AIM
RIGHT ON TARGET

Avionics Databus Solutions

ARINC664 / AFDX

Interface Module

Reference Manual

V19.6.x Rev. A

March 2020

www.aim-online.com

ARINC664 / AFDX

Software Library
Reference Manual

Reference Manual

V19.6.x Rev. A

March 2020

AIM NO.
60-15900-36-19.6.X

AIM – Gesellschaft für angewandte Informatik und Mikroelektronik mbH

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

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

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

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

Table of Contents

List of Tables

List of Figures

1. INTRODUCTION

1.1 General

The AIM-AFDX High Level Application Interface Library provides a comprehensive set of 'C' functions for interfacing application programs to the AIM AFDX Interface Modules listed below. The 'y' in the 'AyC' is an AIM standard placeholder for encoding the module's platform (where 'y' can be replaced with either C, V, or M as shown below).

C:	***ACC-FDX-2/4***	Compact PCI (cPCI) 6U
V:	***AVC-FDX-2/4***	VME
M:	***AMC-FDX-2***	PMC Module

Other Standard Modules are:

API-FDX-2	PCI Module
APM-FDX-2	PC-Card Module (with limited simulation functionality)
fdXTap	USB Module for monitor only
APU-FDX-2	USB Module (full function)
APE-FDX-2	PCIe Module
AXC-FDX-2	XMCModule PCIe based
AMCX-FDX-2	PMC Module PCIe based

The AIM AFDX High Level Application Interface encapsulates operating system specific handling of Host-to-Target communication in order to support the platform independent implementation of the user's applications by providing a unique set of functions for hardware communication to the AFDX target.

To access the boards and also the resources on the boards a client server interface for a network environment is supported. The AIM AFDX Application Interface currently supports all 32-bit and 64-bit Windows® platforms (Win7 and newer). All other systems, e.g. embedded VME with VxWorks, LINUX, LynxOS etc. currently support 'local' servers.

The AIM AFDX High Level Application Interface for PCI and cPCI is available as Dynamic Link Libraries (DLLs) for the platforms mentioned above (Microsoft compatible). The AFDX High Level Application Interface DLL can be used by each programming tool having the capability of interfacing DLLs (32-Bit and 64-Bit). Also dynamic Libraries for LINUX (32-Bit and 64-Bit) are available.

For embedded VME applications (e.g. AMC mounted on a VME Carrier → ACC), the Application Interface Library is provided with the source code, in order to support integration into a customer specific Operating System Environment. Support for the more popular embedded Operating Systems, like VxWorks and LynxOS is built in per Standard.

Each command to the Interface Library will be translated to AFDX Target commands. Long parameter lists of some driver commands are substituted by specific data types (C-structures) in order to reduce the number of function parameters. In addition to the target access functions, a set of administration functions are provided for handling general driver communication, and the client server interface and login mechanism to gain access to the hardware resources. Due to the common core architecture of the AIM bus interface modules, the Driver Software rubs On-Board on the Application Support Processor, with Real-Time-Operating System support. Therefore the command set, provided by the Application Interface does not show significant differences between the platforms.

Since it is possible to have concurrent access to the AIM AFDX High Level Application Interface, (e.g. using multiple thread/task techniques), the AFDX High Level Application Interface handles those conditions via operating system specific capabilities, using Mutexes and Semaphores.

The number of AFDX boards accessible through the High Level Application Interface Library is only limited by memory.

1.2 Applicable Documents

The following documents shall be considered to be part of this document to the extent that they are referenced herein.

[1] PCI Local Bus Specification, Revision 2.1, June 1991

[2] AFDX End System Detailed Functional Specification
AIRBUS Issue: 4.0, Date: 24/10/2001, Ref.:L42D1515045801

[3] AFDX Switch Detailed Functional Specification
AIRBUS Issue: 2.0, Date: 14/09/2001, REF.:515.0519/2001

[4] Arinc664 Programmer's Guide

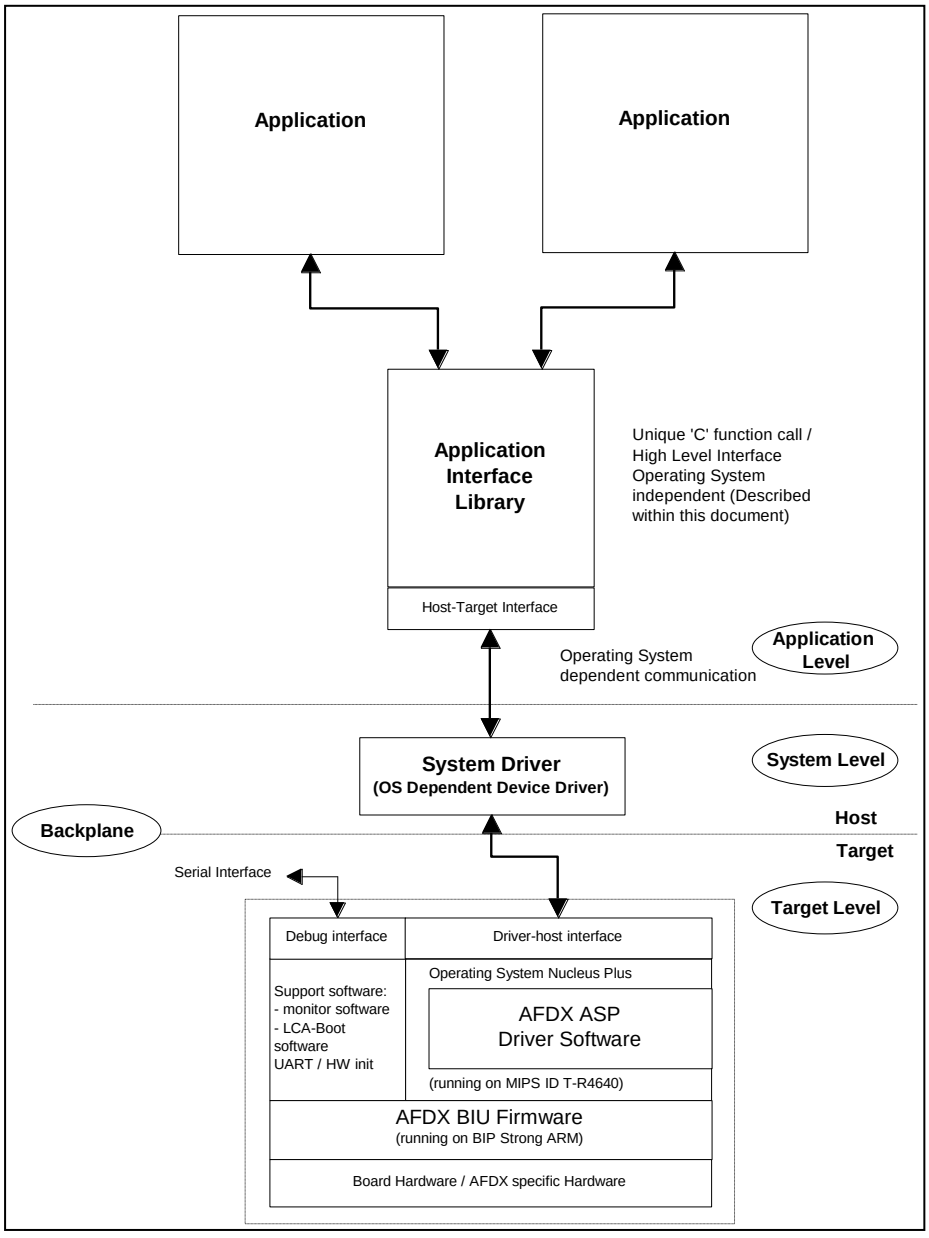
[5] Arinc664 Getting Started for Windows

[6] Arinc664 Getting Started for Linux

2. SOFTWARE AND HARDWARE STRUCTURE

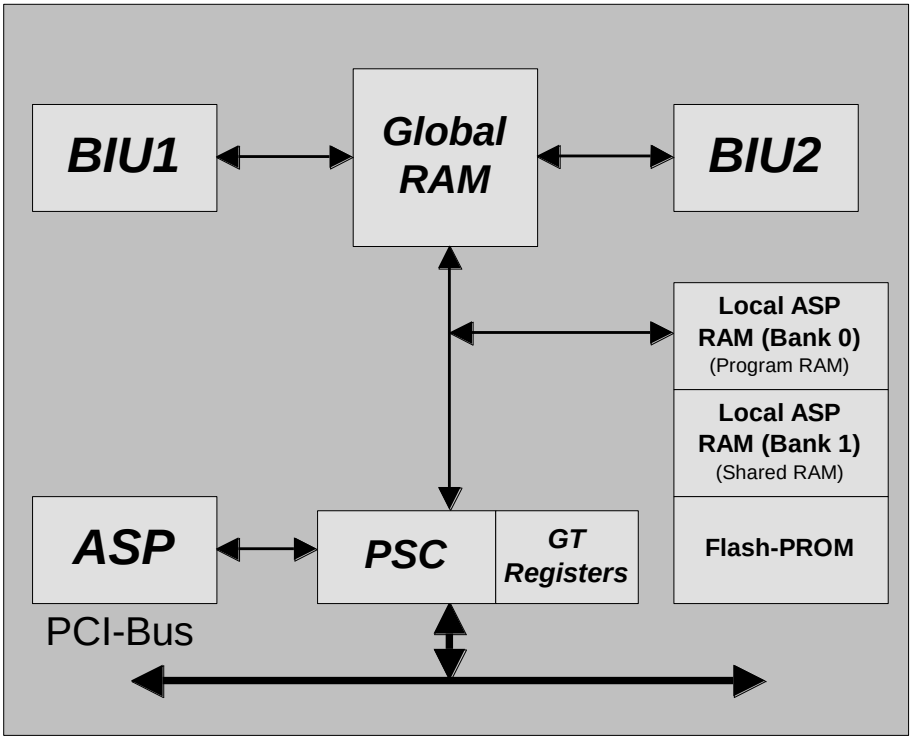
2.1 AFDX Software Structure

Figure 2.1-1: Partitioning of the AFDX Driver Software



2.2 Hardware Structure

Figure 1.2-1: Simplified AyI-FDX Hardware Structure



Explanation

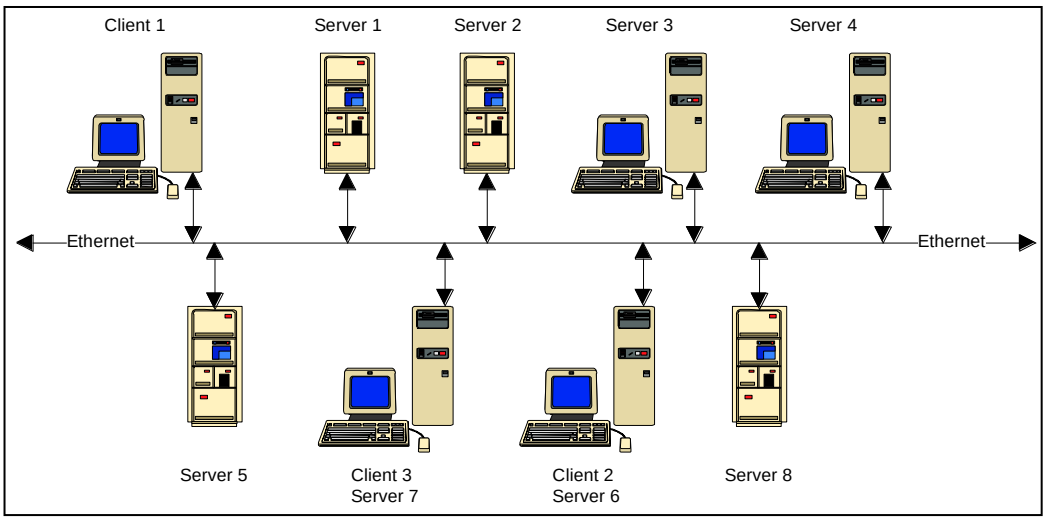
ASP	Application Support Processor - RISC Processor Intel XScale 80200 (ARM-family)
BIU	Bus Interface Unit RISC Processor Intel XScale 80200 (ARM-family)
PSC	- PCI and System Controller - AIM PCI-LCA
Global RAM	Static RAM module Shared between BIU, ASP and PCI bus
Local ASP RAM	- Bank 0 used for ASP local programs (ASP Driver Software, Monitor) - Bank 1 used as Shared RAM area for Host-to-Target communication
Flash-PROM	Board Configuration Memory

