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. ALL OTHER LIABILITY ARISING FROM RELIANCE ON ANY INFORMATION CONTAINED HEREIN IS EXPRESSLY DISCLAIMED.

Microsoft is a registered trademark of Microsoft Corporation.
Windows is a registered trademark of Microsoft Corporation.
VxWorks is a registered trademark of WindRiver Systems Corporation.
INTEGRITY is a registered trademark of Green Hills Software Incorporated.
LabVIEW is a registered trademark of National Instruments Corporation.

Abaco Systems, Inc. acknowledges the trademarks of other organizations for their respective products or services mentioned in this document.

CEI-x30-SW Software User's Manual (1500-048)

Software Revision: 5.13A
Document Revision: 5.13
Document Date: 12 October 2021

Abaco Systems, Inc.
26 Castilian Drive, Suite B
Goleta, CA 93117
Main +1 805-965-8000 or +1 877-429-1553
Support +1 805-883-6097

avionics.support@abaco.com (email)

<https://www.abaco.com/products/avionics>

Additional Resources

For more information, please visit the Abaco Systems website at:

www.abaco.com

Contents and Tables

Contents

Chapter 1	The CEI-x30 ARINC Product Line.....	1
	Overview.....	1
	Common Features.....	2
	Multiprotocol Boards	2
	Operating Systems Supported.....	4
	Summary.....	4
Chapter 2	Windows Installation.....	5
	Software Installation under Windows.....	5
	Hardware Installation	6
	Device Driver Installation under Windows.....	6
	Installation Verification	7
Chapter 3	VxWorks Installation	8
	Overview.....	8
	Building a VxWorks Image	8
	Using the Sample Program.....	13
	Building the API and Sample Program with Workbench	14
	Target-specific Compiler Directives.....	17
Chapter 4	Linux Installation	19
	Overview.....	19
	Software Installation.....	19
	Building Applications.....	20
	Automatic Installation (Builds LSP and API).....	20
	Manual Installation	21
	Linux Driver Operation.....	21
	Troubleshooting.....	22
	Useful Linux system utilities	22
	Compilation Errors.....	22

	Run-time Errors.....	23
Chapter 5	INTEGRITY® Support	24
	Introduction.....	24
	INTEGRITY Installation.....	24
	INTEGRITY PCI Driver Installation.....	25
	Building the CEI-x30 API with Multi.....	25
	Building INTEGRITY Applications	28
Chapter 6	BusTools/ARINC™ Data Bus Analyzer	30
	General Information	30
	BusTools/ARINC Demo Software	30
Chapter 7	Application Development.....	31
	Overview.....	31
	Windows Libraries	31
	Data Types, Constants, and API Routine Prototypes.....	32
	Time-tag Structure Definition	32
	Setting the Device Time.....	33
	Return Status Values	34
	Programming with the CEI-x30 API Interface.....	35
	Example Routines in C – Summary.....	36
	Tst_cnfg.c.....	37
	Config_from_file.c.....	38
	Multiprocess_test.c	38
	C# Support	39
	C# Managed Wrapper Functions.....	41
	Visual Basic and VB.NET Support	43
	Working with Unsigned Integers in Visual Basic.....	43
	AutoConfig ARINC Configuration File Generator.....	44
	Dealing with Complex Message Scheduler Transmit Scenarios	44
Chapter 8	Application Programming Interface	46
	Overview.....	46
	CEI-x30 API Source Files.....	46
	CEI-x30 API Header Files	49
	Building the API for Embedded and Certified Systems.....	51
	Defining Custom Content in Your API Build.....	51
	Defining the Data Types Used in Your API Build	54
	API Data Types	55
	API Routines - Summary	56
	Initialization and Control Routines.....	56
	Device Control Routines.....	56

Termination Routines.....	56
Receive/Transmit Channel-level Configuration Routines	56
Device-level Configuration Routines	57
Receive Data Processing Routines	57
Transmit Data Processing Routines.....	59
Timer-related Routines	59
Information and Status Routines	60
Utility Routines	60
AR_ASSIGN_SCHEDULER_START_OFFSETS.....	62
AR_BOARD_TEST.....	63
AR_BYPASS_WRAP_TEST.....	64
AR_CLR_RX_COUNT	65
AR_CLOSE.....	66
AR_CHANNEL_CONFIGURATION_FROM_XML_FILE.....	67
AR_CONFIG_CHANNEL_FROM_TXT_FILE.....	68
AR_CONVERT_1553_TIME_TO_STRING.....	69
AR_CONVERT_TIME_TO_STRING	70
AR_DEFINE_MESSAGES_FROM_TXT_FILE	71
AR_DEFINE_MESSAGES_FROM_XML_FILE	72
AR_DEFINE_MSG	73
AR_DEFINE_MSG_BLOCK	74
AR_ENH_LABEL_FILTER	76
Label Filtering.....	76
Interrupt Generation.....	76
AR_EXECUTE_BIT.....	78
AR_GET_573_FRAME.....	80
AR_GET_429_MESSAGE.....	82
AR_GET_BASE_ADDR.....	84
AR_GETBLOCK	85
AR_GETBLOCK_T	87
AR_GET_BOARDNAME	89
AR_GET_BOARDTYPE	90
AR_GET_CHANNEL_INDEX_INFO	91
AR_GET_CONFIG.....	93
AR_GET_DATA	97
AR_GET_DATA_XT.....	99
AR_GET_DEVICE_CONFIG.....	101
AR_GET_573_CONFIG	109
AR_GET_ERROR	112
AR_GETFILTER	113
AR_GET_LABEL_FILTER.....	115
AR_GET_LATEST.....	116
AR_GET_LATEST_T	117
AR_GETNEXT	118

AR_GETNEXTT	119
AR_GETNEXT_XT	121
AR_GET_RX_CHANNEL_STATUS	123
AR_GET_RX_COUNT	125
AR_GET_SNAP_DATA	126
AR_GET_STATUS	127
AR_GET_STORAGE_MODE	128
AR_GET_TIME	129
AR_GET_TIMERCNTL	131
AR_GETWORD	132
AR_GETWORDT	134
AR_GETWORD_XT	136
AR_GO	138
AR_HW_INTERRUPT_BUFFER_READ	139
AR_INTERRUPT_QUEUE_READ	140
AR_INITIALIZE_API	141
AR_INITIALIZE_DEVICE	143
AR_HW_INTERRUPT_BUFFER_READ	144
AR_INTERRUPT_QUEUE_READ	145
AR_LABEL_FILTER	146
AR_LOADSLV	147
AR_MODIFY_MSG	149
AR_MODIFY_MSG_BLOCK	151
AR_NUM_RCHANS	153
AR_NUM_XCHANS	154
AR_OPEN	155
AR_PUT_429_MESSAGE	157
AR_PUT_573_FRAME	158
AR_PUTBLOCK	159
AR_PUTBLOCK_MULTI_CHAN	160
AR_PUTFILTER	162
AR_PUTWORD	164
AR_QUERY_DEVICE	165
AR_READ_SCHEDULED_MSG_BLOCK	167
AR_RESET	169
AR_RESET_TIMERCNT	170
AR_SET_CONFIG	171
AR_SET_DEVICE_CONFIG	176
AR_SET_573_CONFIG	182
AR_SET_MULTITHREAD_PROTECT	185
AR_SET_ISR_FUNCTION	186
AR_SET_PRELOAD_CONFIG	187
AR_SET_RAW_MODE	189
AR_SET_STORAGE_MODE	191

AR_SET_TIME.....	192
AR_SLEEP.....	194
AR_SET_TIMERRATE.....	195
AR_STOP.....	196
AR_UPDATE_MSG_BLOCK.....	197
AR_VERSION.....	198
AR_WAIT.....	199
AR_XMIT_SYNC.....	200

Figures

Figure 1. Linux Installation Directory Structure	20
Figure 2. Integrity PCI Driver Installation.....	25
Figure 3. Example CEI-x30 Integrity API Library Project Setup.....	27
Figure 4. Example CEI-x30 Integrity Library Project Options.....	27
Figure 5. Example CEI-x30 Integrity Application Project Setup	28
Figure 6. Example CEI-x30 INTEGRITY Application Project Option	28
Figure 7. Adding a MemoryPoolSize Entry	29
Figure 8. Modifying the Value for the DefaultStartIt Attribute.....	29

Tables

Table 1. CEI-x30 ARINC Products	1
Table 2. VxWorks PCI Driver Component Definition Files.....	11
Table 3. CEI-x30 API C Source File Content for a Custom Build.....	54
Table 4. CEI-x30 API C Data Type Definitions	55

The CEI-x30 ARINC Product Line

Overview

The CEI-x30 ARINC Product Line is a multiple-channel ARINC interface design available in several form factors and channel configurations supporting ARINC 429, ARINC 573/717, Avionics Discrete I/O, and IRIG time synchronization.

Products included in the CEI-x30 ARINC Product Line are listed with their channel configurations and available/optional features in the following table:

Table 1. CEI-x30 ARINC Products

Product Name	Form Factor	ARINC 429 Maximum Channel Count	Available/Optional Features
RCEI-530	PCI	16 Receivers 16 Transmitters	ARINC 573/717 Rx/Tx IRIG Timecode Rx/Tx 16 Discrete Inputs 16 Discrete Outputs
RAR-PCIE	PCI Express	16 Receivers 16 Transmitters	ARINC 573/717 Rx/Tx IRIG Timecode Rx/Tx 16 Discrete Inputs 16 Discrete Outputs
RAR-MPCIE	Mini PCI Express	8 Receivers 4 Transmitters	ARINC 573/717 Rx/Tx IRIG Timecode Rx/Tx 4 Bidirectional Discrete Input/Output
RAR-CPCI	Compact PCI	16 Receivers 16 Transmitters	ARINC 573/717 Rx/Tx IRIG Timecode Rx/Tx 16 Bidirectional Discrete Input/Output
RAR-XMC	XMC	16 Fixed Receivers plus 16 Channels either Fixed Transmit or Receive, or Programmable Transmit/Receive	ARINC 573/717 Rx/Tx IRIG Timecode Rx/Tx 4 Discrete Inputs 4 Discrete Outputs

Product Name	Form Factor	ARINC 429 Maximum Channel Count	Available/Optional Features
RAR-EC	ExpressCard	7 Receivers 4 Transmitters	ARINC 573/717 Rx/Tx IRIG Timecode Rx/Tx 4 Bidirectional Discrete Input/Output
CEI-830 RCEI-830A	PMC	16 Receivers 16 Transmitters	ARINC 573/717 Rx/Tx IRIG Timecode Rx/Tx 4 Discrete Inputs 4 Discrete Outputs
R830RX	PMC	32 Receivers	IRIG Timecode Rx/Tx
CEI-430	PC/104-Plus	12 Receivers 12 Transmitters	ARINC 573/717 Rx/Tx IRIG Timecode Rx/Tx 16 Bidirectional Discrete Input/Output
RCEI-430A	PC/104-Plus	24 Receivers 4 Transmitters	ARINC 573/717 Rx/Tx IRIG Timecode Rx/Tx 16 Bidirectional Discrete Input/Output
AMC-A30	AMC	12 Receivers 12 Transmitters	IRIG Timecode Rx/Tx 4 Bidirectional Discrete Input/Output
RCEI-830X820	PMC	8 Receivers 8 Transmitters	None

For the CEI-830, RCEI-830A, RCEI-830X820, and R830RX PMC products, a variety of bus adapter configurations are also available, supporting both front and rear-I/O access for PCI, PCI Express, and CompactPCI platforms. For the RAR-XMC and RAR-MPCIE products, PCI Express bus adapters are available.

Common Features

The CEI-x30 ARINC Product Line incorporates a common firmware design across all products. Features available on most CEI-x30 products include the following, with IRIG optional on many products.

General Features

- 64-bit 1 microsecond resolution on-board timer
- Fixed ARINC 429 Receive Threshold Levels
- IRIG-B reception supporting AM or DC/TTL input
- IRIG-B generator supporting DC/TTL output

ARINC 429 Transmit Features

- Fixed transmit signal levels

- Individually programmable channel configuration attributes:
 - transmit bit rate
 - slew rate
 - automatic parity generation
 - message bit count and gap error injection
- Individual 2048 message aperiodic transmit buffer for each channel
- 2048 entry message scheduler table supporting high accuracy periodic message transmission for all channels (exception of a 1024 entry limit on the CEI-830)
- 1 millisecond periodic message scheduler transmission accuracy

ARINC 429 Receive Features

- Fixed receive threshold levels
- Message label filtering/triggering and PCI event/interrupt generation
- 64-bit message time-stamp with a 1 microsecond resolution
- Individually programmable channel configuration attributes:
 - receive bit rate
 - parity detection
 - message length/gap error detection
- Multiple message buffering schemes:
 - Individual 2048 message circular buffer for each channel
 - 16384 message merged mode receive buffer with individual receive channel merged buffer selection
 - Independent label/SDI field value-based snapshot storage

Multiprotocol Boards

The CEI-x30 ARINC common firmware design is also included in the **RAR15X** (rear-I/O) and **RAR15XF** (front-I/O) multiprotocol product configurations, supporting both MIL-STD-1553 and ARINC 429. While not formally considered part of the CEI-x30 ARINC product line, the respective software distribution CEI-x30-SW supports any of the Abaco Systems multiprotocol boards in which the CEI-x30 common firmware design is provided.

Operating Systems Supported

The CEI-x30-SW software distribution supports the following products on a range of operating systems, with installation instructions provided in individual chapters of this manual:

- **32-bit and 64-bit Windows 10, 8.1, 8, 7, Server 2012 R1/R2 and 2008 R2, and 32-bit XP** – all products
- **32-bit and 64-bit Linux** – all products
- **VxWorks 6.x** – CEI-830, RCEI-830A, R830RX, CEI-430, RCEI-430A, RCEI-530, RCEI-830X820, RAR-XMC, RAR15-XMC, RAR-MCPIE
- **VxWorks 7** –RCEI-830A, RCEI-830X820, RAR-XMC, RAR15-XMC, RAR-MCPIE
- **Integrity 5.10 and 5.11** – CEI-830, RCEI-830A, RAR-XMC, RAR15-XMC

Summary

Each board in the CEI-x30 ARINC Product Line is described in detail in the CEI-x30 Product Line Hardware User's Manual. For information regarding the CEI-x30-SW distribution and application programmer's interface, see the chapter in this manual titled "Application Programming Interface".

Windows Installation

Software Installation under Windows

Although system resources may limit the number of boards installed on a system, the CEI-x30-SW distribution supports up to 128 devices when installed under 32-bit and 64-bit versions of the Windows 10, 8.1, 8, 7, Server 2008 R2, and Server 2012 operating systems, and 32-bit Windows XP.

Prior to physically installing the CEI-x30 product, the CEI-x30-SW software distribution should be installed on your host computer. Failure to properly install the software prior to the hardware may result in corruption of the Windows Device Manager Registry settings and require restoration to a previous configuration.

To install the software, follow these steps:

1. Exit all programs.
2. Insert the CEI-x30-SW CD into your CD drive.
3. If the installation does not automatically start after 10 seconds:
 - Click Start from the Windows Task Bar and select Run.
 - Use the Browse button to locate the Setup.exe file in the **Setup\Disk1** folder.
 - Double-click the file **setup.exe**. Then, click OK to launch the setup program.
4. Follow the on-screen instructions for the installation.
5. Note which device number is allocated during the installation.
6. Following successful completion of the installation, **turn off the computer**.

Hardware Installation

Once the software has been successfully installed, follow these steps to install the hardware and Windows device driver:

With the computer powered off, install the CEI-x30 board into any available slot, PMC/XMC site, or PC/104-Plus stack location. The CEI-430 and RCEI-430A require card jumpers to agree with the installed stack location. Refer to the *PCI Interface* description in either chapter “CEI-430” or “CEI-430A” of the *CEI-x30 Product Line Hardware User Manual* for the applicable jumper settings.

Device Driver Installation under Windows

To install the hardware under all supported Windows operating systems, follow these steps:

1. Power-up the PC.

With Windows 10, 8.1, 8, 7 and Server 2008R2/2012, device installation should occur automatically; for other Windows versions, the Windows Plug and Play hardware manager should detect the CEI-x30 device, and the *Found New Hardware* dialog box may automatically startup. Decline any request to query the Microsoft web site to obtain drivers for this device.

- For Windows XP, select the *Install the software automatically* option and click **Next**. Under the *Completing the Found New Hardware* dialog, click Finish.

2. If Windows does not detect the new hardware, you must manually install the device driver:

- Click Start and point to Settings.
- Select the Control Panel.
- Select Add New Hardware.

3. If the device drivers are not automatically detected and you are prompted for a path to the driver location, enter the full path to the distribution driver folder located beneath

C:\Program Files\Condor Engineering\CEI-x30-SW\Drivers

and click **OK**.

4. To check for proper driver installation, open the Device Manager as follows:

- Under Windows 7, XP and Server 2008R2, use the “Start -> Run” command prompt, enter “devmgmt.msc” to open the Windows Device Manager.

- Under Windows 10, 8.1, 8 and Server 2012, open the “Run” application and enter “devmgmt.msc” to open the Windows Device Manager.
 - Expand the **Abaco Avionics Devices** folder.
5. Verify the device entry *Product Name* for your device is shown with no exclamation point overlaying the icon. If this is true, the device driver was properly installed. You have completed installation of the hardware.

Installation Verification

To verify the device driver is properly installed, execute the Test Configuration program.

1. Under Windows 7, XP and Server 2008R2
 - a. Click Start, then Programs.
 - b. Find and expand the **Abaco CEI-x30-SW** program group
 - c. Invoke the **CEI-x30 Test Installation** shortcut located therein.
2. Under Windows 8, 8.1 and Server 2012
 - a. Display “Apps by name”.
 - b. Find and invoke the **CEI-x30 Test Installation** shortcut.
3. Under Windows 10
 - a. Expand “All apps”.
 - b. Scroll down and expand the **Abaco CEI-x30-SW** application group.
 - c. Invoke the **CEI-x30 Test Installation** shortcut beneath.

This program executes an internal wrap test on all available channels and notifies you of success or failure. If the program reports success on all channels tested, you are ready to use your new board.

VxWorks Installation

Overview

VxWorks is an embedded real-time operating system supporting flexible hardware configurations. The CEI-x30-SW API source compiles and runs under the PowerPC and x86 VxWorks Board Support Packages.

The CEI-x30-SW API supports up to sixteen boards on a single VxWorks host, using device numbers 0-15 to designate the device on which to operate. VxWorks assigns the device numbers based on the order it encounters the devices on the bus. The first device is device 0, the next device is 1, and so on. If you have only a single board in your system, it is always device 0. Use this value when programming a board using the supplied API.

To use the CEI-x30-SW API with VxWorks you must first build a VxWorks image supporting the desired board configuration. Upon boot of this image, you may download and execute the client application. These basic steps are described in this chapter.

Building a VxWorks Image

To incorporate your CEI-x30-SW API source with VxWorks you must rebuild your VxWorks image. The procedure provided for including the CEI-x30-SW API and common avionics driver in your VxWorks image utilizes the VxWorks Component Installation method, and applies to both Tornado and Workbench development environments. Since these methods differ greatly, these instructions are written in a generic fashion and must be interpreted for your environment.

In addition to these general instructions are specific dependencies on the use of the VxBus Gen1 and Gen2 device drivers with your target processor and BSP, also described below.

VxBus Gen 1 Driver Support (VxWorks 6.8 and 6.9)

1. To install the generic Abaco Avionics common VxBus Gen 1 driver support for VxWorks 6.x, copy the file *40avioVxwDrv.cdf* from the *Component Installation File* folder beneath:

/Program Files/Condor Engineering/CEI-x30-SW/Source/VxWorks/VxBus Gen1 Driver

To

[*Workbench_directory_path*]/target/config/comps/vxWorks

2. Copy the following source and header files from the folder:

/Program Files/Condor Engineering/CEI-x30-SW/Source/VxWorks/VxBus Gen1 Driver

to either folder

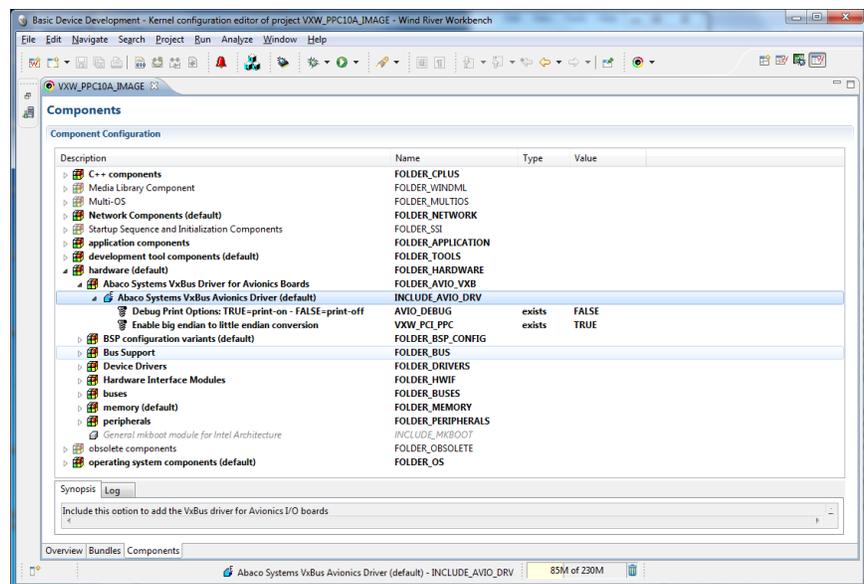
[*Workbench_directory_path*]/target/config/<BSP Folder> or

[*Workbench_directory_path*]/target/config/comps/src:

avioVxwDrv.c

avioVxwDrv.h

Continue with *Common Build Components* below to build the CEI-x30 API. A sample Abaco Avionics common PPC VxBus Gen 1 Kernel Configuration file setup is shown below:



VxBus Gen 2 Driver Support (VxWorks 7)

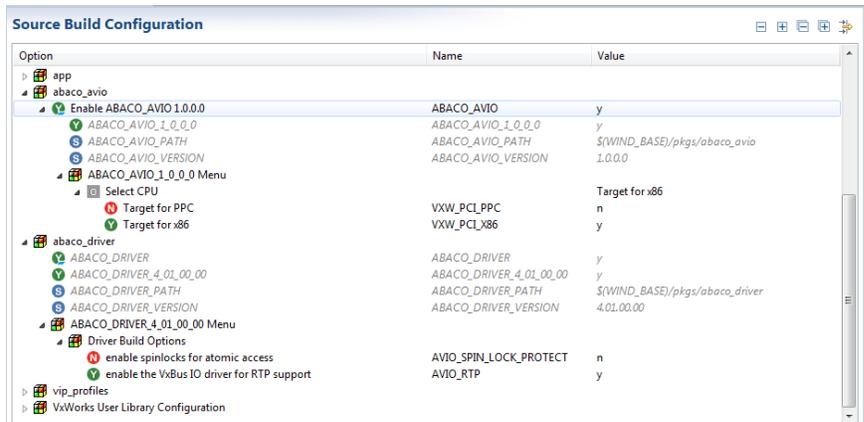
1. To install the generic Abaco Avionics common VxBus Gen2 driver support for VxWorks 7, assure Workbench is closed then copy the entire folders *abaco_avio* and *abaco_driver* (not just the contents) from the folder:

/Program Files/Condor Engineering/CEI-x30-SW/Source/VxWorks/VxBus Gen2 Driver

To the folder:

[Workbench_directory_path]/VxWorks-7/pkggs

2. After opening Workbench, the build options for the Abaco Systems Avionics driver(s) will be available in a VxWorks 7 Source Build project as shown:



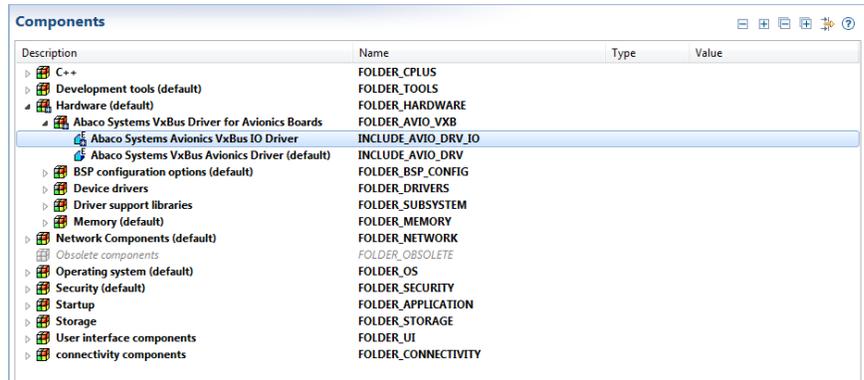
The Abaco Systems Avionics VxBus Gen 2 driver has the following build options.

- *Select CPU*: select either PPC or x86
- *Enable the VxBus IO Driver for RTP support*: only needed if requiring operation in an RTP environment.
- *Enable spinlocks for atomic access*: optional protection in ISR

If you plan to create a new VxWorks 7 Source Build Project, once installed the driver should be included in the build automatically; however, if you are adding the Abaco Avionics driver to an existing VxWorks 7 Source Build Project, you must perform the following steps:

- a. Double-click on your VSB Source Build Configuration entry to refresh the project.
- b. Expand the Build Targets selection
- c. Expand the Layers selection
- d. Right-click on the “ABACO_DRIVER_4_02_00_00” layer and select Build Layer
- e. Right-click on the AVIO_1_01_0_0 layer and select Build Layer
- f. The Abaco Systems VxBus Driver for Avionics Boards should now be available for selection within any VxWorks 7 Image Project based on this VSB.

- After the VxWorks 7 Source Build project has included the drivers a VxWorks 7 Build Image project that includes this VSB will have the drivers available to be included as shown:



- Right-click on the *Abaco Systems VxBus Driver for Avionics Boards* entry and select *Include*. This will include both drivers:
 - Abaco Systems VxBus Avionics Driver* (primary driver)
 - Abaco Systems Avionics VxBus IO Driver* (secondary driver for operating within an RTP environment)
- See the section, “Target-specific Compiler Directives” for information on selecting VxBus Gen2 RTP and DKM driver options, then continue to *Common Build Components* to build the kernel image.
- If you prefer to include the Abaco Systems Avionics driver(s) into your project via RPM installer, the files are provided in the distribution folder:

/Program Files/Condor Engineering/CEI-x30-SW/Source/VxWorks/VxBus Gen2 Driver/installer

Legacy PCI Driver Support

- To build the generic legacy (non-VxBus) Abaco Avionics common VxWorks PCI driver, copy the appropriate component installation file as specified in the following table:

Kernel Version	Platform	Component Installation File
VxWorks 6.0 - 6.5	x86	51_AVIO_x86_RTP_6x_PCI.cdf
	PPC	51_AVIO_PPC_RTP_6x_PCI.cdf
VxWorks 6.6 - 6.9	x86	51_AVIO_x86_RTP_66_PCI.cdf
	PPC	51_AVIO_PPC_RTP_6x_PCI.cdf

Table 2. VxWorks PCI Driver Component Definition Files

from the respective processor-specific folder in the *VxWorks Legacy PCI Driver/Component Installation File* folder located beneath:

/Program Files/Condor Engineering/CEI-x30-SW/Source/VxWorks

To

[*Workbench_directory_path*]/target/config/comps/vxWorks

- Copy the following source files from the *VxWorks Legacy PCI Driver* folder beneath the folder:

/Program Files/Condor Engineering/CEI-x30-SW/Source/VxWorks

And the file *cei_types.h* from the distribution Include folder, to either folder

[*Workbench_directory_path*]/target/config/<BSP Folder> or
[*Workbench_directory_path*]/target/config/comps/src:

CondorVxWRTPDrv.c
CondorVxWRTPDrv.h
avio_ioctl.h
lowlevel.h
target_defines.h

Common Build Components

- You may choose to include the CEI-x30-SW API source files in the BSP kernel source folder, create a new project folder for the CEI-x30-SW API and application development, or reference the distribution installation Source and Include folders. If you are not using the distribution folder, copy the following API source files from the folder /Program Files/Condor Engineering/CEI-x30-SW/Source and VxWorks folder beneath, to the folder of your choice:

cdev_api.c	cdev_vxw.c
cdev_api_a717.c	cdev_api_exp_rx.c
cdev_api_exp_tx.c	cdev_api_intrpt.c
cdev_api_irig.c	cdev_api_legacy_api.c
cdev_api_plx_pgm.c	cdev_api_rx_filter.c
cdev_api_sched.c	cdev_api_utility.c

See the section titled *Defining Custom Content in Your API Build* for a discussion on how to customize the CEI-x30 API for your target application. The use of selective API content can affect which source files you include in the API build.

- If you choose not to reference the Include folder in your compilation Include Path, copy the following C header files from the folder /Program Files/Condor Engineering/CEI-x30-SW/Include and driver source folder to that same folder, with some files dependent on the driver method used:

ar_error.h	cdev_api.h	cdev_fw.h
cdev_glb.h	cdev_hw.h	fpga430.h
fpga430A.h	fpga530.h	fpga630.h
fpga830.h	fpga830a.h	fpga830rx.h

```
fpga830x820.h fpga_ec.h fpgaA30.h
fpgax30n.h    cei_types.h avioVxwDrv.h
target_defines.h lowlevel.h
```

See the section titled *CDEV_FW.H - Firmware Load Files* for a discussion on how to customize the specific firmware load files included with the CEI-x30 API for your target application. The use of selective firmware for your specific board can drastically affect the size of static data allocation in your API build.

3. Open the workspace containing your VxWorks target image project and access the Kernel Configuration setup for the VxWorks image.
4. Beneath the hardware component, right-click the “*Abaco Systems VxBus Driver for Avionics Boards*” (or “*Abaco Systems PCI Avionics Products*”) component and select Include (quick include).
5. Modify the default values for any definitions as required for your target system. Examples of such modifications for the legacy PCI driver include:
 - For x86/Pentium kernel images, change the default state of “*Define PCI Compile for PowerPC*” to FALSE.
 - To modify the maximum number of overall Abaco Avionics boards supported in a system using the legacy PCI driver, change the value of “*Defines the maximum number of devices*” to the desired number, up to a maximum number of 16.
6. If you choose to build the API outside of the BSP source folder, you will have to manually define the directive VXW_PCI_X86 for an x86/Pentium target C source compilation/build or the directive VXW_PCI_PPC for a PowerPC target C source compilation/build, (normally defined in the configuration definition file).
7. Once you have your VxWorks kernel image built and running on your target with your board installed, open a shell to the target and invoke the function *avioDeviceShow*. This routine lists all detected Abaco Avionics products in the discovered “Device ID” order that should be referenced from your application for the respective boards.

See the section, “Target-specific Compiler Directives” for more information on the various ways to customize the CEI-x30-SW API source code compilation for your target BSP.

Using the Sample Program

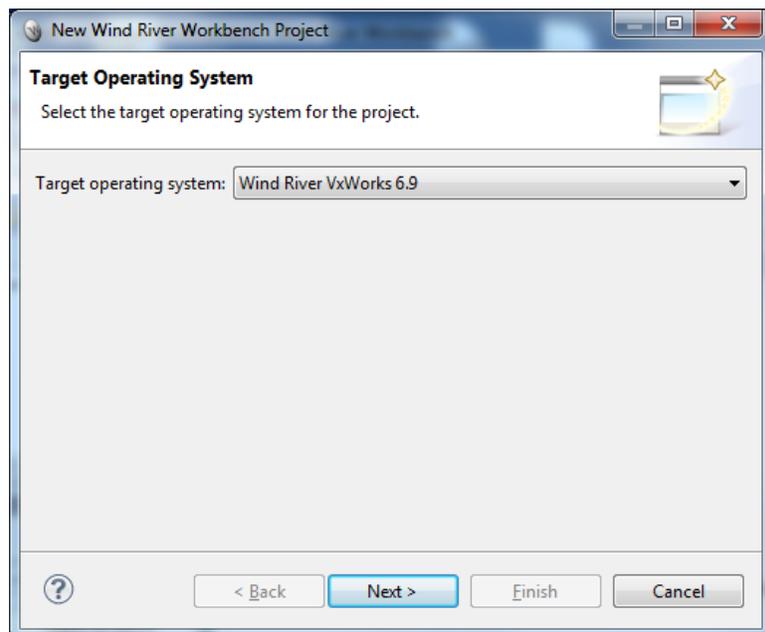
The CEI-x30-SW API distribution includes an example program named TST_CNFG.C. The source code is located within the Windows distribution Source folder or on the distribution disk in the CEI-x30-SW/Source folder. You can use this program to test your VxWorks installation, as it simply executes an internal wrap on all receiver-

transmitter channel pairs. You can also use TST_CNFG.C as a guide for programming with the CEI-x30-SW API.

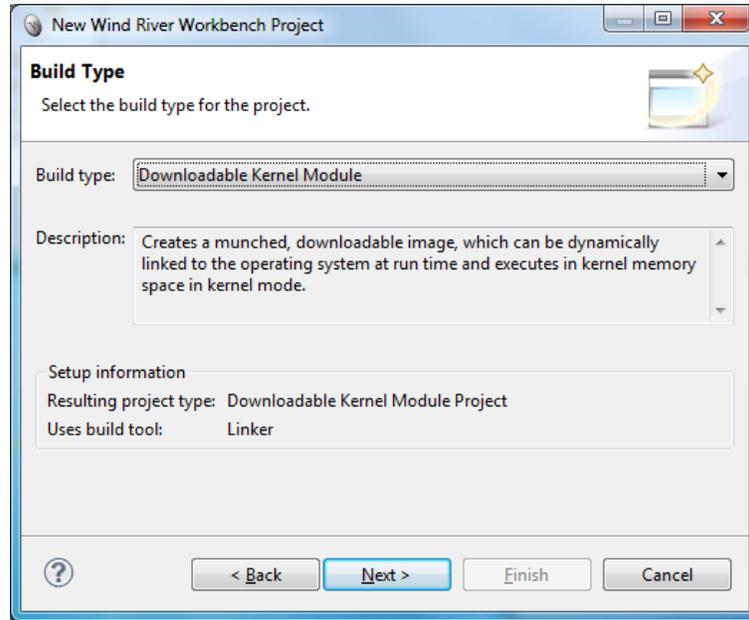
Building the API and Sample Program with Workbench

The CEI-x30-SW API and sample program can be built in the image, or as a downloadable object. The following steps explain how to build the API and sample program together as a downloadable object with Workbench 3 for a VxWorks 6.9 PowerPC target supporting the VxBus Gen1 driver, but can be easily adapted to Workbench 4/VxWorks 7 or Tornado/VxWorks 5.5, as well as other build environments.

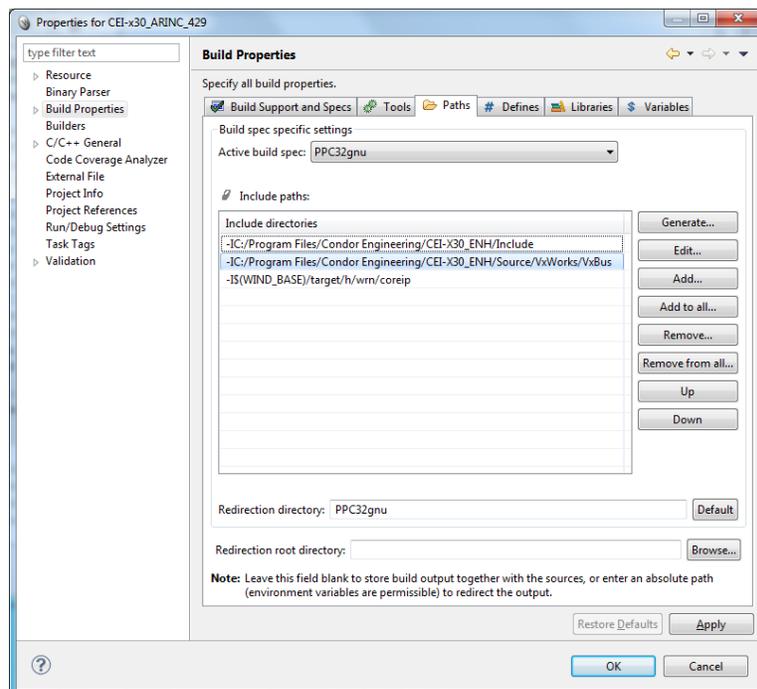
1. From the File pulldown, select New->Wind River Workbench Project... to initiate a new project for your downloadable application. Specify your target operating system, then select **Next**.



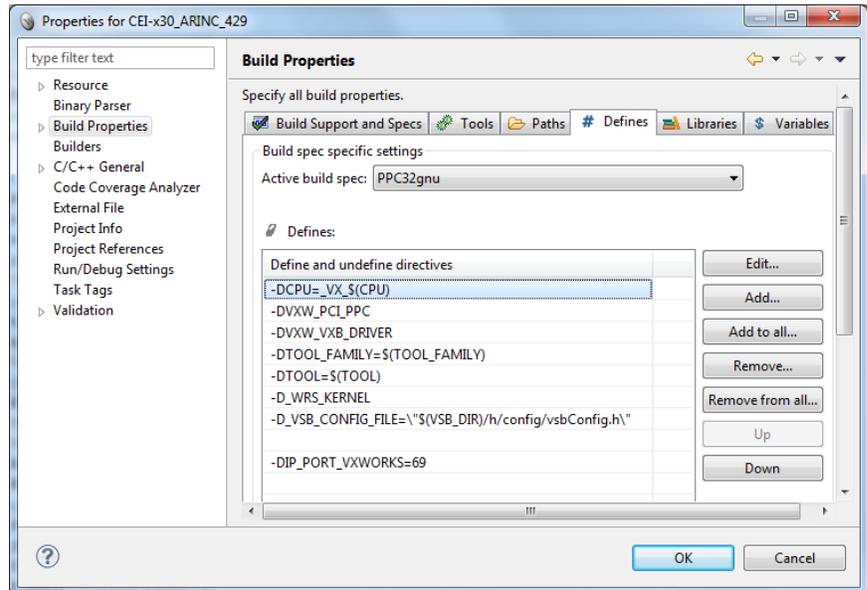
2. For initial testing of the board installation, select the build type "Downloadable Kernel Module". Click **Next**.



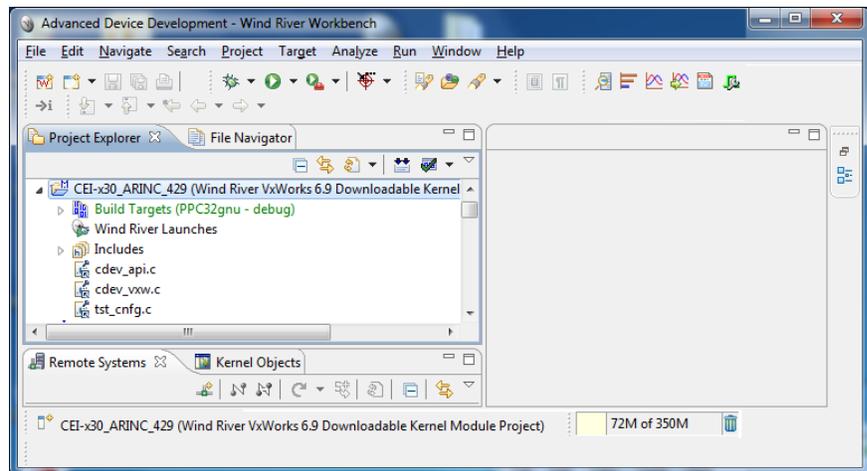
3. Enter your project name. “CEI-x30_ARINC_429” was used for this example. Click **Finish**.
4. Right-click the new project and select *Properties*.
5. Click on *Build Properties* and select the *Paths* tab.
6. Add paths to the CEI-x30-SW distribution’s Include and VxBus folders, then click on **Apply**.



- Click on the Defines tab, and add defines for VXW_PCI_PPC and VXW_VXB_DRIVER. Click on **Apply**, then **OK**.



- Open a view of the CEI-x30-SW/Source folder in the Windows Explorer. Drag the files cdev_api.c, cdev_vxw.c, and tst_cfg.c to the top level project, so they are included as follows:



- Right-click the build specification and select **Build Project**.
- Assuming you have already connected to the target via target server, in the expanded project under the Project Explorer right-click on the **Wind River Launches** item and select **Download->VxWorks Kernel Task** to download the build file you created.
- Open a host shell to the target and invoke the application by typing **wrap** at the shell prompt.

This program executes an internal wrap test on all available channels and notifies you of success or failure. If the program reports success on all channels tested, you are ready to use your new board.

Target-specific Compiler Directives

The CEI-x30 API accounts for specific target requirements using compilation directives. There are a few directives that may be required for the driver and board-specific VxWorks support provided with your target. For VxBus2, RTP and DKM driver support is implemented via the exclusive compiler directives *VXW_VXB_DRIVER* and *VXW_DRIVER_OPTION*. In addition to VxBus support, there are also differing methods for Legacy PCIbus driver mapping to a CEI-x30 board's PCI memory regions, *sysMmuMapAdd*, *sysPciMemToLocalAdrs* and *sysBusToLocalAdrs*. Two alternatives to the standard *taskDelay* method to pause execution are provided for select board support packages, *sysUsDelay* and *sysMsDelay*. You should determine the specific requirements for your target BSP and take the appropriate action prior to building the CEI-x30 API into your system.

The following table contains compiler directives that are defined to include both general and specific features required for compiling for various VxWorks target BSPs. The source of the directive may reside in the component installation file or the include file *target_defines.h*, while some are required to be defined in the build project.

Parameter	Description	Target	Options
Common Avionics API and Driver Specific Parameters and Directives			
VXW_VXB_DRIVER	When this directive is defined in the kernel image build project, it designates the use of the VxWorks VxBus DKM device driver in place of the legacy PCI driver. When this directive is defined in a VxWorks 7 VxBus Gen2 VIP project, the driver build is specific to a DKM kernel build. This option should be used exclusive of the <i>VXW_DRIVER_OPTION</i> parameter.	Both VxBus drivers for VxWorks 6.x and VxWorks 7	N/A
VXW_DRIVER_OPTION	When this directive is defined in a VxWorks 7 VxBus Gen2 VIP project, the driver build is specific to an RTP project. This option should be used exclusive of the <i>VXW_VXB_DRIVER</i> parameter.	VxBus Gen2 driver for VxWorks 7	N/A
VXW_PCI_PPC	When this directive is present or defined, it designates the target processor architecture as either PowerPC or x86.	All Kernel Versions	TRUE (PPC) FALSE (x86)
VXW_PCI_X86	Used to enable x86 target processor specific interrupt processing.	Legacy PCI driver for VxWorks 6.x	TRUE (x86) FALSE (N/A)
AVIO_DEBUG	Used to enable console printout of debug information during driver initialization.	VxBus Gen1 driver for VxWorks 6.7 through 6.9	TRUE FALSE (default)
vwxdebug	Used to enable console printout of debug	Legacy PCI driver	TRUE

Parameter	Description	Target	Options
	information during driver initialization.	for VxWorks 6.x	FALSE (default)
VXW_X86_MAP_ADD	Adds the board's PCI memory to <i>sysPhysMemDesc</i> via invocation of <i>sysMmuMapAdd</i> . Optional for an x86 target processor	Legacy PCI driver for VxWorks 6.0 to 6.5	TRUE FALSE (default)
SPIN_LOCK_PROTECT	When this directive is present or defined TRUE, the respective driver includes spinlock atomic access protection in interrupt service routine processing.	VxBus driver and Legacy PCI driver for VxWorks 6.x and 7	TRUE FALSE (default)
IRQ_OFFSET	This directive is defined in the respective CDF file for the Legacy PCI driver, and is a value added to the base IRQ Offset value '0' to represent the value of interrupt vector assigned to the Avionics board, typically used in an invocation of <i>pciIntConnect</i> or <i>intConnect</i> both with and without use of the macro <i>INUM_TO_IVEC</i> .	Legacy PCI driver for VxWorks 6.x	Refer to your target BSP documentation 0 (default)
VXW_PCI_INT_CON	When this directive is present or defined TRUE, the interrupt connect method <i>pciIntConnect</i> is used in place of <i>intConnect</i> with PowerPC target processors.	Legacy PCI driver for VxWorks 6.x	N/A
VXW_INT_LINE_CON	This directive defines interrupt processing based on a computed interrupt level instead of the IRQ assigned to the board, defined via invocation of <i>intConnect</i> . The default interrupt level 1 value is 25.	Legacy PCI driver for VxWorks 6.x	N/A
default	If neither directive <i>VXW_PCI_INT_CON</i> or <i>VXW_INT_LINE_CON</i> are defined, the interrupt processing based on an interrupt level that is the IRQ value, and can be offset by the value <i>IRQ_OFFSET</i> as defined in the CDF file.	Legacy PCI driver for VxWorks 6.x	N/A
CEI-x30 API Specific Parameters and Directives			
FLASH_BASED_TARGET	This directive is intended for use when building for the RAR-XMC, RAR-PCIE, RAR-MPCIE and RAR15-XMC products. When defined the API build excludes the firmware load modules and PLX programming routines for the PCI products.	All Kernel Versions	N/A
NON_INTEL_WORD_ORDER	This directive is intended for use when building for a Big Endian mapped target (typical for PowerPC and usually handled via <i>target_defines.h</i>).	All Kernel Versions	N/A

Linux Installation

Overview

CEI-x30-SW provides support for all CEI-x30 products under most Linux Kernel 2.4, 2.6, 3.x, 4.x and 5.x revisions. The install process builds the API as a shared library and installs the driver as a module. Application programs link with the shared library to access the respective device. Up to eight boards can be installed under supported Linux distributions.

Refer to the files *Linux_support.txt* and *Linux_install.txt* located in the Linux distribution file for the latest information on installing and building the common driver along with the CEI-x30 API and example program.

Software Installation

The Linux installation process requires your CEI-x30 hardware be installed prior to execution of the installation:

1. Turn OFF your computer system.
2. Ground yourself before handling the board. All hardware devices are static sensitive.
3. Insert the board into any respective slot for your form-factor. If possible, secure it in place.

Install the software as follows:

1. You must log on as "root" (you may use "su")
2. Copy the Linux distribution compressed tar file (*linux_x30_vnnn.tgz*) to the /root directory.
3. Uncompress and extract the installation file using the following:

```
tar -zxvf linux_x30_vnnn.tgz
```

After the tarball extraction completes, the directory structure as shown in Figure 1 will be created:

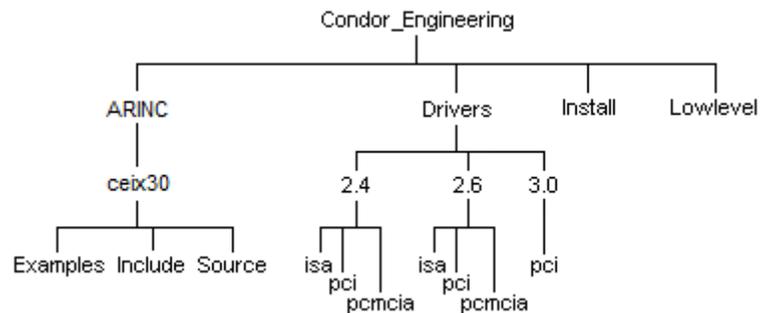


Figure 1. Linux Installation Directory Structure

Building Applications

Automatic Installation (Builds LSP and API)

Navigate to the Install directory and run the installation script by typing

```
./install
```

The PCI device driver builds and loads if the installation script detects a Abaco Systems avionics PCI board in the "procfs" file system. If the system does not have a "proc" file system, perform a manual install of the device driver.

These are the configuration arguments that are accepted by the "install" script:

1. To remove support for SYSFS (the SYS file system), include "no_sysfs" in the "./install" command line. If the system does not have the SYS file system or is based on kernel 2.6.10+ and does not agree with the "Proprietary/GPL" license, then SYSFS support must be removed.
2. To debug the kernel device driver(s), include the option "debug_drv=<DEBUG LEVEL>" in the "./install" command line. The debug statements will be printed out to the kernel message log. The <DEBUG LEVEL> provides increasing debug information with a range of "0" (none) to "3" (all).
3. To debug the low level library, include "debug_ll" in the "./install" command line. The debug statements will be printed to stdout.
4. To build only the device drivers and libraries, include "no_install" in the "./install" command line. If support for SYSFS has been removed,

the "ceidev.conf" will not be generated. Need to follow the instructions that are displayed on the screen during the installation. Refer to *Linux_install.txt* in your distribution, sub-section 4 in the section "Manual Install" concerning the "ceidev.conf" file.

5. To build the low-level and API libraries as 32-bit libraries to run in 32-bit emulation mode for 64-bit systems, include "32bit" in the "./install" command line.
6. To disable hardware interrupt support in the kernel 2.6 PCI/ISA drivers, include "no_hwint" in the "./install" command line.
7. To disable using POSIX RT signals in the kernel 2.6 PCI/ISA drivers, include "no_hwint_signal" in the "./install" command line.
8. To disable using a "wait queue" for the kernel 2.6/3.x/4.x PCI driver, include "no_hwint_waitqueue" in the "./install" command line.

The installation is finished. Check the "install" script output and the kernel message log for any errors. If there are no errors then the device driver(s) are loaded into the kernel, the low-level library is built as well as all detected API(s) distributions.

The installation installs the driver, builds and installs the API, (including the Linux common low-level interface), and compiles the example program. To test the installation, navigate to the Examples directory and execute the *tst_cnfg* application.

Manual Installation

Refer to the file *Linux_install.txt* in your distribution, section "Manual Install" concerning the manual installation of the Linux distribution and/or driver.

Linux Driver Operation

Linux compiles drivers as modules that dynamically link with the Linux kernel. The installation script automatically compiles the correct driver for the boards you are installing and the Linux kernel version. You can recompile using one of the Make files in the */Drivers/kernel/pci* directory, where *kernel* is either 2.4, 2.6 or 3.0. The module installation script *load_pci* is supplied in the Driver folder, which loads the module. You can manually load the driver by typing ".load_pci", and unload the driver by typing ".unload_pci".

Installation automatically invokes the driver load script. However, if you reboot the system you need to re-execute this script. You can put the script in the rc.local initialization file, which should automatically execute on power-up. The installation instructions located in the distribution file *Linux_Install.txt* explain how to manually execute the script.

Compiler Directives and Build Features

The following compiler directives should be used when building the CEI-x30 API for your target (when you are not using the supplied makefile):

- **MULTI_PROCESS_SUPPORT** should be defined for any application that will utilize multiple processes with a single board.
- **_LINUX_PPC_** selects the target compilation for a PowerPC host in the API source files.
- **_LINUX_X86_** selects the target compilation for an Intel host in the API source files.

When building the API for single process applications, thread protection is implemented using POSIX mutex functionality.

When building the API for multiple process applications, a System V global memory block is created to share API global status and control structures between processes and thread protection is implemented using semaphore functionality.

Troubleshooting

When installing any API distribution, you will need to be logged on as "root". Use the "su" command to gain "root" permissions. Root permissions are necessary when building device drivers and loading the modules into the kernel.

Useful Linux system utilities

dmesg: displays the kernel message log.

lsmod: displays the current modules loaded in the kernel.

lspci displays the PCI config space for all PCI devices

strace: displays the system calls that the driver or application calls.

ltrace: displays calls to the dynamic libraries that the application calls.

gdb: the GNU debugger.

modinfo: displays the module information for a driver.

Compilation Errors

If there are compilation errors, check that the path to the kernel headers is valid. If different than the default ("`/lib/modules/<KERNEL>/build`"), include the path in the applicable driver's makefile by including a "-I" with

the path. If there are system calls that cannot be resolved, check the `"/proc/kallsyms"` file to verify that they are compiled into the kernel.

Run-time Errors

Run-time error resolution may involve one or more of the following:

1. Check that the device driver, `uceipci`, is loaded with `"lsmod"`.
2. Examine the kernel message log for error output from the device driver `uceipci`. Use `"dmesg"` and/or look directed at the kernel message log located in `"/var/log/messages"`.
3. If there are version errors when loading the driver, the driver's version string (magic) may not coincide with current running kernel. Use `"modinfo"` to get the driver's magic number. Refer to the `"/usr/src/linux/makefile"` and `"/usr/src/linux/.config"`.
4. To determine where an error may be occurring in the application or API libraries use `"gdb"`. Make sure when compiling to provide the `"-g"` to GCC.
5. If a device driver fails to unload with `"modprobe"`, use `"rmmod"`.
6. If loading the 2.6 or 3.x/4.x/5.x PCI device driver and receive errors indicating missing symbols with `"sysfs"` in the symbol name, then build the distribution without support for SYSFS.

Integrity® Support

Introduction

Green Hills Integrity® is a secure, high-reliability real-time operating system (RTOS) intended for use in mission critical systems. The CEI-x30-SW distribution supports multiple CEI-x30 devices with Integrity on PowerPC and Intel processors, using the standard CEI-x30 API C source files in conjunction with the supplied Integrity-specific driver interface and additional kernel C source file set.

Integrity is flexible in how it builds the kernel and application software. You can build a monolith containing the kernel, BSP, and application software, or you can build a separate kernel/BSP and the application as a Dynamic Download. This Integrity distribution supports either method. This distribution provides the Integrity PCI driver source file, API source files, and an example application source file. You must compile and link the API to form a static library, which can then be linked with your application to achieve support for your board.

Integrity Installation

There are two options for installing CEI-x30-SW support for use with Integrity. If you are running the Multi IDE on a Windows system, you can install the source code using the Windows installation and select the *Source installation only for VxWorks/Integrity* option at the Target Installation selection prompt. You can also copy the desired folders directly from the Installation CD-ROM. The installation contains the Abaco Systems Avionics Integrity PCI driver, the source code for the API and driver interface, the example application source, and documentation.

After installing your board, you need to copy the PCI device driver source file into the Integrity BSP project for your target systems and rebuild the kernel. The Abaco Systems Avionics Integrity driver works with most

PowerPC and Intel BSPs. You can then build your own static library and example application projects.

Integrity PCI Driver Installation

You must install the PCI device driver as part of the BSP project in the default.gpj. The driver is a C file named `cei_int_pci_drv.c` that is BSP independent. Add that file *into* the libbsp.gpj project. Figure 2 shows the driver file installed in a Dy4 DMV181 project. To add the file, right-click the libbsp.gpj line and select the **Add File Into libbsp.gpj** option.

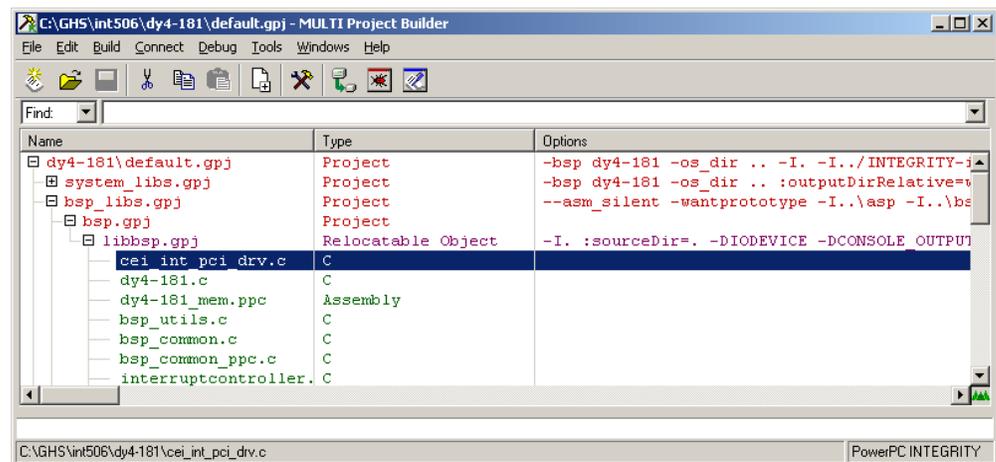


Figure 2. Integrity PCI Driver Installation

Building the CEI-x30 API with Multi

The CEI-x30-SW distribution contains all of the source code required to build a static CEI-x30 API library for use with your application. Build your static library with the supplied source and include files via the following procedure. The source files required are:

```

cdev_api.c           cdev_int.c
cdev_api_a717.c     cdev_api_exp_rx.c
cdev_api_exp_tx.c   cdev_api_intrpt.c
cdev_api_irig.c     cdev_api_legacy_api.c
cdev_api_plx_pgm.c  cdev_api_rx_filter.c
cdev_api_sched.c    cdev_api_utility.c
mem_integrity.c

```

See the section titled *Defining Custom Content in Your API Build* for a discussion on how to customize the CEI-x30 API for your target application. The use of selective API content can affect which source files you include in the API build.

The following list shows the include files (.h) needed to build the CEI-x30 API library.

```

ar_error.h      cdev_api.h      cdev_fw.h
cdev_glb.h      cdev_hw.h      fpga830.h
fpga830a.h      fpga630.h      fpga530.h
fpga430.h      fpga830rx.h    fpgax30n.h
cei_types.h     lowlevel.h     lowlevel.h
target_defines.h

```

See the section titled *CDEV_FW.H - Firmware Load Files* for a discussion on how to customize the specific firmware load files included with the CEI-x30 API for your target application. The use of selective firmware for your specific board can drastically affect the size of static data allocation in your API build.

Compiler Directives

The following compiler directives should be used when building the CEI-x30 API for your target:

- **INTEGRITY_POSIX** should be defined for any Integrity project using POSIX support, which can be any build other than for an Integrity 178B host.
- **INTEGRITY_PCI_PPC** selects the Integrity target compilation for a PowerPC host in the API source files.
- **INTEGRITY_PCI_X86** selects the Integrity target compilation for an Intel host in the API source files.
- **PPC_SYNC** introduces a processor pipeline flush operation during execution of the CEI-x30 API SRAM memory test, required if the SRAM memory test fails when the API is built without this directed defined.
- **FLASH_BASED_TARGET** intended for use when building for the RAR-XMC and RAR15-XMC products, when defined the build excludes the firmware load modules and PLX programming routines for the PCI products.

Monolith Image versus Dynamic Download

During development, if building your application and CEI-x30 API library as a separate **virtual AddressSpace project** to be deployed via Dynamic Download, no further directives are required. The Integrity dynamic loader will determine the virtual addresses of your CEI-x30 board memory regions at load time and pass those addresses to the API during the open session resource acquisition process.

If you are building your application and/or CEI-x30 API library with the kernel as a single image via **Monolith Integrity Application**

Project, the symbol **GHS_KERNEL** must be defined in the compilation of the file `mem_integrity.c`.

The compiler define for `NON_INTEL_WORD_ORDER` is now provided via the file `TARGET_DEFINES.H`.

CEI-x30 API Project Setup

Select a stand-alone project for your host architecture (generic PowerPC or Intel) and select a processor option matching your system.

For Project Type, select Library (empty). You can then add the C source files to the project and add the path to the include files. See Figure 3 for a sample CEI-x30 Integrity API library project for a PowerPC host.

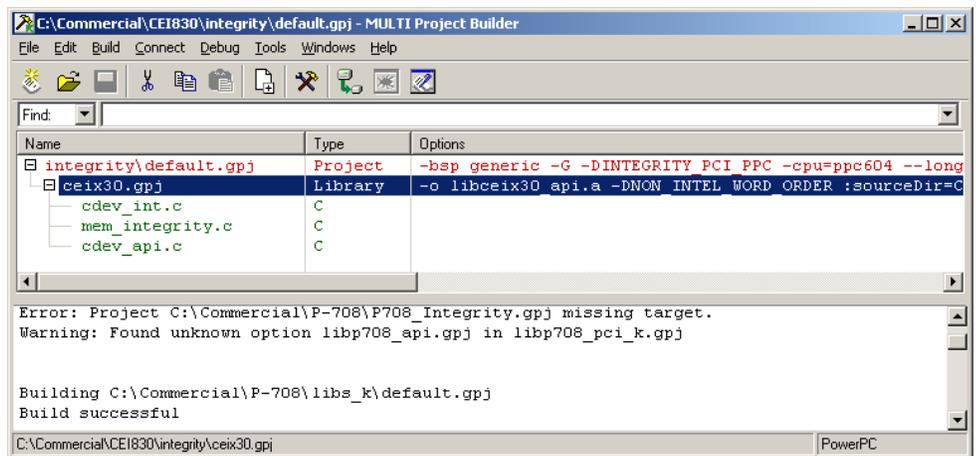


Figure 3. Example CEI-x30 Integrity API Library Project Setup

If you are building a library to run in a Monolith, you also need to define the symbol `GHS_KERNEL` under *Define Preprocessor Symbol*. You can name the output library to anything and use that library name to link with your application(s).

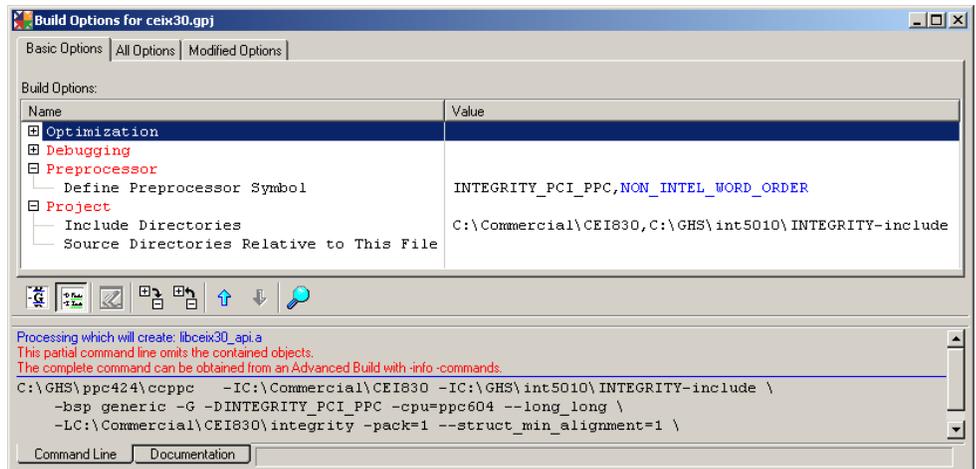


Figure 4. Example CEI-x30 Integrity Library Project Options

Building Integrity Applications

Once the PCI device driver has been built as part of the kernel and the CEI-x30 API static library build is complete, you are ready to build your INTEGRITY application program.

Figure 5 below shows a typical Dynamic Download project using the CEI-x30 API library.

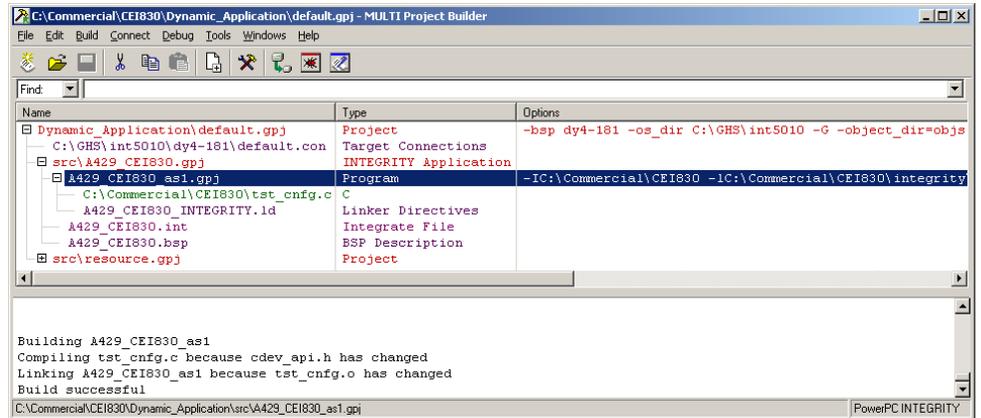


Figure 5. Example CEI-x30 Integrity Application Project Setup

Build the application using the following steps:

1. Define the host processor-specific preprocessor symbol INTERGITY_PCI_PPC or INTERGITY_PCI_X86.
2. Link with your CEI-x30 API static library, and include libposix.a and libsocket.a. You should review the Integrity POSIX chapter to make sure this POSIX option meets your application’s needs.

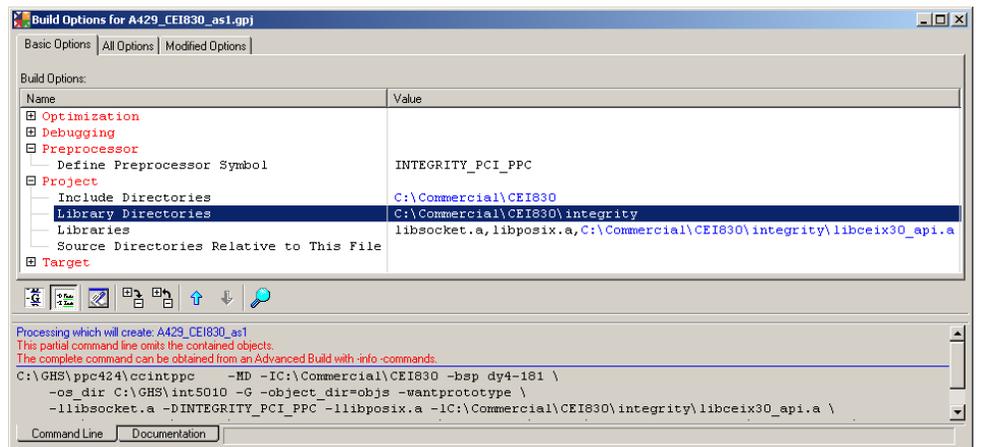


Figure 6. Example CEI-x30 INTEGRITY Application Project Option

3. Add sufficient MemoryPoolSize to create POSIX threads.

The example below uses 0x1000000, but your application may need more. Add the MemoryPoolSize entry between the Filename and Language entries in the Integrate file options.

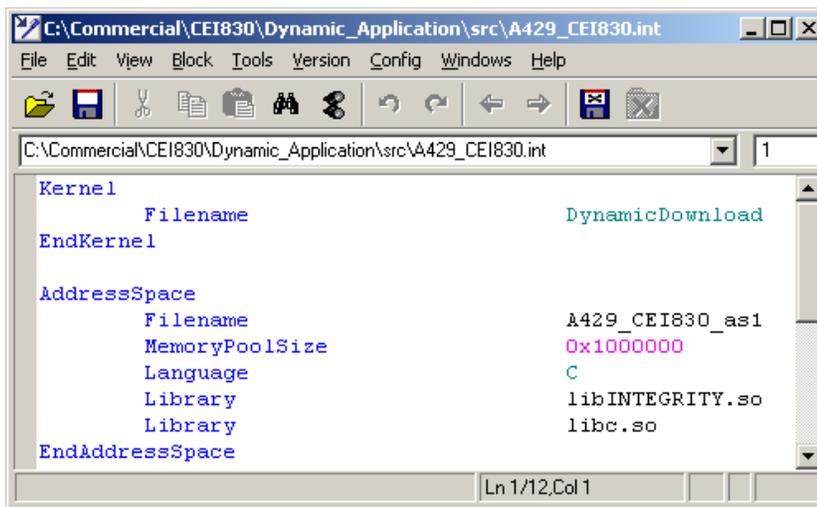


Figure 7. Adding a MemoryPoolSize Entry

4. Build the project, then download and run your application. If you desire the application to execute upon download, modify the value for the **DefaultStartIt** attribute to be “true” as follows:

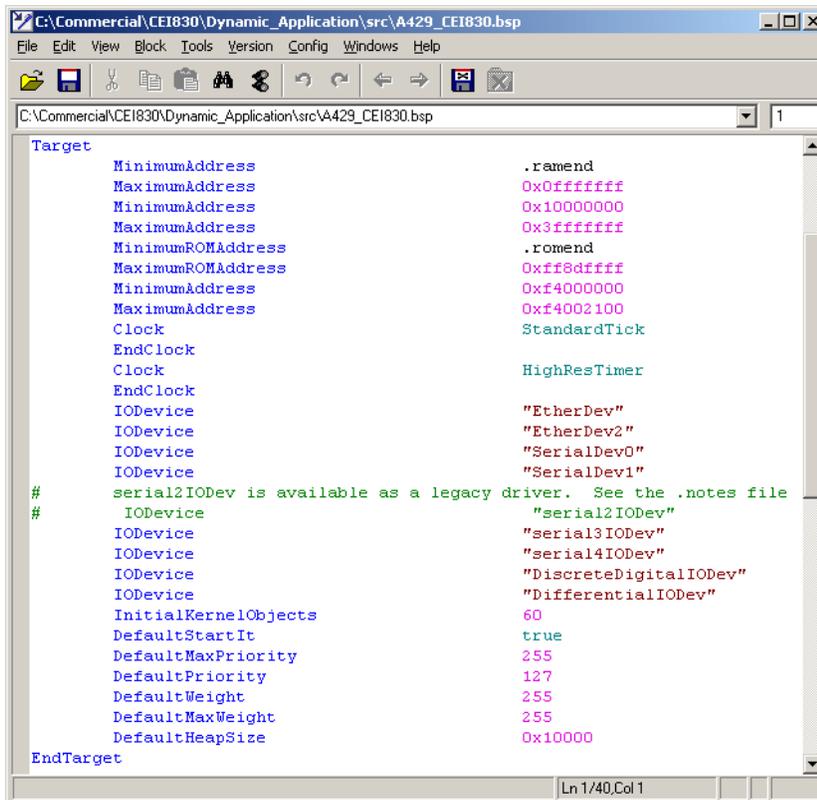


Figure 8. Modifying the Value for the DefaultStartIt Attribute

BusTools/ARINC™ Data Bus Analyzer

General Information

BusTools/ARINC™ is an optional ARINC 429 analysis and simulation utility which runs under Windows. It enhances the utility of an underlying ARINC 429 interface board by expanding your scope of control and by providing additional instrumentation and analytical tools. Additionally, BusTools/ARINC™ provides support for devices configured with ARINC 561, 717, or Commercial Standard Digital Bus (CSDB) channels.

BusTools/ARINC™ supports usage of up to four boards at the same time or independently and allows simultaneous control of all channels on each board.

Its data logging function streams data to disk or memory and replays it in a time-sequenced display. It provides multiple buffering mechanisms, including real-time display of data in engineering units. Strings of outgoing messages are generated, repeated, or automatically stepped through a sequence. Strings of incoming messages are filtered and captured for current or future analysis. A database of standard ARINC 429 translations is included. Translation among binary, hexadecimal, and engineering units is provided, as is a powerful user-defined label facility.

BusTools/ARINC Demo Software

A free demo version of BusTools/ARINC™ is available on our web site at 'www.abaco.com/products/bt-arinc-bustools-software-analyzer'. The demo software operates over a simulated ARINC 429 interface board, but is otherwise identical to the full version.

Application Development with CEI-x30-SW

Overview

The CEI-x30-SW software distribution contains all of the Application Programming Interface (API) and example source files, and additional features necessary to support CEI-x30 board installation and application development under the most common desktop and embedded programming environments (Windows, Linux, Integrity, and VxWorks).

The example application programs are written in C and delivered in a generic ANSI C compiler-compatible format. The API routines can be called from other languages by adhering to the procedures defined in the applicable documentation.

When installed under Windows, this distribution includes the necessary device driver files and Windows libraries required for application execution on the host system. The Windows distribution should be installed on any Windows-based system in which a CEI-x30 board will be used.

Windows Libraries

For the CEI-x30-SW supported products, separate 32-bit and 64-bit Windows API Libraries are provided. For Windows OS target implementation, all API function prototypes are declared “_stdcall”. The CEI-x30 API library included in the installation is referenced as:

- CDEV_API.LIB 32-bit Microsoft VS6.0 Library
- CDEV_API.DLL 32-bit Microsoft VS6.0 DLL
- CDEV_API64.LIB 64-bit Microsoft VS2008 Library
- CDEV_API64.DLL 64-bit Microsoft VS2008 DLL

Included with the installation are the Abaco Systems Common Low-level driver interface and installation verification libraries (not required for linking application programs):

- CEI_Install.DLL 32-bit Microsoft VS6.0 DLL
- CEI_Install64.DLL 64-bit Microsoft VS2008 DLL

All DLLs are installed in the Windows “System” folder. The exact folder name depends on the host version of Windows operating system. The 32-bit versions of these DLLs are typically installed in either ‘c:\winnt\system32’ or ‘c:\windows\system32’ under 32-bit Windows or ‘c:\windows\syswow64’ under 64-bit Windows. The 64-bit versions of these DLLs will be installed in the 64-bit Windows system folder (typically ‘c:\windows\system32’ under 64-bit Windows).