2.3 Client-Server Structure in a Network Environment (Windows Environment)

It is possible to have several ACI-FDX cards in one computer, but it is also possible to have the cards distributed to several computers in a Network. A sample configuration is shown in Figure 2.3-1.

- Each computer, which has ACI-FDX hardware installed acts as **Server**. The server computer provides board functionality to the connected clients. To access the hardware from a client it is necessary to run the AIM Network Server (ANS) on the Server computer.
- The computer which contains the application software acts as **Client**. A client does not require ACI-FDX hardware.
- However, ACI-FDX hardware can be installed on the computer which contains the application software (a Client). In this configuration, local access from the application to the hardware is provided. Therefore, it is not necessary to run the ANS.

Figure 2.3-1: ACI-FDX Network Structure

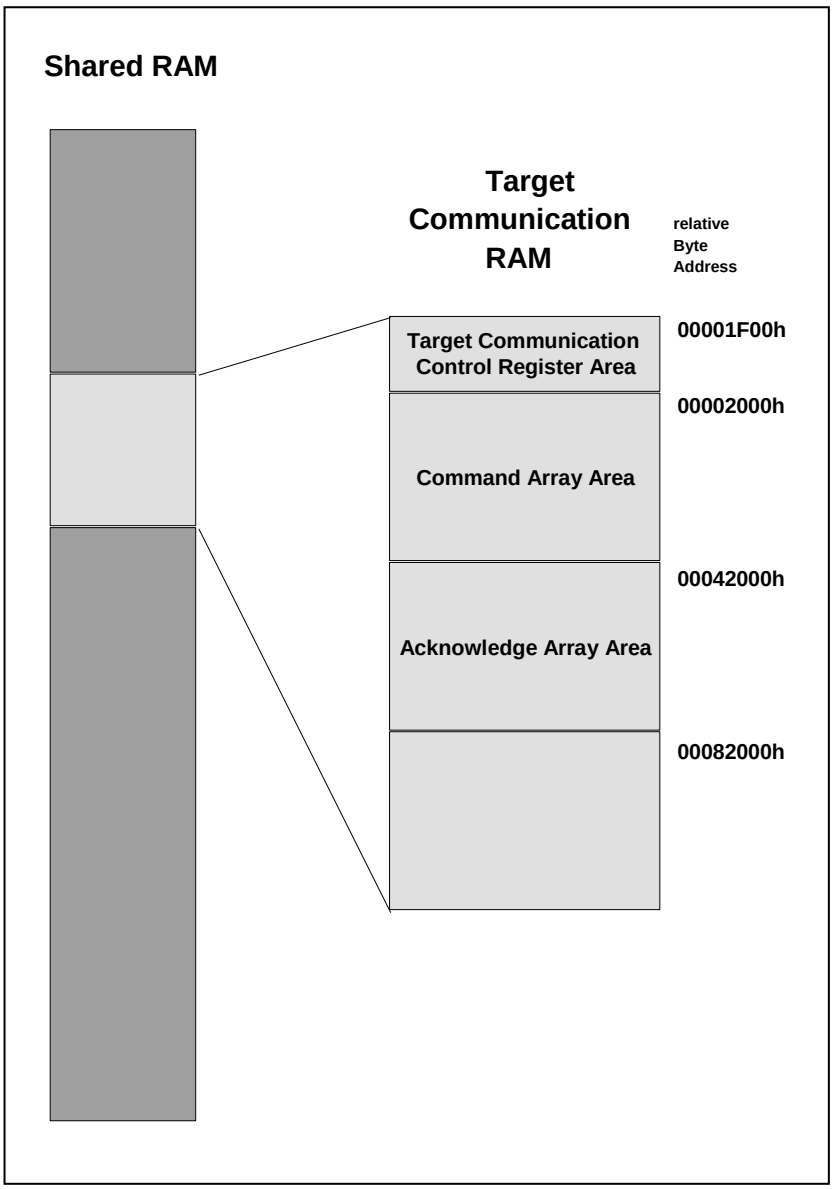


2.4 Host-to-Target Communication

For Host-to-Target communication a small area of the ASP Shared RAM is used and partitioned as shown in Figure 2.4-1.

The FDX High Level Application Interface System Level Drivers for the applicable operating system communicate with the FDX Target Software by writing command information to the Target Command Array (send message). Acknowledge information is returned in the Target Acknowledge Array (receive Message) from the FDX Target Software.

Figure 2.4-1: Host-to-Target Communication



3. APPLICATION INTERFACING

3.1 General

To interface the user's application program to the target hardware, the application program is required to call the basic functions of the FDX Application Interface Library.

Before any driver function can be executed the FDX Application Interface Library must be initialized using the following function:

FdxInit(...)

This function performs the basic initialization of the library and returns a list of servers found in the network environment. The basic case is to find the server named 'local' which describes the computer where the application is running (can also be a stand alone system without any network). *FdxInit* shall be called as the first function.

Note:	***Using the Interface Modules in an environment, which does not support an automatic mapping and resource assign, like PCI Plug&Play, specific Initialization functions are necessary, before the application can continue with the FdxInit.***

To get the number of boards and their configuration the following command should be performed:

FdxQueryServerConfig(...)

This function returns a list of available resources of one server, where a resource can be a board or a physical port of one board.

To establish target communication for a specific resource the following function shall be called:

FdxLogin(...)

This function must be called as part of the FDX device initialization procedure. **FdxLogin** returns a unique handle which identifies the selected resource for the calling application and initializes an internal structure for communication. Upon successful execution of the **FdxLogin** function all driver functions related to the selected resource can be called in order to control the required operation. So functionality is distinguished between board related and port related. To execute board functionality the user must be logged in to a board resource. To execute port functionality the user must be logged in to a port resource.

Any application program shall finish communication to a resource with the following function:

FdxLogout (...)

This function performs a cleanup of the specified FDX device and must be called to shut-down communication for the specified resource. After calling this function, the handle is invalid and it is not possible to use it for further function calls.

FdxExit()

This function performs a cleanup of Library internal used memory. This must be called as the last function before unloading the Libary

3.2 Generic Initialization Functions (VME, local PMC slot)

For VME based FDX modules (AMC-FDX-2 on AVC carrier or AMC-FDX-2 e.g. on PMC slot of CPU) the following initialization has to be done **before** general initialization defined in chapter 3.1.

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

Examines the board(s) on a specific VME slot/ A16 address and runs a PCI config cycle:

AiVmeExamineSlot()

Checks the type of module:

AiAfdxCheckModule()

Scans the local PCI bus for known boards and returns the number of boards handled by this driver:

AiPciScan()

Provides the PCI header information of a board on the local PCI bus:

AiPciGetHeader()

Sets interrupt specific parameters of the board:

AiVmeInitGenericInterrupt()

Maps a board to the VME bus:

AiVmeAfdxMapModule()

Removes a board from the VME bus:

AiVmeAfdxUnmapModule()

3.3 Error Reporting

Each function of the FDX Interface Library has defined return values. For a successful function call the function returns a zero. For an unsuccessful function call the function returns a negative return value. These return values may be classified in prioritized groups of e.g. errors, warnings and information.

In addition to the return value, a defined Error Handler for special error reporting will be invoked by the Library. The error handler is an encapsulated function inside the Library with a defined interface.

3.4 Necessary Files and Defines

For all platforms two C-syntax header files, **AI_CDEF.H** and **AIFDX_DEF.H,** are provided which contain all information concerning constants, data types and function prototypes. The application program only has to include **AIFDX_DEF.H**.

For VME platforms an additional C syntax header files is required: **AIVME_DEF.H**..

The application program must enter the following preprocessor definition (e.g. usually /D or -D option of the C-compiler):

_AIM_FDX and _AIM_WINDOWS	For using the 32-Bit DLL (Windows applications) the **aim_fdx.lib** import library must be linked to the application.

The calling convention for 32-Bit AIM_FDX Application Interface DLLs is ***stdcall***.

The AFDX import library is generated with MICROSOFT Visual C/C++ compiler and is available for 32-Bit applications.

_AIM_FDX and _AIM_VME	For using the Library in embedded VME environment, running under any operating system.
__LYNX_OS__	Additional define, for using the Library under LynxOS (supported version: 3.1.0) .
__VXWORKS__	Additional define, for using the Library under VxWorks.
HOST_ENDIAN_BIG or	
HOST_ENDIAN_LITTLE	For switching endian order to support Little Endian and Big Endian Systems.
_AIM_CPCI_VXWORKS	Additional define, for using cPCI under VxWorks. e.g. AmcFdx on ACC carrier.

When using the Library in embedded VME environment, or LINUX the source code is provided.

3.5 Files of the FDX High Level Interface

3.5.1 Include Files

The following FDX Application Interface Library 'C' syntax include file is valid for all platforms:

aifdx_def.h

It defines all Function Prototypes, Data types and Constants.

In addition, the following header file is required for VME platforms:

aivme_def.h.

3.5.2 Libraries and Files

Table 3.5.2-I: Necessary Application Interface Level Files

Operating System Platform	*Files*	*Comment*
Windows XP and later Windows XP and later	aim_fdx.dll aim_fdx.lib	32 or 64-bit DLL Corresponding 32 or 64-bit import library
LINUX	libtest.a or libaim_fdx.so	Library is delivered in Source Code and will be built at insallation time.
VxWorks	None	Library is delivered in Source Code
LynxOS	All Library Source Files + pre-compiled LynxOS Library lib_aim_fdx.a + makefiles	Library is delivered in Source Code

3.5.3 System Level Driver Files

Table 3.5.3-I: Necessary System Level Driver Files

Operating System Platform	*Files*	*Comment*
Windows XP and later [1)]	Aim_Fdx.sys AimFdxUsb.sys AiUBulk.sys and AiUGen.sys	WDM Kernel Mode Device Driver for all PCI based interface modules APU-FDX WDF Kernel Mode Device Driver WDM Kernel Mode Device Driver for USB based fdXTap
LINUX	aim_fdx.ko aimfdxusb.ko	System Driver is delivered in Source Code and will be built at insallation time.
VxWorks	None	Currently, the VMEbus access is performed via the Library.
LynxOS	None	Currently, the VMEbus access is performed via the Library, using “smem_create(..)” system function for getting User Level access to the VMEbus A16 and A32 areas. Future versions may use a LynxOS Device Driver, whereby this is transparent for the Library User.