Data Types, Constants, and API Routine Prototypes

The C header file CDEV_API.H is the only header file required to be included in application C/C++ source. It either contains or includes the other header files that contain all data type, constant, and prototype definitions required for application development in the C/C++ programming language.

Time-tag Structure Definition

The following API routines use the AR_TIMETAG_TYPE data structure definition when providing a timer or time-tag reference, or as an initial value for the internal timer reset:

- AR_GET_TIME
- AR_SET_TIME
- AR_GETNEXT_XT
- AR_GETWORD_XT
- AR_GET_DATA_XT
- AR_CONVERT_1553_TIME_TO_STRING
- AR_CONVERT_TIME_TO_STRING

Under the Windows and VxWorks operating systems, the AR_TIMETAG_TYPE data structure and pAR_TIMETAG_TYPE pointer types used by these routines are defined to use 64-bit integer values, as follows:

timeTagFormat __int64 or long long

As an input to the AR_GET*_XT routines, this structure member specifies the desired format for the respective invocation,

overriding the global time-tag mode setup for the respective board. As an output from all routines, this structure member indicates the format (size and resolution) of the corresponding *timeTag* structure member. Valid values for this structure member are:

AR_TIMETAG_EXT_IRIG_64BIT	0
AR_TIMETAG_INT_USEC_64BIT	1
AR_TIMETAG_HOST_USEC_64BIT	2
AR_TIMETAG_INT_20USEC_32BIT	3
AR_TIMETAG_INT_MSEC_32BIT	4
AR_TIMETAG_SYNC_1553_CH1	11
AR_TIMETAG_SYNC_1553_CH2	12
AR_TIMETAG_SYNC_1553_CH3	13
AR_TIMETAG_SYNC_1553_CH4	14
AR_TIMETAG_USE_PROGRAMMED_MODE	99

timeTag	__int64 or long long	The timer-referenced time-tag, formatted as specified in the <i>timeTagFormat</i> structure member.
referenceTimeTag	__int64 or long long	The original 64-bit, one microsecond CEI-x30 board timer/time-stamp value reference for the time value supplied in the timeTag member.

Setting the Device Time

When assigning an initial time reference, the host application may choose to set either the device 1 microsecond timer or the IRIG generator timer via invocation of AR_SET_TIME.

When AR_SET_TIME is invoked with an AR_TIMETAG_TYPE data structure parameter *timeTagFormat* member defined to be AR_TIMETAG_EXT_IRIG_64BIT, the format of the *timeTag* member is defined as a 30-bit entity of BCD-like values using the following format:

29-28	27-24	23-20	19-18	17-14	13-11	10-7	6-4	3-0
hundreds of days	tens of days	days	tens of hours	hours	tens of minutes	minutes	tens of seconds	seconds

When AR_SET_TIME is invoked with a *timeTagFormat* member defined to be AR_TIMETAG_INT_USEC_64BIT, the *timeTag* member is referenced as a 64-bit 1 microsecond timer value.

Return Status Values

The following return status values are used by the CEI-x30 API routines. They are defined in the C header file CDEV_API.H and are used in the following context:

C Constant	Value	Constant Definition
ARS_FAILURE	-1	Requested operation failed
ARS_NODATA	0	No data was detected or received
ARS_NORMAL	1	Normal successful completion
ARS_GOTDATA	4	Data was received
ARS_BAD_MESSAGE	5	Receipt of an invalid ARINC 429 message was detected
ARS_RX_BUFFER_OVERRUN	6	Receive buffer overrun detected
ARS_INVHARCMD	1002	Invalid chn cfg in XML/Text file
ARS_INVHARVAL	1003	Invalid configuration value
ARS_XMITOVRFLO	1004	Transmit buffer overflow
ARS_INVBOARD	1005	Invalid board argument
ARS_NOSYNC	1006	Transmit buffer flush failed
ARS_BADLOAD	1007	Firmware download failed
ARS_MEMWRERR	1013	SRAM memory test error
ARS_INVARG	1019	General invalid argument value
ARS_DRIVERFAIL	1021	Driver failed to install or uninstall the ISR
ARS_WINRTFAIL	1022	Legacy driver open failure
ARS_CHAN_TIMEOUT	1023	Channel timeout in receive function
ARS_NO_HW_SUPRT	1024	Function not supported by specified hardware

C Constant	Value	Constant Definition
ARS_BAD_STATIC	1027	Register write/read/verify failure
ARS_HW_CONSISTENCY	1029	Device is not programmed for Enhanced Firmware operations
ARS_HW_DETECT	1030	Session opened but a CEI-x30 board was not detected
ARS_WRAP_DATA_FAIL	1031	BIT wrap test data read-back fail
ARS_WRAP_FLUSH_FAIL	1035	BIT cannot execute external wrap test due to unknown external data reception
ARS_WRAP_DROP_FAIL	1036	BIT wrap test data not received
ARS_INT_ISR	1037	Driver failed to install or uninstall API interrupt support
ARS_BOARD_MUTEX	1038	API routine failed to acquire or release a board lock mechanism
ARS_NO_OS_SUPPORT	1041	There is no operating system support for the requested feature
ARS_NO_INT_SUPPORT	1043	Interrupt handling not supported
ARS_NO_INT_ENABLED	1044	Interrupt handling not enabled
ARS_ERR_SH_MEM_OBJ	1050	API failed to allocate a shared object (semaphore or mutex)
ARS_ERR_SH_MEM_MAP	1051	API failed to allocate a shared memory region (multi-process)
ARS_FW_NOT_SUPPORTED	1052	The firmware programmed on the board is not compatible with the API version in use.
ARS_RX_BIT_CMD_ERROR	1100	Rx BIT CMD read-back failure
ARS_RX_BIT_CHnn_ERROR	1101-1132	Receive channel “nn” BIT failure detected, 01-32

Programming with the CEI-x30 API Interface

Following the outline below, you can easily incorporate the CEI-x30 API into your application.

1. For any Windows or Linux application incorporating multiple threads or processes, invoke the AR_SET_PRELOAD_CONFIG routine to prepare the API for a multi-threaded or multi-process operating environment.

For any VxWorks or Integrity application incorporating multiple threads or tasks, invoke the AR_SET_MULTITHREAD_PROTECT routine after invoking AR_OPEN instead.

2. For your application to interface to any CEI-x30 supported products the device must first be initialized. Invoke the AR_OPEN routine with the respective device number.

For Windows the device number is assigned during installation of CEI-x30-SW. For all other operating systems the device number is dependent on the PCI(e) controller bus/slot query order.

3. Assign general board level configuration settings along with the characteristics of the individual transmit and receive channels, if the default configuration is not appropriate. This is performed with multiple invocations of AR_SET_DEVICE_CONFIG.
4. Perform receiver buffer mode selection based on individual channel/protocol usage via the routine AR_SET_DEVICE_CONFIG. Using buffered mode for multiple channels of the same protocol provides channel-specific access to received data, where merged mode provides for single receive channel access to selected channel data.
5. Once channel configuration is complete, invoke AR_GO to initiate data processing.
6. Then invoke AR_PUTWORD and AR_GETWORD to send and receive single ARINC 429 messages, respectively. See the respective API routines summary section for *transmission* and *reception* routines for a broader selection of support for these operations.
7. When communication is complete, invoke AR_STOP to suspend active data processing. Subsequently, you could invoke AR_GO again to restart message processing.
8. On termination of the application, invoke AR_CLOSE to release all resources acquired during initialization. It is very important that all applications invoke AR_CLOSE upon termination; otherwise, the operating system does not release the memory acquired when the API was initialized.

The example wrap program source code, contained in TST_CNFG.C, is supplied with your installation. This program demonstrates the use of the API for the ARINC 429 and equivalent protocols.

When calling the utility routines that return a status value, it is important to verify the returned status indicates success; otherwise, the application may not be aware that an important function may have failed to fulfill a requested operation.

Example Routines in C – Summary

Example applications demonstrating various CEI-x30 API features are provided in the C programming language, as described in the following paragraphs.

Tst_cnfg.c

The example source file TST_CNFG.C is included with your installation. To access this example executable under the Windows operating system:

1. Click Start, and then Programs.
2. Select Abaco CEI-x30-SW and then Test Configuration.

Within TST_CNFG.C are application-style routines demonstrating use of the API routines for the ARINC 429 protocol:

test_basic_arinc_429	An internal wrap test designed to demonstrate ARINC 429 API usage. This routine enables internal wrap on all 429 receive channels. It also assigns a bus speed of 12.5kbps and ODD parity to both transmit and receive channels. Ten ARINC 429 messages are sent on each transmit channel and proper reception verified on the respective receive channel.
demo_advanced_arinc_429	A demonstration of the following advanced features available with CEI-x30 products: Transmit Message Scheduling Enhanced Label/SDI/ESSM Data Filtering Snapshot Message Data Acquisition Enhanced Time-tag Reset and Conversion IRIG Time-tag Selection (if installed on hardware)
demo_discrete_io_features	A demonstration on the use of the Discrete I/O Channels and the respective API routines
demo_irig_features	A demonstration of IRIG features requiring an external IRIG connection: IRIG DAC Threshold Adjustment IRIG Bias (Offset) Time Assignment IRIG Validity Determination IRIG Time Conversion and Display
test_arinc_717	An internal wrap test designed to demonstrate ARINC 717 protocol support. This routine enables internal wrap on the ARINC 717 receive channel. It also assigns a bus speed 768bps, a sub-frame size of 64 words, and a BPRZ selection to the ARINC 717 transmit and receive channels. A frame consisting of a data pattern incrementing from \$01 to \$FF and sync words of \$123,

	\$224, \$325, and \$426 is transmitted and proper reception verified.
demo_pci_interrupts	This routine demonstrates how to setup a custom interrupt service routine, setup Label Filter Table triggers generating interrupt events, enabling interrupts, and retrieving interrupt event data from the queue.
custom_interrupt_handler	This routine is the custom interrupt service routine assigned within the example routine demo_pci_interrupts.
demo_programmable_channels	Demonstration of how to define software programmable/shared channels as transmitters or receivers.

Config_from_file.c

This application-level source code demonstrates the channel configuration and scheduled message definition from files features available with the CEI-x30-SW distribution. Two files are available for each selected file type, XML and text, one for channel configuration of a two-channel board and the other defining four scheduled messages on each of two transmit channels. The four data files used with this application are:

- 8 msg def.txt
- 8 msg def.xml
- 2 chan cfg.txt
- 2 chan cfg.xml

Multiprocess_test.c

Provisions for simultaneous multiple process access to a CEI-x30 board are supported under the Windows and Linux operating systems. This feature is implemented in the standard API with minimal requirements on the application developer. The example program contained in the source file multiprocess_test.c describes how one method of multi-process application may be implemented.

This example application is based on separate processes, one for ARINC 429 scheduled message transmission on multiple transmit channels, and others for ARINC 429 message reception on individual receive channels. Regardless of the process invocation type, setup for the multiple process application is performed via invocation of the API routine AR_SET_PRELOAD_CONFIG, prior to invocation of AR_OPEN.

A single "transmit" process should be launched first, with subsequent invocation of one or more "receive" processes. The "transmit" process loads the board using AR_OPEN, configure board and transmit channel specific parameters using AR_SET_DEVICE_CONFIG, and activate data processing using AR_GO. After the "transmit" process invokes AR_GO, it is permissible to launch one or more "receive" processes. A "receive" process first attaches to the board using AR_OPEN, then execute operations strictly confined to the particular channel to which that process is associated (in this case AR_SET_DEVICE_CONFIG and AR_GETWORDT). When finished, each "receive" process invokes AR_CLOSE and terminates. The "transmit" process should remain running until all "receive" processes have terminated; upon termination the "transmit" process invokes AR_STOP and AR_CLOSE.

While the previously discussed application API invocation order is recommended, it is not strictly required. You may actually invoke a "receive" process first and terminate it last; however, multi-process applications should rely on a primary board-control process as the focus for board initialization, BIT functionality, and control of h/w message processing.

C# Support

The CEI-x30 API is an unmanaged DLL, not built using the .NET framework; however, a managed DLL wrapper allows you to use the CEI-x30 API library in your application development. This reference solution consists of:

- A managed wrapper class written in C# that encapsulates the CEI-x30 API function prototypes, constants, and data types.
- A sample managed GUI written in C# that demonstrates use of the wrapper class to interact with a CEI-x30 product.

The managed wrapper class can be used with C#, VB.NET, or any of the managed languages .NET supports. This documentation assumes familiarity with Visual Studio 2008 or later, and creating and running .NET applications.

The Reference Solution

The reference solution is a Microsoft Visual Studio 2008 solution in the **CEI-x30-SW\Examples\C#\C# Wrapper Example** folder, named **ArincCsApplication.sln**. When you open this solution, you will see two projects in the solution explorer window.

- **ArincCsApp:** This project creates the C# application **ArincCsApp.exe**, which allows you to perform a simple single ARINC 429 channel internal or external wrap test or monitor reception on a specific ARINC 429 receiver.

- **ArincCsWrapper:** This project creates the C# class library **ArincCsWrapper.dll**, which wraps the unmanaged CEI-x30 API functions, constants, and data types.

The Managed Wrapper Class

The *ArincAPI* namespace encapsulates all the API functions, Data Types, and Constant definitions. It is recommended you do not change this namespace name, as it identifies the wrapper and provides name separation when loaded into other projects.

The *API* static class contains managed entry points for most of the CEI-x30 API functions from the unmanaged CEI-x30 API library (with the prefix “ar_” omitted). The .NET interop environment requires that managed entry points be contained in a static class. This class is found in file *API.cs*, with supported functions listed below.

The *DataTypes* namespace contains managed equivalents of the structures required by the unmanaged CEI-x30 API library. The managed equivalents are implemented using C# structures, classes, and unions. This namespace is found in file *DataTypes.cs*.

The *Constants* static class contains managed definitions of the constants required by the unmanaged CEI-x30 API library. This class is found in file *Constants.cs*.

Adding the Managed Wrapper to an Existing .NET Application

First, it is suggested (but not required) that you add the C# project **ArincCsWrapper** to your existing .NET solution.

Then, in the Solution Explorer, right-click your project and select "Add Reference". If you added *ArincCsWrapper* to your solution, click the Projects tab and select *ArincCsWrapper*. Otherwise, select the Browse tab and locate *ArincCsWrapper.dll* on your disk.

At the top of each of your code pages, add the following lines:

```
using ArincAPI;  
using ArincAPI.DataTypes;  
using System.Runtime.InteropServices;
```

You may need to edit the file *API.cs* in the *ArincCsWrapper* project. At the top of this file is a statement that defines exactly where *cdev_api.dll* should be found.

You can now use the managed wrapper classes in your project.

C# Managed Wrapper Functions

The following is an alphabetical function list showing the link between the managed wrapper functions in the namespace/class ArincAPI.API, to the respective unmanaged CDEV_API.DLL function exports.

Assign_Scheduler_Start_Offsets	ar_assign_scheduler_start_offsets
Bypass_Wrap_Test	ar_bypass_wrap_test
Close_Session	ar_close
Clear_Rx_Count	ar_clr_rx_count
Define_Periodic_Message	ar_define_msg
Define_Msg_Block	ar_define_msg_block
Enh_Label_Filter	ar_enh_label_filter
Execute_Bit	ar_execute_bit
Get_Board_Name (see Note)	ar_get_boardname
Get_Board_Type	ar_get_boardtype
Get_Config	ar_get_config
Get_Data	ar_get_data
Get_Data_XT	ar_get_data_xt
Get_Device_Config	ar_get_device_config
Get_Error (see Note)	ar_get_error
Get_Filter_Table	ar_getfilter
Get_429_Message	ar_get_429_message
Get_573_Config	ar_get_573_config
Get_573_Frame	ar_get_573_frame
Get_Label_Filter	ar_get_label_filter
Get_Latest	ar_get_latest
Get_Latest_T	ar_get_latest_t
Get_Msg_Block	ar_getblock
Get_Msg_Block_T	ar_getblock_t
Get_Next	ar_getnext
Get_Next_T	ar_getnextt
Get_Next_XT	ar_getnext_xt
Get_Snap_Data	ar_get_snap_data
Get_RX_Channel_Status	ar_get_rx_channel_status
Get_RX_Count	ar_get_rx_count
Get_RX_Status	ar_get_status
Get_RX_Storage_Mode	ar_get_storage_mode
Get_Time	ar_get_time
Get_Transmitter_Mode	ar_get_transmitter_mode

Get_Word	ar_getword
Get_Word_T	ar_getwordt
Get_Word_XT	ar_getword_xt
Go	ar_go
Interrupt_Buffer_Read	ar_hw_interrupt_buffer_read
Interrupt_Queue_Read	ar_interrupt_queue_read
Label_Filter	ar_label_filter
Modify_Msg	ar_modify_msg
Modify_Msg_Block	ar_modify_msg_block
Num_RX_Chans	ar_num_rchans
Num_TX_Chans	ar_num_xchans
Open_Session	ar_open
Put_Block	ar_putblock
Put_Block_Multi_Chan	ar_putblock_multi_chan
Put_Filter_Table	ar_putfilter
Put_429_Message	ar_put_429_message
Put_573_Frame	ar_put_573_frame
Put_Word	ar_putword
Query_Device	ar_query_device
Read_Scheduled_Msg_Block	ar_read_scheduled_msg_block
Read_Message_Schedule_Table_Entry	ar_read_message_schedule_table_entry
Reset_Device	ar_reset
Reset_Timer_Count	ar_reset_timercnt
Set_Config	ar_set_config
Set_Device_Config	ar_set_device_config
Set_573_Config	ar_set_573_config
Set_Multi_Thread_Protect	ar_set_multithread_protect
Set_ISR_Function	ar_set_isr_function
Set_Preload_Config	ar_set_preload_config
Set_Raw_Mode	ar_set_raw_mode
Set_Storage_Mode	ar_set_storage_mode
Set_Time	ar_set_time
Set_Timer_Rate	ar_set_timerrate
Set_Transmitter_Mode	ar_set_transmitter_mode
Sleep	ar_sleep
Stop	ar_stop
Update_Message_Block	ar_update_message_block
Transmit_Sync	ar_xmit_sync

Note:

Dynamic memory allocation issues exist under 64-bit Windows when a C DLL function that returns a string. See the routines `ar_get_error_as_managed_string` and `ar_get_boardname_as_managed_string` in the C# source file `ArincFunctions.cs` for the managed-string solution for these API routines.

Visual Basic and VB.NET Support

The CEI-x30-SW distribution contains sample Visual Studio 2008 .NET solutions providing support for the Visual Basic and C# programming languages. These solutions are located in folders beneath the `Examples\C#` folder within the CEI-x30-SW software distribution.

Beneath the first folder “.Net Console Example”, the first sub-folder `CEIx30NetClass` contains a solution called `x30Wrap.sln`, which builds a VB wrapper class library over the standard API library `cdev_api.dll`, called **`CeIx30ClassLib.dll`**. Within this library is the class “Arinc”, containing many of the necessary constants, data types, and function references to support application development with the CEI-x30 API. The second folder `Arinc429Example`, contains a solution called `Arinc429Example.sln`, which is a very basic C# console application based on the source file `CEIx30Class.cs`, and demonstrating how to use the “Arinc” class for .NET application development.

Visual Basic Support

A text file, `CDEV_API_VB.TXT`, is provided to aid the Visual Basic programmer in using the `CDEV_API DLL` in the `Examples\VB` folder of the software distribution. This text file contains the Function Declaration and Global Constant statements required to interface to `CDEV_API.DLL`. You can manually copy and paste text from this file to your project, or you can use the Microsoft API Text Viewer utility included with Visual Basic. For more information on the API Text Viewer, consult Microsoft Visual Basic documentation.

The `CDEV_API_VB.TXT` is designed for use with any 32-bit version of Visual Basic; however, the VB example and API interface are recommended for use with Visual Basic version 6.0 or later.

Working with Unsigned Integers in Visual Basic

Visual Basic doesn't support unsigned integers. Since the CEI-x30 API library uses unsigned integers for some function parameters, problems can arise when attempting to set values in the upper half of the range.

Example

When a CEI-x30 API function uses an argument of C type *unsigned short* the equivalent type is *integer* in Visual Basic or *short* in VB.Net. All are 16-bit values but the Visual Basic variable has a range of -32768 to 32767. The C argument has a range of 0 to 65535. A Visual Basic error is generated if an *integer* type is set to value greater than 32767.

Solutions

There are two solutions to this problem. The easiest is to set the value of variables directly in Hex. To set an integer variable to 65535 use: `myVariable = &HFFFF`. (note the &H syntax) . For setting long variables use the ending "&" as in `myvar = &H12&`.

The second solution is to convert the desired unsigned value to the signed equivalent. This can be accomplished in a small utility function:

```
Function u_conv (unsigned as Long) as Integer
    Dim signed as Integer
    If unsigned > 32767 then
        signed = unsigned - 65536
    Else
        signed = unsigned
    End If
    u_conv = signed
End Function
```

When using returned values from CEI-x30 API functions the opposite conversion can be made. Often, the returned values from CEI-x30 API functions simply need to be compared to the predefined values.

AutoConfig ARINC Configuration File Generator

The capability to create channel configuration and scheduled message definition files, and initialize a CEI-x30 board using the corresponding VIs is available with this distribution. See the "AutoConfig ARINC User's Manual" supplied in the folder `\Utilities\AutoConfig ARINC` within this distribution for a detailed description on the use of this application.

Dealing with Complex Message Scheduler Transmit Scenarios

Whether a transmit channel is operating at 100Kbps or 12.5Kbps, any scheduled message scenario that introduces as little as 50% bus loading is susceptible to message rate skew. The ability to assign proper offsets to the scheduled message rate definitions can become cumbersome in such situations. For these cases two options are provided:

The Start Offset Assistant utility will accept an input text file containing a C-like message scheduler structure array layout with defined channel message scenarios and generate an output text file with assigned start offset values. The contents of the output text file can then be copied as a C data structure array into the application source, to be used with the API's message scheduler support routines. See the "Start Offset Assistant" User's Manual in the folder **C:\Users\Public\Documents\Condor Engineering\CEI-x30-SW\Utilities\Start Offset Assistant** for a detailed description on the use of this tool.

The CEI-x30 API also provides a utility routine supported under the Windows and Linux operating systems that will update the contents of the existing message scheduler table with start offset values calculated in a best attempt to avoid rate skew based on the programmed bus speed for individual transmit channels. The start offset values are computed based on the message count and rates defined per transmit channel. For more details, see the routine `AR_ASSIGN_SCHEDULER_START_OFFSETS` within this document.

Application Programming Interface

Overview

Abaco Systems supplies an extensive software Application Programming Interface (API) for the CEI-x30 family of ARINC products. API routines are supplied to setup the interface, configure channel attributes, and transmit and receive ARINC 429 and 717 messages.

CEI-x30 API Source Files

The CEI-x30 API is written in C and delivered in a generic ANSI C compiler-compatible format. The API source file set consists of the following files:

CDEV_API.C

This file contains a limited set of API routines providing access to the most used features and functionality for an embedded application environment.

CDEV_API_A717.C

This file contains the set of API routines specifically related to the ARINC 717 protocol features and functionality provided with the CEI-x30 products.

CDEV_API_CFG_FILE.C

This file contains the set of API routines specifically related to the configuration file accessible board configuration and scheduled message setup features.

CDEV_API_EXP_RX.C

This file contains the set of API routines specifically related to both the expanded and legacy message receive functions.

CDEV_API_EXP_TX.C

This file contains the set of API routines specifically related to both the expanded and legacy message transmit functions.

CDEV_API_INTRPT.C

This file contains the set of API routines specifically related to the PCI Interrupt features and functionality provided with the CEI-x30 products.

CDEV_API_IRIG.C

This file contains the set of private API utility routines specifically related to the IRIG generator and receiver features and functionality provided with the CEI-x30 products.

CDEV_API_LEGACY_API.C

This file contains the set of API routines specifically provided to support application migration from older ARINC products and API's.

CDEV_API_PLX_PGM.C

This file contains the set of private API utility routines specifically related programming the firmware on CEI-x30 products utilizing a PLX PCI interface design.

CDEV_API_RX_FILTER.C

This file contains the set of API routines specifically related to the Receive Label Filtering features and functionality provided with the CEI-x30 products.

CDEV_API_SCHED.C

This file contains the set of API routines specifically related to the onboard Transmit Message Scheduler features and functionality provided with the CEI-x30 products.

CDEV_API_UTILITY.C

This file contains the set of API routines specifically related to utilities not typically incorporated into embedded or simulation based applications.

CDEV_WIN.C

This file contains the C routines that interface directly with the Avionics common low-level driver interface library, CEI_INSTALL.LIB/DLL, supporting all Windows operating systems.

CDEV_VXW.C

This file contains the C routines that interface directly with the Avionics common VxWorks kernel driver (VxBus or Legacy PCI).

CDEV_LNX.C

This file contains the routines that interface directly with the Linux kernel driver provided with the CEI-x30 Linux distribution archive.

CDEV_INT.C

This file contains the routines that interface directly with the Avionics common Integrity PCI driver.

CDEV_LRT.C

This file contains the routines that interface directly with the LabVIEW Real-Time operating environment.

CEI-x30 API Header Files

The only C header file required to be included in application source is CDEV_API.H. This file and the remaining C header files utilized in the CEI-x30 API are described as follows:

CDEV_API.H

This header file contains the majority of the definition for API constants, data types, and function prototypes, and should be included in all C/C++ application source files that reference one or more CEI-x30 API utility routines.

CDEV_GLB.H

This header file contains the majority of the API internal global variables, internal definitions, and data structures.

CEIX30_TYPES.H

This header file contains all of the CEI-x30 API build-specific parameter and structure data types, based on the Abaco Systems Avionics product common types defined in CDEV_API.H; included in CDEV_API.H.

CEI_TYPES.H

This header file contains all of the Abaco Systems Avionics product common type defines for the various data types used with the respective operating system and compiler; included in CEIX30_TYPES.H.

CDEV_API_CFG.H

This header file contains #define declarations that affect the compilation (inclusion) of select routines and functionality in the C source files, and C function prototypes defined in CDEV_API.H and CDEV_GLB.H. It is specifically provided for building a custom API in an embedded target environment; included in CDEV_API.H.

AR_ERROR.H

This header file contains the error string constant definitions utilized by the API routine `AR_Get_Error`, describing each of the potential error codes returned by the CEI-x30 API utility routines.

CDEV_HW.H

This header file contains all of the API constants that define the hardware interface for the CEI-x30 architecture; included in `CDEV_API.H`.

CEI_INSTALL.H

This header file contains all of the Abaco Systems Avionics product common type defines for the Windows Common Driver Interface library.

CDEV_FW.H - Firmware Load Files

The header file `CDEV_FW.H` is included in the source file `CDEV_API.C`, containing the array declarations for all API-loaded CEI-x30 board firmware. The default compilation of `CDEV_API.C` includes the firmware load modules for the entire CEI-x30 product line. When the compiler directive `LABVIEW_RT` is defined, only the firmware for the CEI-830, RCEI-830A, R830RX, RCEI-530, and RAR-CPCI boards are included in the build, as these are the products currently supported with LabVIEW Real-Time. When the compiler directive `INTEGRITY_PCI_PPC` is defined, only the CEI-830, RCEI-830A, R830RX, RAR-CPCI, CEI-430, and RCEI-530 firmware is included in the build. The RAR-PCIE, RAR-MPCIE and RAR15-XMC combo-card firmware is loaded from Flash Memory, and is not subject to a required firmware header file.

If for any reason you wish to reduce the API library or object module size by omitting the firmware load modules for extraneous boards, you may replace the header file reference in the respective include statement(s) with the header file `FPGAX30N.H`. For example, to omit the RAR-CPCI firmware load module you would modify line 86 of the file `CDEV_FW.H` as follows:

```
static CEI_UINT32 const fpga_630[]={
    #include "fpgax30n.h"
};
```

The firmware header files are referenced as follows:

<code>FPGA830.H</code>	CEI-830 Firmware
<code>FPGA830A.H</code>	RCEI-830A Firmware
<code>FPGA830RX.H</code>	R830RX Firmware
<code>FPGA430.H</code>	CEI-430 Firmware

FPGA430A.H	CEI-430A Firmware
FPGA530.H	CEI-530 Firmware
FPGA630.H	RAR-CPCI Firmware
FPGAA30.H	AMC-A30 Firmware
FPGA_EC.H	RAR-EC Firmware
FPGA830X820.H	RCEI-830X820 Firmware
FPGAX30N.H	Two element array (empty f/w allocation)

Building the API for Embedded and Certified Systems

The CEI-x30 API file set contains provisions for limiting the scope of the API source included in a custom API build. Limiting API content in a build can drastically reduce effort involved in both deliverable code analysis and software validation, tasks typically encountered in embedded and certified systems.

Defining Custom Content in Your API Build

For any API build, the file CDEV_API.C is the basic API source file that must be included in a custom CEI-x30 API source build. It contains the rudimentary API routines for accessing basic board configuration and ARINC 429 message transmission and reception operations. This file may be modified, and content removed by the user; however, any functions removed should have the respective prototype removed from CDEV_API.H (and/or CDEV_GLB.H).

You may add API source code supporting the desired CEI-x30 features by selectively including any of the remaining C source files for the respective functionality in your API project (see section *CEI-x30 API C Source Files*). In conjunction with the source files included in your build, control of which API and local utility function prototypes are defined in CDEV_API.H and CDEV_GLB.H is provided in the file CDEV_API_CFG.H.

The relationship between the definitions in CDEV_API_CFG.H, the source files containing the referenced functions, the CDEV_API.H prototype definitions affected by those definitions, and a list of the respective functions, are all documented in the following table.

Feature/Functionality	CDEV_API_CFG.H Definition	Associated C Source File	API Functions Included and Prototypes Affected
Default CEI-x30 API Routine Set	N/A	cdev_api.c	ar_board_test ar_bypass_wrap_test ar_close ar_clr_rx_count

Feature/Functionality	CDEV_API_CFG.H Definition	Associated C Source File	API Functions Included and Prototypes Affected
			ar_execute_bit ar_get_device_config ar_get_error ar_get_rx_channel_status ar_get_rx_count ar_get_time ar_getblock_t ar_getword ar_getword_xt ar_get_latest_t ar_get_snap_data ar_go ar_initialize_api ar_initialize_device ar_num_rchans ar_num_xchans ar_open ar_putblock ar_putword ar_reset ar_set_device_config ar_set_multithread_protect ar_set_preload_config ar_set_storage_mode ar_set_time ar_sleep ar_stop
API Utility Routines	INCLUDE_API_UTILITIES	cdev_api_utility.c	ar_convert_time_to_string ar_convert_1553_time_to_string ar_get_base_addr ar_get_boardname ar_get_boardnameLV ar_get_boardtype ar_get_channel_index_info ar_get_config ar_get_storage_mode ar_query_device ar_version
ARINC 717 Support	INCLUDE_ARINC_717	cdev_api_a717.c	ar_get_573_config ar_get_573_frame ar_get_transmitter_mode ar_put_573_frame ar_set_573_config ar_set_transmitter_mode
PCI Interrupt Support	INCLUDE_INTERRUPTS	cdev_api_intrpt.c	ar_hw_interrupt_buffer_read ar_int_control ar_interrupt_queue_read ar_set_isr_function
IRIG Time Support	INCLUDE_IRIG	cdev_api_irig.c	API private routines only
PMC/PCI/Express Card and PC104/Plus Support	INCLUDE_PLX_SUPPORT	cdev_api_plx_pgm.c	API private routines only

Feature/Functionality	CDEV_API_CFG.H Definition	Associated C Source File	API Functions Included and Prototypes Affected
Receive Label Filter Support	INCLUDE_LABEL_FILTER	cdev_api_rx_filter.c	ar_enh_label_filter ar_getfilter ar_get_label_filter ar_label_filter ar_putfilter
Transmit Message Scheduler Support	INCLUDE_MSG_SCHEDULER	cdev_api_sched.c	ar_assign_scheduler_start_offsets ar_define_msg ar_define_msg_block ar_define_msg_block_lv ar_modify_msg ar_modify_msg_block ar_modify_msg_block_lv ar_read_scheduled_msg_block ar_read_message_schedule_table_entry ar_update_message_block
Expanded and Legacy Message Reception and Retrieval Support	INCLUDE_EXPANDED_RX	cdev_api_exp_rx.c	ar_get_data ar_get_data_xt ar_get_429_message ar_get_latest ar_getblock ar_getnext ar_getnextt ar_getwordt ar_getnext_xt
Expanded Message Transmit Support	INCLUDE_EXPANDED_TX	cdev_api_exp_tx.c	ar_putblock_multi_chan ar_put_429_message
Configuration File Support	INCLUDE_CFG_FILE	cdev_api_cfg_file.c	ar_config_channels_from_txt_file ar_channel_configuration_from_xml_file ar_define_messages_from_txt_file ar_define_messages_from_xml_file
Legacy API Support	INCLUDE_LEGACY_API	cdev_api_legacy_api.c	ar_get_status ar_get_timercntl ar_get_timercnt ar_get_raw_mode ar_init_dual_port ar_init_slave ar_reset_timercnt ar_set_raw_mode ar_set_config ar_set_timerrate ar_wait ar_int_control ar_recreate_parity ar_reset_int ar_setinterrupts ar_setchparms ar_xmit_sync ar_dump_dp ar_force_version ar_formatarinclabel ar_formatbinarylabel ar_get_errorLV ar_get_harris

Feature/Functionality	CDEV_API_CFG.H Definition	Associated C Source File	API Functions Included and Prototypes Affected
			ar_int_set ar_int_slave ar_loadslv_init_disc ar_msg_control ar_putword2x16 ar_reformat ar_set_arinc_config ar_set_harris ar_timetag_control ar_unformat ar_flush_receiver

Table 3. CEI-x30 API C Source File Content for a Custom Build

Excluding Parameter Validation and Thread Protection

In many embedded and certified target systems, parameter validation and thread protection processing are not required or desired. Defining the compiler directive `TARGET_EMBEDDED` allows you to build the API source code such that it excludes most lines of code invoking parameter validation and all thread protection from compilation. This can optimize API code execution and reduce the deliverable code analysis and software validation effort.

Defining the Data Types Used in Your API Build

The default data types used for the CEI-x30 API function return parameters/status arguments match those types used in the legacy API's that preceded it (for ease of application transition to the latest ARINC products). These default data types and the mixed datatype usage that accompanies it do not meet the requirements of most code analysis tools, and tend to be less desirable for embedded and certified systems usage.

For this type of target build, provisions for defining the common CEI-x30 API data types within the file `CEIX30_TYPES.H` exists, in which most return parameter and API function argument data types can be defined as a mix of signed and unsigned 32-bit integer. The compiler directive `TARGET_OPTIMAL_TYPES` is the mechanism by which the alternate data type definitions are introduced into the API source.

Note:

If you build the CEI-x30 API with the directive `TARGET_OPTIMAL_TYPES` defined and use the CEI-x30 data type declarations in your application source, you must define this directive for your application source compilation.

API Data Types

The actual data types used in the CEI-x30 return parameters and function arguments are documented as follows, both for the default API build and the alternate API build (when building the API with the compiler directive `TARGET_OPTIMAL_TYPES` defined). The C header files in which these type definitions reside are `CEIX30_TYPES.H` and `CEI_TYPES.H`.

Data Type Definition	Default Data Types	
	TARGET_OPTIMAL_TYPES not defined	TARGET_OPTIMAL_TYPES defined
CDEV_API_RET_TYPE	CEI_INT16 signed short integer	CEI_INT32 signed integer
CDEV_BOARD_TYPE	CEI_INT16 signed short integer	CEI_UINT32 unsigned integer
CDEV_CHAN_TYPE	CEI_INT16 signed short integer	CEI_UINT32 unsigned integer
CDEV_PARM_USI_TYPE	CEI_UINT16 unsigned short integer	CEI_UINT32 unsigned integer
CDEV_PARM_SSI_TYPE	CEI_INT16 signed short integer	CEI_UINT32 unsigned integer
CDEV_PARM_SSI_PTR_TYPE	CEI_INT16 * signed short integer pointer	CEI_UINT32 * unsigned integer pointer
CDEV_PARM_SI_TYPE	CEI_INT32 signed integer	CEI_UINT32 unsigned integer
CDEV_PARM_SI_PTR_TYPE	CEI_INT32 * signed integer pointer	CEI_UINT32 * unsigned integer pointer
CDEV_PARM_VOID_PTR_TYPE	CEI_VOID * (pCEI_VOID) void pointer	CEI_UINT32 * (pCEI_UINT32) unsigned integer pointer
TIME_TAG_TYPE	CEI_UINT64 unsigned 64-bit integer (long long)	CEI_UINT64 unsigned 64-bit integer (long long)
pCEI_UINT32	unsigned integer pointer	unsigned integer pointer
pCEI_INT32	integer pointer	integer pointer
pCEI_CHAR	character pointer	character pointer
CEI_VOID	void	void

Table 4. CEI-x30 API C Data Type Definitions

API Routines - Summary

The routines provided in the API supporting the CEI-x30 device features are defined in the following pages, categorized and summarized:

Initialization and Control Routines

<code>ar_open</code>	The main initialization routine acquiring the necessary resources and initializing the CEI-x30 device.
<code>ar_board_test</code>	Verifies the CEI-x30 data processing capabilities via internal/external data wrap.
<code>ar_bypass_wrap_test</code>	Controls conditional execution of the ARINC 429 internal wrap test within <i>ar_initialize_device</i> .
<code>ar_initialize_device</code>	Initializes the CEI-x30 device to the default state.
<code>ar_loadslv</code>	A legacy routine replicating the <code>ar_open</code> routine processing.

Device Control Routines

<code>ar_go</code>	Enables CEI-x30 ARINC data processing.
<code>ar_reset</code>	Disables CEI-x30 ARINC data processing and initializes the device to the default state.
<code>ar_stop</code>	Disables CEI-x30 ARINC data processing.

Termination Routines

<code>ar_close</code>	Releases all resources for the specified device.
-----------------------	--

Receive/Transmit Channel-level Configuration Routines

<code>ar_channel_configuration_from_xml_file</code>	Configures channels based on settings supplied from an XML file.
<code>ar_config_channels_from_txt_file</code>	Configures channels based on settings supplied from a text file.

ar_set_device_config	As the main channel configuration routine, it assigns ARINC 429-specific transmitter and receiver channel configuration information.
ar_get_device_config	Retrieves the value of a bit field for I/O and ARINC 429 transmitter or receiver channel configuration registers.
ar_enh_label_filter	Assigns the enhanced label filter table definition for each CEI-x30 device receiver.
ar_get_config	Retrieves board-level configuration and API local attribute values.
ar_get_573_config	Retrieves the value of a bit field for an ARINC 573/717 transmitter or receiver channel configuration register.
ar_get_filter	Retrieves the specified label filter buffer entry from the enhanced label filter table.
ar_get_label_filter	Retrieves the active state of label filtering for a single label on all receivers.
ar_label_filter	Assigns ARINC 429 label values to be filtered by the specified receive channel.
ar_putfilter	Places the specified label filter buffer entry in the enhanced label filter table.
ar_set_config	Assigns board-level configuration and API local attribute values.
ar_set_573_config	Assigns ARINC 573/717 transmitter and receiver channel configuration information.

Device-level Configuration Routines

ar_get_storage_mode	Retrieves the API state of a device-generic receive data storage mode.
ar_set_raw_mode	Assigns both transmitter and receiver parity state on the specified ARINC 429 channel.
ar_set_storage_mode	Assigns the device-level receive data storage mode.

Receive Data Processing Routines

ar_get_429_message	Retrieves the next ARINC 429 message from a receive buffer. Optionally, it waits up to ½ second for data to become available.
--------------------	---

<code>ar_get_573_frame</code>	Retrieves a specified number of ARINC 573 data words from the receive buffer.
<code>ar_getnext</code>	Retrieves the next message from the specified receive buffer. It waits up to ½ second for data to become available.
<code>ar_getnextt</code>	Retrieves the next message and a 32-bit, 20 µsec time-tag from the specified receive buffer. It waits up to ½ second for data to become available.
<code>ar_getnext_xt</code>	Retrieves the next message with a 64-bit, user-programmable time-tag from the specified receive buffer. It waits up to ½ second for data to become available.
<code>ar_getword</code>	Retrieves the next message from the specified receive buffer.
<code>ar_getwordt</code>	Retrieves the next message and a 32-bit, 20 µsec time-tag from the specified receive buffer.
<code>ar_getwordt_xt</code>	Retrieves the next message from the specified receive buffer with a 64-bit, user-programmable time-tag.
<code>ar_get_data</code>	Retrieves the next available data and the 64-bit, 1 µsec time-tag from a receive buffer.
<code>ar_get_data_xt</code>	Retrieves the next available data from a receive buffer with a 64-bit, user-programmable time-tag.
<code>ar_getblock</code>	Retrieves all of the available ARINC 429 messages from the requested receive buffer with 32-bit time-tags.
<code>ar_getblock_t</code>	Retrieves all of the available ARINC 429 messages from the requested receive buffer, with 64-bit time-tags.
<code>ar_get_latest</code>	Retrieves the latest message from the snapshot buffer for the specified channel/label combination.
<code>ar_get_latest_t</code>	Retrieves the latest message and time-tag from the snapshot buffer for the specified channel/label combination.
<code>ar_get_snap_data</code>	Retrieves the latest message from the snapshot buffer for the specified channel/label/sdi combination.

Transmit Data Processing Routines

<code>ar_assign_scheduler_start_offsets</code>	Computes and applies best fit start offset values for each transmit channel message scenario defined in the onboard message scheduler table.
<code>ar_define_messages_from_txt_file</code>	Defines a set of scheduled ARINC 429 messages supplied from a text file.
<code>ar_define_messages_from_xml_file</code>	Defines a set of scheduled ARINC 429 messages supplied from an XML file.
<code>ar_define_msg</code>	Defines a scheduled ARINC 429 messages.
<code>ar_define_msg_block</code>	Defines a block of scheduled ARINC 429 messages.
<code>ar_modify_msg</code>	Modifies an existing ARINC 429 message already defined for periodic transmission.
<code>ar_modify_msg_block</code>	Modifies a block of ARINC 429 messages already defined for periodic transmission.
<code>ar_put_429_message</code>	Places a single message in the specified ARINC 429 transmit buffer.
<code>ar_put_573_frame</code>	Places a specified number of ARINC 573 data words in the transmit buffer.
<code>ar_putword</code>	Places a single message in the specified ARINC 429 transmit buffer.
<code>ar_putblock</code>	Places multiple messages in a single ARINC 429 transmit buffer.
<code>ar_putblock_multi_chan</code>	Places multiple messages in multiple ARINC 429 transmit buffers.
<code>ar_read_message_schedule_table_entry</code>	Returns the contents of the specified block of message scheduler entries
<code>ar_read_message_schedule_table_entry</code>	Returns the contents of the specified message scheduler entry

Timer-related Routines

<code>ar_get_time</code>	Retrieves the current hardware reference time based on the selected timer mode.
<code>ar_get_timercntl</code>	Retrieves the current value of the OS timer.

ar_reset_timercnt	Resets the internal timer/time-tag reference to zero.
ar_set_timerrate	Assigns the CEI-x30 compatible timer resolution for use with ar_get_time() and any ARINC 429 receive data routines returning 32-bit time-tag values.
ar_set_time	Sets the internal clock/timer or IRIG time generator to an application supplied value.

Information and Status Routines

ar_get_base_addr	Retrieves a pointer to the base address of the address space allocated to the specified device.
ar_get_boardname	Returns a string description of the specified device.
ar_get_boardtype	Retrieves the target device configuration.
ar_get_error	Retrieves a message string associated with a given error status code.
ar_get_rx_channel_status	Reports the current buffer state of the specified ARINC 429 receive channel.
ar_get_status	Retrieves the combined state of each receive FIFO status register Data Available bit.
ar_num_rchans	Retrieves the number of receive channels supplied by the CEI-x30 device.
ar_num_xchans	Retrieves the number of transmit channels supplied by the CEI-x30 device.

Utility Routines

ar_clr_rx_count	Resets the counter of received messages for the specified receive channel.
ar_convert_time_to_string	Converts a standard CEI-x30 64-bit time value to character string.
ar_convert_1553_time_to_string	Converts a multiprotocol, 1553 channel synchronized time value to character string.
ar_execute_bit	Verifies the CEI-x30 operational state through various data wrap and timer tests.

ar_get_rx_count	Returns the number of messages received for the specified receive channel.
ar_hw_interrupt_buffer_read	Returns the contents of the API maintained interrupt buffer entries.
ar_interrupt_queue_read	Provides host access to read the device interrupt queue.
ar_set_isr_function	Provides the method for the host application to define a custom interrupt service routine.
ar_set_multithread_protect	Enable/disable multithread access protection to all API routines accessing the hardware interface of the device.
ar_set_preload_config	Defines the process and thread setup for the calling application.
ar_sleep	Suspends the calling thread for a specified number of milliseconds.
ar_wait	Blocks execution of the calling thread for the specified number of seconds.
ar_version	Retrieves the current API software version for the CEI-x30 API, in a string format.

AR_ASSIGN_SCHEDULER_START_OFFSETS

Syntax CEI_INT32 ar_assign_scheduler_start_offsets (CDEV_BOARD_TYPE board)

Description This routine determines the appropriate start offset values for each transmit channel message scenario as a best estimate to avoid rate skew on the respective channel. It first reads all defined messages from the message scheduler table (any message having a non-zero rate attribute), determines the appropriate start offset value for all messages on a channel-by-channel basis, and then updates the respective start offset values in the table.

This routine should be called immediately following the last invocation of AR_DEFINE_MSG or AR_DEFINE_MSG_BLOCK, and must be called prior to the invocation of AR_GO, (I.E. the *Global Enable* must be disabled). The transmit channel bus speed for all channels referenced in the message scheduler table entries must also be assigned prior to calling this routine.

To assign start offset values to a scheduled message scenario prior to the invocation of AR_DEFINE_MSG or AR_DEFINE_MSG_BLOCK, see the Start Offset Assistant utility.

Notes: The start offset values defined by this routine do not account for bus timing issues and/or message rate skew due to bursts of aperiodic messages invoked by the host application.

All start offset values assigned assume a continuous transmission of all messages defined in the message scheduler table.

Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_FAILURE	Message processing is enabled on the board.

Arguments CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

AR_BOARD_TEST

Syntax

CDEV_API_RET_TYPE ar_board_test (CDEV_BOARD_TYPE board, CDEV_PARM_SSI_TYPE testType)

Description

This routine performs a single message internal or external wrap test on each matched ARINC 429 transmit/receive channel pair. On successful completion of the wrap test, the board is initialized to the default state via invocation of AR_INITIALIZE_DEVICE.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_INVBOARD	An uninitialized or invalid <i>board</i> value was provided.
ARS_MEMWRERR	The SRAM test write/read/verify failed.
ARS_WRAP_DROP_FAIL	ARINC 429 wrap test data missing.
ARS_WRAP_DATA_FAIL	ARINC 429 wrap test data pattern mismatch.
ARS_WRAP_FLUSH_FAIL	Unexpected data from an external source was received during wrap test execution.
ARS_XMITOVRFLO	A transmit buffer overrun occurred.

Arguments

CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

CDEV_PARM_SSI_TYPE testType (input) Type of test to execute. Valid values for this parameter are:

INTERNAL_WRAP (4)

EXTERNAL_WRAP (5)

AR_BYPASS_WRAP_TEST

Syntax

CDEV_API_RET_TYPE ar_bypass_wrap_test (CDEV_BOARD_TYPE board, CDEV_PARM_SSI_TYPE bypass)

Description

This routine defines an internal flag used to control the invocation of the AR_BOARD_TEST routine during execution of AR_OPEN. The routine AR_BOARD_TEST will perform a single-message internal wrap test for each matching ARINC 429 transmit/receive channel pair. The default state of this internal flag is ON, indicating no internal wrap test is executed during the board/API initialization process.

This routine should be invoked with the bypass parameter set to AR_OFF if you wish the API to perform an internal wrap test as part of the board/API initialization process. Note that if AR_OPEN is invoked with any ARINC 429 receive channel connected to an actively transmitting LRU, execution of AR_BOARD_TEST may return a false failure status indication.

Return Value

ARS_NORMAL Routine execution was successful.

ARS_INVBOARD An invalid *board* value was supplied.

Arguments

CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

CDEV_PARM_SSI_TYPE bypass (input) Bypass flag controlling execution of the internal wrap test when the routine AR_OPEN is invoked. Valid values for this parameter are:

AR_ON (7) bypass internal test

AR_OFF (8) execute internal test

AR_CLR_RX_COUNT

Syntax

CEI_VOID ar_clr_rx_count (CDEV_BOARD_TYPE board,
CDEV_CHAN_TYPE channel)

Description

This routine resets the API-tracked count of ARINC data words received by the specified channel to zero. The CEI-x30 device maintains a count of ARINC messages received over the interface for each channel since the device was initialized. When this routine is invoked, the API saves the current count, to be used as the most recent reset reference and subtracted from the device count when the count value is requested by AR_GET_RX_COUNT.

Return Value

None

Arguments

CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

CEI_UINT32 channel (input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.

AR_CLOSE

Syntax	CDEV_API_RET_TYPE ar_close (CDEV_BOARD_TYPE board)	
Description	This routine releases all resources acquired during the initialization of the specified device. Once this routine has been executed, invocation of other API routines results in the return of an invalid status.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_DRIVERFAIL	Windows device driver failed to close the session with the device.
	ARS_INT_ISR	Failed to relinquish the interrupt resources
	ARS_ERR_SH_MEM_OBJ	Failed to relinquish the shared memory resource in a multi-process application configuration
	ARS_FAILURE	Under Windows only, indicates a failure to access the low-level driver library.
Arguments	CDEV_BOARD_TYPE board	(input) Device to access. Valid range is 0-127.

AR_CHANNEL_CONFIGURATION_FROM_XML_FILE

Syntax

CDEV_API_RET_TYPE ar_channel_configuration_from_xml_file
(pCEI_CHAR pcCfgFileName, CDEV_BOARD_TYPE cintBrdIdx)

Description

This routine assigns the state of the following channel configuration attributes from the application supplied XML-based channel configuration file generated via the AutoConfig ARINC utility:

- Buffer Enable
- Bit Rate
- Parity Check/Generation
- Receive Buffering Mode
- Receive Internal Wrap
- Transmit Protocol Error Injection

See the AutoConfig ARINC User's Manual for a description of the acceptable Channel Configuration XML file format.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_BADLOAD	The specified file not found.
ARS_INVBOARD	An uninitialized board or invalid <i>cint16BrdIdx</i> value was provided.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_FAILURE	Invalid configuration file format.
ARS_INVHARVAL	Invalid configuration file channel value.
ARS_INVHARCMD	Invalid configuration file attribute entry.

Arguments

pCEI_CHAR pcCfgFileName (input) The complete path and file name defining the channel configuration file to be referenced by this routine. A valid file name suffix can only be “.xml” for a channel configuration XML file.

CDEV_BOARD_TYPE cintBrdIdx (input) Device to access. Valid range is 0-127.

AR_CONFIG_CHANNEL_FROM_TXT_FILE

Syntax

CDEV_API_RET_TYPE ar_config_channels_from_txt_file
(CDEV_BOARD_TYPE BoardIdx, pCEI_CHAR cfgFileName)

Description

This routine assigns the state of the following channel configuration attributes from the application supplied text-based channel configuration file generated via the AutoConfig ARINC utility:

- Buffer Enable
- Bit Rate
- Parity Check/Generation
- Receive Buffering Mode
- Receive Internal Wrap
- Transmit Protocol Error Injection

See the AutoConfig ARINC User's Manual for a description of the acceptable Channel Configuration text file format.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_BADLOAD	The specified file not found.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_FAILURE	Invalid configuration file designation or format.
ARS_INVHARCMD	Invalid configuration file attribute entry.

Arguments

CDEV_BOARD_TYPE BoardIdx (input) Device to access. Valid range is 0-127.

pCEI_CHAR cfgFileName (input) The complete path and file name defining the channel configuration file to be referenced by this routine. A valid file name suffix can only be ".txt" for a channel configuration text file.

AR_CONVERT_1553_TIME_TO_STRING

Syntax

CEI_VOID ar_convert_1553_time_to_string (CDEV_BOARD_TYPE board, CDEV_PARM_SSI_TYPE displayFormat, pAR_TIMETAG_TYPE timeIn, pCEI_CHAR timeString)

Description

This routine should be used for conversion of any ARINC 429 message time-stamp or board timer value with any multiprotocol board when the time-tag mode is set to AR_TIMETAG_SYNC_1553_CH n (1-4) via the API routine AR_SET_DEVICE_CONFIG with the option ARU_RX_TIMETAG_MODE.

This routine invokes the BusTools/1553-API time conversion routine *BusTools_TimeGetFmtString* to convert the MIL-STD-1553 channel-specific synchronized time value provided in the timeIn structure to a character string representation of date/time. The format of the conversion should be based on the active display format for the specified 1553 channel, acquired via invocation of the API routine AR_GET_DEVICE_CONFIG for the respective 1553 channel with an option of ARU_1553_TIME_TAG_DISPLAY. The format type returned from that invocation should be provided in the displayFormat parameter of this routine invocation. The supplied time format (LSB resolution) is a fixed one nanosecond resolution with this conversion routine.

Return Value

none.

Arguments

CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

CDEV_PARM_SSI_TYPE displayFormat (input) Format for returned string acquired as documented in the description above.

pAR_TIMETAG_TYPE timeIn (input) Source 64-bit time structure

pCEI_CHAR timeString (output) Pointer to destination text string

AR_CONVERT_TIME_TO_STRING

Syntax	CEI_VOID ar_convert_time_to_string (CDEV_BOARD_TYPE board, CDEV_PARM_SSI_TYPE displayFormat, pAR_TIMETAG_TYPE timeIn, pCEI_CHAR timeString)						
Description	This routine converts the time value provided in the <i>timeIn</i> structure to a character string representation of date/time, format based on what is specified via the <i>displayFormat</i> parameter. The supplied time format (LSB resolution) must be specified in the <i>timeIn</i> structure member <i>timeTagFormat</i> , representing the resolution of the respective <i>timeTag</i> member data.						
Return Value	none.						
Arguments	<p>CDEV_BOARD_TYPE board board (input) Device to access. Valid range is 0-127.</p> <p>CDEV_PARM_SSI_TYPE displayFormat (input) Format for returned string:</p> <table style="margin-left: 40px;"> <tr> <td>AR_TD_REL_MIDNIGHT</td> <td>Relative to Midnight Format and Full IRIG Format, defined as "(DDD)hh:mm:ss.uuuuuu"</td> </tr> <tr> <td>AR_TD_IRIG</td> <td>Full IRIG Format, defined as "(DDD)hh:mm:ss.uuuuuu"</td> </tr> <tr> <td>AR_TD_DATE</td> <td>Date Format defined as "(MM/DD)hh:mm:ss.uuuuuu"</td> </tr> </table> <p>pAR_TIMETAG_TYPE timeIn (input) Source 64-bit time structure</p> <p>pCEI_CHAR timeString (output) Pointer to destination text string</p>	AR_TD_REL_MIDNIGHT	Relative to Midnight Format and Full IRIG Format, defined as "(DDD)hh:mm:ss.uuuuuu"	AR_TD_IRIG	Full IRIG Format, defined as "(DDD)hh:mm:ss.uuuuuu"	AR_TD_DATE	Date Format defined as "(MM/DD)hh:mm:ss.uuuuuu"
AR_TD_REL_MIDNIGHT	Relative to Midnight Format and Full IRIG Format, defined as "(DDD)hh:mm:ss.uuuuuu"						
AR_TD_IRIG	Full IRIG Format, defined as "(DDD)hh:mm:ss.uuuuuu"						
AR_TD_DATE	Date Format defined as "(MM/DD)hh:mm:ss.uuuuuu"						

AR_DEFINE_MESSAGES_FROM_TXT_FILE

Syntax

CDEV_API_RET_TYPE ar_define_messages_from_txt_file
(pCEI_CHAR cfgFileName)

Description

This routine defines a series of ARINC 429 messages for periodic retransmission at the specified rate from the supplied text scheduled message definition file generated by the AutoConfig ARINC utility.

See the AutoConfig ARINC User's Manual for a description of the acceptable Scheduled Message Definition text file format.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_BADLOAD	The specified file not found.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided in the text file.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_FAILURE	The number of entries requested exceeds the available number of remaining table entries on a referenced board.
ARS_INVHARCMD	Invalid message scheduler attribute entry was detected in the text file.

Arguments

pCEI_CHAR cfgFileName (input) The complete path and file name defining the scheduled message definition file to be referenced by this routine. A valid file name suffix can only be ".txt" for a scheduled message definition text file.

AR_DEFINE_MESSAGES_FROM_XML_FILE

Syntax

CDEV_API_RET_TYPE ar_define_messages_from_xml_file
(pCEI_CHAR pcSMFileName)

Description

This routine defines a series of ARINC 429 messages for periodic retransmission at the specified rate from the supplied XML scheduled message definition file generated by the AutoConfig ARINC utility.

See the AutoConfig ARINC User's Manual for a description of the acceptable Scheduled Message Definition XML file format.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_BADLOAD	The specified file not found.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided in the text file.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_FAILURE	The number of entries requested exceeds the available number of remaining table entries on a referenced board.
ARS_INVHARCMD	Invalid message scheduler attribute entry was detected in the XML file.

Arguments

pCEI_CHAR pcSMFileName (input) The complete path and file name defining the scheduled message definition file to be referenced by this routine. A valid file name suffix can only be ".xml" for a scheduled message definition XML file.

AR_DEFINE_MSG

Syntax

CDEV_API_RET_TYPE ar_define_msg (CDEV_BOARD_TYPE board, CDEV_CHAN_TYPE channel, CDEV_PARM_SSI_TYPE rate, CDEV_PARM_USI_TYPE start, CDEV_PARM_SI_TYPE data)

Description

This routine defines a 32-bit ARINC 429 message for periodic retransmission at the specified rate. Once defined, the message rate, content, or assigned channel may be altered through AR_MODIFY_MSG.

Return Value

Any positive value between 0 and 2047 is the unique message scheduler table entry index assigned to this message.

ARS_FAILURE Indicates the routine encountered an uninitialized board, an invalid board/channel parameter value, or a full message scheduler table.

Arguments

CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

CDEV_CHAN_TYPE channel (input) Channel on which to transmit this message. The valid range is 0 to one less than the number of installed transmit channels.

CDEV_PARM_SSI_TYPE rate (input) Periodic transmission rate, defined in milliseconds by default. For backward compatibility to the CEI-x20 tick-timer message rate method, when AR_SET_TIMERRATE has been executed to simulate the CEI-x20 tick-timer resolution assignment within the CEI-x30 API, the rate and start parameters is scaled to the specified tick-timer resolution.

CDEV_PARM_USI_TYPE start (input) Offset, (in milliseconds), from the start of CEI-x30 device message processing at which this message will begin its initial periodic transmission.

CDEV_PARM_SI_TYPE data (input) The 32-bit ARINC 429 message to transmit.

AR_DEFINE_MSG_BLOCK

Syntax	CDEV_API_RET_TYPE ar_define_msg_block (CEI_INT32 numberOfEntries, pAR_SCHEDULED_MSG_ENTRY_TYPE messageEntry)	
Description	This routine defines a series of 32-bit ARINC 429 messages for periodic retransmission at the specified rate. Once defined, the message rate, content, or assigned channel for any individual message scheduler table entry within this same structure may be altered via invocation of AR_MODIFY_MSG_BLOCK.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	Invalid <i>channel</i> parameter value.
	ARS_INVHARVAL	Message scheduling is not supported on the specified channel.
	ARS_FAILURE	Message scheduler table full indication.
Arguments	CEI_INT32 numberOfEntries (input)	The number of entries to define from the subsequent structure pointer parameter, messageEntry.
	pAR_SCHEDULED_MSG_ENTRY_TYPE messageEntry (input)	Array of structures of message definition content, defined as follows:
	unsigned long messageIndex	The unique message scheduler table entry index assigned to this message. Upon completion of this routine, the messageIndex structure member will have been updated to reflect the message scheduler table index assigned to the respective message. Valid range is 0-2047.
	unsigned long board	Device to access. Valid range is 0-127.
	unsigned long channel	Channel on which to transmit this message. The valid range is 0 to one less than the number of installed transmit channels.

unsigned long rate	Periodic transmission rate, defined in milliseconds by default. For backward compatibility to the CEI-x20 tick-timer message rate method, when AR_SET_TIMERRATE has been executed to simulate the CEI-x20 tick-timer resolution assignment within the CEI-x30 API, the rate and start parameters will be scaled to the specified tick-timer resolution.
unsigned long start	Offset, (in milliseconds), from the start of CEI-x30 device message processing at which this message will begin its initial periodic transmission.
unsigned long txCount	The total number of times this message will be transmitted. The constant value ARU_SCHED_MSG_INFINITE (0xFFFFFFFF) indicates infinite transmission of this message is requested.
unsigned long data	The 32-bit ARINC 429 message to transmit.

AR_ENH_LABEL_FILTER

Syntax

CEI_INT32 ar_enh_label_filter (CDEV_BOARD_TYPE board, CDEV_CHAN_TYPE channel, CDEV_PARM_USI_TYPE label, CDEV_PARM_USI_TYPE sdi, CDEV_PARM_USI_TYPE essm, CDEV_PARM_SSI_TYPE action)

Description

This routine supports the assignment of both a single entry in the enhanced label filter table for the specified receive channel and channel-wide field definitions for the entire channel filter table. The CEI-x30 device enhanced label filtering feature supports the ability to both filter ARINC 429 messages and generate a hardware interrupt based on reception of a message matching the combined 8-bit label value, 2-bit SDI value, and 3-bit Enhanced SSM value.

Bit Field Name	ESSM	SDI	Label
Message Bit Field Location	30 29 28	9 8	7 6 5 4 3 2 1 0

Note:

This routine should be used exclusive of the use of the legacy API routine AR_LABEL_FILTER, as any filter table value assigned with one routine supersedes a previous assignment with another.

Label Filtering

Once message reception filtering has been enabled for a specified channel/label/sdi/essm combination, data received with matching bit field values will be discarded until label filtering for that specified message has been disabled.

Interrupt Generation

Once interrupt generation filtering has been enabled for a specified channel/label/sdi/essm combination, data received with matching bit field values induce an entry in the CEI-x30 device interrupt queue. If the device hardware interrupt has been enabled by invoking AR_SET_DEVICE_CONFIG with the option ARU_HW_INTERRUPT_ENABLE set to AR_ON, the device generates a PCI Interrupt to be serviced by the default interrupt service routine provided with the API or a custom ISR assigned by the host application.

The label filtering feature is disabled for all labels/sdi/essm combinations by default. Label filtering changes are effective immediately on completion of this routine.

Return Value

ARS_NORMAL

Routine execution was successful.

ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVHARVAL	Invalid <i>channel</i> parameter value.
ARS_INVARG	Invalid <i>label</i> , <i>sdi</i> , <i>essm</i> , or <i>action</i> parameter value.

Arguments

CDEV_BOARD_TYPE	board	(input) Device to access. Valid range is 0-127.
CDEV_CHAN_TYPE	channel	(input) Channel label filter table this routine is to access. The valid range is 0 to one less than the installed receive channel count.
CDEV_PARM_USI_TYPE	label	(input) The <i>label</i> of interest. Valid range is 0-255. Also valid is ARU_ALL_LABELS (511), which invokes the action for all labels.
CDEV_PARM_USI_TYPE	sdi	(input) The <i>SDI</i> field value of interest. Valid range is 0-3. Also valid is ARU_ALL_SDI (4), which invokes the action for all SDI entries for the specified label.
CDEV_PARM_USI_TYPE	essm	(input) The <i>ESSM</i> field value of interest. Valid range is 0-7. Also valid is ARU_ALL_ESSM (8), which invokes the action for all ESSM entries for the specified label.
CDEV_PARM_SSI_TYPE	action	(input) Enable or disable filtering action for this table entry. Valid values are:
FILTER_SEQUENTIAL	0x10	If CLEAR add the respective message to the sequential receive buffer; if SET filter the respective message from the sequential receive buffer.
FILTER_SNAPSHOT	0x20	If CLEAR add the respective message to the snapshot buffer; if SET filter the respective message from the snapshot buffer.
FILTER_INTERRUPT	0x40	If CLEAR does nothing; if SET, receipt of the respective message creates a receive channel index entry in the device's interrupt queue and if enabled, generates a PCI interrupt.

AR_EXECUTE_BIT

Syntax

CDEV_API_RET_TYPE ar_execute_bit (CDEV_BOARD_TYPE board, CDEV_PARM_SSI_TYPE testType)

Description

This routine performs hardware test functionality normally associated with board-level Built-In-Test (BIT). Testing ranges from a full SRAM memory test to verification of ARINC 429 message wrap on transmit/receive channel pair on the specified device.

Note:

This routine bypasses execution and returns a failure status if you invoke it when multi-process execution is enabled by AR_SET_PRELOAD_CONFIG and multiple processes are attached to a single board.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_MEMWRERR	Device SRAM test write/read/verify failure.
ARS_WRAP_DROP_FAIL	ARINC 429 wrap test data missing.
ARS_WRAP_DATA_FAIL	ARINC 429 wrap test data pattern mismatch.
ARS_WRAP_FLUSH_FAIL	Unknown external data received during wrap test execution.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	Invalid <i>testType</i> parameter value.
ARS_FAILURE	Timer-deviation test failed, or multi-process execution is enabled.
ARS_RX_BIT_CMD_ERROR	RAR15-XMC only – Receiver BIT command register write/read/verify failure.
ARS_RX_BIT_CH01_ERROR to ARS_RX_BIT_CH32_ERROR	RAR15-XMC only – Receiver “nn” BIT loopback execution failure, where a valid range for “nn” is 1 to 32.

Arguments

CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

CDEV_PARM_SSI_TYPE testType (input) Type of test to execute, defined as follows:

AR_BIT_BASIC_STARTUP (0) invokes a basic device initialization to a reset state (all buffers are flushed and channel configurations reset); however, the device firmware is not reloaded or restarted.

AR_BIT_FULL_STARTUP (1) invokes device initialization, a full, destructive SRAM memory test, and an internal wrap test of all matched transmit/ receive channels, (regardless of prior invocation of the API routine AR_BYPASS_WRAP_TEST). The duration of this test is approximately 17 seconds.

AR_BIT_PERIODIC (2) invokes a short, non-destructive SRAM memory test and a timer-deviation test, providing verification of the basic health status of the device.

AR_BIT_INT_LOOPBACK (3) invokes an internal wrap test of all matched transmit/receive channels.

AR_BIT_EXT_LOOPBACK (4) invokes an external wrap test of all matched transmit/receive channels.

AR_BIT_PARTIAL_SRAM (8) invokes a short destructive test of select, unused SRAM locations

AR_BIT_FULL_SRAM (9) invokes a destructive test of all SRAM locations.

AR_BIT_RX_LOOPBACK (10) exclusive to the RAR15-XMC products, invokes the Receiver BIT loopback functional test.

AR_BIT_SELECT_SRAM_MIN to
AR_BIT_SELECT_SRAM_MAX (100 to 1123)
invokes a destructive test of a select block of SRAM, parsed into 1024 blocks of 512 locations each.

AR_GET_573_FRAME

Syntax	CDEV_API_RET_TYPE ar_get_573_frame (CDEV_BOARD_TYPE board, pCEI_UINT32 numberWords, pCEI_UINT16 arincData)	
Description	<p>This function retrieves <i>numberWords</i> of ARINC 573/717 data from the ARINC 573/717 receive channel. If any data is available, the actual number of words received is indicated in the return value of <i>numberWords</i>. If auto-synchronization is configured for the ARINC 573/717 channel, this function will search the receive buffer for any occurrence of the first sub-frame sync word (defined via invocation of AR_SET_573_CONFIG with the item set to ARU_573_SYNC_WORD1) and return the specified number of words of frame data following the instance of that sync word. With automatic synchronization selected and the full frame size specified in <i>numberWords</i>, this function will wait until the full frame is received and copied to the destination array. The acquisition of an entire ARINC 573/717 frame may require up to four seconds to complete.</p>	
Return Value	ARS_NODATA	No frame data was available.
	ARS_GOTDATA	At least one ARINC 573/717 data word has been retrieved.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVHARVAL	ARINC 573 support is not available on device.
	ARS_INVARG	Invalid <i>numberWords</i> or <i>arincData</i> parameter.
Arguments	CDEV_BOARD_TYPE board	(input) Device to access. Valid range is 0-127.
	pCEI_UINT32 numberWords	(input/output) As an input this specifies the number of words to retrieve from the receive buffer. As an output this indicates how many words were retrieved from the receive buffer, less than or equal to the input value of <i>numberWords</i> .
	pCEI_UINT16 arincData	(output) The address that is to receive the frame data. The format of each data word in the ARINC 573/717 frame is defined as follows:

15	14	13 – 12	11 - 0
sync word	RESERVED	subframe	data

sync word: indicates this word was detected as a sync word, where a value of 1 indicates sync word and 0 indicates data word.

subframe: identifies the sub-frame assignment for this word, where 1 indicates sub-frame 1, 2 indicates sub-frame 2, 3 indicates sub-frame 3, and 0 indicates sub-frame 4.

data: the 12-bit ARINC 573/717 data.

AR_GET_429_MESSAGE

Syntax

CDEV_API_RET_TYPE ar_get_429_message (CDEV_BOARD_TYPE board, CDEV_CHAN_TYPE channel, CDEV_PARM_SSI_TYPE waitState, CDEV_PARM_VOID_PTR_TYPE data, CDEV_PARM_VOID_PTR_TYPE timetag)

Description

This routine retrieves the most recent ARINC 429 data and 32-bit time-tag from the specified channel. If no data is present in the receiver buffer, this routine attempts to retrieve data for up to one-half second. If no data is present after one-half second, a time-out status is returned. If **no wait** is specified and no data is available, the return status is so indicated.

Return Value

ARS_GOTDATA	An ARINC 429 message and its time-tag have been retrieved.
ARS_CHAN_TIMEOUT	The receive buffer was empty (if waitState is AR_ON).
ARS_NODATA	The receive buffer was empty (if waitState is AR_OFF).
ARS_BAD_MESSAGE	Receipt of an invalid message has been detected on this receiver; a message was returned if available in the buffer.
ARS_RX_BUFFER_OVERRUN	A receive buffer overrun was detected, no message was returned, and the receive buffer is potentially stale.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVHARVAL	Channel is not available on the device.
ARS_INVARG	A NULL <i>data</i> parameter value was provided.

Arguments

CDEV_BOARD_TYPE board	(input) Device this routine is to access. Valid range is 0-127.
CDEV_CHAN_TYPE channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.

CDEV_PARM_SSI_TYPE waitState(input) Whether or not to wait for data. A value of AR_ON specifies to wait ½ second for data; a value of AR_OFF specifies to return if no data is immediately available.

CDEV_PARM_VOID_PTR_TYPE data (output) The address that is to receive the data. The returned ARINC 429 data is always in normal ARINC format.

CDEV_PARM_VOID_PTR_TYPE timetag (output) The address that is to receive the 32-bit time-tag associated with the data, (resolution is programmable). If the merged receive mode is active for the specified channel, the upper five bits of the 32-bit time-tag word will contain the receive channel number on which the data was received. If the *timetag* parameter is NULL, time-tag information will not be provided.