Note: 1) Windows System drivers are available for 32bit and 64bit Windows. The 32bit versions can be used for Windows XP and later. The 64bit versions are built for Windows 7 and later.

4. FUNCTION REFERENCE

This chapter contains a reference for all FDX High Level Library 'C' functions. Special data type definitions are described with the corresponding 'C' function which is using the data type.

The first parameter of each function is called "***ul_Handle***" and determines the FDX destination resource. This handle is returned by the login function at login time as a unique handle to that resource.

This parameter is mentioned but not described for each function since the parameter must be given for all functions with the exception of the system related functions.

All Functions with parameter "***ul_Handle***" can additionally return the following error codes:

FDX_CLIENTHANDLE_INVALID

FDX_RESOURCEID_INVALID

FDX_RESOURCETYPE_INVALID

4.1 Library Administration Functions

This section describes the commands used to gain general access to the physical resources provided on the FDX-2/4 board. There are also functions to observe the resources. The resources are divided in board- and port-resources.

Table 4.1-I: Library Administration Functions

Function	*Description*
Platform Independent	
FdxInit	Initializes the Interface Library. Returns a list of servers.
FdxExit	Cleanup the Library internal used memory structures.
FdxQueryServerConfig	Returns a list of resources of one server. Connects additional server (additional to local available resources)
FdxQueryResource	Gets detailed information about a resource
FdxInstallServerConfigCallback	Provides mechanism to notify PnP device changes
FdxLogin	Login for one resource
FdxLogout	Logout from a resource
FdxInstIntHandler	Installs a user-defined interrupt handler function
FdxDelIntHandler	Deletes the user-defined interrupt handler function
FdxQueryLoginInfo	Get Information about clients logged in to a specific resource
Platform Dependent - VME-Initialisation Function	
AiVmeExamineSlot	Examines the board(s) on a specific VME slot/ A16 address and runs a PCI config cycle
AiAfdxCheckModule	Checks the type of module
AiPciScan	Configures the board's interrupt level and vector # and defines the pointer to the Interrupt Routine
AiPciGetHeader	Provides the PCI header information of a board on the local PCI bus.
AiVmeInitGenericInterrupt	Sets interrupt specific parameters of the board.
AiVmeAfdxMapModule	Maps a board to the VME bus
AiVmeAfdxUnmapModule	Removes a board from the VME bus

4.1.1 Platform Independent Functions

4.1.1.1 FdxInit

Prototype:

AiReturn FdxInit(TY_SERVER_LIST **ppx_ServerNames);

Purpose:

This function initialises the entire application interface and must be called at first in an application program, before any other function is applied. This function returns a list of computer names of the network environment, where the ANS (AIM network server) is running and FDX boards are available to work with.

Note: For this version this function will return only "local" - server. Use function FdxQueryServerConfig to connect another server.

Input:

None

Output:

TY_FDX_SERVER_LIST **ppx_ServerNames

Pointer to a pointer to a list of structured elements, containing the names of the available servers (e.g. "\\SW-PC-06" or "192.168.0.119") and a pointer to the next element. The end of the list is indicated by a NULL pointer in the next pointer entry. A special case is the name is "local" which describes the computer where the Interface Library is running

```
#define MAX_SERVER_NAME_LEN 32

typedef struct _server_list{
   struct _server_list *px_Next;
   AiChar auc_ServerName[MAX_SERVER_NAME_LEN];
   const AiUInt32 ul_StructId;
} TY_SERVER_LIST;
```

struct _server_list *px_Next

Pointer to the next element of the List. A NULL pointer indicates the last element of the list.

AiChar auc_ServerName[MAX_SERVER_NAME_LEN]

Server name (e.g. "\\SW-PC-06" or "192.168.0.119"). The name 'local' indicates that the server is the machine the interface library is running.

const AiUInt32 ul_StructId

Element, which identifies the type of this structure (see ***FdxCmdFreeMemory***)

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.1.1.2 **FdxExit**

Prototype:

AiReturn FdxExit(void);

Purpose:

This function performs a cleanup of all internal used memory structures. This shall be used as last function before unloading the Library to guarantee no memory leaks at time of unloading.

Input:

None

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.1.1.3 FdxLogin

Prototype:

```
AiReturn FdxLogin(  const AiChar *ac_SrvName,
                    const TY_FDX_CLIENT_INFO *px_ClientInfo,
                    AiUInt32 ul_ResourceID,
                    AiUInt32 ul_Privileges,
                    AiUInt32 *pul_Handle );
```

Purpose:

This function provides log in to a resource.

Note: ***For other than "local" available resources the server has to be connected by using function FdxQueryServerConfig before using this function.***

Input

*AiChar *ac_SrvName*

Address of the server that is hosting the AFDX resource.

Value	*Description*
"local"	Resource is hosted on the local PC
<SrvName>	Name or IP address of the PC, where the ANS664 Server (AIM Network Server) is running (e.g. "myhost.mydomain.com" or "192.168.0.119")

*const TY_FDX_CLIENT_INFO *px_ClientInfo*

Pointer to an information structure about the calling client, describing the client application and the client computer environment.

```
#define  MAX_FDX_CLIENT_HOST_NAME   32
#define  MAX_FDX_CLIENT_USER_NAME   32
#define  MAX_FDX_CLIENT_APPLI_NAME  32
#define  MAX_FDX_CLIENT_APPLI_VERS  16

typedef struct {
   AiChar ac_ClHostName[MAX_FDX_CLIENT_HOST_NAME ];
   AiChar ac_ClUser[MAX_FDX_CLIENT_USER_NAME ];
   AiChar ac_ClApplication[MAX_FDX_CLIENT_APPLI_NAME ];
   AiChar ac_ClApplicationVersion[MAX_FDX_CLIENT_APPLI_VERS];
} TY_FDX_CLIENT_INFO;
```

AiUInt32 ul_ResourceID

The Resource ID identifies one resource of a server. This resource can be either a board or an Ethernet port of a board. The resource ID is obtained by a calling the function FdxQueryServerConfig(...).

AiUInt32 ul_Privileges

Defines the access mode and access rights the client has to the selected resource.

Value	*Description*
PRIVILEGES_ADMIN	Administrator Privileges
other values for future expansion	

Output

*AiUInt32 *pul_Handle*

Unique handle to the resource, which can be either a board for board level functions or an Ethernet port for port functionality. This handle is necessary for all future calls to this resource. If login to a resource fails, this handle will be NULL.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.1.1.4 **FdxLogout**

Prototype:

AiReturn FdxLogout (AiUInt32 ul_Handle);

Purpose:

This function closes the application interface for the specified resource and must be called last in an application program for all opened resources. After calling this function the handle is invalid and it is not possible to use it for further function calls.

Input

None

Output

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.1.1.5 **FdxDelIntHandler**

Prototype:

AiReturn FdxDelIntHandler (AiUInt32 ul_Handle,
AiUInt8 uc_Type);

Purpose:

Uninstalls an user interrupt handler function, which has been installed previously with the function "**FdxInstIntHandler**".

Input:

AiUInt8 uc_Type

Interrupt Type

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

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.1.1.6 FdxInstIntHandler

Prototype:

AiReturn FdxInstIntHandler (AiUInt32 ul_Handle,
AiUInt8 uc_Type,
TY_INT_FUNC_PTR pf_IntFunc);

Purpose:

This function is used to install a user-defined interrupt handler function. It is possible to define interrupt handler functions for TBD related interrupts.

If there is the need of an interrupt handler function that handles several interrupt types, it is necessary to call this function for all wanted different interrupt types each with the same given interrupt handler function "***pf_IntFunc***".

Input:

AiUInt8 uc_Type

Interrupt Type

Defines the type of interrupt which will be connected to the interrupt handler function given in "***pf_IntFunc***". 0 <= uc_Type <= FDX_INT_MAX.

Constant	*Description*
FDX_INT_TX	Interrupt on TX related events
FDX_INT_RX	Interrupt for RX related events

TY_INT_FUNC_PTR pf_IntFunc

Pointer to the interrupt handler function of the user application.

```
typedef void (*TY_INT_FUNC_PTR)( AiUInt8 bModule,
                                 AiUInt8 uc_Port,
                                 AiUInt8 uc_Type,
                                 TY_FDX_INTR_LOGLIST_ENTRY x_Info );
```

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

AiUInt8 b_Module

Module Number of the AIM board that generated the interrupt.

AiUInt8 uc_Port

Port Number of the AIM board that generated the interrupt.

Value	*Description*
1	Port number 1
2	Port number 2
3	Port number 3
4	Port number 4

AiUInt8 uc_Type

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

TY_FDX_INTR_LOGLIST_ENTRY x_Info

Contains detailed information about the cause of the interrupt.

```
typedef struct
{
   TY_LOGLIST_1_ENTRY x_LWordA;
   AiUInt32 ul_LWordB;
   AiUInt32 ul_LWordC;
   AiUInt32 ul_LWordD;
   AiUInt32 ul_LWordE;
   AiUInt32 ul_LWordF;
} TY_FDX_INTR_LOGLIST_ENTRY;
```

TY_LOGLIST x_LWordA

```
typedef union {
   AiUInt32 ul_All;
   struct {
      AiUInt Info:24;
      AiUInt port:3;
      AiUInt type:5;
   }t;
   struct {
      AiUInt reserved:24;
      AiUInt port:3;
      AiUInt dma:1;
      AiUInt cmd:1;
      AiUInt target:1;
      AiUInt biu1:1;
      AiUInt biu2:1;
   }b;
} TY_LOGLIST_1_ENTRY;
```

Interrupt Loglist Event, Entry Word 1

AiUInt Info;

TBD

AiUInt port;

Port which has interrupted

AiUInt type;

Description	***Interrupt Type***
HOST_INT_DMA	0x0001
HOST_INT_CMD	0x0002
HOST_INT_TRG	0x0004
HOST_INT_BIU1	0x0008
HOST_INT_BIU2	0x0010

AiUInt32 ul_LWordB

Interrupt Loglist Event, Entry Word 2

1) For interrupt type FDX_INT_RX and Loglist Entry Word A type HOST_INT_BIU1 or HOST_INT_BIU2, for PINT and MBF:
2) For interrupt type FDX_INT_RX and Losglist Entry Word A type HOST_INT_TRG

TYPE	Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
FDX_INT_RX [1]	VL number							
FDX_INT_RX [2]	Type			UDF	Reserved		UBF	Reserved
FDX_INT_TX [3]	Interrupt Identifier							

TYPE	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
FDX_INT_RX [1]	VL number							
FDX_INT_RX [2]	Reserved							
FDX_INT_TX [3]	Interrupt Identifier							

TYPE	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
FDX_INT_RX [1]	VL number							
FDX_INT_RX [2]	Reserved							
FDX_INT_TX [3]	Interrupt Identifier							

TYPE	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FDX_INT_RX [1]	VL number							
FDX_INT_RX [2]	Reserved							
FDX_INT_TX [3]	Interrupt Identifier							

Type

0x04 Udp Interrupt
0x05 Continuous Capture Interrupt

UDF

Update Flag. Set to 1 when the interrupt loglist entry is written.