AR_GET_BASE_ADDR

Syntax	pCEI_UINT32 ar_get_base_addr (CDEV_BOARD_TYPE board)
Description	This routine returns the driver-acquired virtual base address for the PCI memory region for the host interface of the specified device. This routine should be invoked only after successfully invoking AR_LOADLSV.
Return Value	Any positive value exceeding \$2000 is the driver-acquired virtual base address for the host interface PCI memory region of the specified device. ARS_INVBOARD An uninitialized board or invalid <i>board</i> value was provided.
Arguments	CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

AR_GETBLOCK

Syntax

CEI_INT32 ar_getblock (CEI_UINT32 board, CEI_UINT32 channel, CEI_INT32 maxMessages, CEI_INT32 offset, pCEI_INT32 actualCount, pCEI_INT32 data, pCEI_INT32 timeTags);

Description

This routine retrieves all of the available ARINC messages from the requested receive channel buffer and copies them to the desired destination. If the *timeTags* parameter is not NULL, the 32-bit time-tag data associated with each retrieved message is also copied. The *actualCount*, *data*, and *timeTags* parameters are only when *Return Value* is ARS_GOTDATA or ARS_BAD_MESSAGE.

The channel value passed to this routine corresponds to the ARINC 429 receive channel index, starting with zero. If that value exceeds the 429 receive channel count and an ARINC 717 receiver exists, the ARINC 717 receiver is used as the designated channel buffer.

Return Value

ARS_GOTDATA Message(s) and time-tag(s) were retrieved.

ARS_NODATA The receive buffer was empty.

ARS_BAD_MESSAGE Receipt of an invalid message has been detected on this receiver; message(s) were returned if available in the buffer.

ARS_RX_BUFFER_OVERRUN A receive buffer overrun was detected, no message was returned, and the receive buffer is potentially stale.

ARS_BOARD_MUTEX Access to the Board Lock timed-out/failed.

ARS_INVBOARD An uninitialized board or invalid *board* value was provided.

ARS_INVHARVAL Channel is not available on device.

ARS_INVARG Invalid *maxMessages*, *actualCount*, or *data* parameter was encountered.

Arguments

CEI_UINT32 board (input) Device to access. Valid range is 0-127.

CEI_UINT32 channel (input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.

CEI_INT32 maxMessages (input) The number of messages to retrieve.

- CEI_INT32 offset unused parameter, retained for legacy API support
- pCEI_INT32 actualCount (output) The number of messages retrieved.
- pCEI_INT32 data (output) Array to store 32-bit ARINC data. The format of the 32-bit value is dependent on the protocol assigned to the respective receive channel:

ARINC 429/575 Data Format

31	30 – 10	9 - 8	7 - 0
Parity Indication or ARINC Data MSB	ARINC Data	SDI bits or ARINC Data bits 0-1	ARINC Label (MSB – LSB)

ARINC 717 Data Format

31 – 16	15	14	13 - 12	11 – 0
Unused	Sync Indication	Unused	Sub-frame Identification	Data Word

See the *Channel Buffer Word 4 - Receive* description for more details.

- pCEI_INT32 timeTags (output) Array to store 32-bit message time-tags; can be set to NULL if time-tags are not desired.

AR_GETBLOCK_T

Syntax

CEI_INT32 ar_getblock_t (CEI_UINT32 board, CEI_UINT32 channel, CEI_INT32 maxMessages, pCEI_INT32 actualCount, pCEI_UINT32 msgChan, pCEI_INT32 data, pCEI_INT32 timeTagMsw, pCEI_INT32 timeTagLsw)

Description

This routine retrieves the available ARINC 429 messages from the requested receive channel buffer and copies them to the desired destination. If the *msgChan*, *timeTagMsw*, and *timeTagLsw* parameters are not NULL, the receive channel and 64-bit time-tag data associated with each retrieved message are also copied. The *actualCount*, *msgChan*, *data*, *timeTagMsw* and *timeTagLsw* parameters are only valid when *Return Value* is ARS_GOTDATA or ARS_BAD_MESSAGE.

Return Value

ARS_GOTDATA	Message(s) and time-tag(s) were retrieved.
ARS_BAD_MESSAGE	Receipt of an invalid message has been detected on this receiver; message(s) were returned if available in the buffer.
ARS_RX_BUFFER_OVERRUN	A receive buffer overrun was detected, no message was returned, and the receive buffer is potentially stale.
ARS_NODATA	The receive buffer was empty.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVHARVAL	Channel is not available on device.
ARS_INVARG	Invalid <i>maxMessages</i> , <i>actualCount</i> , or <i>data</i> parameter was encountered.

Arguments

CEI_UINT32 board	(input) Device this routine is to access. Valid range is 0-127.
CEI_UINT32 channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
CEI_INT32 maxMessages	(input) The number of messages to retrieve.
pCEI_INT32 actualCount	(output) The number of messages retrieved.

pCEI_UINT32 msgChan	(output) Array to store the receiver channel indication, necessary for actual receive channel determination when using the Merged Receive Mode
pCEI_INT32 data	(output) Array to store 32-bit ARINC 429 data.
pCEI_INT32 timeTagMsw	(output) Array to store the most significant 32-bits of the 64-bit time-tag data.
pCEI_INT32 timeTagLsw	(output) Array to store the least significant 32-bits of the 64-bit time-tag data.

AR_GET_BOARDNAME

Syntax	pCEI_CHAR ar_get_boardname (CDEV_BOARD_TYPE board, pCEI_CHAR boardName)	
Description	This routine returns a character string describing the board name for the specified device. It should only be invoked after successful invocation of AR_LOADLSV.	
Return Value	NULL	An uninitialized board or invalid <i>board</i> value was provided.

For any valid detected board, the return value is a character string description of board associated with the supplied *board* value:

“AMC-A30”
 “CEI-430”
 “CEI-430A”
 “CEI-530”
 “CEI-830”
 “R830RX”
 “RAR-CPCI”
 “RAR-EC”
 “RAR-PCIE”
 “RAR15-XMC”
 “RCEI-830X820”
 ”RAR-XMC”
 ”RCEI-830A”
 "RAR-MPCIE"

Arguments	CDEV_BOARD_TYPE board	(input) Device to access. Valid range is 0-127.
	pCEI_CHAR boardName	(output) If a valid board is detected and this parameter is not NULL, the character description of that board is copied to the location referenced by this parameter. A minimum of 12 bytes of allocation is required for the destination array.

AR_GET_BOARDTYPE

Syntax CDEV_API_RET_TYPE ar_get_boardtype (CDEV_BOARD_TYPE board)

Description This routine returns the API/device type for the specified device. It should only be invoked after successful invocation of AR_LOADLSV.

Return Value ARS_INVBOARD An uninitialized board or invalid *board* value was provided.

For any value less than ARS_INVBOARD, the return value indicates the type of board associated with the supplied *board* value:

CEI-830	(19)
CEI-430	(21)
AMC-A30	(22)
CEI-530	(26)
R830RX	(27)
RAR-CPCI	(28)
RAR-EC	(29)
RAR-PCIE	(30)
CEI-430A	(31)
RCEI-830X820	(34)
RAR-XMC	(35)
RCEI-830A	(36)
RAR-MPCIE	(38)
RAR15-XMC	(119)

Arguments CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

AR_GET_CHANNEL_INDEX_INFO

Syntax	CDEV_API_RET_TYPE ar_get_channel_index_info (CDEV_BOARD_TYPE board, AR_CHANNEL_INDEX_INFO_TYPE * chan_index_info)	
Description	<p>This routine returns the returns the CEI-x30 channel indexing configuration for the specified board, specifically for use with boards supporting programmable channel configurations. It should only be invoked after successful invocation of AR_LOADLSV.</p> <p>This routine is typically used during application initialization of a programmable channel board configuration to both determine programmable channel index values and subsequently assign those programmable channels as either a receiver or transmitter (reference the ar_set_device_config ARU_CONFIG_PROGRAMMABLE_CHAN option).</p>	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
Arguments	<p>CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.</p> <p>AR_CHANNEL_INDEX_INFO_TYPE * chan_index_info</p> <p>(output) Channel index information structure. This structure defines the board's channel configuration and transmit/receive indexing information. The structure type is defined as follows:</p> <p>unsigned long num_channels The total number of channels installed on the board, where programmable channels are counted as a single channel.</p> <p>unsigned long chan_type[256] An array of ARINC429 channel type values associated with each of the 256 virtual channels allocated on in CEI-x30 device host interface, indexed from 0 to 255. Supported channel types are defined as:</p>	

AR_CHAN_TYPE_429_RX (0) indicates this channel is implemented as a fixed receiver in the host interface.

AR_CHAN_TYPE_429_TX (1) indicates this channel is implemented as a fixed transmitter in the host interface.

AR_CHAN_TYPE_429_RX_OR_TX (2) indicates this channel is implemented as a programmable transmitter or receiver in the host interface

AR_CHAN_TYPE_UNDEFINED (255) indicates this channel is either not assigned or assigned to a channel type other than ARINC 429 (e.g. Discrete Input/Output).

unsigned long chan_rx_index[256] The receive channel index values assigned to corresponding *chan_type* array entries assigned a value of either AR_CHAN_TYPE_429_RX or AR_CHAN_TYPE_429_RX_OR_TX.

unsigned long chan_tx_index[256] The transmit channel index values assigned to corresponding *chan_type* array entries assigned a value of either AR_CHAN_TYPE_429_TX or AR_CHAN_TYPE_429_RX_OR_TX.

AR_GET_CONFIG

Syntax

CEI_INT32 ar_get_config (CDEV_BOARD_TYPE board,
CDEV_PARM_SSI_TYPE item)

Description

This routine returns the active state of API information, board level settings, and limited ARINC 429 channel configuration register bit fields. It is provided for backward compatibility to CEI-x20 based applications. The routine AR_GET_DEVICE_CONFIG is the desired routine for acquiring information regarding channel and board-level configuration.

See the ARU_* definitions in the file CDEV_API.H for the most current list of parameter options supported by this routine and the values associated with those definitions.

Return Value

If the requested item is ARU_RX_CH nn _BIT_RATE (500-531), where nn is the receiver channel (01 - 32), this routine returns the current value of the channel configuration register baud rate field:

AR_HIGH	(0) high rate (100Kbs)
AR_LOW	(1) low rate (12.5Kbs)

Any other value is returned as a frequency value in Hertz.

If the requested item is ARU_TX_CH nn _BIT_RATE (700-731), where nn is the transmitter channel (01 - 32), this routine returns the current value of the channel configuration register baud rate field:

AR_HIGH	(0) high rate (100Kbs)
AR_LOW	(1) low rate (12.5Kbs)

Any other value is returned as a frequency value in Hertz.

If the requested item is ARU_RX_CH nn _PARITY (900-931), where nn is the receiver channel (01 - 32), this routine returns the current state of the specified receiver channel configuration register parity field:

AR_ODD	(0) receiver parity check enabled
AR_OFF	(8) receiver parity check disabled

If the requested item is ARU_TX_CH nn _PARITY (1100-1131), where nn is the transmitter channel (01 - 32), this routine returns the current state of the specified transmitter channel configuration register parity field:

AR_ODD	(0) odd transmitter parity
AR_EVEN	(1) even transmitter parity

If the requested item is ARU_TX_CH nn _SHUT_OFF (1700-1731), where nn is the transmitter channel (01 - 32), this routine returns the current state of the specified transmitter channel configuration register transmit disable field:

AR_ON	(7) external transmission is disabled
AR_OFF	(8) external transmission is enabled

If the requested item is ARU_TX_CH nn _HB_INJ (3300-3331), where nn is the transmitter channel (01 - 32), this routine returns the current state of the specified transmitter channel configuration register high-bit error injection field:

AR_ON (7) 33-bit transmission is enabled
 AR_OFF (8) standard 32-bit transmission is enabled

If the requested item is ARU_TX_CH nn _LB_INJ (3500-3531), where nn is the transmitter channel (01 - 32), this routine returns the current state of the specified transmitter channel configuration register low-bit error injection field:

AR_ON (7) 31-bit transmission is enabled
 AR_OFF (8) standard 32-bit transmission is enabled

If the requested item is ARU_TX_CH nn _GAP_INJ (3700-3731), where nn is the transmitter channel (01 - 32), this routine returns the current state of the specified transmitter channel configuration register message gap error injection field:

AR_ON (7) 3-bit message gap is used
 AR_OFF (8) standard 4-bit message gap is used

If the requested item is ARU_CONFIGURATION (21), this routine returns the value of the CEI-x30 Board Configuration, defined as follows:

CEIDEV_CONFIG_CEI830	(7)	CEI-830
CEIDEV_CONFIG_CEI430	(8)	CEI-430
CEIDEV_CONFIG_AMCA30	(9)	AMC-A30
CEIDEV_CONFIG_CEI530	(10)	CEI-530
CEIDEV_CONFIG_R830RX	(11)	R830RX
CEIDEV_CONFIG_RAR_CPCI	(12)	RAR-CPCI
CEIDEV_CONFIG_RAR_EC	(13)	RAR-EC
CEIDEV_CONFIG_RAR_PCIE	(14)	RAR-PCIE
CEIDEV_CONFIG_CEI430A	(15)	CEI-430A
CEIDEV_CONFIG_RAR15XT	(17)	RAR15-XMC
CEIDEV_CONFIG_R830X820	(18)	RCEI-830X820
CEIDEV_CONFIG_RAR_XMC	(19)	RAR-XMC
CEIDEV_CONFIG_RCEI830A	(20)	RCEI-830A
CEIDEV_CONFIG_RAR_MPCIE	(21)	RAR-MPCIE

If the requested item is ARU_RX_TIMETAG_MODE (440), this routine returns a value representing the currently selected timer/time-tag source and resolution. This value indicates the resolution of any timer-read or receive data time-tag value obtained via the API, and is defined as follows:

AR_TIMETAG_EXT_IRIG_64BIT	(0)
AR_TIMETAG_INT_USEC_64BIT	(1)
AR_TIMETAG_INT_20USEC_32BIT	(3)
AR_TIMETAG_INT_MSEC_32BIT	(4)
AR_TIMER_X20_COMPAT_32BIT	(5)
AR_TIMETAG_SYNC_1553_CH1	(11)

AR_TIMETAG_SYNC_1553_CH2 (12)
 AR_TIMETAG_SYNC_1553_CH3 (13)
 AR_TIMETAG_SYNC_1553_CH4 (14)

A value of AR_TIMETAG_EXT_IRIG_64BIT indicates the source is the external IRIG receiver, if connected; otherwise, if the IRIG signal is not internally wrapped this selection would be invalid.

Any 1553 synchronized time value indicates the timer source is the respective 1553 channel on a Multi-protocol device.

All other values represent various timer/time-tag LSB resolution values based on the internal CEI-x30 device timer.

If the requested item is ARU_ACCESS_SNAPSHOT_BUFFER (38), this routine returns the currently selected Snapshot Buffer storage mode:

ARU_LABEL_ONLY (0) messages stored based on label
 ARU_LABEL_WITH_SDI (1) messages stored based on the combined label and SDI field values

If the requested item is ARU_IRIG_WRAP_ENABLE (441), this routine returns the current state of the IRIG Receiver internal wrap feature:

AR_ON (7) IRIG Receiver is patched into the IRIG Generator
 AR_OFF (8) IRIG Receiver is configured for external IRIG source

If the requested item is ARU_IRIG_AVAILALBE (445), this routine returns TRUE if IRIG-B support is available, FALSE if it is not.

If the requested item is ARU_IRIG_CALIBRATED (447), this routine verifies the ability to capture consecutive IRIG time samples at a one second interval. If this results in a return status of FALSE (0), the IRIG signal is not consistent; otherwise, a return value of TRUE (1) indicates the signal is valid, or an error status (any value greater than 1) indicates a failure occurred.

If the requested item is not on this list or in the list of valid items for AR_GET_DEVICE_CONFIG, this routine will return a value of ARS_INVARG.

If the requested item is not valid for the specified device, this routine returns a value of ARS_INVHARCMD.

If the specified board is invalid or has not been initialized, this routine returns ARS_INVBOARD.

If access to the Board Lock timed-out or failed, this routine returns ARS_BOARD_MUTEX.

If the requested item is ARU_HW_FPGA_TEMPERATURE (453), this routine returns the current temperature of the RAR-PCIE or RAR-XMC FPGA core, in units of “milli”degrees Celsius.

Arguments

CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

CDEV_PARM_SSI_TYPE item (input) Control function about which to return information:

ARU_RX_CH01_BIT_RATE –
ARU_RX_CH32_BIT_RATE receiver 1 – 32 bit rate selection.

ARU_TX_CH01_BIT_RATE –
ARU_TX_CH32_BIT_RATE transmitter 1 – 32 bit rate selection.

ARU_RX_CH01_PARITY –
ARU_RX_CH32_PARITY receiver 1 – 32 parity state.

ARU_TX_CH01_PARITY –
ARU_TX_CH32_PARITY transmitter 1 – 32 parity state.

ARU_TX_CH01_SHUT_OFF –
ARU_TX_CH32_SHUT_OFF transmitter 1 – 32 enable state.

ARU_TX_CH01_LB_INJ – transmitter 1 – 32 low bit error
ARU_TX_CH32_LB_INJ enable state.

ARU_TX_CH01_HB_INJ – transmitter 1 – 32 high bit error
ARU_TX_CH32_HB_INJ enable state.

ARU_TX_CH01_GAP_INJ – transmitter 1 – 32 message
ARU_TX_CH32_GAP_INJ gap error enable state.

ARU_IRIG_AVAILABLE IRIG Receiver installed state.
ARU_IRIG_WRAP_ENABLE IRIG Receiver internal wrap state.
ARU_IRIG_CALIBRATED IRIG signal validity.
ARU_ACCESS_SNAPSHOT_BUFFER snapshot storage mode.
ARU_FW_VERSION Hardware Version reg. value.
ARU_CONFIGURATION configuration of the device.
ARU_RX_TIMETAG_MODE active timer/time-tagging mode.
ARU_HW_FPGA_TEMPERATURE read FPGA temperature

AR_GET_DATA

Syntax

CDEV_API_RET_TYPE ar_get_data (CDEV_BOARD_TYPE board, CDEV_CHAN_TYPE * channel, pCEI_UINT32 data, pCEI_UINT32 timeTagLo, pCEI_UINT32 timeTagHi)

Description

This routine retrieves the next unread message and 64-bit time-tag from the specified receive channel. If it successfully returns data, there may or may not be more data in the buffer. It means only that there was at least one message in the buffer. Subsequent calls are required to determine if more data words are available in the buffer. If this routine returns a status value of ARS_NODATA, the buffer is empty.

If the specified channel was configured for merged mode operation along with other receive channels, this routine returns the next unread message from the merged receive buffer and indicate on which channel the message was received via the *channel* parameter.

Return Value

ARS_GOTDATA	A message and time-tag have been received.
ARS_NODATA	The receive buffer was empty.
ARS_BAD_MESSAGE	Receipt of an invalid message has been detected on this receiver; a message was returned if available in the buffer.
ARS_RX_BUFFER_OVERRUN	A receive buffer overrun was detected, no message was returned, and the receive buffer is potentially stale.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVHARVAL	The channel is not available on the device.

Arguments

CDEV_BOARD_TYPE board	(input) Device this routine is to access. Valid range is 0-127.
CDEV_CHAN_TYPE * channel	(input/output) As an input, specifies which hardware receive channel this routine is to access, (see the description of the Hardware Channel Assignments in the CEI-x30 Product Line Hardware Manual for the list of valid hardware receive channel values). As an output, indicates the receive

	channel number on which the data was received (for merged-mode channel reporting).
pCEI_UINT32 data	(output) Address that is to receive the data.
pCEI_UINT32 timeTagLo	(output) Address that is to receive the least-significant 32 bits of the 64-bit time-tag associated with the data, (resolution of the combined time-tag words is 1 μ sec).
pCEI_UINT32 timeTagHi	(output) Address that is to receive the most-significant 32 bits of the 64-bit time-tag associated with the data, (resolution of the combined time-tag words is 1 μ sec).

AR_GET_DATA_XT

Syntax

CDEV_API_RET_TYPE ar_get_data (CDEV_BOARD_TYPE board,
CDEV_CHAN_TYPE * channel, pCEI_UINT32 data,
pAR_TIMETAG_TYPE timeTagRef)

Description

This routine retrieves the next unread message and the associated time-tag structure from the specified receive channel. If it successfully returns data, there may or may not be more data in the buffer. It means only that there was at least one message in the buffer. Subsequent calls are required to determine if more data words are available in the buffer. If this routine returns a status value of ARS_NODATA, the buffer is empty.

If the specified channel was configured for merged mode operation along with other receive channels, this routine returns the next unread message from the merged receive buffer with a reference to which channel received the message via the *channel* parameter.

Return Value

ARS_GOTDATA A message and time-tag have been received.

ARS_NODATA The receive buffer was empty.

ARS_BAD_MESSAGE Receipt of an invalid message has been detected on this receiver; a message was returned if available in the buffer.

ARS_RX_BUFFER_OVERRUN A receive buffer overrun was detected, no message was returned, and the receive buffer is potentially stale.

ARS_BOARD_MUTEX Access to the Board Lock timed-out/failed.

ARS_INVBOARD An uninitialized board or invalid *board* value was provided.

ARS_INVHARVAL The channel is not available on the device.

Arguments

CDEV_BOARD_TYPE board (input) Device this routine is to access. Valid range is 0-127.

CDEV_CHAN_TYPE * channel (input/output) As an input, specifies which hardware receive channel this routine is to access, (see the description of the Hardware Channel Assignments in the CEI-x30 Product Line Hardware Manual for the list of valid hardware receive channel values). As an output, indicates the receive channel number on which the data was

received (for merged-mode channel reporting).

pCEI_UINT32 data (output) Address that is to receive the data.

pAR_TIMETAG_TYPE timeTagRef(input)

The timeTagRef structure member *timeTagFormat* specifies the desired format of the time-tag value returned in the *timeTag* structure member. Valid values are:

AR_TIMETAG_EXT_IRIG_64BIT	0
AR_TIMETAG_INT_USEC_64BIT	1
AR_TIMETAG_HOST_USEC_64BIT	2
AR_TIMETAG_INT_20USEC_32BIT	3
AR_TIMETAG_INT_MSEC_32BIT	4
AR_TIMETAG_SYNC_1553_CH1	11
AR_TIMETAG_SYNC_1553_CH2	12
AR_TIMETAG_SYNC_1553_CH3	13
AR_TIMETAG_SYNC_1553_CH4	14
AR_TIMETAG_USE_PROGRAMMED_MODE	99

(output)

The address that is to receive the time-tag data structure associated with the data.

AR_GET_DEVICE_CONFIG

Syntax	CDEV_API_RET_TYPE ar_get_device_config (CDEV_BOARD_TYPE board, CDEV_CHAN_TYPE channel, CDEV_PARM_SSI_TYPE item, CDEV_PARM_SSI_PTR_TYPE value)	
Description	This routine returns the state of the device configuration register attribute based on the combined item/value parameters. It is designed to support all ARINC 429 channel configuration register bit fields available to the device.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	The item argument value is not supported by this API routine.
	ARS_INVHARVAL	The item argument value is not supported by this device configuration.
	ARS_HW_CONSISTENCY	Indicates the board is compatible with the CEI-x30 Enhanced Operations (Version 2.00 API or later).
Arguments	CDEV_BOARD_TYPE board	(input) Device to access. Valid range is 0-127.
	CDEV_CHAN_TYPE channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed channel count.
		For the ARU_xx_FIFO_COUNT options, channel 32 is allowed to access the ARINC 717 buffer count, and 63 to access the Merged Receive Buffer count.
		For board-level configuration items, this parameter is not used.
	CDEV_PARM_SSI_TYPE item	(input) configuration register or board level attribute for which to return the current state:
	ARU_RX_PARITY	receive channel parity enable.

ARU_RX_BITRATE	receive channel bit rate.
ARU_RX_FIFO_ENABLE	receive channel enable.
ARU_RX_DISABLE	receive channel enable.
ARU_RECV_MODE	receiver internal wrap.
ARU_RX_MERGED_MODE	receiver merged mode enable.
ARU_RX_MSG_SIZE_24BIT	receiver 24-bit message mode.
ARU_TX_BITRATE	transmit channel bit rate.
ARU_TX_PARITY	transmit channel parity select.
ARU_TX_FIFO_ENABLE	transmit channel enable.
ARU_TX_DISABLE	transmit channel transceiver disable.
ARU_TX_BIT_ERROR	transmit channel bit error enable.
ARU_TX_GAP_ERROR	transmit channel gap error enable.
ARU_TX_MSG_SIZE_24BIT	transmitter 24-bit message mode.
ARU_FAST_SLEW_RATE	transmit channel slew rate select.
ARU_ACCESS_SNAPSHOT_BUFFER	snapshot storage mode.
ARU_IRIG_WRAP_ENABLE	IRIG receiver internal wrap state.
ARU_IRIG_AVAILABLE	IRIG receiver installed state.
ARU_IRIG_OUTPUT_ENABLE	R830RX IRIG Tx state.
ARU_IRIG_INPUT_TIME	IRIG received sample value.
ARU_IRIG_CALIBRATED	IRIG signal validity.
ARU_DEVICE_DISABLE	CEI-430 device disabled state.
ARU_DISCRETE_IN	discrete input state.
ARU_DIFFERENTIAL_IN	differential input state.
ARU_DIFFERENTIAL_OUT	differential output enable state.
ARU_RX_TIMETAG_MODE	active timer/time-tagging mode.
ARU_CHAN_COUNT_429	ARINC 429 Tx channel count.
ARU_CHAN_COUNT_573	ARINC 573/717 channel count.
ARU_CHAN_COUNT_DISC	discrete I/O channel count.
ARU_CHAN_COUNT_DIFF	differential I/O channel count.
ARU_RX_FIFO_COUNT	receive FIFO buffer fill count.
ARU_TX_FIFO_COUNT	transmit FIFO buffer fill count.
ARU_RX_MSG_COUNT	receive message count.
ARU_TX_MSG_COUNT	transmit message count.
ARU_FW_VERSION	current programmed firmware.
ARU_HW_ENHANCE_CHECK	current device BAR2 size.
ARU_HW_INTERRUPT_ENABLE	current PCI interrupt state
ARU_HW_1PT0V_PWR_SUPPLY	RAR-PCIE 1.0V supply value
ARU_HW_2PT5V_PWR_SUPPLY	RAR-PCIE 2.5V supply value
ARU_SPECIAL_PROGRAMMABLE_CONFIG	detect pgm chan cfg

Options for the item parameter only
available with multiprotocol boards:

ARU_1553_CHANNEL_INITIALIZED	1553 chan init flag.
ARU_1553_TIME_TAG_MODE	1553 chan time mode.
ARU_1553_TIME_TAG_DISPLAY	1553 chan time string conversion format.

CDEV_PARM_SSI_PTR_TYPE value (output) state of the
configuration register attribute:

If the requested item is ARU_RX_FIFO_ENABLE (16), ARU_RX_DISABLE (9), or ARU_TX_FIFO_ENABLE (17), this routine will return the current value of the specified channel configuration register FIFO Enable field:

AR_ON	(7) FIFO operation enabled
AR_OFF	(8) FIFO operation disabled

If the requested item is ARU_RX_BITRATE (1) or ARU_TX_BITRATE (2), this routine will return the current value of the specified channel configuration register baud rate field:

ARU_SPEED_HIGH	(0) high rate (100Kbs)
ARU_SPEED_LOW	(1) low rate (12.5Kbs)

Any other value is translated as a non-standard bus speed value divisor for the 16MHz device clock reference. This value and the respective baud rate may be interpreted using the following formula:

$$\text{Baud Rate} = 16,000,000 / (\text{Value}+2)$$

If the requested item is ARU_RX_PARITY (3), this routine returns the current state of the specified receive channel configuration register parity field:

AR_ON	(7) receiver parity check enabled
AR_OFF	(8) receiver parity check disabled

If the requested item is ARU_TX_PARITY (4), this routine returns the current state of the specified transmitter channel configuration register parity field:

ARU_PARITY_ODD	(0) odd transmitter parity
ARU_PARITY_EVEN	(1) even transmitter parity
ARU_PARITY_NONE	(2) transmitter parity disabled

If the requested item is ARU_RECV_MODE (5), this routine returns the current state of the specified receive channel configuration register Internal Wrap Enable field:

AR_WRAP_ON	(0) internal wrap reception enabled
AR_WRAP_OFF	(1) internal wrap reception disabled

If the requested item is ARU_RX_MERGED_MODE (18), this routine returns the current state of the specified receive channel configuration register Merge Mode enable field:

AR_ON	(7) Merged Mode enabled
AR_OFF	(8) Merged Mode disabled

If the requested item is ARU_RX_MSG_SIZE_24BIT (459), this routine returns the current state of the specified receive channel configuration register 24-bit Message Size enable field:

AR_ON	(7) ARINC 585 24-bit protocol enabled
AR_OFF	(8) ARINC 429/575 32-bit protocol enabled

If the requested item is ARU_TX_DISABLE (10), this routine returns the current state of the specified receive channel configuration register Transmit Disable field:

AR_ON (7) external transmission disabled
 AR_OFF (8) external transmission enabled

If the requested item is ARU_TX_BIT_ERROR (6), this routine returns the current state of the specified transmitter channel configuration register Bit Count Hi and Low fields:

AR_LO (0) Bit Count Low enabled
 AR_HI (1) Bit Count High enabled
 AR_OFF (8) both Bit Count Low and High disabled

If the requested item is ARU_TX_GAP_ERROR (8), this routine returns the current state of the specified transmitter channel configuration register Gap Error field:

AR_ON (7) Gap Error enabled
 AR_OFF (8) Gap Error disabled

If the requested item is ARU_FAST_SLEW_RATE (323), this routine returns the current state of the specified transmitter channel configuration register Slew Rate field:

AR_ON (7) Fast Slew Rate selected (1.5 μ sec rise time)
 AR_OFF (8) Slow Slew Rate selected (10 μ sec rise time)

If the requested item is ARU_TX_MSG_SIZE_24BIT (460), this routine returns the current state of the specified transmit channel configuration register 24-bit Message Size enable field:

AR_ON (7) ARINC 585 24-bit protocol enabled
 AR_OFF (8) ARINC 429/575 32-bit protocol enabled

If the requested item is ARU_ACCESS_SNAPSHOT_BUFFER (38), this routine returns the current state of the device snapshot storage mode:

ARU_LABEL_ONLY (0) message storage on a label basis
 ARU_LABEL_WITH_SDI (1) message storage on a combined label/SDI basis.

If the requested item is ARU_IRIG_AVAILALBE (445), this routine returns TRUE if IRIG-B support is available, FALSE if it is not.

If the requested item is ARU_IRIG_OUTPUT_ENABLE (26), this routine returns the enable state of the R830RX IRIG Generator Enable:

AR_ON (7) IRIG output is enabled
 AR_OFF (8) IRIG output is disabled

If the requested item is ARU_IRIG_INPUT_TIME (27), this routine returns the most recent received IRIG received sample value.

If the requested item is ARU_IRIG_WRAP_ENABLE (441), this routine returns the current state of the IRIG Receiver internal wrap feature:

AR_ON (7) IRIG Receiver is patched into the IRIG Generator
 AR_OFF (8) IRIG Receiver is configured for external IRIG source

If the requested item is ARU_IRIG_CALIBRATED (447), this routine verifies the ability to capture consecutive IRIG time samples at a one second interval. If this results in a return status of FALSE (0), the IRIG signal is not consistent; otherwise, a return value of TRUE (1) indicates the signal is valid, or an error status (any value greater than 1) indicates a failure occurred.

If the requested item is ARU_DISCRETE_IN (14), this routine returns the current state of the specified discrete I/O channel:

AR_HI (1) the discrete input is High
 AR_LO (0) the discrete input is Low

If the requested item is ARU_DIFFERENTIAL_IN (22), this routine returns the current state of the specified differential I/O channel:

AR_HI (1) the differential input is High
 AR_LO (0) the differential input is Low

If the requested item is ARU_DIFFERENTIAL_OUT (23), this routine returns the enable state of the specified differential I/O channel:

AR_ON (7) the differential output enabled
 AR_OFF (8) the differential output disabled

If the requested item is ARU_RX_TIMETAG_MODE (440), this routine returns a value representing the currently selected timer/time-tag source and resolution. This value indicates the resolution of any timer-read or receive data time-tag value obtained via the API, and is defined as follows:

AR_TIMETAG_EXT_IRIG_64BIT (0)
 AR_TIMETAG_INT_USEC_64BIT (1)
 AR_TIMETAG_INT_20USEC_32BIT (3)
 AR_TIMETAG_INT_MSEC_32BIT (4)
 AR_TIMER_X20_COMPAT_32BIT (5)
 AR_TIMETAG_SYNC_1553_CH1 (11)
 AR_TIMETAG_SYNC_1553_CH2 (12)
 AR_TIMETAG_SYNC_1553_CH3 (13)
 AR_TIMETAG_SYNC_1553_CH4 (14)

A value of AR_TIMETAG_EXT_IRIG_64BIT indicates the source is the external IRIG receiver, if connected; otherwise, if the IRIG signal is not internally wrapped this selection would be invalid.

Any 1553 synchronized time value indicates the timer source is the respective 1553 channel on a Multi-protocol device.

All other values represent various timer/time-tag LSB resolution values based on the internal CEI-x30 device timer.

If the requested item is ARU_CONFIGURATION (21), this routine returns the value of the CEI-x30 Board Configuration, defined as follows:

CEIDEV_CONFIG_CEI830	(7)	CEI-830
CEIDEV_CONFIG_CEI430	(8)	CEI-430
CEIDEV_CONFIG_AMCA30	(9)	AMC-A30
CEIDEV_CONFIG_CEI530	(10)	CEI-530
CEIDEV_CONFIG_R830RX	(11)	R830RX
CEIDEV_CONFIG_RAR_CPCI	(12)	RAR-CPCI
CEIDEV_CONFIG_RAR_EC	(13)	RAR-EC
CEIDEV_CONFIG_RAR_PCIE	(14)	RAR-PCIE
CEIDEV_CONFIG_CEI430A	(15)	CEI-430A
CEIDEV_CONFIG_RAR15XT	(17)	RAR15-XMC
CEIDEV_CONFIG_R830X820	(18)	RCEI-830X820
CEIDEV_CONFIG_RAR_XMC	(19)	RAR-XMC
CEIDEV_CONFIG_RCEI830A	(20)	RCEI-830A
CEIDEV_CONFIG_RAR_MPCIE	(21)	RAR-MPCIE

Intended for use only with the RAR-XMC and RAR15-XMC board configurations, if ARU_SPECIAL_PROGRAMMABLE_CONFIG (456) is the supplied item value and this routine returns any value other than zero, this is an indication the CEI-x30 Board Configuration contains a special programmable transmit/receive channel I/O configuration, defined as one of the following:

RAR_XMC_16P16_SPECIAL_CFG (5) This value indicates the RAR-XMC configuration contains 16 fixed receive channels and 16 transmit/receive programmable channels (where the programmable transmit channels are referenced as transmitters 0-15 while programmable receiver channels are referenced as receivers 16-31).

RAR_XMC_P16_SPECIAL_CFG (14) This value indicates the RAR-XMC configuration contains a fixed 16-channel transmit/receive programmable configuration (all channels are programmable as either transmitters or receivers).

RAR_XMC_102_SPECIAL_CFG (15) This value indicates the RAR-XMC configuration contains 10 fixed receive channels and 2 fixed transmit channels.

RAR15_XMC_SPECIAL_CFG (16) This value indicates the RAR15-XMC combo card configuration contains at least 10 fixed receive channels and from 2 to 8 transmit/receive programmable channels.

If the requested item is ARU_DEVICE_DISABLE (39), this routine returns the current value of the CEI-430 Global Enable Register – Device Disabled bit.

If the requested item is ARU_CHAN_COUNT_429 (448), this routine returns the ARINC 429 transmit channel count detected on the board.