UBF

UDP Buffer event flag. Set to 1 when a message is written the UDP port buffer.

AiUInt32 ul_LWordC

Interrupt Loglist Event, Entry Word 3

TYPE	Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
FDX_INT_RX	3h (Type)			UDF	PINT	MTI	MBF	MSI
FDX_INT_RX[2]	Admin Structure Pointer (shared RAM) for Continuous Capture							
FDX_INT_TX[3]	2h (Type)			UDF	PINT		Instruction	

TYPE	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
FDX_INT_RX	MST	ROV	Reserved					
FDX_INT_RX[2]	Admin Structure Pointer (shared RAM) for Continuous Capture							
FDX_INT_TX[3]	Instruction				FTI			

TYPE	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
FDX_INT_RX	Physical Port No.							
FDX_INT_RX[2]	Admin Structure Pointer (shared RAM) for Continuous Capture							
FDX_INT_TX[3]	Physical Port No.							

TYPE	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FDX_INT_RX	Trigger Control Block Index							
FDX_INT_RX[2]	Admin Structure Pointer (shared RAM) for Continuous Capture							
FDX_INT_TX[3]	Reserved							

Type

Value	Description
0h	TX Simulator related Interrupt Entry (STM)
1h	Replay related Interrupt Entry (RP)
2h	Generic Transmit Operation related Interrupt Entry (GTM)
3h	Monitor / Receive Operation related Interrupt Entry (RX/MN)
4h	Reserved

UDF

Update Flag. Set to one when Interrupt Loglist Entry is written.

Physical Port No

This field indicates the Physical Port, which releases the current interrupt. For Port A the field will be set to 0x00 and for Port B the field will be set to 0x01. If the module is configured as redundant Interface for transmit and receive operation, only the identifier for the Physical Port A will be shown in this field.

PINT

Monitor/Receive Operation Packet Interruptor Simulator/ Generic Transmit Operation Interrupt.

Logical '1': This interrupt is asserted, if the interrupt flag in the related transmit package is set (STM)

Logical '1': This interrupt is asserted, if a defined condition related to a transmit packet type 1 instruction becomes true (GTM)

Logical '1': This interrupt is asserted, if a defined condition related to a received packet becomes true (RX/MN)

FTI

Frame Transmitted Interrupt (GTM, STM, RP-Fifo)

Logical '1': The Interrupt is asserted, after a Frame was physically transmitted where the Frame Transmit Event Interrupt bit in the Frame Header was set.

INSTR

Generic Transmit Operation Instruction Type

This bit field shows the related instruction type, which releases this interrupt. (GTM)

MTI

Monitor Trigger Interrupt

Logical '1': This interrupt is asserted, if an trigger event becomes valid during the Trigger Control Block Processing.

MBF

Monitor / Receive Operation Buffer Full (or half full) Interrupt.

Logical '1': This interrupt is asserted due to the Monitor or Receive Operation Buffer Full event or the Half Buffer Full event. In Monitor Standard or selective Capture Mode only the Buffer Full event may assert an Interrupt (RX/MN)

MSI

Monitor Start Interrupt

Logical '1': This interrupt is asserted due to Monitor Start Trigger Event (MN)

MST

Monitor Stop Interrupt

Logical '1': This interrupt is asserted due to Monitor Stop Trigger Event (MN)

ROV

Receiver Overflow Interrupt

Logical '1': This interrupt is asserted, if the physical decoder device is stopped, due to an Overflow or an Overload condition, respectively. In this case the RX-port keeps still enabled , but no more frames will be received.

AiUInt32 ul_LWordD

Interrupt Loglist Event, Entry Word 4

TYPE	Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
FDX_INT_TX[3]	Transmit Buffer / Instruction Base Address							
FDX_INT_RX[1]	Reserved							
FDX_INT_RX[2]	UDP Port Handle (Bits 31-0) / Capture Buffer pointer (shared RAM) for Continuous Capture							

TYPE	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
FDX_INT_TX[3]	Transmit Buffer / Instruction Base Address							
FDX_INT_RX[1]	Receive / Monitor Buffer Pointer (Bits 25 - 0)							
FDX_INT_RX[2]	UDP Port Handle (Bits 31-0) / Capture Buffer pointer (shared RAM) for Continuous Capture							

TYPE	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
FDX_INT_TX[3]	Transmit Buffer / Instruction Base Address							
FDX_INT_RX[1]	Receive / Monitor Buffer Pointer (Bits 25 - 0)							
FDX_INT_RX[2]	UDP Port Handle (Bits 31-0) / Capture Buffer pointer (shared RAM) for Continuous Capture							

TYPE	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FDX_INT_TX[3]	Transmit Buffer / Instruction Base Address							
FDX_INT_RX[1]	Receive / Monitor Buffer Pointer (Bits 25 - 0)							
FDX_INT_RX[2]	UDP Port Handle (Bits 31-0) / Capture Buffer pointer (shared RAM) for Continuous Capture							

1) For interrupt type FDX_INT_RX and Loglist Entry Word A type HOST_INT_BIU1 or HOST_INT_BIU2:
2) For interrupt type FDX_INT_RX and Losglist Entry Word A type HOST_INT_TRG
3) For interrupt type FDX_INT_TX and Loglist Entry Word A type HOST_INT_BIU1 or HOST_INT_BIU2:

AiUInt32 ul_LWordE

Interrupt Loglist Event, Entry Word 5

TYPE	Bit 31		Bit 0
FDX_INT_TX[3]	Time Tag High		
FDX_INT_RX[1]			
FDX_INT_RX[2]			

AiUInt32 ul_LWordF

Interrupt Loglist Event, Entry Word 6

TYPE	Bit 31		Bit 0
FDX_INT_TX[3]	Time Tag Low		
FDX_INT_RX[1]			
FDX_INT_RX[2]			

Word ul_LWordE and ul_LWordF are extensions to get timing information for transmitted frames. Both words together are equal to structure TY_FDX_FW_IRIG_TIME.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.1.1.7 FdxQueryResource

Prototype:

AiReturn FdxQueryResource (const AiChar *ac_SrvName,
const AiUInt32 ul_ResourceID,
void *px_ResourceInfo);

Purpose:

To get information about one resource which clients are using this resource.

Note: For other than "local" available resources the server has to be connected by using function FdxQueryServerConfig before using this function.

Input

AiChar *ac_SrvName

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

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

AiUInt32 ul_ResourceID

Resource ID for this resource, returned by FdxQueryServerConfig(...).

Output

void *px_ResourceInfo

Pointer to a pointer of a resource information. Dependent on the input resource ID this pointer points to one of the following described structures (TY_FDX_BOARD_RESOURCE or TY_FDX_PORT_RESOURCE).

```
typedef struct{
   AiChar ac_BoardName[MAX_STRING_1];
   AiUInt32 ul_BoardSerialNo;
   AiUInt32 ul_NumOfEthernetPorts;
   AiUInt32 ul_Clients;
   AiUInt32 ul_GlobalMemSize;
   AiUInt32 ul_SharedMemSize;
   AiUInt32 ul_StructId;
} TY_FDX_BOARD_RESOURCE;
```

AiChar ac_BoardName[]

String, which contains the name of the board.
For the FDX-2/4 board this name is " AyI-FDX-2/4" "AMC-FDX-2".

AiUInt32 ul_BoardSerialNo

The Serial Number of the Board

AiUInt32 ul_NumOfEthernetPorts

Physical ports available on the board.

AiUInt32 ul_Clients

Reserved.

AiUInt32 ul_GlobalMemSize

Size of the Global Memory (Firmware Interface) in Byte

AiUInt32 ul_SharedMemSize

Size of Shared Memory (in Byte).

AiUInt32 ul_StructId

Element, which identifies the type of this structure (see ***FdxCmdFreeMemory***)

```
typedef struct{
   AiChar ac_PortName[MAX_STRING_1];
   AiUInt32 ul_BoardResourceID;
   AiUInt32 ul_StructId;
   AiUInt8  uc_PortNo;
   AiUInt8  uc_PortMode;
} TY_FDX_PORT_RESOURCE;
```

AiChar ac_ChnName

A special, definable name for this port.
For the FDX-2/4 board this name is "FDX Port X" where X represents the port number (1..4)

AiUInt32 ul_BoardResourceID

This is the resource ID of the board, where this port resource is located.

AiUInt8 uc_PortNo

Port Number.
For the FDX-2/4 board this value counts from 1 (port 1) to 4 (port 4).

AiUInt8 uc_PortlMode

NOTE: Port Mode is not supported in this version.

Value	*Description*
FDX_SINGLE	Single Mode
FDX_REDUNDANT	Redundant Mode

AiUInt32 ul_StructId

Element, which identifies the type of this structure (see ***FdxCmdFreeMemory***)

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.1.1.8 FdxInstallServerConfigCallback

Prototype:

AiReturn FdxInstallServerConfigCallback (const AiChar *ac_SrvName, FDX_SERVER_CALLBACK_FUNC *f_CallbackFunction);

Purpose:

This function provides mechanism to notify PnP device changes. For example adding or removing fdXTap at runtime requires an update of the resourcelist.

Note: It is recommended, that in case of removal of a device no action to the device shall be active.

Input:

AiChar ac_SrvName

Server name (e.g. "\\SW-PC-06" or "192.168.0.119"). The name 'local' indicates that the server is the machine the interface library is running.

Note: For this version this function can only be used on a local server

FDX_SERVER_CALLBACK_FUNC *f_CallbackFunction

Function pointer to a function of the application which shall be called when a device was added or removed to/from the system.

Prototype:

AiReturn FDX_SERVER_CALLBACK_FUNC (const AiChar ac_SrvName, const AiUInt32 ul_ChangeType, TY_RESOURCE_LIST_ELEMENT *px_ResourceList);

AiChar ac_SrvName

Name of the Server on which a PnP event occurred

AiUInt32 ul_ChangeType

Indicates if a device was added or removed.

Value	*Description*
FDX_RESOURCE_CHANGE_DELETE	device was removed
FDX_RESOURCE_CHANGE_ARRIVE	device was added

TY_RESOURCE_LIST_ELEMENT *px_ResourceList

Pointer of a list of resources as described for the function FdxQueryServerConfig. This list contains the resources of the added/removed device. In case of removing a device, the struct contains only the ResourceId of the related resources.