If the requested item is ARU_CHAN_COUNT_573 (449), this routine returns the ARINC 573/717 transmit channel count detected on the board.

If the requested item is ARU_CHAN_COUNT_DISC (450), this routine returns the discrete output channel count detected on the board.

If the requested item is ARU_CHAN_COUNT_DIFF (451), this routine returns the differential output channel count detected on the board.

If the requested item is ARU_TX_FIFO_COUNT (19), this routine returns the current buffer count of messages in the specified ARINC 429 transmit FIFO awaiting transmission.

If the requested item is ARU_RX_FIFO_COUNT (28), this routine returns the current buffer count of messages in the ARINC 429 receive FIFO available to be read by the host application.

If the requested item is ARU_RX_MSG_COUNT (35), this routine returns the number of messages received on this channel since the board was last initialized.

If the requested item is ARU_TX_MSG_COUNT (36), this routine returns the number of messages transmitted on this channel since the board was last initialized.

If the requested item is ARU_FW_VERSION (20), this routine returns the current programmed firmware.

If the requested item is ARU_HW_ENHANCE_CHECK (30), this routine returns either ARS_NORMAL to indicate the board is compatible with the CEI-x30 Enhanced Operations (Version 2.00 API), or ARS_HW_CONSISTENCY to indicate it is not.

If the requested item is ARU_HW_INTERRUPT_ENABLE (29), this routine returns the current state of the PCI Interrupt Enable bit:

AR_ON	(7) PCI Interrupts are enabled
AR_OFF	(8) PCI Interrupts are disabled

If the requested item is ARU_HW_1PT0V_PWR_SUPPLY (454), this routine returns the current level of the RAR-PCIE on-board 1.0V supply in millivolts.

If the requested item is ARU_HW_2PT5V_PWR_SUPPLY (455), this routine returns the current level of the RAR-PCIE on-board 2.5V supply in millivolts.

If the requested item is ARU_1553_CHANNEL_INITIALIZED (10000), this routine returns the initialization state of the specified 1553 channel, indicating whether or not the channel is available for time-tag synchronization:

TRUE	(1) specified 1553 channel has been initialized
------	---

FALSE (0) specified 1553 channel has not been initialized

If the requested item is ARU_1553_TIME_TAG_MODE (10006), this routine returns the current BusTools/1553-API programmed time-tag mode for the specified 1553 channel, (reference the routine description for BusTools_TimeTagMode in the BusTools/1553-API Reference Manual for a description of the various time-tag mode options).

If the requested item is ARU_1553_TIME_TAG_DISPLAY (10008), this routine returns the current BusTools/1553-API programmed time-tag string conversion display format, (reference the routine description for BusTools_TimeTagMode in the BusTools/1553-API Reference Manual for a description of the various time-tag string conversion options).

AR_GET_573_CONFIG

Syntax	CDEV_API_RET_TYPE ar_get_573_config (CDEV_BOARD_TYPE board, CDEV_PARM_SSI_TYPE item, CDEV_PARM_SI_PTR_TYPE value)	
Description	This routine returns the state of the device configuration register attribute based on the combined item/value. It is designed to support the ARINC 573/717 configuration register attributes available to the device.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	The item argument value is not supported by this API routine.
	ARS_INVHARVAL	The item argument value is not supported by this device configuration.
Arguments	CDEV_BOARD_TYPE board (input) Device this routine is to access. Valid range is 0-127.	
	CDEV_PARM_SSI_TYPE item (input) Configuration item about which to return information:	
	ARU_RECV_MODE	receiver internal wrap.
	ARU_RX_BITRATE	receive channel bit rate.
	ARU_RX_FIFO_ENABLE	receive channel enable.
	ARU_RX_MERGED_MODE	receiver merged mode enable
	ARU_573_RX_AUTO_DETECT	receiver frame auto-detect enable.
	ARU_573_RX_BPRZ_SELECT	receiver BPRZ/HBP selection.
	ARU_TX_BITRATE	transmit channel bit rate.
	ARU_TX_FIFO_ENABLE	transmit channel enable.
	ARU_573_TX_BPRZ_SELECT	transmitter BPRZ encoder enable.
	ARU_573_TX_HBP_SELECT	transmitter HBP encoder enable.
	ARU_573_TX_SLEW_RATE	transmitter slew rate select.
	ARU_TX_FIFO_COUNT	transmit FIFO buffer fill count.
	ARU_573_SYNC_WORD1	receiver auto-detect sync word 1.
	ARU_573_SYNC_WORD2	receiver auto-detect sync word 2.
	ARU_573_SYNC_WORD3	receiver auto-detect sync word 3.
	ARU_573_SYNC_WORD4	receiver auto-detect sync word 4.

CDEV_PARM_SI_PTR_TYPE value (output) The address that receives the state of the item requested:

If the requested item is ARU_RECV_MODE (5), this routine returns the current state of the ARINC 573/717 receive channel Internal Wrap Enable:

AR_WRAP_ON (0) internal wrap reception enabled
 AR_WRAP_OFF (1) internal wrap reception disabled

If the requested item is ARU_RX_FIFO_ENABLE (16) or ARU_TX_FIFO_ENABLE (17), then this routine will return the current value of the ARINC 573/717 channel FIFO Enable:

AR_ON (7) FIFO operation enabled
 AR_OFF (8) FIFO operation disabled

If the requested item is ARU_RX_MERGED_MODE (18), this routine returns the current state of the ARINC 573/717 receive channel Merge Mode Enable:

AR_ON (7) Merge mode enabled
 AR_OFF (8) Merge mode disabled

If the requested item is ARU_573_RX_AUTO_DETECT (301), this routine returns the current state of the ARINC 573/717 receive channel Auto-synchronization Enable:

AR_ON (7) ARINC 573/717 frame auto-detection enabled
 AR_OFF (8) ARINC 573/717 frame auto-detection disabled

If the requested item is ARU_573_RX_BPRZ_SELECT (302), this routine returns the current state of the ARINC 573/717 receive channel Encoding Enable:

AR_ON (7) ARINC 573/717 BPRZ encoding enabled
 AR_OFF (8) ARINC 573/717 HBP encoding enabled

If the requested item is ARU_RX_BITRATE (1) or ARU_TX_BITRATE (2), this routine returns the current state of the ARINC 573/717 channel Baud Rate/Subframe Size selection (ranging from 0 to 7):

ARU_573_RATE_SIZE_384_32 384 bps, 32 word sub-frame
 ARU_573_RATE_SIZE_768_64 768 bps, 64 word sub-frame
 ARU_573_RATE_SIZE_1536_128 1536 bps, 128 word sub-frame
 ARU_573_RATE_SIZE_3072_256 3072 bps, 256 word sub-frame
 ARU_573_RATE_SIZE_6144_512 6144 bps, 512 word sub-frame
 ARU_573_RATE_SIZE_12288_1024 12288 bps, 1024 word sub-frame
 ARU_573_RATE_SIZE_24576_2048 24576 bps, 2048 word sub-frame
 ARU_573_RATE_SIZE_49152_4096 49152 bps, 4096 word sub-frame

If the requested item is ARU_573_TX_BPRZ_SELECT (313), this routine returns the current state of the ARINC 573/717 transmit channel Encoding Enable:

AR_ON (7) ARINC 573/717 BPRZ encoding enabled
AR_OFF (8) ARINC 573/717 BPRZ encoding disabled

If the requested item is ARU_573_TX_HBP_SELECT (314), this routine returns the current state of the ARINC 573/717 transmit channel Encoding Enable:

AR_ON (7) ARINC 573/717 HBP encoding enabled
AR_OFF (8) ARINC 573/717 HBP encoding disabled

If the requested item is ARU_573_TX_SLEW_RATE (305) this routine returns the current state of the ARINC 573/717 transmit channel Slew Rate selection:

ARU_573_TX_SLEW_1PT5 (1) 1.5µsec rise time
ARU_573_TX_SLEW_10PT0 (0) 10.0µsec rise time

If the requested item is ARU_TX_FIFO_COUNT (19), this routine returns the current buffer count of messages in the ARINC 717 transmit FIFO awaiting transmission.

If the requested item is ARU_573_SYNC_WORD1 (307), ARU_573_SYNC_WORD2 (308), ARU_573_SYNC_WORD3 (309), or ARU_573_SYNC_WORD4 (310), this routine returns the 12-bit value for the respective receiver sub-frame sync word.

AR_GET_ERROR

Syntax

pCEI_CHAR ar_get_error (CDEV_API_RET_TYPE status)

Description

Most of the API routines return status values, a majority of which indicate either success or some form of an error condition. When supplied with such an error value, this routine returns a pointer to a string describing the error.

Review the section, “Return Status Values”, for the current list of possible error codes and their explanations.

If the application intends to copy the string referenced via ar_get_error, a minimum allocation of 512 characters is required.

Return Value

A pointer to the error message character string.

Arguments

CDEV_API_RET_TYPE status (input) a status value returned by any of the API utilities.

AR_GETFILTER

Syntax

CEI_UINT32 ar_getfilter (CEI_UINT32 board, CEI_UINT32 channel, pCEI_CHAR filterTable)

Description

This routine returns the contents of a single channel label filter table from the device. Each receive channel has a separate section within the label filter table, is used by the firmware to control FIFO storage of received labels and generate hardware interrupts. Each element of the filter buffer consists of a bit field defined for compatibility with the CEI-x20 product line as follows:

FILTER_SEQUENTIAL	0x10	If CLEAR add label to Sequential receive buffer
FILTER_SNAPSHOT	0x20	If CLEAR add label to Snapshot receiver buffer
FILTER_INTERRUPT	0x40	If SET on reception insert the respective receive channel tag (ranging from 64-95) in the interrupt queue and if enabled generate a PCI interrupt.

The filter buffer for a single channel is defined as follows:

```
filterTable[MAX_ESSM][MAX_SDI][MAX_LABEL]
```

and accessed as:

```
filterTable[eSSM][SDI][label]
```

where the bits of the ARINC word are split up as follows:

eSSM	SDI	label
30, 29, 28	9, 8	7, 6, 5, 4, 3, 2, 1, 0

To write an entry to the label interrupt and filter table, refer to the API routines AR_ENH_LABEL_FILTER, AR_PUTFILTER, and AR_LABEL_FILTER.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	Invalid <i>channel</i> or <i>filterTable</i> parameter.

Arguments

CEI_UINT32 board	(input) Device to access. Valid range is 0-127.
CEI_UINT32 channel	(input) Specifies which receive channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
pCEI_CHAR filterTable	(input) Array to receive the contents of the specified channel's label filter table. This array must have a minimum allocation of 8Kbytes.

AR_GET_LABEL_FILTER

Syntax	CDEV_API_RET_TYPE ar_get_label_filter (CDEV_BOARD_TYPE board, CDEV_PARM_USI_TYPE label)
Description	This routine returns the active state of label filtering for the specified label on each of the first sixteen installed receive channels.
Return Value	<p>Given the routine is supplied with a valid board and label value, the return value indicates the active state of the specified label on each receive channel through the respective bit state, where the label filter state on receive channel zero (zero-referenced) is indicated via b0 as “1” to indicate the label is filtered and “0” to indicate either the label is not filtered or the receive channel is not installed. Subsequent bits in the value indicate the label filter state for the respective receive channel.</p> <p>A label is indicated to be “filtered” if the respective entry in the Label Filter Table is defined to filter the label from either the Sequential (FIFO) or Snapshot buffers.</p>
Arguments	<p>CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.</p> <p>CDEV_PARM_USI_TYPE label (input) Specifies which label to query. Valid range is 0 to 255.</p>

AR_GET_LATEST

Syntax

CEI_VOID ar_get_latest (CDEV_BOARD_TYPE board,
CDEV_CHAN_TYPE channel, CDEV_PARM_USI_TYPE label,
CDEV_PARM_VOID_PTR_TYPE data, pCEI_CHAR seq_num)

Description

In support of backward compatibility to previous ARINC product APIs, this routine returns the latest ARINC 429 message received for the specified channel/label combination from the snapshot buffer.

If the *label* parameter value requested is either 256 or the value ARU_ALL_LABELS (511), this routine treats the *data* parameter as an array reference and returns the most recent received ARINC message for all 256 valid ARINC labels for the specified channel, in successive *data* array elements. This function assumes that the caller has allocated at least 1024 bytes for *data* when used in this mode.

When using this routine, the host application should set the snapshot storage mode to **label field only**, (see the documentation on the routine AR_SET_DEVICE_CONFIG, for the configuration option ARU_ACCESS_SNAPSHOT_BUFFER). This sets up the x30 device to store snapshot data based on the label field value only, ignoring the SDI bit field value.

If no message has been received for the specified channel/label since the last initialization of the device, a data value of zero will be returned.

Arguments

CDEV_BOARD_TYPE board (input) Device this routine is to access.
Valid range is 0-127.

CDEV_CHAN_TYPE channel (input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.

CDEV_PARM_USI_TYPE label (input) The label value of interest.

CDEV_PARM_VOID_PTR_TYPE data (output) Location to store 32-bit ARINC data.

pCEI_CHAR seq_num (output) Unsupported legacy parameter.

AR_GET_LATEST_T

Syntax

CEI_UINT32 ar_get_latest_t (CDEV_BOARD_TYPE board, CDEV_CHAN_TYPE channel, CDEV_PARM_USI_TYPE label, pCEI_UINT32 data, TIME_TAG_TYPE * timeTag)

Description

This routine returns the latest ARINC 429 message and time-stamp received for the specified channel/label combination from the snapshot buffer. When using this routine, the host application should set the snapshot storage mode to **label field only**, (see the documentation on the routine AR_SET_DEVICE_CONFIG, for the configuration option ARU_ACCESS_SNAPSHOT_BUFFER). This sets up the x30 device to store the ARINC message and time-stamp in the snapshot buffer based on the label field value only, ignoring the SDI bit field value.

If no message has been received for the specified channel/label since the last initialization of the device, a data value of zero is returned for the message and time-stamp. If the supplied timeTag parameter is NULL, no time-stamp information will be returned.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	Invalid <i>label</i> or null <i>data</i> parameter.
ARS_INVHARVAL	Channel is not available on device.

Arguments

CDEV_BOARD_TYPE board	(input) Device this routine is to access. Valid range is 0-127.
CDEV_CHAN_TYPE channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
CDEV_PARM_USI_TYPE label	(input) The label value of interest.
pCEI_UINT32 data	(output) Location to store ARINC message.
TIME_TAG_TYPE * timeTag	(output) The address that is to receive the 64-bit message time-stamp, the format of which is determined by the current API time-tag format.

AR_GETNEXT

Syntax

CDEV_API_RET_TYPE ar_getnext (CDEV_BOARD_TYPE board, CDEV_CHAN_TYPE channel, CDEV_PARM_VOID_PTR_TYPE destination)

Description

This routine retrieves the next unread message from the specified receive channel. If no message is present in the receiver FIFO buffer upon invocation, this routine polls the buffer waiting for the presence of a received message for up to one-half second. If no message is present after one-half second, a time-out status is returned.

Return Value

ARS_GOTDATA	A message has been retrieved.
ARS_CHAN_TIMEOUT	No message was available or received during the time-out period.
ARS_BAD_MESSAGE	Receipt of an invalid message has been detected on this receiver; a message was returned if available in the buffer.
ARS_RX_BUFFER_OVERRUN	A receive buffer overrun was detected, no message was returned, and the receive buffer is potentially stale.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	Invalid <i>destination</i> parameter.
ARS_INVHARVAL	Channel is not available on device.

Arguments

CDEV_BOARD_TYPE board	(input) Device this routine is to access. Valid range is 0-127.
CDEV_CHAN_TYPE channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
CDEV_PARM_VOID_PTR_TYPE destination	(output) The address that is to receive the message.

AR_GETNEXTT

Syntax

CDEV_API_RET_TYPE ar_getnextt (CDEV_BOARD_TYPE board, CDEV_CHAN_TYPE channel, CDEV_PARM_VOID_PTR_TYPE destination, CDEV_PARM_VOID_PTR_TYPE timetag)

Description

This routine retrieves the next unread message and scaled 32-bit time-stamp from the specified receive channel. If no message is present in the receiver FIFO buffer when invoked this routine polls the buffer waiting for the presence of a received message for up to one-half second. If no message is present after one-half second, a time-out status is returned.

If the *timetag* parameter is not NULL, the 32-bit translation of the 64-bit message time-stamp will be returned, scaled to the active legacy 32-bit time-tag mode, (1 millisecond resolution by default).

Return Value

ARS_GOTDATA	A message has been retrieved.
ARS_CHAN_TIMEOUT	No message was available or received during the time-out period.
ARS_BAD_MESSAGE	Receipt of an invalid message has been detected on this receiver; a message was returned if available in the buffer.
ARS_RX_BUFFER_OVERRUN	A receive buffer overrun was detected, no message was returned, and the receive buffer is potentially stale.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	Invalid <i>destination</i> parameter.
ARS_INVHARVAL	Channel is not available on device.

Arguments

CDEV_BOARD_TYPE board (input) Device this routine is to access. Valid range is 0-127.

CDEV_CHAN_TYPE channel (input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.

CDEV_PARM_VOID_PTR_TYPE destination (output) The address that is to receive the message.

CDEV_PARM_VOID_PTR_TYPE timetag (output) The address that is to receive the 32-bit time-tag associated with the data, (resolution is programmable). If the merged receive mode is active for the specified channel, the upper five bits of the 32-bit time-tag word contain the receive channel number on which the data was received.

AR_GETNEXT_XT

Syntax

CDEV_API_RET_TYPE ar_getnext_xt (CDEV_BOARD_TYPE board, CDEV_CHAN_TYPE channel, pCEI_UINT32 data, pAR_TIMETAG_TYPE timeTagRef)

Description

This routine retrieves the next unread message and the associated time-tag structure from the specified receive channel. If no message is present in the receiver FIFO buffer when invoked, this routine polls the buffer waiting for the presence of a received message for up to one-half second. If no message is present after one-half second, a time-out status is returned.

If the *timeTagRef* parameter is not NULL, the time-tag structure containing the message time-stamp will be returned.

Return Value

ARS_GOTDATA	A message has been retrieved.
ARS_CHAN_TIMEOUT	No message was available or received during the time-out period.
ARS_BAD_MESSAGE	Receipt of an invalid message has been detected on this receiver; a message was returned if available in the buffer.
ARS_RX_BUFFER_OVERRUN	A receive buffer overrun was detected, no message was returned, and the receive buffer is potentially stale.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	Invalid <i>data</i> parameter.
ARS_INVHARVAL	Channel is not available on device.

Arguments

CDEV_BOARD_TYPE board	(input) Device this routine is to access. Valid range is 0-127.
CDEV_CHAN_TYPE channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
pCEI_UINT32 data	(output) The address that is to receive the message.
pAR_TIMETAG_TYPE timeTagRef	(input)

The `timeTagRef` structure member `timeTagFormat` specifies the desired format of the time-tag value returned in the `timeTag` structure member. Valid values are:

AR_TIMETAG_EXT_IRIG_64BIT	0
AR_TIMETAG_INT_USEC_64BIT	1
AR_TIMETAG_HOST_USEC_64BIT	2
AR_TIMETAG_INT_20USEC_32BIT	3
AR_TIMETAG_INT_MSEC_32BIT	4
AR_TIMETAG_SYNC_1553_CH1	11
AR_TIMETAG_SYNC_1553_CH2	12
AR_TIMETAG_SYNC_1553_CH3	13
AR_TIMETAG_SYNC_1553_CH4	14
AR_TIMETAG_USE_PROGRAMMED_MODE	99

(output)

The address that is to receive the time-tag data structure associated with the message.

AR_GET_RX_CHANNEL_STATUS

Syntax	CEI_UINT32 ar_get_rx_channel_status (CDEV_BOARD_TYPE board, CDEV_CHAN_TYPE channel, CDEV_PARM_SI_PTR_TYPE channelStatus, CDEV_PARM_SI_PTR_TYPE messageCount)	
Description	This routine returns the current status of the specified receive channel buffer. If either an ARINC 429 protocol error or buffer overflow bit was set in the receive channel buffer status register, it is cleared on return from this routine.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	Invalid <i>channelStatus</i> or <i>messageCount</i> parameter.
	ARS_INVHARVAL	ARINC 429 channel is not available on the device.
Arguments	CDEV_BOARD_TYPE board	(input) Device to access. Valid range is 0-127.
	CDEV_CHAN_TYPE channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count. Channel 32 is allowed to access the ARINC 717 receive channel status, and 63 to access the Merged Receive Buffer status.
	CDEV_PARM_SI_PTR_TYPE channelStatus	(output) Location to store the bitwise representation of the current receiver buffer status bits. The Status Register Bit assignments are defined as follows:
	b0 - AR_BUFFER_MSG_AVAILABLE (1)	Set indicates at least one message is ready to read from the buffer. Clear indicates the buffer is empty.
	b1 - AR_INVALID_MSG_DETECTED (2)	

Set indicates at least one ARINC 429 message protocol error was detected since either this routine was previously invoked or a message was last retrieved from the buffer. Clear indicates no protocol error has been encountered.

b2 - AR_BUFFER_OVERFLOW_DETECTED (4)

Set indicates the respective receive channel encountered a message buffer overflow since this routine was previously invoked. Clear indicates no buffer overflow has been encountered.

CDEV_PARM_SI_PTR_TYPE messageCount (output) Location to store the number of messages currently available in the respective receive buffer, acquired from the respective receive channel status register. This value will only be valid if b0 in the *channelStatus* return value is set, and has a valid range of 1 - 2047.

AR_GET_RX_COUNT

Syntax

CEI_UINT32 ar_get_rx_count (CDEV_BOARD_TYPE board,
CDEV_CHAN_TYPE channel)

Description

The device maintains a count of the number of ARINC data messages received over the interface for each channel since the device was last initialized (see AR_OPEN). This routine returns that number.

If the API routine AR_CLR_RX_COUNT has been invoked by the host application prior to this routine's invocation, the API logs the current value of the message count and returns the difference between that value and the value read from the device upon invocation.

Return Value

Current count of ARINC messages received on the specified channel.

Arguments

CDEV_BOARD_TYPE board (input) Device this routine is to access.
Valid range is 0-127.

CDEV_CHAN_TYPE channel (input) Specifies which receive channel
this routine is to access.

AR_GET_SNAP_DATA

Syntax

CEI_UINT32 ar_get_snap_data (CDEV_BOARD_TYPE board, CDEV_CHAN_TYPE channel, CDEV_PARM_USI_TYPE label, CDEV_PARM_USI_TYPE sdi, pCEI_UINT32 data)

Description

This routine returns the latest ARINC 429 message received for the specified channel/label combination from the snapshot buffer.

When using this routine, the host application should set the snapshot storage mode to **label/sdi storage**, (see the documentation on the routine AR_SET_DEVICE_CONFIG, for the configuration option ARU_ACCESS_SNAPSHOT_BUFFER). This sets up the x30 device to store snapshot data based on the label field value in combination with the SDI bit field value.

If no message has been received for the specified channel/label since the last initialization of the device, a data value of zero is returned for the message.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	Invalid <i>label</i> , <i>sdi</i> , or null <i>data</i> parameter.
ARS_INVHARVAL	Channel is not available on device.

Arguments

CDEV_BOARD_TYPE board	(input) Device this routine is to access. Valid range is 0-127.
CDEV_CHAN_TYPE channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
CEI_UINT16 label	(input) The label value of interest.
CEI_UINT16 sdi	(input) The SDI value of interest.
pCEI_UINT32 data	(output) Location to store 32-bit ARINC data.

AR_GET_STATUS

Syntax	CEI_UINT32 ar_get_status (CDEV_BOARD_TYPE board, CDEV_PARM_SSI_PTR_TYPE state)	
Description	This routine returns the state of the FIFO Data Available bit for up to 16 receivers in a bitwise 16-bit value.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
Arguments	CDEV_BOARD_TYPE board	(input) Device to access. Valid range is 0-127.
	CDEV_PARM_SSI_PTR_TYPE state	(output) Location to store the receiver FIFO status. The Status Register Bit Assignments are defined as follows, ("1" indicates Data Available, "0" indicates No Data Available):
	b0	- ARINC 429 Receiver 1
	b1	- ARINC 429 Receiver 2
	...	
	b14	- ARINC 429 Receiver 15
	b15	- ARINC 429 Receiver 16

AR_GET_STORAGE_MODE

Syntax	CDEV_API_RET_TYPE ar_get_storage_mode (CDEV_BOARD_TYPE board, CDEV_PARM_SSI_PTR_TYPE mode)	
Description	<p>This routine is designed to provide compatibility with the CEI-x20 ARINC device API. It returns the current state of the API receive storage mode. When the API receive storage mode is <i>buffered</i>, each receiver is assigned an independent circular buffer for data storage (merged mode is disabled). When the storage mode is <i>merged</i>, all receivers are set to enable merged receive mode and data received on each is stored in the merged FIFO buffer. Each receive data API routine processes the active storage mode internally, acquiring data from the appropriate buffer. Since each receive channel can be independently programmed to store data in <i>buffered</i> or <i>merged</i> mode through AR_SET_DEVICE_CONFIG, this routine should only be used in conjunction with the AR_SET_STORAGE_MODE routine.</p>	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
Arguments	<p>CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.</p> <p>CDEV_PARM_SSI_PTR_TYPE mode (output) The address that is to receive the state of the current API storage mode. Valid return values for this parameter are:</p> <p style="margin-left: 40px;">ARU_BUFFERED (0) buffered receive mode</p> <p style="margin-left: 40px;">ARU_MERGED (2) merged receive mode</p>	

AR_GET_TIME

Syntax

CDEV_API_RET_TYPE ar_get_time (CDEV_BOARD_TYPE board, CDEV_PARM_SSI_TYPE format, pAR_TIMETAG_TYPE timeTag)

Description

This routine returns the current time reference value scaled from either the CEI-x30 device internal 64-bit timer, the most recently received IRIG timer reference, or the specified 1553 channel (for multi-protocol boards), as specified via the *format* parameter.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	An invalid format parameter value was provided.
ARS_INVHARVAL	IRIG or MIL-STD-1553 time format was requested via the <i>format</i> parameter but is not available on the specified device.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.

Arguments

CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

CDEV_PARM_SSI_TYPE format (input) Time format requested. Valid options are:

AR_TIMETAG_EXT_IRIG_64BIT	0
AR_TIMETAG_INT_USEC_64BIT	1
AR_TIMETAG_HOST_USEC_64BIT	2
AR_TIMETAG_INT_20USEC_32BIT	3
AR_TIMETAG_INT_MSEC_32BIT	4
AR_TIMER_X20_COMPAT_32BIT	6
AR_TIMETAG_SYNC_1553_CH1	11
AR_TIMETAG_SYNC_1553_CH2	12
AR_TIMETAG_SYNC_1553_CH3	13
AR_TIMETAG_SYNC_1553_CH4	14

pAR_TIMETAG_TYPE timeTag

(output) Current device timer or translated IRIG sample value. The *timeTag.timeTag* structure member will be defined as follows based on the supplied *format* parameter value:

AR_TIMETAG_EXT_IRIG_64BIT - 64-bit IRIG sample time in microseconds since beginning of current year. The returned *timeTag.R.referenceTimeTag* structure member will contain the board internal timer-referenced time-stamp assigned when the last bit of the IRIG sample was processed by the CEI-x30 IRIG receiver.

AR_TIMETAG_INT_USEC_64BIT - 64-bit internal board timer in microseconds.

AR_TIMETAG_HOST_USEC_64BIT - 64-bit host operating system time scaled to have a 1 microsecond resolution.

AR_TIMETAG_INT_20USEC_32BIT - 32-bit internal board timer in microseconds.

AR_TIMETAG_INT_MSEC_32BIT - 32-bit internal board timer scaled to have a 20 microsecond resolution.

AR_TIMER_X20_COMPAT_32BIT - 32-bit internal board timer scaled to have a 1 millisecond resolution.

AR_TIMETAG_SYNC_1553_CH1 – 64-bit timer value synchronized to the 1553 channel1 timer, scaled to have a 1 microsecond resolution.

AR_TIMETAG_SYNC_1553_CH2 – 64-bit timer value synchronized to the 1553 channel2 timer, scaled to have a 1 microsecond resolution.

AR_TIMETAG_SYNC_1553_CH3 – 64-bit timer value synchronized to the 1553 channel3 timer, scaled to have a 1 microsecond resolution.

AR_TIMETAG_SYNC_1553_CH4 – 64-bit timer value synchronized to the 1553 channel4 timer, scaled to have a 1 microsecond resolution.

AR_GET_TIMERCNTL

Syntax	CEI_UINT32 ar_get_timercntl (CDEV_BOARD_TYPE board)
Description	This routine is provided for legacy support of the CEI-x20 ARINC API, returning the current 32-bit, 1 millisecond resolution time reference value based on the current application-specified timer mode, (specified through AR_SET_CONFIG using the attribute ARU_RX_TIMETAG_MODE). If the current timer mode is assigned to any 64-bit timer, the least-significant 32-bits of the internal device timer will be returned (this applies to IRIG, host, or internal timer). If the current timer mode is assigned to either of the 32-bit CEI-x20 API compatibility or 20 microsecond (IP-AVIONICS) resolution modes, the respective 32-bit adjusted timer value will be returned.
Return Value	The 32-bit timer value.
Arguments	CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

AR_GETWORD

Syntax

CDEV_API_RET_TYPE ar_getword (CDEV_BOARD_TYPE board, CDEV_CHAN_TYPE channel, CDEV_PARM_VOID_PTR_TYPE destination)

Description

This routine retrieves the next unread message from the specified receive channel. If it successfully returns data message, there may or may not be more messages in the buffer. It only means there was at least one message in the buffer. Subsequent calls would be required to determine if more messages are available in the buffer. If this routine returns a status value of ARS_NODATA, the buffer is empty.

The channel value passed to this routine corresponds to the ARINC 429 receive channel index, starting with zero. If that value exceeds the 429 receive channel count and an ARINC 717 receiver exists, the ARINC 717 receiver is used as the designated channel buffer.

Return Value

ARS_GOTDATA	A message has been retrieved.
ARS_NODATA	The receive buffer was empty.
ARS_BAD_MESSAGE	Receipt of an invalid message has been detected on this receiver; a message was returned if available in the buffer.
ARS_RX_BUFFER_OVERRUN	A receive buffer overrun was detected, no message was returned, and the receive buffer is potentially stale.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	A null data parameter value was provided.
ARS_INVHARVAL	Channel is not available on device.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.

Arguments

CDEV_BOARD_TYPE board	(input) Device this routine is to access. Valid range is 0-127.
CDEV_CHAN_TYPE channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
CDEV_PARM_VOID_PTR_TYPE destination	(output) The address that is to receive the message. The format of the

32-bit value is dependent on the protocol assigned to the respective receive channel:

ARINC 429/575 Data Format

31	30 – 10	9 - 8	7 - 0
Parity Indication or ARINC Data MSB	ARINC Data	SDI bits or ARINC Data bits 0-1	ARINC Label (MSB – LSB)

ARINC 717 Data Format

31 – 16	15	14	13 - 12	11 – 0
Unused	Sync Indication	Unused	Sub-frame Identification	Data Word

AR_GETWORDT

Syntax

CDEV_API_RET_TYPE ar_getwordt (CDEV_BOARD_TYPE board, CDEV_CHAN_TYPE channel, CDEV_PARM_VOID_PTR_TYPE destination, CDEV_PARM_VOID_PTR_TYPE timetag)

Description

This routine retrieves the next unread message from the specified receive channel. If it successfully returns data message, there may or may not be more messages data in the buffer. It means only that there was at least one message in the buffer. Subsequent calls are required to determine if more messages are available in the buffer. If this routine returns a status value of ARS_NODATA, the buffer is empty.

Return Value

ARS_GOTDATA	A message has been retrieved.
ARS_NODATA	The receive buffer was empty.
ARS_BAD_MESSAGE	Receipt of an invalid message has been detected on this receiver; a message was returned if available in the buffer.
ARS_RX_BUFFER_OVERRUN	A receive buffer overrun was detected, no message was returned, and the receive buffer is potentially stale.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	A null <i>data</i> parameter value was provided.
ARS_INVHARVAL	Channel is not available on device.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.

Arguments

CDEV_BOARD_TYPE board (input) Device this routine is to access. Valid range is 0-127.

CDEV_CHAN_TYPE channel (input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.

CDEV_PARM_VOID_PTR_TYPE destination (output) The address that is to receive the message. The format of the 32-bit value is dependent on the protocol assigned to the respective receive channel. See the chapter “CEI-x30 Hardware Interface” in the *CEI-x30 Product Line*

Hardware Manual for more details on receive buffer message formats.

CDEV_PARM_VOID_PTR_TYPE timetag (output) The address that is to receive the 32-bit time-tag associated with the data, (resolution is programmable). If the merged receive mode is active for the specified channel, the upper five bits of the 32-bit time-tag word contain the receive channel number on which the data was received.

AR_GETWORD_XT

Syntax

CDEV_API_RET_TYPE ar_getword_xt (CDEV_BOARD_TYPE board, CDEV_CHAN_TYPE channel, pCEI_UINT32 data, pAR_TIMETAG_TYPE timeTagRef)

Description

This routine retrieves the next unread message and the associated time-tag structure from the specified receive channel. If it successfully returns a message, there may or may not be more messages in the buffer. It means only that there was at least one message in the buffer. Subsequent calls are required to determine if more messages are available in the buffer. If this routine returns a status value of ARS_NODATA, the buffer is empty.

Return Value

ARS_GOTDATA	A message has been retrieved.
ARS_NODATA	The receive buffer was empty.
ARS_BAD_MESSAGE	Receipt of an invalid message has been detected on this receiver; a message was returned if available in the buffer.
ARS_RX_BUFFER_OVERRUN	A receive buffer overrun was detected, no message was returned, and the receive buffer is potentially stale.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	A null <i>data</i> parameter value was provided.
ARS_INVHARVAL	Channel is not available on device.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.

Arguments

CDEV_BOARD_TYPE board	(input) Device this routine is to access. Valid range is 0-127.
CDEV_CHAN_TYPE channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
pCEI_UINT32 data	(output) The address that is to receive the 32-bit message.
pAR_TIMETAG_TYPE timeTagRef	(input) The timeTagRef structure member <i>timeTagFormat</i> specifies the desired format

of the time-tag value returned in the *timeTag* structure member. Valid values are:

AR_TIMETAG_EXT_IRIG_64BIT	0
AR_TIMETAG_INT_USEC_64BIT	1
AR_TIMETAG_HOST_USEC_64BIT	2
AR_TIMETAG_INT_20USEC_32BIT	3
AR_TIMETAG_INT_MSEC_32BIT	4
AR_TIMETAG_SYNC_1553_CH1	11
AR_TIMETAG_SYNC_1553_CH2	12
AR_TIMETAG_SYNC_1553_CH3	13
AR_TIMETAG_SYNC_1553_CH4	14
AR_TIMETAG_USE_PROGRAMMED_MODE	99

(output)

The address that is to receive the time-tag data structure associated with the message.

AR_GO

Syntax	CDEV_API_RET_TYPE ar_go (CDEV_BOARD_TYPE board)	
Description	This routine assigns the global enable register Global Enable bit to be <i>enabled</i> for the specified device. All message processing on the device is activated when this routine executes.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
Arguments	CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.	

AR_HW_INTERRUPT_BUFFER_READ

Syntax	CEI_UINT32 ar_hw_interrupt_buffer_read (CDEV_BOARD_TYPE board, pCEI_UINT32 numberOfWords, pCEI_UINT32 data)	
Description	<p>This routine provides read access to the local API copy of the CEI-x30 device interrupt queue. The local API copy is filled by hardware interrupt processing within the default API ISR. If the host application replaces the default API ISR with a custom ISR, this routine is not usable.</p> <p>Each time this routine is invoked, the specified number of queue entries is read from the buffer region starting at the location last referenced by the API in a previous invocation and ending at the location written by the most recent execution of the default API interrupt service routine.</p>	
Return Value	ARS_GOTDATA	Routine execution was successful and one or more interrupt buffer entries were returned.
	ARS_NODATA	No unread interrupt buffer entries were available or returned.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	A NULL <i>data</i> buffer pointer was supplied.
Arguments	CDEV_BOARD_TYPE board	(input) Device to access. Valid range is 0-127.
	pCEI_UINT32 numberOfWords	(input/output) As an input, this argument specified the number of interrupt buffer entries to read and return. As an output, this argument indicates the number of interrupt buffer entries actually read, if there were fewer unread entries available than what was requested.
	pCEI_UINT32 data	(output) The location to store the interrupt buffer entries read.