Note: ***The callbackfunction of the Application may not free the memory of the px_ResourceList as it is recommended for the FdxQueryServerConfig function call.***

Output

none

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.1.1.9 **FdxQueryServerConfig**

Prototype:

AiReturn FdxQueryServerConfig (const AiChar *ac_SrvName,
TY_RESOURCE_LIST_ELEMENT
*****ppx_ResourceList);***

Purpose:

This function is to get the configuration of AFDX boards of one computer or server. The function returns a list of resources available on that computer

Note:	***For this version this function can be used to connect a server. If a ac_SrvName other than "local" is specified this function checks that PC if a valid ANS Server is running. If a valid ANS Server is found on specified PC the function connects to that server and returns a list of available resources of that PC.***

Input:

AiChar ac_SrvName

Name of the PC, where the ANS (AIM Network Server) Server is running.

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

Output:

TY_RESOURCE_LIST_ELEMENT **ppx_ResourceList

Pointer to a pointer of list of resources. The list is a single pointered list, with the last element indexed to NULL. This means, the first entry in one list element is the pointer to the next list element. The last element is marked by a NULL pointer at this entry.

The memory of the list is allocated by the Application Interface Library. It is under control of the application to free or release this memory.

```
#define MAX_STRING_1 20

typedef struct _resource_list_element
{
   struct _resource_list_element *px_Next;
   AiUInt32 ul_ResourceID;
   AiUInt32 ul_ResourceType;
   AiChar ac_ResourceInfo[MAX_STRING_1];
   const AiUInt32 ul_StructId;
}TY_RESOURCE_LIST_ELEMENT;
```

TY_RESOURCE_LIST_ELEMENT *px_Next

Pointer to the next list element. If this pointer is a NULL pointer this is the last element in the list.

AiUInt32 ul_ResourceID

A number to the Resource which is unique over a complete server.

AiUInt32 ul_ResourceType

Describes the type of the following information:

Value	Type
RESOURCETYPE_BOARD	Board Information
RESOURCETYPE_PORT	Port Information

AiChar ac_ResourceInfo[]

String which can give some more information about this resource.

For example, the API-FDX-2 V2 board, this string will contain the name " API-FDX-2 V2". For the ports of this board this string must contain "FDX Port X", where X represents the number of the port (1..4).

const AiUInt32 ul_StructId

Element, which identifies the type of this structure (see ***FdxCmdFreeMemory***)

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.1.1.10 FdxQueryLoginInfo

Prototype:

```
AiReturn FdxQueryLoginInfo (  const AiChar *ac_SrvName,
                              const AiUInt32 ul_ResourceID,
                               AiUInt32 *ul_NumOfClients
                              TY_FDX_RESOURCE_LOGIN_INFO
                              **ppx_ResourceLoginInfo );
```

Purpose:

To get information about clients that are logged in to a specific resource.

Input

AiChar *ac_SrvName

Address of the server that is hosting the AFDX resource.

Value	*Constant*	*Description*
"local"	-	Resource is hosted on the local PC
<SrvName>		Name or IP address of the PC, where the ANS664 Server (AIM Network Server) is running (e.g. “myhost.mydomain.com” or “192.168.0.119”)

AiUInt32 ul_ResourceID

ID of an AFDX board or port resource.

The function provides information about all clients that are logged in to the resource specified by ul_ResourceID.

Use FdxQueryServerConfig(...) to obtain the resource IDs of a server.

Output

AiUInt32 *ul_NumOfClients

Number of Clients logged in to this resource.

TY_FDX_RESOURCE_LOGIN_INFO
*****ppx_ResourceLoginInfo***

Pointer to a pointer to a list of structures, which describe the clients logged in to this resource. The length of this list is described by ul_NumOfClients.

The memory of the array is allocated by the Application Interface Library. It is under control of the application to free or release this memory with the function

4.1.2 FdxAddIrigStructIrig and FdxSubIrigStructIrig

Prototype:

TY_FDX_IRIG_TIME FdxAddIrigStructIrig (const TY_FDX_IRIG_TIME *px_IrigTimeA
const TY_FDX_IRIG_TIME *px_IrigTimeB);

TY_FDX_IRIG_TIME FdxSubIrigStructIrig (const TY_FDX_IRIG_TIME *px_IrigTimeA
const TY_FDX_IRIG_TIME *px_IrigTimeB);

Purpose:

These two functions are used to calculate time tag sums and differences.

Result = IRIG Time A + IRIG Time B (add) or
Result = IRIG Time A - IRIG Time B (sub).
(Calculates with 366 Days / Year)

Input:

TY_FDX_IRIG_TIME *px_IrigTimeA

Format see FdxFwIrig2StructIrig function above.

TY_FDX_IRIG_TIME *px_IrigTimeB

Format see FdxFwIrig2StructIrig function above.

Output:

None

Return Value

TY_FDX_IRIG_TIME. The result of the IRIG time calculation. Format can be absolute or relative (see definition of TY_FDX_IRIG_TIME above)

Px_IrigTimeA	Operation	Px_IrigTimeB	Result	Function
Absolute	Add	Absolute	Relative	FdxAddIrigStructIrig
Absolute	Add	Relative	Absolute	FdxAddIrigStructIrig
Relative	Add	Absolute	Absolute	FdxAddIrigStructIrig
Relative	Add	Relative	Relative	FdxAddIrigStructIrig
Absolute	Sub	Absolute	Relative	FdxSubIrigStructIrig
Absolute	Sub	Relative	Absolute	FdxSubIrigStructIrig
Relative	Sub	Absolute	Absolute	FdxSubIrigStructIrig
Relative	Sub	Relative	Relative	FdxSubIrigStructIrig

(...).

```
typedef struct _fdx_resource_login_info {
   struct _fdx_resource_login_info *px_Next;
   TY_FDX_CLIENT_INFO x_ClientInfo;
   AiUInt32 ul_Privileges;
   AiUInt32 ul_Info;
   AiUInt32 ul_StructId;
} TY_FDX_RESOURCE_LOGIN_INFO;
```

struct _fdx_resource_login_info *px_Next

Pointer to the next element of the list. If this structure is the last in the list, this pointer will be NULL.

TY_FDX_CLIENT_INFO x_ClientInfo

A structured information about the logged in client, described in the command FdxLogin(...).

AiUInt32 ul_Privileges

Indication of the privileges the logged in client has (see ***FdxLogin***).

AiUInt32 ul_Info

Additional Information for future expansion.

AiUInt32 ul_StructId

Element which identifies the type of this structure (see ***FdxFreeMemory***).

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.1.3 Generic Initialisation Functions (VME, PMC slot on CPU)

This section describes the commands used to initialize the FDX module and configure for interrupts.

4.1.3.1 AiVmeExamineSlot

Prototype:

AiUInt32 AiVmeExamineSlot (TY_VME_EXAMINE_SLOT *in, TY_PCI_INFO *px_PCI_Info1, TY_PCI_INFO *px_PCI_Info2);

Purpose:

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

Input

TY_VME_EXAMINE_SLOT *in

```
typedef struct ty_vme_examine_slot
{
  AiUInt32  ul_A16Addr;
  AiUInt32  ul_Force;

  /* only needed for PMC on AVC */
  AiUInt32  ul_TempA32Addr;
  AiUInt32  ul_TempA32UserAccess;
} TY_VME_EXAMINE_SLOT;
```

AiUInt32 ul_A16Addr

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

AiUInt32 ul_Force

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

AiUInt32 ul_TempA32Addr

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

Only used for PMC on AVC

AiUInt32 ul_TempA32UserAccess

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

Only used for PMC on AVC

Output

TY_PCI_INFO *px_PCI_Info1

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

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

TY_PCI_CONFIGSPACE_HEADER x_PCIConfHd

```
typedef struct {
   AiUInt16 uw_DeviceID; // Device ID
   AiUInt16 uw_VendorID; // Vendor ID
   AiUInt16 uw_Status; // PCI status register
   AiUInt16 uw_Command; // PCI command register
   AiUInt32 ul_ClassCode_RevID; // PCI Class code / Revision ID
   AiUInt8 uc_Bist; // PCI BIST register
   AiUInt8 uc_HeaderType; // PCI header type
   AiUInt8 uc_LatencyTimer; // PCI latency timer
   AiUInt8 uc_CacheLineSize; // PCI cache line size
   AiUInt32 ul_BAR[6]; // Base address registers
   AiUInt32 ul_CardbusCisPtr; // Card Bus CIS Ptr
   AiUInt16 uw_SubsystemID; // Subsystem ID
   AiUInt16 uw_SubsystemVendID; // Subsystem Vendor ID
   AiUInt32 ul_ExpRomBaseAddr; // Expansion ROM Base Address
   AiUInt16 uw_Reserved1;
   AiUInt8 uc_Reserved2;
   AiUInt8 uc_CapabilitiesPtr; // Capabilities Ptr
   AiUInt32 ul_Reserved3;
   AiUInt8 uc_MaxLat; // Max latency
   AiUInt8 uc_MinGnt; // Min grant
   AiUInt8 uc_intr_pin; // Interrupt pin
   AiUInt8 uc_intr_line; // Interrupt Line
} TY_PCI_CONFIGSPACE_HEADER;


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

*TY_PCI_INFO *px_PCI_Info2*

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

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.1.3.2 **AiAfdxCheckModule**

Prototype:

AiUInt32 AiAfdxCheckModule(TY_VME_MAP_MODULE_IN *in);

Purpose:

This commands checks if a board is a known AFDX board.

Input

TY_VME_MAP_MODULE *in

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

Output

None

Return Value

Value	Constant	Description
0	**FDX_MODULE_TYPE_OTHER**	The board is not a known Arinc Afdx board
1	**FDX_MODULE_TYPE_PMC**	The board is an AMC-FDX board.
2	**FDX_MODULE_TYPE_AVI**	The board is an AVI-FDX
3	**FDX_MODULE_TYPE_PCI**	The board is an API/ACI FDX

4.1.3.3 **AiPciScan**

Prototype:

AiUInt32 AiPciScan(void);

Purpose:

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

Input

None

Output

None

Return Value

The number of boards found

4.1.3.4 **AiPciGetHeader**

Prototype:

TY_PCI_INFO* AiPciGetHeader(AiUInt32 ulModuleIndex);

Purpose:

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

Input

AiUInt32 ulModuleIndex

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

Output

None

Return Value

The PCI header of the board identified by the ulModuleIndex.

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

4.1.3.5 AiVmeInitGenericInterrupt

Prototype:

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

Purpose:

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

Input

*TY_PCI_INFO *px_PCI_Info*

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

*TY_INIT_VMEGENERIC_INT *in*

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

AiUInt32 ul_IrLevel

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

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

AiUInt32 ul_IrVector

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

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

*INTERRUPT_SET_FUNC *intSetFunction*

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

typedef AiUint8 INTERRUPT_SET_FUNC (AiUint8 vector,

AiUint8 level, VOID_FUNC *intFuntion);

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

The type VOID_FUNC is defined as follows:

typedef void VOID_FUNC(void);

INTERRUPT_FUNC *userIntFunction

This is a function pointer to a user function to do some own activities on a interrupt. This function must be from the type INTERRUPT_FUNC which is defined as follows:

typedef void INTERRUPT_FUNC(AiUInt8 uc_Module, AiUInt8 uc_Biu,

AiUInt8 uc_Type, TY_API_INTR_LOGLIST_ENTRY x_Info);

The user interrupt routine will be described in a seperate section.

Output

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.1.3.6 **AiVmeAfdxMapModule**

Prototype:

AiUInt32 AiVmeAfdxMapModule(TY_VME_MAP_MODULE_IN *in)

Purpose:

This Function initializes the FDX Board. The Function internally takes care, that the resources of the initialized board will be reported in the internal resource list. This function must be called to establish connectivity between the Application Interface and the AIM memory area for host-to-target communication.

Input

TY_VME_MAP_MODULE_IN *in

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

AiUInt32 A32Address

Is a User defined address in the VME A32 address space where the user wants to see the memory of the VME Carrier's AMCFDX Module in the VME address space (real, physical A32 address). The memory size for the module in the A32 space is dependant on the amount of memory on the module.

The real memory depends on the settings of the PCI-BAR register of the AIM Module. For all requested memory of the AIM Module one image is mapped on the VME-bus.

To access this mapped VME-memory for each BAR register memory request one

VME base pointer is returned.

Note: this adress has to be 16MB alligned.

AiUInt32 A32UserAccess

According to your VME-CPU system or operating system, the access address of

your CPU to the VME A32 range can be different to the real physical A32 address.

This parameter defines the CPU VME A32 access address.

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

AiUInt32 ul_Force

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

AiUInt32 ul_cPCI

This parameter has to be set to '1' if the boards are located on a cPCI bus..

For boards connected by VME bus (e.g. AMC-FDX on AVC-2 carrier) or for AMC-FDX located in the local PMC slots of a VME CPU, this value shall be set to zero.

TY_PCI_INFO *px_PCI_Info

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

Output

None

Return Value

Value	Description
0..32	Module ID. To be used intern for further Afdx commands
0x80000001	In->px_PCI_Info points not to an Afdx board
0x80000002	Internal error
0x80000003	Not enough memory left. Malloc failed

4.1.3.7 **AiVmeAfdxUnmapModule**

Prototype:

AiUInt32 AiVmeAfdxUnmapModule(TY_VME_MAP_MODULE_IN *in)

Purpose:

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

Input

TY_VME_MAP_MODULE_IN *in

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

Output

None

Return Value

Value	Constant	Description
0	**FDX_OK**	Execution success
Other value		Execution error

4.2 System Functions (Board level functions)

The Handle input parameter to the following functions must be a board related one.

Table 4.2-I: System Functions

Function	*Description*
FdxCmdBoardControl	Controls and resets the board operation mode.
FdxCmdIrigTimeControl	Reads and writes the onboard IRIG Time
FdxCmdStrobeTriggerLine	Provides a trigger output strobe on system command.
FdxReadBSPVersion	Reads version numbers of board software package components.
FdxVersionGet	New unified command to read a Version of a component.
FdxVersionGetAll	New unified command to read all Version of all available component.
FdxCmdBITETransfer	Performs transfer tests using available port resources of one FDX board.

4.2.1 FdxCmdBITETransfer

Prototype:

AiReturn FdxCmdBITETransfer (const AiChar * ac_SrvName,
const AiUInt32 ul_Board_ResourceID);

Purpose:

This function performs some transfer tests using available port resources of one FDX board. This function will determine the number of ports on the board. If only two ports, it will test them against each other. If four ports are used, Port 1 and Port 2 will be tested against each other and Port 3 and Port 4 will be tested against each other.
Port 1 and Port 2 must be connected with a Loop-Back cable (crossover), if available Port 3 and Port 4 must be connected with a Loop-Back cable (crossover).
The resources of the board under test shall be not logged in.

Note: For this version there is only "local" operation of the resources supported. This function will operate only with local available resources.

Input

AiChar ac_SrvName

Name of the PC, where the ANS (AIM Network Server) Server is running.

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

AiUInt32 ul_Board_ResourceID

The Resource ID identifies one resource of a server. For this function a board resource ID has to be used.. This resource ID is obtained by calling the function FdxQueryServerConfig(...).

Output

None

Return Value

Returns true on success or false on any kind of error.

4.2.2 FdxCmdBoardControl

Prototype:

AiReturn FdxCmdBoardControl (AiUInt32 ul_Handle,
AiUInt32 ul_Control,
const TY_FDX_BOARD_CTRL_IN *px_BoardControlIn,
TY_FDX_BOARD_CTRL_OUT *px_BoardControlOut);

Purpose:

This function is used to control the global settings of the board. For the AIM-FDX board this means the organization of the physical ports to logical single or redundant ports.

Input:

AiUInt32 ul_Control

Control the consequence of this function.

Value	*Description*
FDX_READ	Read back values
FDX_WRITE	Write values of aul_PortConfig to get a new configuration and perform a RESET

The output values will be updated in any control case.

Note: Writing a new configuration and Reset can only be done by a privileged user!

TY_FDX_BOARD_CTRL_IN px_BoardControlIN*

Pointer to a board control input structure.

```
typedef struct {
   AiUInt32 aul_PortConfig[FDX_MAX_BOARD_PORTS];
   AiUInt32 aul_PortSpeed[FDX_MAX_BOARD_PORTS];
   AiUInt32 aul_ExpertMode[FDX_MAX_BOARD_PORTS];
   AiUInt32 ul_RxVeriMode;
   AiUInt32 aul_RxVeriData[8];
   AiUInt32 aul_RxVeriMask[8];
} TY_FDX_BOARD_CTRL_IN;
```

AiUInt32 aul_PortConfig

The physical ports of the board are configurable in two different ways. This array over the maximum count of ports per board sets or shows the configuration of those ports:

1. The port works as a single port. This means that the accessible port is represented by one physical port. In this mode it is possible to do traffic policing for this port.
2. The port works as a redundant port. This means that the accessible port is represented by two physical ports which are redundant. In this mode, the AFDX Redundancy Management Algorithm (RMA) is active and only the RMA passed frame is transmitted to the application. For login to that port only the first resource ID of the two physical ports can be used. A login to the second resource will cause an error. If the first of two ports is set to redundant mode, the second port will also be set to redundant mode.
3. The ports work as inline Monitor. This port configuration is used for data monitor tap mode. (Rxa -> Txb, Rxb -> Txa).

For the redundant port mode it is restricted that both physical ports are managed by one BIU.

Note:	This configuration can only by changed if the ports are not used by any other client!

Value	Description
FDX_SINGLE	Single Mode
FDX_REDUNDANT	Redundant Mode
FDX_TAP	Tap Mode

AiUInt32 aul_PortSpeed

To configure the port wire speed

Value	Description
FDX_10MBIT	Network bit rate 10Mbit
FDX_100MBIT	Network bit rate 100Mbit (default)
FDX_1000MBIT	Network bit rate 1Gbit (only APX-GNET and APE-FDX)
FDX_AUTO_100MBIT or FDX_AUTO_10MBIT	Port mode is set to Auto Negotiation (port speed) adapted to the port speed of the other side of the connection.

Note:	Two physical boards are managed by one BIU. So port speed can be configured for two ports together! If the mode is set to auto negotiation the speed of the ports can be different!

AiUInt32 aul_ExpertMode

Flags to configure the port. Default value is FDX_EXPERT_MODE (at options disabled.) Otherwise, it can be a combination of the following options:

Value	Description
FDX_DISA_GHOST	Disable Ghost Frames. Frames containing more than four physical errors or no valid SFD (start frame delimiter) are silently discarded while capturing.

AiUInt32 ul_RxVeriMode

Flags to configure the receive error Verification Register.

Value	Description
FDX_BOARD_VERIFICATION_TYPE_DEFAULT	default Verification Register in accordance with the Board type
FDX_BOARD_VERIFICATION_TYPE_AFDX	AFDX specific Verification Register is used
FDX_BOARD_VERIFICATION_TYPE_BOEING	Boeing specific Verification Register is used
FDX_BOARD_VERIFICATION_TYPE_A664	ARINC664 specific Verification Register is used
FDX_BOARD_VERIFICATION_TYPE_DEFINED	Verification Register defined by customer

The AIM AFDX board has the capability to check specified filds inside the MAC and IP header against constant values. By violation of this check a MAC or IP Error will be reported with this frame. The board supports several standard values and is also able to define the first 32 bytes of a frame by mask and compare values with the next parameters.

AiUInt32 aul_RxVeriData

Compare values for receive frame check. This is an array of 8 AiUInt32 values. The frame starts with the LSB of the first AiUInt32 value.

The compare values define the values of the corresponding frame bit locations. It will be compared only if the corresponding bit in the mask value is set..

AiUInt32 aul_RxVeriMask

Mask values for receive frame check. This is an array of 8 AiUInt32 values. The frame starts with the LSB of the first AiUInt32 value.

Setting a bit to logical 1 means the value will be compared against the Data value. Setting a bit to logical 0 means the data value dont cares.

Setting a bit to logical 1 means the value will be compared against the Data value. Setting a bit to logical 0 means the data value don't cares.

This parameter is only valid when verification mode FDX_BOARD_VERIFICATION_TYPE_DEFINED is used.

Note: The figure below demonstrates how the aul_RxVeriData and aul_RxVeriMask parameters are automatically set when ul_RxVeriMode is set to FDX_BOARD_VERIFICATION_TYPE_AFDX. This figure also demonstrates how to set ul_RxVeriMode and aul_RxVeriMask in order to direct the cards to check and verify received frames when ul_RxVeriMode is set to FDX_BOARD_VERIFICATION_TYPE_DEFINED.

Figure 4.2.2.1 Rx Verification Data and Mask

FDX_BOARD_VERIFICATION_TYPE_AFDX						
Byte	aul_RxVeriData	aul_RxVeriMask	Hex or Bin	Field Identification		
0	03	FF	03	Constant	MAC Dest	MAC Header
1	00	FF	00			
2	00	FF	00			
3	00	FF	00			
4	00	00	XX	VL		
5	00	00	XX			
6	02	FF	02	Const	MAC Source	
7	00	FF	00			
8	00	FF	00			
9	00	F0	X0	Network ID		
10	00	00	XX	Equipment ID		
11	00	1F	XXX00000b	Interface ID		
12	08	FF	08	Type		
13	00	FF	00			
14	45	FF	45	Version / IHL		IP Header
15	00	FF	00	Type of Service		
16	00	00	XX	Total Length		
17	00	00	XX			
18	00	00	XX	Fragment ID		
19	00	00	XX			
20	00	80	0XXXXXXXb	Control Flag / Frag Offset		
21	00	00	XX			
22	01	FF	01	Time to live		
23	00	00	XX	Protocol		
24	00	00	XX	Header Checksum		
25	00	00	XX			
26	0A	FF	0A	Constant	IP Source	
27	00	F0	0X	Network ID		
28	00	00	XX	Equipment ID		
29	00	E0	000XXXXXb	Partition ID		
30	00	00	XX		IP Dest	
31	00	00	XX			

Output:

TY_FDX_BOARD_CTRL_OUT* px_BoardControlOUT