AR_INTERRUPT_QUEUE_READ

Syntax	CEI_INT32 ar_interrupt_queue_read (CDEV_BOARD_TYPE board, pCEI_UINT32 numberOfWords, pCEI_UINT32 data)	
Description	This routine provides read access directly to the CEI-x30 device hardware interrupt queue. Each time this routine is invoked, the specified number of queue entries will be read from the interrupt queue starting at the location referenced by last invocation of this routine, and ending at the location indicated by the device interrupt queue pointer.	
Return Value	ARS_GOTDATA	Routine execution was successful and one or more interrupt queue entries were returned.
	ARS_NODATA	No unread interrupt queue entries were available or returned.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	A NULL <i>data</i> buffer pointer was supplied.
Arguments	CDEV_BOARD_TYPE board	(input) Device to access. Valid range is 0-127.
	pCEI_UINT32 numberOfWords	(input/output) As an input, this argument specified the number of interrupt buffer entries to read and return. As an output, this argument indicates the number of interrupt buffer entries actually read, if there were fewer unread entries available in the interrupt queue than what was requested.
	pCEI_UINT32 data	(output) The location to store the interrupt queue entries read.

AR_INITIALIZE_API

Syntax

CDEV_API_RET_TYPE ar_initialize_api (CDEV_BOARD_TYPE board)

Description

This routine acquires the resources for the device and initializes API local variables. With the exception of the RAR-PCIE and RAR15-XMC-XT boards, it also downloads the CEI-x30 firmware program, resetting the device to an initial power-up state. While this routine is available for use, applications should utilize the AR_OPEN routine to open a session with a CEI-x30 board.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_DRIVERFAIL	The device driver failed to open a session with the device, either because the device is not properly installed in the host system or a resource conflict is inhibiting device driver initialization.
ARS_BOARD_MUTEX	For Windows only, indicates creation of the Board Lock mutex timed-out/failed.
ARS_FW_NOT_SUPPORTED	The specified flash-based device firmware is not compatible with this API version.
ARS_BADLOAD	The device driver session was opened successfully but the device firmware download failed.
ARS_HW_DETECT	The device driver session was opened but the detected device is not recognized as a CEI-x30 product.
ARS_FAILURE	For <i>Windows</i> only, indicates library CEI_Install.dll failed to load in memory; for <i>Linux</i> only, indicates creation of the Board Lock mutex timed-out/failed; for <i>VxWorks</i> only, indicates a legacy PCI driver open session failure.
API_VXB_METHOD_ERR	For VxWorks VxBus driver usage only, indicates a device open session failure.

Arguments

CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

AR_INITIALIZE_DEVICE

Syntax	CDEV_API_RET_TYPE ar_initialize_device (CDEV_BOARD_TYPE board)	
Description	<p>This routine performs a non-destructive SRAM memory test, flushes the receiver FIFO buffers, and assigns the default state of all channel configuration registers. It also initializes the label filtering and message scheduling features. The default state of the CEI-x30 board is defined as follows:</p> <ul style="list-style-type: none"> ■ ARINC 429 Transmitter FIFOs enabled, speed set for 100Kbps and ODD parity enabled. ■ ARINC 429 Receiver FIFOs enabled, speed set for 100Kbps, ODD parity, Merged Mode disabled, and Internal Wrap disabled. ■ ARINC 717 Transmitter and Receiver FIFOs disabled, set for BPRZ encoding at 768BPS (64-word subframe), auto-detect enabled, and Internal Wrap disabled. ■ Message Scheduler enabled, no messages defined. ■ All receive label filtering disabled 	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_MEMWRERR	Device SRAM test write/read/verify failure.
	ARS_BAD_STATIC	Device register write/read/verify failure.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
Arguments	CDEV_BOARD_TYPE board	(input) Device to access. Valid range is 0-127.

AR_HW_INTERRUPT_BUFFER_READ

Syntax	CEI_INT32 ar_hw_interrupt_buffer_read (CDEV_BOARD_TYPE board, pCEI_UINT32 numberOfWords, pCEI_UINT32 data)	
Description	<p>This routine provides read access to the local API copy of the CEI-x30 device interrupt queue. The local API copy of the current device interrupt queue is maintained by hardware interrupt processing within the default API interrupt service routine (ISR). If the host application replaces the default API ISR with a custom ISR, this routine is not usable.</p> <p>Each time this routine is invoked, the specified number of queue entries will be read from the buffer region starting at the location last referenced by the API and ending at the location referenced by the interrupt queue pointer. If fewer than the requested number of entries are found, only those entries available will be returned.</p>	
Return Value	ARS_GOTDATA	At least one interrupt queue entry has been retrieved.
	ARS_NODATA	No unread interrupt queue entries are available.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	Invalid or null <i>data</i> buffer parameter.
Arguments	CDEV_BOARD_TYPE board	(input) Device to access. Valid range is 0-127.
	pCEI_UINT32 numberOfWords	(input/output) As an input this parameter specifies the number of interrupt queue entries to read; as an output this parameter indicates how many interrupt queue entries were actually copied to the <i>data</i> array.
	pCEI_UINT32 data	(output) An array referencing the location to store the requested 32-bit interrupt queue entry/entries. Valid Interrupt Queue entry values range from 64 to 95, and 255. The value 64 indicates receive channel 0 label filter triggered, where 95 indicates receive channel 31 label filter triggered, and 255 is reserved for host-triggered interrupt.

AR_INTERRUPT_QUEUE_READ

Syntax	CEI_UINT32 ar_interrupt_queue_read (CDEV_BOARD_TYPE board, pCEI_UINT32 numberOfWords, pCEI_UINT32 data)	
Description	This routine provides read access directly to the CEI-x30 device interrupt queue. Each time you invoke this routine, the specified number of queue entries is read from the buffer region starting at the location last referenced by the host/API from this routine. If fewer than the requested number of entries are found, only those entries available are returned.	
Return Value	ARS_GOTDATA	At least one interrupt queue entry has been retrieved.
	ARS_NODATA	No unread interrupt queue entries are available.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	Invalid or null <i>data</i> buffer parameter.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
Arguments	CDEV_BOARD_TYPE board	(input) Device to access. Valid range is 0-127.
	pCEI_UINT32 numberOfWords	(input/output) As an input this parameter specifies the number of interrupt queue entries to read; as an output this parameter indicates how many interrupt queue entries were actually copied to the <i>data</i> array.
	pCEI_UINT32 data	(output) An array referencing the location to store the requested 32-bit interrupt queue entry/entries. Valid Interrupt Queue entry values range from 64 to 95, and 255. The value 64 indicates receive channel 0 label filter triggered, where 95 indicates receive channel 31 label filter triggered, and 255 is reserved for host-triggered interrupt.

AR_LABEL_FILTER

Syntax

CDEV_API_RET_TYPE ar_label_filter (CDEV_BOARD_TYPE board, CDEV_CHAN_TYPE channel, CDEV_PARM_USI_TYPE label, CDEV_PARM_USI_TYPE action)

Description

CEI-x30 devices support the ability to filter ARINC 429 messages by the 8-bit label value. Once filtering has been enabled for a specified channel/label combination, data received with that label value would be discarded until label filtering for the specified label has been disabled. Label filtering is disabled for all labels by default. Label filtering changes are effective immediately on completion of this routine.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	An invalid <i>label</i> or <i>action</i> value was provided.
ARS_INHARVAL	The specified <i>channel</i> does not support label filtering.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.

Arguments

CDEV_BOARD_TYPE board	(input) Device to access. Valid range is 0-127.
CDEV_CHAN_TYPE channel	(input) channel label filter table this routine is to access. The valid range is 0 to one less than the installed receive channel count.
CDEV_PARM_USI_TYPE label	(input) The label of interest. Valid range is 0-255. Also valid is ARU_ALL_LABELS (511), which invokes the action for all labels on the specified channel.
CDEV_PARM_USI_TYPE action	(input) Enable or disable filtering for this combination of board/channel/label. Valid values are: ARU_FILTER_ON (1) enable filtering ARU_FILTER_OFF (0) disable filtering (default state is to not filter any labels).

AR_LOADSLV

Syntax

CDEV_API_RET_TYPE ar_loadslv (CDEV_BOARD_TYPE board, CEI_UINT32 base_seg, CEI_INT32 base_port, CEI_UINT16 ram_size)

Description

This is a legacy routine providing a backward compatible method to open a session with a CEI-x30 board. It has been superseded by the routine AR_OPEN.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_DRIVERFAIL	The device driver failed to open a session with the device, either because the device is not properly installed in the host system or a resource conflict is inhibiting device driver initialization.
ARS_BOARD_MUTEX	For Windows only, indicates creation of the Board Lock mutex timed-out/failed.
ARS_FW_NOT_SUPPORTED	The specified flash-based device firmware is not compatible with this API version.
ARS_BADLOAD	The device driver session was opened successfully but the device firmware download failed.
ARS_HW_DETECT	The device driver session was opened but the detected device is not recognized as a CEI-x30 product.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_ERR_SH_MEM_OBJ	For <i>Windows multi-process application use only</i> , indicates the API failed to allocate a shared memory block.
ARS_ERR_SH_MEM_MAP	For <i>Windows multi-process application use only</i> , indicates the API failed to map shared memory block.
ARS_WRAP_DROP_FAIL	ARINC 429 wrap test data missing.
ARS_WRAP_DATA_FAIL	ARINC 429 wrap test data pattern mismatch.

ARS_WRAP_FLUSH_FAIL	Unknown external messages were received during the internal wrap test execution.
ARS_MEMWRERR	Device SRAM test write/read/verify failure.
ARS_FAILURE	For <i>Windows</i> only, indicates library CEI_Install.dll failed to load in memory; for <i>Linux multi-process application use</i> only, indicates the API either failed to allocate a shared memory block or create the Board Lock semaphore; for <i>VxWorks</i> only, indicates a legacy PCI driver open session failure.
API_VXB_METHOD_ERR	For <i>VxWorks VxBus driver usage</i> only, indicates a device open session failure.

Arguments

CDEV_BOARD_TYPE board	(input) Device to access. Valid range is 0-127.
CEI_UINT32 base_seg	(input) This parameter is ignored, (supplied for ARINC API compatibility only).
CEI_INT32 base_port	(input) This parameter is ignored, (supplied for ARINC API compatibility only).
CEI_UINT16 ram_size	(input) This parameter is ignored, (supplied for ARINC API compatibility only).

AR_MODIFY_MSG

Syntax

CDEV_API_RET_TYPE ar_modify_msg (CDEV_BOARD_TYPE board, CDEV_CHAN_TYPE channel, CDEV_PARM_SSI_TYPE msgNumber, CDEV_PARM_SSI_TYPE rate, CDEV_PARM_SI_TYPE data)

Description

This routine modifies an existing 32-bit ARINC message for periodic retransmission, originally created through use of the AR_DEFINE_MSG or AR_DEFINE_MSG_BLOCK API routines.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	An invalid <i>channel</i> value was provided.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_INVHARVAL	Message scheduling is not supported on the specified channel.
ARS_FAILURE	The supplied message table index exceeds the available number of table entries.

Arguments

CDEV_BOARD_TYPE board	(input) Device to access. Valid range is 0-127.
CDEV_CHAN_TYPE channel	(input) Channel message scheduler table this routine is to access. The valid range is 0 to one less than the number of installed transmit channels.
CDEV_PARM_SSI_TYPE msgNumber	(input) The unique message scheduler table entry index assigned to this message, as returned for the respective message from the routine AR_DEFINE_MSG or AR_DEFINE_MSG_BLOCK. Valid range is 0-2047.
CDEV_PARM_SSI_TYPE rate	(input) Periodic transmission rate, defined in milliseconds. A rate value of zero will disable message transmission for this message scheduler table entry and make this entry available for reuse on the next invocation of AR_DEFINE_MSG or AR_DEFINE_MSG_BLOCK.

CDEV_PARM_SI_TYPE data (input) The updated 32-bit ARINC message to transmit.

AR_MODIFY_MSG_BLOCK

Syntax	CDEV_API_RET_TYPE ar_modify_msg_block (CEI_INT32 numberOfEntries, pAR_SCHEDULED_MSG_ENTRY_TYPE messageEntry)	
Description	This routine provides a method to modify the channel assignment or rate and data values on a series of 32-bit ARINC messages previously defined for periodic retransmission via AR_DEFINE_MSG_BLOCK.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	An invalid <i>channel</i> structure member value was provided.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVHARVAL	Message scheduling is not supported on the specified channel.
	ARS_FAILURE	A supplied message table index exceeds the available number of table entries.
Arguments	<p>CEI_INT32 numberOfEntries (input) The number of entries to modify using the subsequent structure pointer parameter, messageEntry.</p> <p>pAR_SCHEDULED_MSG_ENTRY_TYPE messageEntry (input)</p> <p>array of structures of message definition content, defined as follows:</p> <p>unsigned long messageIndex The unique message scheduler table entry index assigned to this message. This messageIndex structure member will have been defined in a previous invocation of AR_DEFINE_MSG_BLOCK. Valid range is 0-2047.</p> <p>unsigned long board Device to access. Valid range is 0-127.</p>	

unsigned long channel	Which channel portion of the message scheduler table this routine is to access. The valid range is 0 to one less than the number of installed transmit channels.
unsigned long rate	Periodic transmission rate, in milliseconds. A rate value of zero will disable message transmission.
unsigned long start	Not supported during message modification.
unsigned long txCount	Not supported during message modification.
unsigned long data	The 32-bit ARINC message to transmit.

AR_NUM_RCHANS

Syntax	CDEV_CHAN_TYPE ar_num_rchans (CDEV_BOARD_TYPE board)
Description	This routine retrieves the number of receive channels installed on the specified device.
Return Value	Any value less than 40 number of installed receive channels. ARS_INVBOARD An uninitialized board or invalid <i>board</i> value was provided.
Arguments	CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

AR_NUM_XCHANS

Syntax CDEV_CHAN_TYPE ar_num_xchans (CDEV_BOARD_TYPE board)

Description This routine retrieves the number of transmit channels installed on the specified device.

Return Value

Any value less than 40	number of installed transmit channels.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.

Arguments CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

AR_OPEN

Syntax

CDEV_API_RET_TYPE ar_open (CDEV_BOARD_TYPE board)

Description

This routine opens a session and acquires the memory resources allocated to the device, downloads the firmware to the FPGA, and invokes an initialization/reset procedure. Following API and device initialization, optional invocation of AR_BOARD_TEST may provide verification of internal message wrap operation, (execution controlled via invocation of AR_BYPASS_WRAP_TEST).

See the routine descriptions under AP_INITIALIZE_API and AR_INITIALIZE_DEVICE for details regarding the default setup of the API and the device following execution of this routine.

If any portion of the initialization fails or the board is not detected, a status other than ARS_NORMAL is returned.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_DRIVERFAIL	The device driver failed to open a session with the device, either because the device is not properly installed in the host system or a resource conflict is inhibiting device driver initialization.
ARS_BOARD_MUTEX	For Windows only, indicates creation of the Board Lock mutex timed-out/failed.
ARS_FW_NOT_SUPPORTED	The specified flash-based device firmware is not compatible with this API version.
ARS_BADLOAD	The device driver session was opened successfully but the device firmware download failed.
ARS_HW_DETECT	The device driver session was opened but the detected device is not recognized as a CEI-x30 product.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_ERR_SH_MEM_OBJ	For <i>Windows multi-process application use</i> only, indicates the API failed to allocate a shared memory block.

ARS_ERR_SH_MEM_MAP For *Windows multi-process application use* only, indicates the API failed to map shared memory block.

ARS_WRAP_DROP_FAIL ARINC 429 wrap test data missing.

ARS_WRAP_DATA_FAIL ARINC 429 wrap test data pattern mismatch.

ARS_WRAP_FLUSH_FAIL Unknown external messages were received during the internal wrap test execution.

ARS_MEMWRERR Device SRAM test write/read/verify failure.

ARS_FAILURE For *Windows* only, indicates library CEI_Install.dll failed to load in memory; for *Linux multi-process application use* only, indicates the API either failed to allocate a shared memory block or create the Board Lock semaphore; for *VxWorks* only, indicates a legacy PCI driver open session failure.

API_VXB_METHOD_ERR For *VxWorks VxBus driver usage* only, indicates a device open session failure.

Arguments

CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

AR_PUT_429_MESSAGE

Syntax CDEV_API_RET_TYPE ar_put_429_message (CDEV_BOARD_TYPE board, CDEV_CHAN_TYPE channel, CDEV_PARM_SI_TYPE data)

Description This routine places the provided ARINC 429 message data in the specified channel transmit buffer. If the specified transmit buffer is full, an overflow status is returned.

Note: Since ARINC 429 transmit data rates are relatively slow, almost any host can generate transmit data at a much faster rate than data is transmitted.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INHARVAL	The specified <i>channel</i> is invalid or does not support the ARINC 429 protocol.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_XMITOVRFLO	A transmit buffer overrun occurred.

Arguments

CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

CDEV_CHAN_TYPE channel (input) ARINC 429 transmit channel this routine is to access. The valid range is 0 to one less than the number of installed transmit channels.

CDEV_PARM_SI_TYPE data (input) ARINC 429 message to transmit in standard ARINC 429 format.

AR_PUT_573_FRAME

Syntax

CDEV_API_RET_TYPE ar_put_573_frame (CDEV_BOARD_TYPE board, CEI_UINT32 numberWords, pCEI_UINT32 transmitCount, pCEI_INT16 arincData)

Description

This routine attempts to transfer *numberWords* of ARINC 573/717 data from the *arincData* source to the device ARINC 573/717 transmit buffer. The amount of data transferred to the transmitter is based on what is available in the buffer, with the actual number of words transferred indicated in the return value of *transmitCount*.

Note:

Since ARINC 573/717 transmit data rates are relatively slow, almost any host can generate transmit frame data at a much faster rate than frame data is transmitted.

Return Value

ARS_NORMAL Routine execution was successful.

ARS_INVBOARD An uninitialized board or invalid *board* value was provided.

ARS_INHARVAL The specified *board* does not support the ARINC 573/717 protocol.

ARS_BOARD_MUTEX Access to the Board Lock timed-out/failed.

Arguments

CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

CEI_UINT32 numberWords (input) Number of words to copy from the source 573 frame to the transmit buffer.

pCEI_UINT32 transmitCount(output) Indicates how many words were copied from the source 573 frame to the transmit buffer, either less than or equal to the value of *numberWords*.

pCEI_INT16 arincData (output) Pointer to the array of ARINC 573/717 frame data. The format of each data word in the source ARINC 573/717 frame is defined as follows:

15 – 12	11 - 0
RESERVED	data

data: the 12-bit ARINC 573/717 data.

AR_PUTBLOCK

Syntax

CEI_INT32 ar_putblock (CEI_UINT32 board, CEI_UINT32 channel, CEI_INT32 maxMessages, CEI_INT32 offset, pCEI_INT32 data, pCEI_INT32 actualCount)

Description

This routine transfers the array of ARINC 429 messages to the specified transmit channel buffer. When this routine returns, the data has not been transmitted, it has only been placed in the transmit buffer. If other data is in the transmit buffer ahead of it, this data is transmitted in turn.

Note:

Since ARINC 429 transmit data rates are relatively slow, almost any host can generate transmit data at a much faster rate than data is transmitted.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INHARVAL	The specified <i>channel</i> is invalid or does not support the ARINC 429 protocol.
ARS_INVARG	An invalid or null <i>maxMessages</i> , <i>data</i> , or <i>actualCount</i> parameter was provided.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_XMITOVRFLO	A transmit buffer overrun occurred.

Arguments

CEI_UINT32 board	(input) Device to access. Valid range is 0-127.
CEI_UINT32 channel	(input) ARINC 429 transmit channel this routine is to access. The valid range is 0 to one less than the number of installed transmit channels.
CEI_INT32 maxMessages	(input) The number of messages to transmit.
CEI_INT32 offset	Unused legacy API parameter.
pCEI_INT32 data	(input) Array supplying 32-bit ARINC data values.
pCEI_INT32 actualCount	(output) The number of messages copied to the transmit buffer.

AR_PUTBLOCK_MULTI_CHAN

Syntax CEI_INT32 ar_putblock_multi_chan (CEI_UINT32 board, CEI_INT32 maxMessages, pCEI_UINT32 channels, pCEI_INT32 data, pCEI_INT32 actualCount)

Description This routine transfers messages from the *data* array source to the channel transmit buffer corresponding to the respective transmit channel element of the *channels* array. When this routine returns, the data has not necessarily been transmitted, it has only been placed in the respective transmit buffer(s). If other data is in the transmit buffer ahead of it, this data will be transmitted in turn.

Note: Since ARINC 429 transmit data rates are relatively slow, almost any host can generate transmit data at a much faster rate than data is transmitted.

Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INHARVAL	One of the specified <i>channel</i> array elements is invalid or does not support the ARINC 429 protocol.
	ARS_INVARG	An invalid or null <i>maxMessages</i> , <i>data</i> , or NULL <i>actualCount</i> parameter was provided.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_XMITOVRFLO	A transmit buffer overrun occurred.
Arguments	CEI_UINT32 board	(input) Device to access. Valid range is 0-127.
	CEI_INT32 maxMessages	(input) The number of messages to transmit.
	pCEI_UINT32 channels	(input) Array supplying the ARINC 429 transmit channel on which this routine is to transmit the respective ARINC 429 data. The transmit channel index in each element of this array corresponds directly to the ARINC 429 message defined in the respective element of the <i>data</i> array. The valid range for each element of this array is 0 to one less than the number of installed transmit channels.

pCEI_INT32 data	(input) Array supplying 32-bit ARINC data values.
pCEI_INT32 actualCount	(input) The number of messages transmitted.

AR_PUTFILTER

Syntax

CEI_INT32 ar_putfilter (CEI_UINT32 board, CEI_UINT32 channel, pCEI_CHAR filterTable)

Description

This routine assigns an entire channel portion of the label filter table for the specified receive channel. Each receive channel has a separate area in the device label filter table, which is used by the firmware to control storage of received labels. Each element of the filter table consists of a three bit field defined for compatibility with the CEI-x20 product line as follows:

FILTER_SEQUENTIAL	0x10	If CLEAR add label to circular receive buffer
FILTER_SNAPSHOT	0x20	If CLEAR add label to snapshot receiver buffer
FILTER_INTERRUPT	0x40	If SET on reception insert the respective receive channel tag (ranging from 64-95) in the interrupt queue and if enabled generate a PCI interrupt.

The filter buffer for a single channel is defined as follows:

filterTable[MAX_ESSM][MAX_SDI][MAX_LABEL]

and accessed as follows in the array referenced by *filterTable*:

filterTable[eSSM][SDI][label]

where the bits of the ARINC word are split up as follows:

eSSM	SDI	label
30, 29, 28	9, 8	7, 6, 5, 4, 3, 2, 1, 0

Note:

For FILTER_INTERRUPT processing, an entry is made into the interrupt queue if specified through receiver interaction with the label filter table definition, even if hardware interrupts are not enabled.

To write individual label filter table elements, refer to the API routines AR_ENH_LABEL_FILTER and AR_LABEL_FILTER.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	An invalid <i>channel</i> or <i>filterTable</i> value was provided.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.

Arguments

CEI_UINT32 board	(input) Device to access. Valid range is 0-127.
CEI_UINT32 channel	(input) Specifies which receive channel this routine is to access. Valid range is 0 to one less than the installed receive channel count.
pCEI_CHAR filterTable	(input) Array containing the contents of the specified channel's label filter table. This array must have an allocation of 8Kbytes.

AR_PUTWORD

Syntax

CDEV_API_RET_TYPE ar_putword (CDEV_BOARD_TYPE board, CDEV_CHAN_TYPE channel, CDEV_PARM_SI_TYPE arincdata)

Description

This routine places the provided message data in the specified channel transmit buffer. When this routine returns, the data has not necessarily been sent, it has only been placed in the transmit buffer. If other data is in the transmit buffer ahead of it, this data will be transmitted in turn. If the specified transmit buffer is full, an overflow status is returned.

The channel value passed to this routine corresponds to the ARINC 429 transmit channel index, starting with zero. If that value exceeds the 429 transmit channel count and an ARINC 573/717 transmitter exists, it is used as the designated transmit channel buffer.

Note:

Since ARINC transmit data rates are relatively slow, almost any host can generate transmit data at a much faster rate than data is transmitted.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INHARVAL	The specified <i>channel</i> is invalid or does not support the ARINC 429 (or 717) protocol.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
ARS_XMITOVRFLO	A transmit buffer overrun occurred.

Arguments

CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

CDEV_CHAN_TYPE channel (input) Transmit channel this routine is to access. The valid range is 0 to one less than the installed transmit channel count.

CDEV_PARM_SI_TYPE arincdata (input) 32-bit ARINC 429 message to transmit.

AR_QUERY_DEVICE

Syntax	CDEV_API_RET_TYPE ar_query_device (CDEV_BOARD_TYPE board, CDEV_BOARD_TYPE * boardType)																																									
Description	This routine opens a session to the specified device, determines the identification of that device, then closes the session with the device and returns the identification to the calling application. This routine should not be invoked with the same board parameter value used in a previous invocation of AR_OPEN without first terminating the session with that device via invocation of AR_CLOSE.																																									
Return Value	ARS_NORMAL	Routine execution was successful.																																								
	ARS_INVBOARD	An invalid <i>board</i> value was provided.																																								
	ARS_BOARD_MUTEX	For Windows only, indicates creation of the Board Lock mutex timed-out/failed.																																								
	ARS_FAILURE	For <i>Windows</i> only, indicates library CEI_Install.dll failed to load in memory.																																								
Arguments	CDEV_BOARD_TYPE board	(input) Device to access. Valid range is 0-127.																																								
	CDEV_BOARD_TYPE * boardType	(output) Identification of the board detected, with valid values defined as follows:																																								
		<table> <tr><td>CEI-830</td><td>19</td></tr> <tr><td>CEI-430</td><td>21</td></tr> <tr><td>AMC-A30</td><td>22</td></tr> <tr><td>RCEI-530</td><td>26</td></tr> <tr><td>R830RX</td><td>27</td></tr> <tr><td>RAR-CPCI</td><td>28</td></tr> <tr><td>RAR-EC</td><td>29</td></tr> <tr><td>RAR-PCIE</td><td>30</td></tr> <tr><td>CEI-430A</td><td>31</td></tr> <tr><td>RCEI-830X820</td><td>34</td></tr> <tr><td>RAR-XMC</td><td>35</td></tr> <tr><td>RCEI-830A</td><td>36</td></tr> <tr><td>RP-708</td><td>37</td></tr> <tr><td>RAR-MPCIE</td><td>38</td></tr> <tr><td>CEI-520</td><td>6</td></tr> <tr><td>CEI-620</td><td>10</td></tr> <tr><td>CEI-820</td><td>11</td></tr> <tr><td>CEI-715</td><td>16</td></tr> <tr><td>P-708</td><td>20</td></tr> <tr><td>P-SER</td><td>104</td></tr> </table>	CEI-830	19	CEI-430	21	AMC-A30	22	RCEI-530	26	R830RX	27	RAR-CPCI	28	RAR-EC	29	RAR-PCIE	30	CEI-430A	31	RCEI-830X820	34	RAR-XMC	35	RCEI-830A	36	RP-708	37	RAR-MPCIE	38	CEI-520	6	CEI-620	10	CEI-820	11	CEI-715	16	P-708	20	P-SER	104
CEI-830	19																																									
CEI-430	21																																									
AMC-A30	22																																									
RCEI-530	26																																									
R830RX	27																																									
RAR-CPCI	28																																									
RAR-EC	29																																									
RAR-PCIE	30																																									
CEI-430A	31																																									
RCEI-830X820	34																																									
RAR-XMC	35																																									
RCEI-830A	36																																									
RP-708	37																																									
RAR-MPCIE	38																																									
CEI-520	6																																									
CEI-620	10																																									
CEI-820	11																																									
CEI-715	16																																									
P-708	20																																									
P-SER	104																																									

P-DIS	106
R15-MPCIE	107
AMC-1553	108
(c)PCI-1553	109
QPCI-1553	110
QCP-1553	111
QPCX-1553	112
R15-EC	113
R15-XMC	114
R15-PCIE	115
R15-LPCIE	116
R15-XMC2	117
RAR15-XMC	118
RAR15-XMC-XT	119

AR_READ_SCHEDULED_MSG_BLOCK

Syntax	CDEV_API_RET_TYPE ar_read_scheduled_msg_block (CDEV_BOARD_TYPE board, CEI_INT32 startingEntry, CEI_INT32 numberOfEntries, pAR_SCHEDULED_MSG_ENTRY_TYPE messageEntry)	
Description	This routine returns the current contents of one or more message scheduler table entries.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	The starting entry plus the number of entries requested to read exceeds the upper boundary of the message scheduler table.
Arguments	CDEV_BOARD_TYPE board	(input) Device to access. Valid range is 0-127.
	CEI_INT32 startingEntry	(input) The first message scheduler table entry to read.
	CEI_INT32 numberOfEntries	(input) The number of entries to read from the message scheduler table.
	pAR_SCHEDULED_MSG_ENTRY_TYPE messageEntry	(input) Array of structures that will receive the message definition content:
	unsigned long messageIndex	The unique message scheduler table entry index assigned to this message. Upon completion of this routine, the messageIndex structure member will have been updated to reflect the message scheduler table index assigned to the respective message.
	unsigned long board	Device to access. Valid range is 0-127.
	unsigned long channel	Channel on which to transmit this message. The valid range is 0 to one less than the number of installed transmit channels.

unsigned long rate	Periodic transmission rate, defined in milliseconds by default.
unsigned long start	Offset, (in milliseconds), from the start of CEI-x30 device message processing at which this message will begin its initial periodic transmission.
unsigned long txCount	The total number of times this message will be transmitted. The constant value ARU_SCHED_MSG_INFINITE (0xFFFFFFFF) indicates infinite transmission of this message is requested.
unsigned long data	The 32-bit ARINC 429 message to transmit.

AR_RESET

Syntax

CDEV_API_RET_TYPE ar_reset (CDEV_BOARD_TYPE board)

Description

This routine assigns the global enable register Global Enable bit to be *disabled* and reinitializes the device to the same channel configuration as that following an invocation of AR_OPEN. See the description for the routine AR_INITIALIZE_DEVICE for more details.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_MEMWRERR	Device SRAM test write/read/verify failure.
ARS_BAD_STATIC	Device register write/read/verify failure.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.

Arguments

CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

AR_RESET_TIMERCNT

Syntax	CEI_VOID ar_reset_timercnt (CDEV_BOARD_TYPE board)
Description	This routine is designed to provide compatibility with the CEI-x20 ARINC API. It resets the CEI-x30 device internal one-microsecond timer to zero.
Arguments	CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

AR_SET_CONFIG

Syntax

CDEV_API_RET_TYPE ar_set_config (CDEV_BOARD_TYPE board, CDEV_PARM_SSI_TYPE item, CEI_UINT32 value)

Description

This routine provides a means to define general device configuration attributes, as well as limited individual channel configuration attributes. It is provided for backward compatibility to CEI-x20 based applications. The routine AR_SET_DEVICE_CONFIG is the desired routine for defining channel and board-level configuration items.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVARG	The <i>item</i> argument value is not supported by this API routine.
ARS_INVHARVAL	The <i>item</i> argument value is not supported by this device configuration.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.

Arguments

CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

CDEV_PARM_SSI_TYPE item (input) Attribute about which to set information:

ARU_XMIT_RATE	transmit rate for all transmitters.
ARU_RECV_RATE	receive rate for all receivers.
ARU_PARITY	parity for all transmitters and receivers.
ARU_INTERNAL_WRAP	enables internal wrap mode for all receivers.
ARU_RX_CH01_BIT_RATE – ARU_RX_CH32_BIT_RATE	receiver 1 - 32 bit rate.
ARU_TX_CH01_BIT_RATE – ARU_TX_CH32_BIT_RATE	transmitter 1 - 32 bit rate.
ARU_RX_CH01_PARITY – ARU_RX_CH32_PARITY	receiver 1 - 32 parity.
ARU_TX_CH01_PARITY – ARU_TX_CH32_PARITY	transmitter 1 - 32 parity.

ARU_TX_CH01_SHUT_OFF –
 ARU_TX_CH32_SHUT_OFF transmitter 1 - 32 disable.

ARU_TX_CH01_LB_INJ – transmitter 1 - 32 low bit
 ARU_TX_CH32_LB_INJ error enable.

ARU_TX_CH01_HB_INJ – transmitter 1 - 32 high bit
 ARU_TX_CH32_HB_INJ error enable.

ARU_TX_CH01_GAP_INJ – transmitter 1 - 32
 ARU_TX_CH32_GAP_INJ message gap error enable.

ARU_RX_TIMETAG_MODE the timer/time-tag source and
 resolution

ARU_ACCESS_SNAPSHOT_BUFFER snapshot storage mode

ARU_IRIG_WRAP_ENABLE enables IRIG receiver internal wrap

ARU_IRIG_INPUT_THRESHOLD sets the IRIG DAC threshold

ARU_IRIG_ADJUST_THRESHOLD invokes a more precise IRIG
 DAC auto-adjustment procedure

ARU_IRIG_QUICK_ADJUSTMENT invokes a quick IRIG DAC
 auto-adjustment procedure

ARU_IRIG_SET_BIAS assigns an offset to the board IRIG
 time value

CEI_UINT32 value (input) the value to set the item.

If the specified item is ARU_XMIT_RATE (1) or ARU_RECV_RATE (2), valid value parameter selections are:

AR_HIGH (0) high rate (100Kbs)
 AR_LOW (1) low rate (12.5Kbs)
 Any other value specifies a frequency value in Hertz.

If the specified item is ARU_RX_CH nn _BIT_RATE (500-531), where nn is the receiver channel (01 - 32), valid value parameter selections are:

AR_HIGH (0) high rate (100Kbs)
 AR_LOW (1) low rate (12.5Kbs)
 Any other value specifies a frequency value in Hertz.

If the specified item is ARU_TX_CH nn _BIT_RATE (700-731), where nn is the transmitter channel (01 - 32), valid value parameter selections are:

AR_HIGH (0) high rate (100Kbs)
 AR_LOW (1) low rate (12.5Kbs)
 Any other value specifies a frequency value in Hertz.

Note

Any specified transmit bus frequency below 15KHz will be assigned to a slow slew rate.
Any specified transmit bus frequency above 15KHz will be assigned to a fast slew rate.

If the specified item is ARU_PARITY (3), the value parameter specifies the parity selection for all transmit and receive channels.

AR_ODD (0) odd transmit parity and receive parity detect enabled
AR_EVEN (1) even transmit parity and rx parity detect enabled
AR_OFF (8) transmit parity and receive parity detect disabled
AR_RAW (0x2000) transmit parity and rx parity detect disabled

If the specified item is ARU_RX_CH nn _PARITY (900-931), where nn is the receiver channel (01 - 32), valid value parameter selections are:

AR_ODD (0) receiver parity detection enabled
AR_OFF (8) receiver parity detection disabled
AR_RAW (0x2000) receiver parity detection disabled

If the specified item is ARU_TX_CH nn _PARITY (1100-1131), where nn is the transmitter channel (01 - 32), valid value parameter selections are:

AR_ODD (0) odd transmitter parity
AR_EVEN (1) even transmitter parity
AR_OFF (8) transmitter parity disabled
AR_RAW (0x2000) transmitter parity disabled

If the requested item is ARU_TX_CH nn _SHUT_OFF (1700-1731), where nn is the transmitter channel (01 - 32), valid value parameter selections are:

AR_ON (7) external transmission is disabled
AR_OFF (8) external transmission is enabled

For the RAR-PCIE and RAR15-XMC-XT boards, disabling external transmission also causes the transmit pins to switch to a tri-state condition; for all other boards the transmit pins will switch to a null condition.