```
typedef struct {
   AiUInt32 aul_PortConfig[FDX_MAX_BOARD_PORTS];
   AiUInt32 aul_PortSpeed[FDX_MAX_BOARD_PORTS];
   AiUInt32 aul_PortUsed[FDX_MAX_BOARD_PORTS];
   AiUInt32 aul_PortGoodLink[FDX_MAX_BOARD_PORTS];
   AiUInt32 aul_ExpertMode[FDX_MAX_BOARD_PORTS];
   AiUInt32 ul_GlobalMemFree;
   AiUInt32 ul_SharedMemFree;
   AiUInt32 ul_RxVeriMode;
} TY_FDX_BOARD_CTRL_OUT;
```

AiUInt32 aul_PortConfig

Reflects the current port configuration

AiUInt32 aul_PortSpeed

Reflects the current port wire speed

Value	Description
FDX_100MBIT	Port bit rate 100Mbit (default)
FDX_10MBIT	Port bit rate 10Mbit
FDX_1000MBIT	Network bit rate 1Gbit (only APX-GNET and not applicable for other Boards)
FDX_AUTO_100MBIT	Port mode is set to Auto Negotiation and bit rate 100Mbit is detected
FDX_AUTO_10MBIT	Port mode is set to Auto Negotiation and bit rate 10Mbit is detected
FDX_AUTO_ERROR	No good link detected

AiUInt32 aul_PortUsed

Each port can be used by different clients. This array over the maximum count of ports per board shows how many clients are using the ports at this time. This information only includes if the port is used or not. For detailed information the function FdxQueryResource(..) can be used

Value	Description
0	Port is not used by any client
>0	Number of clients that are using this port

AiUInt32 aul_GoodLink

Connection status of port.

Value	Description
FDX_NO_LINK	No good link detected
FDX_GOOD_LNK	Good link detected

AiUInt32 aul_ExpertMode

Flags to configure the port. Default value is FDX_EXPERT_MODE (at options disabled.) Otherwise, it can be a combination of the following options:

Value	Description
FDX_DISA_GHOST	Disable Ghost Frames. Frames containing more than four physical errors or no valid SFD (start frame delimiter) are silently discarded while capturing.

AiUInt32 ul_GlobalMemFree

Size of Global Memory (in Bytes) which is not already allocated.

AiUInt32 ul_SharedMemFree

Size of Shared Memory (in Bytes) which is not already allocated.

AiUInt32 ul_RxVeriMode

Rx error verification mode which is used by the Board

Return Value

Returns FDX_OK on success or a negative error code on error.
Error Codes: FDX_ERR

4.2.3 FdxCmdIrigTimeControl

Prototype:

```
AiReturn FdxCmdIrigTimeControl (  AiUInt32 ul_Handle,
                                  AiUInt32 ul_Control,
                                  TY_FDX_IRIG_TIME *px_IrigTime,
                                  AiUInt32 *pul_Mode);
```

Purpose:

This function is used to read or set the on-board IRIG timecode encoder and also to select the IRIG source. There is a board internal IRIG time source, but it is also possible to synchronize the time coding to an external IRIG source.

Input

AiUInt32 ul_Control

Control information for this command:

Value	*Description*	*Comment*
FDX_IRIG_READ	Read the IRIG Time encoder.	Applicable every time.
FDX_IRIG_WRITE	Write the IRIG Time encoder.	This is not directly applicable if the external IRIG time is used. Exceptional when the IRIG Time encoder is looped external to the IRIG Time input and additionally if the output is used to synchronize other boards.
FDX_IRIG_EXTERN [1)]	Use external IRIG Time	Switch to using the IRIG time from an external IRIG source, to synchronize several boards to a common time.
FDX_IRIG_INTERN [1)]	Use internal IRIG Time	Switch to the internal IRIG time. Using the internal IRIG time is default

1) For API-FDX-2 Baords and AMC-FDX-2 boards intern extern is not switchable manually. The board detects a external IRIG signal and switche automatically to the external signal

If the IRIG output of a board is used to synchronize several boards to the same IRIG Time, this signal shall be routed also to the own external IRIG input. This is necessary to eliminate the time shift between IRIG encoder and decoder.

If IRIG Time is set by control FDX_IRIG_WRITE it can take up to 3 seconds until the IRIG decoder is also synchronized to the new time set to the encoder.

*TY_FDX_IRIG_TIME *px_IrigTime*

Pointer to the IRIG Timecode structure. This structure is only used to read and write the IRIG timecode. For other controls, this structure is not used and this pointer can be NULL.

```
typedef struct {
/* AiUInt32  ul_Sign;       sign (0=positiv, !0=negativ only needed for
   calculation)*/
   AiInt32   l_Sign;       /* sign (0=absolute 1=positiv, -1=negativ only
                              needed for calculation)*/
   AiUInt32  ul_Hour;      /* 0..23*/
   AiUInt32  ul_Min;       /* 0..59*/
   AiUInt32  ul_Second;    /* 0..59*/
   AiUInt32  ul_Day;       /* 1..366*/
   AiUInt32  ul_MilliSec;  /* 0..999*/
   AiUInt32  ul_MicroSec;  /* 0..999*/
   AiUInt32  ul_NanoSec;   /* 0..900 in steps of 100 */
   AiUInt32  ul_Info;
} TY_FDX_IRIG_TIME;
```

Note: ***The parameters l_Sign , ul_MilliSec, ul_MicroSec, ul_NanoSec and ul_Info is not relevant for IRIG Time set.***
Parameter ul_Info is for future use. Not used at the moment.

Output

*TY_FDX_IRIG_TIME *px_IrigTime*

See input parameter.

*AiUInt32 *pul_Mode*

Returns the mode of IRIG Time

Value	*Description*
FDX_IRIG_EXTERN	Runs with external IRIG Time
FDX_IRIG_INTERN	Runs with internal IRIG Time
FDX_IRIG_EXTERN_NOT_SYNC [1)]	Runs with external IRIG Time but got no synchronisation
FDX_IRIG_INTERN_NOT_SYNC [1)]	Runs with internal IRIG Time but got no synchronisation

1) These modes can only be seen with APX-GNET.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.2.4 FdxCmdStrobeTriggerLine

Prototype:

AiReturn FdxCmdStrobeTriggerLine (AiUInt32 ul_Handle,
AiUInt32 ul_TriggerLine);

Purpose:

This function provides a Trigger output strobe on a selectable Trigger Output Line on system command.

NOTE!! This function uses a PORT handle as input.

Input

AiUInt32 ul_Handle

A port resrource handle.

AiUInt32 ul_TriggerLine

Values from 0 to 3 are possible for this parameter to select the corresponding Trigger Output lines 1 to 4.

Output

None.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.2.5 FdxReadBSPVersion

Prototype:

AiReturn FdxReadBSPVersion(AiUInt32 ul_Handle,
TY_FDX_BSP_VERSION *px_BspVersion);

Purpose:

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

Input

None

Output

The following structure describes the Version Type information including major version number, a minor version number, a build number, a special major version number and a special minor version number.

```
typedef struct {
   AiUInt32 ul_MajorVer;
   AiUInt32 ul_MinorVer;
   AiUInt32 ul_BuildNr;
   AiUInt32 ul_MajorSpecialVer;
   AiUInt32 ul_MinorSpecialVer;
} TY_VER_NO;
```

*TY_FDX_BSP_VERSION *px_BspVersion*

Pointer to a structure, which contains the full available version information.

```
typedef struct {
   TY_VER_NO x_SysDrvVer;
   TY_VER_NO x_DllVer;
   TY_VER_NO x_TargetVer;
   TY_VER_NO x_FirmwareVerBiu1;
   TY_VER_NO x_FirmwareVerBiu2;
   TY_VER_NO x_MainLcaVer;
   TY_VER_NO x_LcaVerBiu1;
   TY_VER_NO x_LcaVerBiu2;
   TY_VER_NO x_PciLcaVer;
   TY_VER_NO x_TcpVer;
   TY_VER_NO x_Bootstrap;
   TY_VER_NO x_Monitor;
   TY_VER_NO x_TargetOS;
   AiUInt32  ul_BspCompatibility;
} TY_FDX_BSP_VERSION;
```

TY_VER_NO x_SysDrvVer

Version Information of the Board System Level Driver

TY_VER_NO x_DllVer

Version Information of the Application Interface Library

TY_VER_NO x_TargetVer

Version Information of the onboard Application Software Processor.

TY_VER_NO x_FirmwareVerBiu1

Firmware Version Number of the BIU1

TY_VER_NO x_FirmwareVerBiu2

Firmware Version Number of the BIU2

TY_VER_NO x_MainLcaVer

LCA Revision Information of the main board

TY_VER_NO x_LcaVerBiu1

LCA Revision Information of the BIU1

TY_VER_NO x_LcaVerBiu2

LCA Revision Information of the BIU2

TY_VER_NO x_PciLcaVer

LCA Revision Information of the PCI LCA (AMC only)

TY_VER_NO x_TcpVer

TCP Revision Information

TY_VER_NO x_Bootstrap;

Bootsrap Revision Information
(Only relevant for APX-GNET)

TY_VER_NO x_Monitor;

Onboard Monitor Software Revision Information
(Only relevant for APX-GNET)

TY_VER_NO x_TargetOS;

Onboard Target Operating System Revision Information
(At the moment only relevant for APX-GNET)

AiUInt32 ul_BspCompatibility

Compatibility Status of the current BSP (AIM Board Software Package) components relating to the Library

Value	*Description*
FDX_BSP_COMPATIBLE	BSP components are compatible.
FDX_BSP_WRONG_SYS_W98_W2000_WXP_SW	The system driver for Windows 98 / 2000 / XP is not compatible with the DLL.
FDX_BSP_WRONG_SYS_WNT_SW	The system driver for Windows NT is not compatible with the DLL.
FDX_BSP_WRONG_SYS_DRV	The system driver for Linux is not compatible with the DLL.
FDX_BSP_WRONG_TARGET_SW	The target SW is not compatible with the DLL.
FDX_BSP_WRONG_BIU1_SW	The Firmware for Biu1 is not compatible with the DLL.
FDX_BSP_WRONG_BIU2_SW	The Firmware for Biu2 is not compatible with the DLL.
FDX_BSP_WRONG_LCA1_SW	The LCA-SW for Biu1 is not compatible with the DLL.
FDX_BSP_WRONG_LCA2_SW	The LCA-SW for Biu1 is not compatible with the DLL.
FDX_BSP_NOT_COMPATIBLE	BSP components are not compatible due to other errors.

Return Value

Returns FDX_OK on success or a negative error code on error.
Error Codes: FDX_ERR

4.2.6 FdxVersionGetAll

Prototype:

AiReturn FdxVersionGetAll(AiUInt32 ul_Handle,
AiUInt32 ul_Count,
TY_VER_INFO ax_Versions[],
TY_VERSION_OUT* px_VersionInfoOut);

Purpose:

This function fills the ax_Versions array with all versions available on the board. The returned amount of Versions is returned in the px_VersionInfoOut structure.