If the requested item is ARU_TX_CH nn _HB_INJ (3300-3331), where nn is the transmitter channel (01 - 32), valid value parameter selections are:

AR_ON (7) 33-bit transmission is enabled
AR_OFF (8) standard 32-bit transmission is enabled

If the requested item is ARU_TX_CH nn _LB_INJ (3500-3531), where nn is the transmitter channel (01 - 32), valid value parameter selections are:

AR_ON (7) 31-bit transmission is enabled
AR_OFF (8) standard 32-bit transmission is enabled

If the requested item is ARU_TX_CH nn _GAP_INJ (3700-3731), where nn is the transmitter channel (01 - 32), valid value parameter selections are:

AR_ON (7) 3-bit message gap is used
AR_OFF (8) standard 4-bit message gap is used

If the specified item is ARU_INTERNAL_WRAP (4), valid value parameter selections are:

AR_WRAP_ON (0) internal wrap enabled
 AR_WRAP_OFF (1) internal wrap disabled

If the specified item is ARU_RX_TIMETAG_MODE (440), valid value parameter selections represent the timer/time-tag source and resolution. This item specifies the resolution of any timer-read or receive data time-tag value obtained via the API, with value selections defined as follows:

AR_TIMETAG_EXT_IRIG_64BIT (0)
 AR_TIMETAG_INT_USEC_64BIT (1)
 AR_TIMETAG_INT_20USEC_32BIT (3)
 AR_TIMETAG_INT_MSEC_32BIT (4)

A value of AR_TIMETAG_EXT_IRIG_64BIT selects the source as the external IRIG receiver, if connected; otherwise, if the IRIG signal is not internally wrapped this selection would be invalid. All other values represent various timer/time-tag LSB resolution values based on the internal CEI-x30 device timer.

If the specified item is ARU_ACCESS_SNAPSHOT_BUFFER (38), a valid value parameter for selecting the active Snapshot Buffer storage mode is:

ARU_LABEL_ONLY (0) messages stored based on label
 ARU_LABEL_WITH_SDI (1) messages stored based on the combined label and SDI field values

If the specified item is ARU_IRIG_WRAP_ENABLE (441), valid value parameter selections are:

AR_ON (7) IRIG receiver internal wrap enabled
 AR_OFF (8) IRIG receiver internal wrap disabled

If the specified item is ARU_IRIG_INPUT_THRESHOLD (442), the value parameter specifies the IRIG receiver threshold voltage in millivolts.

The item ARU_IRIG_ADJUST_THRESHOLD (443) invokes the IRIG DAC auto-adjustment procedure. This procedure will determine the low and high threshold values at which the incoming IRIG signal is present. It then determines the best threshold level for the IRIG DAC, and returns the value to the application in place of a returned status (failures are indicated via return value of ARS_FAILURE). This procedure may execute for up to, and in some cases in excess of, one minute before finding the best-case threshold value for the incoming IRIG signal. If a printed status of the execution progress within this procedure is desired, assign the *value* parameter to any non-zero value.

If the specified item is ARU_IRIG_QUICK_ADJUSTMENT (444), the API performs a quick adjustment of the IRIG DAC for an external input IRIG signal using signal edge detection for verification of signal presence. This execution of this adjustment should require less than one second.

If the specified item is ARU_IRIG_SET_BIAS (446), a valid value parameter consists of an offset to the board-supplied IRIG time specified in milliseconds. The bias time range is +/-32.768 seconds.

AR_SET_DEVICE_CONFIG

Syntax	CDEV_API_RET_TYPE ar_set_device_config (CDEV_BOARD_TYPE board, CDEV_CHAN_TYPE channel, CDEV_PARM_SSI_TYPE item, CDEV_PARM_SSI_TYPE value)	
Description	This is the recommended routine to define the general device and ARINC 429 channel configuration attributes.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVARG	The item argument value is not supported by this API routine.
	ARS_INVHARVAL	The item argument value is not supported by this device configuration.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
Arguments	CDEV_BOARD_TYPE board	(input) Device to access. Valid range is 0-127.
	CDEV_CHAN_TYPE channel	(input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed channel count for the respective channel type.
	CDEV_PARM_SSI_TYPE item	(input) Specifies the configuration attribute to define:
	ARU_RX_BITRATE	receive rate for specified channel.
	ARU_RX_PARITY	receive parity for specified channel.
	ARU_RX_FIFO_ENABLE	receive channel FIFO enable.
	ARU_RECV_MODE	receive channel internal wrap mode.
	ARU_RX_MERGED_MODE	receive channel merge mode enable.
	ARU_RX_MSG_SIZE_24BIT	receiver 24-bit message mode.
	ARU_TX_BITRATE	transmit rate for specified channel.
	ARU_TX_PARITY	transmit parity for specified channel.
	ARU_TX_FIFO_ENABLE	transmit channel FIFO enable.
	ARU_TX_DISABLE	transmit channel transceiver disable.
	ARU_TX_GAP_ERROR	transmit message gap error enable.
	ARU_TX_BIT_ERROR	transmit message size error enable.
	ARU_TX_MSG_SIZE_24BIT	transmitter 24-bit message mode.
	ARU_FAST_SLEW_RATE	transmit channel slew rate select.
	ARU_ACCESS_SNAPSHOT_BUFFER	snapshot storage mode
	ARU_BYPASS_INIT_WRAP_TEST	bypass initialization wrap test

ARU_MULTITHREAD_PROTECT control use of thread protection
 ARU_RX_TIMETAG_MODE timer/time-tag source and resolution
 ARU_DIFFERENTIAL_OUT differential output enable and state
 ARU_DISCRETE_OUT sets a discrete output state
 ARU_IRIG_WRAP_ENABLE enables IRIG receiver internal wrap
 ARU_IRIG_INPUT_THRESHOLD sets the IRIG DAC threshold
 ARU_IRIG_ADJUST_THRESHOLD both invoke IRIG DAC
 ARU_IRIG_QUICK_ADJUSTMENT auto-adjustment procedures
 ARU_IRIG_SET_BIAS assigns an offset to IRIG time
 ARU_IRIG_OUTPUT_ENABLE R830RX IRIG Tx state
 ARU_HW_ENHANCE_UPDATE update board for enhanced f/w
 ARU_HW_INTERRUPT_ENABLE enable/disable PCI interrupts
 ARU_INSERT_INT_Q_ENTRY insert entry in interrupt queue
 ARU_CONFIG_PROGRAMMABLE_CHAN assign a pgm chan
 state

CDEV_PARM_SSI_TYPE value (input) the value to set the specified item.

If the requested item is ARU_RX_BITRATE (1) or ARU_TX_BITRATE (2), valid value parameter selections are:

ARU_SPEED_HIGH (0) high rate (100Kbs)
 ARU_SPEED_LOW (1) low rate (12.5Kbs)

Any other value assigns a non-standard bus speed, and is translated as a divisor for the 16MHz device clock reference. This value and the respective baud rate may be interpreted using the following formulas:

$$\text{Baud Rate} = 16,000,000 / (\text{Value}+2)$$

$$\text{Value} = (16,000,000 / \text{Desired Baud Rate}) - 2$$

Note:

Any non-standard transmit bus speed value resulting in a baud rate below 15KHz will be assigned to a slow slew rate. Any non-standard transmit bus speed value resulting in a baud rate at or above 15KHz will be assigned to a fast slew rate.

If the requested item is ARU_RX_PARITY (3), valid value parameter selections are:

AR_ON (7) receiver parity enabled
 AR_OFF (8) receiver parity disabled

If the requested item is ARU_TX_PARITY (4), valid value parameter selections are:

ARU_PARITY_ODD (0) odd transmitter parity
 ARU_PARITY_EVEN (1) even transmitter parity
 ARU_PARITY_NONE (2) transmitter parity disabled

If the requested item is ARU_RECV_MODE (5), valid value parameter selections are:

AR_WRAP_ON (0) internal wrap enabled
 AR_WRAP_OFF (1) internal wrap disabled

If the requested item is ARU_RX_FIFO_ENABLE (16) or ARU_TX_FIFO_ENABLE (17), valid value parameter selections are:

AR_ON (7) Rx/Tx FIFO operation enabled
 AR_OFF (8) Rx/Tx FIFO operation disabled

If the requested item is ARU_TX_DISABLE (10), valid value parameter selections are:

AR_ON (7) external transmission disabled
 AR_OFF (8) external transmission enabled

For the RAR-PCIE and RAR15-XMC-XT boards, disabling external transmission also causes the transmit pins to switch to a tri-state condition; for all other boards the transmit pins will switch to a null condition.

If the requested item is ARU_TX_GAP_ERROR (8), valid value parameter selections are:

AR_ON (7) transmit message gap error enabled
 AR_OFF (8) transmit message gap error disabled

If the requested item is ARU_TX_BIT_ERROR (6), valid value parameter selections are:

AR_LO (0) Low Bit Error operation is enabled
 AR_HI (1) High Bit Error operation is enabled
 AR_OFF (8) bit errors are disabled on this transmitter

If the requested item is ARU_FAST_SLEW_RATE (323), valid value parameter selections are:

AR_ON (7) Fast Slew Rate selected (1.5 μ sec rise time)
 AR_OFF (8) Slow Slew Rate selected (10 μ sec rise time)

If the requested item is ARU_RX_MERGED_MODE (18), valid value parameter selections are:

AR_ON (7) receiver merged mode operation enabled
 AR_OFF (8) receiver merged mode operation disabled

If the requested item is ARU_RX_MSG_SIZE_24BIT (459), this routine returns the current state of the specified receive channel configuration register 24-bit Message Size enable field:

AR_ON (7) ARINC 585 24-bit protocol enabled
 AR_OFF (8) ARINC 429/575 32-bit protocol enabled

If the requested item is ARU_TX_MSG_SIZE_24BIT (460), this routine returns the current state of the specified transmit channel configuration register 24-bit Message Size enable field:

AR_ON (7) ARINC 585 24-bit protocol enabled

AR_OFF (8) ARINC 429/575 32-bit protocol enabled

If the requested item is ARU_ACCESS_SNAPSHOT_BUFFER (38), valid value parameter selections are:

ARU_LABEL_ONLY (0) message storage on a label basis

ARU_LABEL_WITH_SDI (1) message storage on a label/sdi basis

If the specified item is ARU_BYPASS_INIT_WRAP_TEST (320), valid value parameter selections are:

AR_ON (7) bypass internal wrap test invocation during init

AR_OFF (8) execute internal wrap test invocation during init

If the specified item is ARU_MULTITHREAD_PROTECT (321), valid value parameter selections are:

AR_ON (7) enables mutex/semaphore thread protection

AR_OFF (8) disables mutex/semaphore thread protection

If the specified item is ARU_RX_TIMETAG_MODE (440), valid value parameter selections represent the timer/time-tag source and resolution. This item specifies the resolution of any timer-read or receive data time-tag value obtained via the API, with value selections defined as follows:

AR_TIMETAG_EXT_IRIG_64BIT (0)

AR_TIMETAG_INT_USEC_64BIT (1)

AR_TIMETAG_INT_20USEC_32BIT (3)

AR_TIMETAG_INT_MSEC_32BIT (4)

AR_TIMETAG_SYNC_1553_CH1 (11)

AR_TIMETAG_SYNC_1553_CH2 (12)

AR_TIMETAG_SYNC_1553_CH3 (13)

AR_TIMETAG_SYNC_1553_CH4 (14)

A value of AR_TIMETAG_EXT_IRIG_64BIT selects the source as the external IRIG receiver, if connected; otherwise, if the IRIG signal is not internally wrapped this selection would be invalid. .

Any 1553 synchronized time value selects the timer source as the respective 1553 channel on a Multi-protocol device.

All other values represent various timer/time-tag LSB resolution values based on the internal CEI-x30 device timer.

If the specified item is ARU_DISCRETE_OUT (12), valid value parameter selections are:

AR_HI (1) Discrete Out set to 0 (FET OFF – tri-state)

AR_LO (0) Discrete Out set to 1 (FET ON – conduct to Ground)

(see paragraph *Avionics Discrete I/O* for the Discrete circuit diagram)

If the specified item is ARU_DIFFERENTIAL_OUT (23), valid value parameter selections assign both the enable state and output state of the differential channel:

AR_HI (1) the differential output is set high
 AR_LO (0) the differential output is set low
 AR_ON (7) the differential output is enabled
 AR_OFF (8) the differential output is disabled

If the specified item is ARU_IRIG_WRAP_ENABLE (441), valid value parameter selections are:

AR_WRAP_ON (0) IRIG receiver internal wrap enabled
 AR_WRAP_OFF (1) IRIG receiver internal wrap disabled

If the specified item is ARU_IRIG_INPUT_THRESHOLD (442), the value parameter specifies the IRIG receiver threshold voltage in millivolts.

The item ARU_IRIG_ADJUST_THRESHOLD (443) invokes the IRIG DAC auto-adjustment procedure. This procedure determines the low and high threshold values at which the incoming IRIG signal is present. It then determines the best threshold level for the IRIG DAC, and returns the value to the application in place of a returned status (failures are indicated through return value of ARS_FAILURE). This procedure may execute for up to and in some cases in excess of one minute before finding the best-case threshold value for the incoming IRIG signal. If a printed status of the execution progress within this procedure is desired, assign the *value* parameter to any non-zero value.

If the specified item is ARU_IRIG_QUICK_ADJUSTMENT (444), the API performs a quick adjustment of the IRIG DAC for an external input IRIG signal, using signal edge detection for verification of signal presence. This execution of this adjustment should require less than one second.

If the specified item is ARU_IRIG_SET_BIAS (446), the API assigns an offset to the board IRIG time value calculation for any time and time-tag retrieval.

If the specified item is ARU_IRIG_OUTPUT_ENABLE (26), valid value parameter selections to control the state of the R830RX IRIG Generator Enable are:

AR_ON (7) IRIG output is enabled
 AR_OFF (8) IRIG output is disabled

If the specified item is ARU_HW_ENHANCE_UPDATE (31), valid value parameter selections to control the board's PCI BAR2 Size allocation residing in an on-board EEPROM are:

AR_ON (7) support for the CEI-x30 Enhanced Firmware Interface is enabled
 AR_OFF (8) support for the CEI-x30 Enhanced Firmware Interface is disabled

Based on the value parameter selection, the board may be reprogrammed to support the 512Kb CEI-x30 Enhanced Firmware Interface (exclusively supported with CEI-x30 API Version 2.00 and later); or it may be

reprogrammed to support only the standard 4Kb CEI-x30 interface, (exclusively supported with CEI-x30 API Versions 1.00 through 1.70).

Note:

Any modification to the current PCI BAR2 Size allocation requires a host restart for those changes to take effect in the system.

If the specified item is ARU_HW_INTERRUPT_ENABLE (29), valid value parameter selections to control the state of the PCI Interrupt Enable state are:

- AR_ON (7) PCI Interrupts are enabled
- AR_OFF (8) PCI Interrupts are disabled

If the specified item is ARU_INSERT_INT_Q_ENTRY (37), the value parameter is ignored and the value 255 is inserted as the next entry in the device interrupt queue.

If the specified item is ARU_CONFIG_PROGRAMMABLE_CHAN (457), valid value parameter selections to control the enable function of a software programmable transmit/receive channel pair are:

ARU_RECEIVER (0) configures the shared I/O pins and respective receive channel for reception (disables the transmitter).

ARU_TRANSMITTER (1) configures the shared I/O pins and respective transmit channel for external transmission (disables the receiver).

ARU_BOTH (2) configures the shared I/O pins and respective transmit and receive channels for external transmission and reception (all transmissions logged to receiver).

ARU_DISABLE (3) disables external transmission and reception on the transmit and receive channels assigned to the respective shared I/O pins.

AR_SET_573_CONFIG

Syntax	CDEV_API_RET_TYPE ar_set_573_config (CDEV_BOARD_TYPE board, CDEV_PARM_SSI_TYPE item, CDEV_PARM_SI_TYPE value)	
Description	This routine provides the method for manipulating the ARINC 573/717 channel configuration attributes.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
	ARS_INVHARVAL	the item argument value is not supported by the device configuration or this API routine.
Arguments	CDEV_BOARD_TYPE board	(input) Device to access. Valid range is 0-127.
	CDEV_PARM_SSI_TYPE item	(input) Specifies the configuration attribute to define:
	ARU_RECV_MODE	receiver internal wrap.
	ARU_RX_MERGED_MODE	receiver merge mode enable.
	ARU_RX_BITRATE	receive channel bit rate.
	ARU_RX_FIFO_ENABLE	receive channel FIFO enable.
	ARU_TX_BITRATE	transmit channel bit rate.
	ARU_TX_FIFO_ENABLE	transmit channel FIFO enable.
	ARU_573_RX_AUTO_DETECT	data frame auto-detect enable.
	ARU_573_RX_BPRZ_SELECT	receiver BPRZ/HBP selection.
	ARU_573_TX_BPRZ_SELECT	transmit BPRZ encoding enable.
	ARU_573_TX_HBP_SELECT	transmit HBP encoding enable.
	ARU_573_TX_SLEW_RATE	transmit slew rate select.
	ARU_573_SYNC_WORD1	receiver auto-detect sync word 1.
	ARU_573_SYNC_WORD2	receiver auto-detect sync word 2.
	ARU_573_SYNC_WORD3	receiver auto-detect sync word 3.
	ARU_573_SYNC_WORD4	receiver auto-detect sync word 4.
	CDEV_PARM_SI_TYPE value	(input) The state to assign to the specified configuration item:

If the requested item is ARU_RECV_MODE (5), valid value parameter selections are:

- AR_WRAP_ON (0) = internal wrap enabled
- AR_WRAP_OFF (1) = internal wrap disabled

If the requested item is ARU_RX_MERGED_MODE (18), valid value parameter selections are:

- AR_ON (7) receiver merged mode operation enabled
- AR_OFF (8) receiver merged mode operation disabled

If the requested item is ARU_RX_FIFO_ENABLE (16) or ARU_TX_FIFO_ENABLE (17), valid value parameter selections are:

- AR_ON (7) FIFO operation enabled
- AR_OFF (8) FIFO operation disabled

If the configuration item is ARU_RX_BITRATE (1) or ARU_TX_BITRATE (2), valid item values are one of the following (0-7):

- ARU_573_RATE_SIZE_384_32 384 bps, 32 word sub-frame
- ARU_573_RATE_SIZE_768_64 768 bps, 64 word sub-frame
- ARU_573_RATE_SIZE_1536_128 1536 bps, 128 word sub-frame
- ARU_573_RATE_SIZE_3072_256 3072 bps, 256 word sub-frame
- ARU_573_RATE_SIZE_6144_512 6144 bps, 512 word sub-frame
- ARU_573_RATE_SIZE_12288_1024 12288 bps, 1024 word sub-frame
- ARU_573_RATE_SIZE_24576_2048 24576 bps, 2048 word sub-frame
- ARU_573_RATE_SIZE_49152_4096 49152 bps, 4096 word sub-frame

If the configuration item is ARU_573_RX_AUTO_DETECT (301), valid item values are one of the following:

- AR_ON (7) ARINC 573/717 frame auto-detection enabled
- AR_OFF (8) ARINC 573/717 frame auto-detection disabled

If the configuration item is ARU_573_RX_BPRZ_SELECT (302), valid item values are one of the following:

- AR_OFF (7) ARINC 573/717 HBP reception enabled
- AR_ON (8) ARINC 573/717 BPRZ reception enabled

If the configuration item is ARU_573_TX_BPRZ_SELECT (313), valid item values are one of the following:

- AR_OFF (7) ARINC 573/717 BPRZ transmission disabled
- AR_ON (8) ARINC 573/717 BPRZ transmission enabled

If the configuration item is ARU_573_TX_HBP_SELECT (314), valid item values are one of the following:

- AR_OFF (7) ARINC 573/717 HBP transmission disabled
- AR_ON (8) ARINC 573/717 HBP transmission enabled

If the configuration item is ARU_573_TX_SLEW_RATE (315), valid item values are one of the following:

- ARU_573_TX_SLEW_1PT5 (1) = fast (1.5µsec rise time)
- ARU_573_TX_SLEW_10PT0 (0) = slow (10.0µsec rise time)

If the configuration item is ARU_573_SYNC_WORD1 (307), ARU_573_SYNC_WORD2 (308), ARU_573_SYNC_WORD3 (309), or ARU_573_SYNC_WORD4 (310), a valid item value is any 12-bit non-zero value.

AR_SET_MULTITHREAD_PROTECT

Syntax	CDEV_API_RET_TYPE ar_set_multithread_protect (CDEV_BOARD_TYPE board, CDEV_PARM_SSI_TYPE state)
Description	This routine controls the use of mutex/semaphore protection around all device channel-specific accesses performed within the API routines. This type of thread protection should be enabled for any multi-threaded application or reentrant API usage.
Return Value	<p>ARS_NORMAL routine was successful.</p> <p>ARS_INVARG An invalid <i>state</i> value was provided.</p> <p>ARS_INVBOARD An invalid <i>board</i> value was provided.</p>
Arguments	<p>CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.</p> <p>CDEV_PARM_SSI_TYPE state (input) Multi-thread protection setting, valid values are defined as follows:</p> <p style="padding-left: 40px;">AR_ON (7) enables mutex/semaphore protection.</p> <p style="padding-left: 40px;">AR_OFF (8) disables mutex/ semaphore protection.</p>

AR_SET_ISR_FUNCTION

Syntax	CEI_INT32 ar_set_isr_function (CEI_INT32 board, pCEI_VOID function)						
Description	<p>This routine allows the host application to define a custom interrupt service routine to be referenced by the operating system-specific hardware interrupt initialization. It assigns the host-supplied function pointer to an array of pointers indexed by the <i>board</i> parameter value. The function referenced by this pointer is invoked from the internal routine <code>cei_utl_interrupt_handler()</code> instead of executing the default API-supplied "flush the h/w interrupt queue" processing.</p> <p>The function declaration for the supplied interrupt service routine should be defined as follows for the respective operating system:</p> <p>Any Windows:</p> <pre>void _stdcall host_interrupt_handler (CEI_INT32 deviceIndex);</pre> <p>Any Linux distribution:</p> <pre>void host_interrupt_handler (CEI_INT32 deviceIndex);</pre> <p>Any VxWorks or Integrity distribution:</p> <pre>void host_interrupt_handler (CEI_INT32 deviceIndex, pCEI_UINT32 data);</pre>						
Note:	<hr/> <p>The data parameter should not be used by the ISR.</p> <p>This interrupt service routine is executed as a separate process or task from the actual host low-level h/w interrupt processing, with an execution priority based on default process/task priority settings for the respective host operating system.</p> <hr/>						
Return Value	<table> <tr> <td>ARS_NORMAL</td> <td>routine was successful.</td> </tr> <tr> <td>ARS_INVARG</td> <td>An null <i>function</i> value was provided.</td> </tr> <tr> <td>ARS_INVBOARD</td> <td>An invalid <i>board</i> value was provided.</td> </tr> </table>	ARS_NORMAL	routine was successful.	ARS_INVARG	An null <i>function</i> value was provided.	ARS_INVBOARD	An invalid <i>board</i> value was provided.
ARS_NORMAL	routine was successful.						
ARS_INVARG	An null <i>function</i> value was provided.						
ARS_INVBOARD	An invalid <i>board</i> value was provided.						
Arguments	<table> <tr> <td>CDEV_BOARD_TYPE board</td> <td>(input) Device index for storing the function pointer. Valid range is 0-127.</td> </tr> <tr> <td>pCEI_VOID function</td> <td>(input) Function pointer to the host specified interrupt service routine.</td> </tr> </table>	CDEV_BOARD_TYPE board	(input) Device index for storing the function pointer. Valid range is 0-127.	pCEI_VOID function	(input) Function pointer to the host specified interrupt service routine.		
CDEV_BOARD_TYPE board	(input) Device index for storing the function pointer. Valid range is 0-127.						
pCEI_VOID function	(input) Function pointer to the host specified interrupt service routine.						

AR_SET_PRELOAD_CONFIG

Syntax

CDEV_API_RET_TYPE ar_set_preload_config (CDEV_BOARD_TYPE board, CDEV_PARM_SSI_TYPE item, CEI_UINT32 value)

Description

This routine is designed to provide protection when executing multi-threaded or multi-process applications with your CEI-x30 device. Call this routine before calling AR_OPEN to update the value of a particular preload API operational configuration setting. This routine should not be called subsequent to any invocation of AR_OPEN.

If *item* is ARU_CONCURRENCY_MODE, the *value* parameter specifies the API concurrency mode. One of three modes may be selected: AR_CONC_NONE, AR_CONC_MULTITHRD, or AR_CONC_MULTIPROC. Note that some modes are only supported on certain operating systems.

The default concurrency mode, AR_CONC_NONE, provides no multi-thread protection to the device and no multi-process API support. The user application must ensure that only one thread is calling into the API at any given time, and only a single process may interface with a particular board.

If AR_CONC_MULTITHRD concurrency mode is selected, thread protection for each device access is provided internally within the API. The user application may call into the API from multiple threads, but all threads must belong to a single process. The main user application thread should initialize the board with a call to AR_OPEN before other threads attempt to call into the API. This mode is supported on all operating systems supported by the CEI-x30 software distribution.

If AR_CONC_MULTIPROC concurrency mode is selected, thread protection is provided internally within the API and multiple processes may interface with a single board. If any process requests multi-process mode, all other processes must also request multi-process mode. This mode is only supported under Windows operating systems and Linux Kernel 2.6/3.x distributions specifically supporting System V features.

Note:

The use of hardware interrupts is prohibited when multi-process operations are enabled under the Windows operating system.

In this mode, all processes must invoke AR_OPEN during initialization of the process and AR_CLOSE upon termination. Failure to follow this strict requirement could result in irrecoverable errors. Note that board setup/initialization is only executed in AR_OPEN if no other processes have the board open. If another process has the board open (that is, if another process has opened the board using AR_OPEN but hasn't yet closed the board using AR_CLOSE), AR_OPEN attaches to the device without re-initializing the board or modifying board settings. Similarly,

AR_CLOSE only shuts down the board if no other processes have the board open. If another process has the board open, AR_CLOSE detaches from the board without shutting it down. Thus, board settings are preserved across all process invocations of AR_OPEN and AR_CLOSE. Multi-process mode is only required when accessing a single board from multiple processes. If multiple boards are installed, AR_CONC_NONE concurrency mode can be used as long as only one process interfaces with each board.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An invalid <i>board</i> value was provided.
ARS_INVARG	An invalid <i>item</i> or <i>value</i> parameter was provided.
ARS_BOARD_MUTEX	Creation of the Board Lock mechanism failed.
ARS_NO_OS_SUPPORT	The item selection not supported with the host operating system.

Arguments

CDEV_BOARD_TYPE	board	(input) Device to access. Valid range is 0-127.
CDEV_PARM_SSI_TYPE	item	(input) Attribute about which to set information, currently limited to a single option, ARU_CONCURRENCY_MODE.
CEI_UINT32	value	(input) the value to set the specified item.
AR_CONC_NONE		no multi-thread or multi-process support (default).
AR_CONC_MULTITHRD		multi-thread concurrency mode (see Description section for details).
AR_CONC_MULTIPROC		multi-process concurrency mode (see Description section for details).

AR_SET_RAW_MODE

Syntax

CDEV_API_RET_TYPE ar_set_raw_mode (CDEV_BOARD_TYPE board, CDEV_PARM_SSI_TYPE direction, CDEV_CHAN_TYPE channel, CDEV_PARM_SSI_TYPE control)

Description

This routine is designed to provide compatibility with the CEI-x20 ARINC APIs. The routine AR_SET_DEVICE_CONFIG is the recommended routine for manipulating the channel parity attribute.

Each transmit and receive channel can be configured to run in *raw* mode, where parity assignment and detection is disabled. When raw mode is selected, every 32-bit ARINC word is transmitted or received with the parity bit (msb) unchanged. This differs from a standard ARINC 429 data transfer in which the message parity is always calculated. Raw mode is typically used for older ARINC specifications such as ARINC 575.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVHARVAL	An invalid <i>channel</i> parameter was provided or the specified <i>channel</i> doesn't support parity selection.
ARS_INVARG	An invalid <i>direction</i> or <i>control</i> parameter was provided.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.

Arguments

CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

CDEV_PARM_SSI_TYPE direction (input) The type of channel specified in the channel argument (transmit or receive). Valid values to select transmit channels are:

TRANSMIT_CHANNEL (0)
ARU_XMIT (34)

Valid values to select receive channels are:

RECEIVE_CHANNEL (1)
ARU_RECV (35)

CDEV_CHAN_TYPE channel (input) Specifies which channel this routine is to access. Valid range is 0 to one less than the installed channel count for the respective channel type.

CDEV_PARM_SSI_TYPE control (input) Enables or disables raw mode.

AR_ON (7) enable "raw" mode, parity is disabled

AR_OFF (8) disable "raw" mode, parity assignment and/or checking is enabled

AR_SET_STORAGE_MODE

Syntax	CDEV_API_RET_TYPE ar_set_storage_mode (CDEV_BOARD_TYPE board, CDEV_PARM_SSI_TYPE mode)	
Description	<p>This routine is designed to provide backward compatibility with legacy ARINC API board-wide ARINC message storage selection. The routine AR_SET_DEVICE_CONFIG is the recommended routine for manipulating individual receive channel data storage modes.</p> <p>While CEI-x30 devices can store received data in either individual receive buffers or in a single merged receive buffer on an individual channel basis, this routine allows you to perform a single invocation to select the universal receive mode for all receive channels on the device, as either BUFFERED (individual) or MERGED.</p> <p>Each receive data API routine detects the respective receive channel assigned storage mode, and will acquire messages from the appropriate buffer as necessary.</p>	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVARG	An invalid <i>mode</i> value was provided.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
Arguments	<p>CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.</p> <p>CDEV_PARM_SSI_TYPE mode (input) The type of receive data storage mode to assign. Valid values are:</p> <p style="padding-left: 40px;">ARU_BUFFERED (0) use individual FIFO buffers</p> <p style="padding-left: 40px;">ARU_MERGED (2) use the merged FIFO buffer</p>	

AR_SET_TIME

Syntax

CDEV_API_RET_TYPE ar_set_time (CDEV_BOARD_TYPE board, pAR_TIMETAG_TYPE timeTag)

Description

This routine assigns a value to the specified CEI-x30 device internal timer or IRIG time generator based on an application-supplied time format and value.

Return Value

ARS_NORMAL	Routine execution was successful.
ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
ARS_INVARG	An invalid <i>timeTag</i> structure member <i>timeTagFormat</i> selection was provided.
ARS_INVHARVAL	The external IRIG time option was requested via the <i>timeTagFormat</i> structure member, but IRIG is not available on the specified device.
ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.

Arguments

CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

pAR_TIMETAG_TYPE timeTag

(input) The 64-bit device timer or 32-bit IRIG time generator value to assign to the respective hardware. Valid options for the *timeTagFormat* structure member are:

AR_TIMETAG_EXT_IRIG_64BIT (0)

AR_TIMETAG_INT_USEC_64BIT (1)

To assign a 32-bit IRIG Day/Time value, the *timeTag* structure member should be defined as a 30-bit value of the following bit format:

29-28	27-24	23-20	19-18	17-14	13-11	10-7	6-4	3-0
hundreds of days	tens of days	days	tens of hours	hours	tens of minutes	minutes	tens of seconds	seconds

To assign a 64-bit internal timer value, the *timeTag* structure member should be defined as a 64-bit 1 microsecond resolution time value.

The *timeTagRef* structure member is not used by this routine.

See the section titled *Time-tag Structure Definition* for more information on the AR_TIMETAG_TYPE data structure.

AR_SLEEP

Syntax

CEI_VOID ar_sleep (CEI_UINT32 sleep_ms)

Description

This routine suspends execution of the calling thread for the specified number of milliseconds. Platform-dependent thread delay methods are used to implement this operation, defined below for the supported operating system. The accuracy of this operation is dependent upon the accuracy of the underlying operating system call.

Return Value

None

Arguments

CEI_INT32 sleep_ms (input) Sleep duration, in milliseconds.

AR_SET_TIMERRATE

Syntax

CEI_VOID ar_set_timerrate (CDEV_BOARD_TYPE board,
CDEV_PARM_SSI_TYPE rate)

Description

This routine assigns the API internal timer reference resolution for compatibility with applications based on the CEI-x20 product family device timer and time-tag operation. When you invoke this routine, the CEI-x30 API sets the current timer usage and time-tag reporting mode to the “CEI-x20 compatibility mode”. In this mode, all scheduled message rate and start offset values and receive message time-stamp values are referenced in terms of the resolution value assigned in the “rate” parameter instead of the standard one millisecond (for scheduled message rate/offset) or one microsecond (for receive message time-stamps).

The actual CEI-x30 hardware device time-tag reference timer resolution is not programmable; rather, it is a fixed one microsecond resolution.

The CEI-x30 message scheduler minimum rate resolution is fixed at a one millisecond resolution. As a result, any timer rate assignment having a resolution that is not divisible by, or is less than, one millisecond, coupled with an attempt to define a message scheduler entry rate or start offset value that is not divisible by one millisecond results in that value being assigned to the nearest 1 millisecond value below the assigned value.

Arguments

CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

CDEV_PARM_SSI_TYPE rate (input) Resolution of the CEI-x30-emulated timer operation, specified as a tick-timer value having a resolution of 250 nanoseconds.

AR_STOP

Syntax	CDEV_API_RET_TYPE ar_stop (CDEV_BOARD_TYPE board)	
Description	This routine assigns the global enable register Global Enable bit to be <i>disabled</i> for the specified device. All active message processing is terminated upon execution of this routine.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_BOARD_MUTEX	Access to the Board Lock timed-out/failed.
Arguments	CDEV_BOARD_TYPE board	(input) Device to access. Valid range is 0-127.

AR_UPDATE_MSG_BLOCK

Syntax	CEI_INT32 ar_update_message_block (CDEV_BOARD_TYPE board, CEI_UINT32 numberOfMsgs, CEI_UINT32 msgIndex, pCEI_UINT32 msgValues)	
Description	This routine modifies only the message value of a successive block of message scheduler table entries previously defined via ar_define_msg or ar_define_msg_block.	
Return Value	ARS_NORMAL	Routine execution was successful.
	ARS_INVBOARD	An uninitialized board or invalid <i>board</i> value was provided.
	ARS_INVARG	An invalid message scheduler table index was provided.
Arguments	CDEV_BOARD_TYPE board	(input) Device to access. Valid range is 0-127.
	CEI_UINT32 numberOfMsgs	(input) The number of message scheduler entries to modify.
	pCEI_UINT32 msgValues	(input) The updated 32-bit ARINC message values to overwrite the values in the specified message scheduler table entries.

AR_VERSION

Syntax

CEI_VOID ar_version (pCEI_CHAR verstr)

Description

This routine retrieves the current software version number of the device API.

Arguments

pCEI_CHAR verstr	(output) String representation of the API Version number consisting of up to 10 characters.
------------------	---

AR_WAIT

Syntax

CEI_VOID ar_wait (CEI_FLOAT nsecs)

Description

This routine delays the calling application by the specified number of seconds. The delay is based on the respective OS system time utility.

Arguments

CEI_FLOAT nsecs (input) Number of seconds to delay.

AR_XMIT_SYNC

Syntax

CDEV_API_RET_TYPE ar_xmit_sync (CDEV_BOARD_TYPE board, CDEV_CHAN_TYPE channel)

Description

This is a utility that waits for all the data in the transmit buffer to be loaded into the ARINC transmitter. It is useful in an application that is sending data out but doesn't want to halt the interface until everything has been sent.

Arguments

CDEV_BOARD_TYPE board (input) Device to access. Valid range is 0-127.

CDEV_CHAN_TYPE channel (input) The transmit channel on which to wait.