```
TY_VERSION_OUT xVerOut = {0};
TY_VER_INFO xVers[AI_MAX_VERSIONS] = { {0} };
FdxVersionGetAll(Handle, AI_MAX_VERSIONS,xVers,&xVerOut);

for(AiUInt32 i=0;i<xVerOut.ul_Count;i++)
{
    printf("%-20s : %s", xVers[i].ac_Description, xVers[i].ac_FullVersion);
}
```

Input

AiUInt32 ul_Handle

A protocol specific handle to the device

AiUInt32 ul_Count

Max number of versions to read. Defines the size of the number of elements of the ax_Versions array passed to the function.

Output

TY_VER_INFO ax_Versions[]

Array of TY_VER_INFO elements where information about all board component versions is stored. The size of the array must be greater or equal to ul_Count elements.

```
typedef struct ty_ver_info{
    AiUInt32 ul_VersionType;
    AiChar   ac_Description[AI_DESCRIPTION_STRINGL];
    AiUInt32 ul_MajorVer;
    AiUInt32 ul_MinorVer;
    AiUInt32 ul_PatchVersion;
    AiUInt32 ul_BuildNr;
    AiChar   ac_FullVersion[AI_VERSION_STRINGL];
} TY_VER_INFO;
```

TY_E_VERSION_ID e_VersionType

The version type indicates the type of component the version is referred to. The enum type lists all components that can be versioned.

```
typedef enum{
   AI_SYS_DRV_VER          = 0,
   AI_ANS_VER              = 1,
   AI_DLL_VER              = 2,
   AI_REMOTE_DLL_VER       = 3,
   AI_TARGET_VER           = 4,
```

```
   AI_FIRMWARE_VER_BIU1    = 5,
   AI_MAIN_LCA_VER         = 6, /*!< MLCA/NOVRAM (PL+Microblaze SW)*/
   AI_IO_LCA_VER_BIU1      = 7,
   AI_PCI_LCA_VER          = 8,
   AI_TCP_VER              = 9,
   AI_BOOTSTRAP_VER        = 10,
   AI_MONITOR_VER          = 11,
   AI_TARGET_OS_VER        = 12,
   AI_FPGA_VER             = 13, /*!< AyE=FPGA overall, AyS=PL */
   AI_FIRMWARE_VER_BIU2    = 14,
   AI_FIRMWARE_VER_BIU3    = 15,
   AI_FIRMWARE_VER_BIU4    = 16,
   AI_IO_LCA_VER_BIU2      = 17,
   AI_IO_LCA_VER_BIU3      = 18,
   AI_IO_LCA_VER_BIU4      = 19,
   AI_SUBDLL1_VER          = 20,
   AI_SUBDLL2_VER          = 21,
   AI_SUBDLL3_VER          = 22,
   __AI_MAX_VERSIONS       /*! Indicates last entry.
} TY_E_VERSION_ID;

#define AI_MAX_VERSIONS __AI_MAX_VERSIONS
```

Values	*Description*
AI_SYS_DRV_VER	Version Information of the Board System Level Driver
AI_ANS_VER	Version Information of the AIM Network Server
AI_DLL_VER	Version Information of the Application Interface Library
AI_REMOTE_DLL_VER	Version Information of the Application Interface Library (Remote PC)
AI_TARGET_VER	Version Information of the onboard Application Software Processor
AI_FIRMWARE_VER_BIUx	Firmware Version Number of the BIU1
AI_MAIN_LCA_VER	LCA Revision Information of the main board
AI_FPGA_VERSION	
AI_IO_LCA_VER_BIUx	LCA Revision Information of the BIU1
AI_PCI_LCA_VER	PCI LCA Revision Information
AI_TCP_VER	TCP Revision Information
AI_BOOTSTRAP_VER	Bootsrap Revision Information
AI_MONITOR_VER	Onboard Monitor Software Revision Information
AI_TARGET_OS_VER	Onboard Target Operating System Revision Information
AI_FPGA_VER	
AISUBDLLx_VER	

AiChar ac_Description[AI_DESCRIPTION_STRINGL]

zero-terminated ASCII string that describes the component the version is for. This string has limited size which is described by AI_DESCRIPTION_STRINGL.

AiUInt32 ul_MajorVer

Major version info as a decimal number.

AiUInt32 ul_MinorVer

Minor version info as a decimal number.

AiUInt32 ul_PatchVersion

Patch version info as a decimal number.

AiUInt32 ul_BuildNr

Buildnumber as a decimal number.

AiChar ac_FullVersion [AI_VERSION_STRINGL]

zero-terminated ASCII version string. Holds main version in dot-separated major.minor.patch.build format. Version extension will be hyphen separated from main version. This string has limited size which is described by AI_VERSION_STRINGL.

*TY_VERSION_OUT *px_VersionOut*

```
typedef struct ty_version_out{
   AiUInt32 ul_Count;
   AiUInt32 ul_MaxCount;
} TY_VERSION_OUT;
```

AiUInt32 ul_Count

Shows the number of returned version structures in array ax_Versions[]. The information in ax_Versions above ul_Count is invalid.

AiUInt32 ul_MaxCount

Shows the number of maximum available component versions for the device. If this value is higher than the number of returned elements ul_Count it is possible to read again with a larger array of ax_Versions[] elements.

Return Value

Returns FDX_OK on success or a error code on error.

4.2.7 FdxVersionGet

Prototype:

AiReturn FdxVersionGet(***AiUInt32 ul_Handle,***
TY_E_VERSION_ID eId,
TY_VER_INFO *pxVersion);

Purpose:

This function returns an structured element containing the version information of the requested version id. If the version id does not exist an appropriate error value is returned.

```
TY_VER_INFO xDLLVer = { 0 };
FdxVersionGet(Handle, AI_DLL_VER, &xDLLVer);
```

Input

AiUInt32 ul_Handle

A protocol specific handel to the device

TY_E_VERSION_ID e_Id

A version type identifier for the type of component the version shall be read for. The version type is identified by an enum value. For detailed description please have a look to function FdxVersionGetAll.

Output

*TY_VER_INFO *pxVersion*

Pointer to a memory that can contain structure version type This memory must be allocated / deallocated by the application. For detailed description please have a look to function FdxVersionGetAll.

Return Value

Returns FDX_OK on success. An error code on error or if the version id is not found.

4.3 Transmitter Functions

This section describes the transmit functionality of the FDX-2/4 Board. The following transmit Modes and sub modes are available:

Mode	*Sub Mode*	*Description*
Generic		The data to send in the generic transmit mode is described as a queue of frames to send continuously. This queue is organized as a part of memory in the BIU associated memory. Functions are provided to setup, observe and write this queue. Error injection in a full blown manner is provided in this mode. The Transmit queue is provided in a cyclic manner. This means, after transmission of the last frame in that queue, the transmitter starts again at the beginning of the queue. At start time, the queue must be set up completely. The timing is organized relative between two frames by a transfer wait time, provided for each frame. ***Note for APE/ACE/AXC/AMCX-FDX boards: The size of the queue is limited to 2048 frames with no more than 255 frames of the same Virtual Link number.***
Replay		The Transmit queue is provided as a reloadable queue. This means the queue is a FIFO queue. The frames in the queue are transmitted out in the order they were put into the queue. The user is able reload data, while transmission is running. The timing is secured by the IRIG time tag provided for each frame. Use this mode for replaying previously captured AFDX traffic via the chronological Receiver Operation.
Individual		The data to send is described in an application oriented UDP port based manner. To send data in this mode, first the Virtual Link must be defined for sending. Based on this Virtual Link, a UDP port can be specified. This UDP port can provide one of the following sub modes according to the definition in ARINC 653.
	Sampling:	A sampling port is mainly characterized by a sampling rate and the fixed frame length. The sampling rate describes the equidistant appearance of this frame on the physical bus. Each time data is written to this port, the data for transmission will be updated.
	Queuing:	The queuing service describes a port, where an application is able to send data packages in an asynchronous way. The message size can be different for each data package up to a defined maximum. The data package will be sent one time without any repetition.

The following sections of function descriptions are reflecting these modes.

The Handle input parameter to the following functions must be a port related handle.

Table 4.3-I: Transmitter Functions

Function	*Description*
Global Transmitter Functions	
FdxCmdTxPortInit	Initializes the transmitter
FdxCmdTxModeControl	Defines the Mode of the transmitter
FdxCmdTxControl	Starts and stops the transmitter
FdxCmdTxStatus	Obtains status information about the transmitter
FdxCmdTxTrgLineCtrl	Controls Transmitter Associated Trigger Lines
FdxCmdTxStaticRegsControl	Controls Static Transmit Registers
FdxCmdTxVLControl	Controls VL (Enable / Disable)
Generic or Replay Transmitter Functions	
FdxCmdTxQueueCreate	Creates an AFDX Frame Queue
FdxCmdTxQueueStatus	Retrieves Status of an AFDX Frame Queue
FdxCmdTxQueueWrite	Writes AFDX Frames to the Queue
FdxCmdTxQueueUpdate	Updates AFDX Frames of a generic Queue on the fly
UDP Port-Oriented Transmitter Functions	
FdxCmdTxCreateVL	Creates a Virtual Link, which can be used for transmission.
FdxCmdTxCreateHiResVL	Creates a Virtual Link, which can be used for transmission with a high resolution bag.
FdxCmdTxUDPCreatePort	Creates a fully described AFDX Comm port for transmission.
FdxCmdTxUDPChgSrcPort	Change source of an UDP port.
FdxCmdTxSAPCreatePort	Create a fully described SAP port for transmission.
FdxCmdTxUDPDestroyPort	Destroys a configured UDP port.
FdxCmdTxUDPWrite	Writes data to a transmission UDP port
FdxCmdTxUDPBlockWrite	Writes data to several transmission UDP ports
FdxCmdTxSAPWrite	Writes data to a transmission SAP port
FdxCmdTxSAPBlockWrite	Writes data to several transmission SAP ports
FdxCmdTxUDPGetStatus	Retrieves the status of a transmission UDP port
FdxCmdTxUDPControl	Controls UDP Port operation (Enable / Disable and error injection)
FdxCmdTxVLWrite	Writes Frames to the VL-Buffer
FdxCmdTxVLWriteEx	Writes Frames to the VL-Buffer with extended frame control possibilities
Transmitter Data Buffer Functions	
FdxCmdTxBufferQueueAlloc	Allocate Transmit Data Buffer Queue
FdxCmdTxBufferQueueFree	Free Transmit Buffer Queue
FdxCmdTxBufferQueueRead	Reads Data from Transmit Buffer Queue
FdxCmdTxBufferQueueWrite	Write Data from Transmit Buffer Queue
FdxCmdTxBufferQueueCtrl	Controls a Transmit Buffer Queue
Generic Transmitter Sub-Queue Functions	
FdxCmdTxSubQueueCreate	Allocate Transmit Data Buffer Queue
FdxCmdTxSubQueueDelete	Free Transmit Buffer Queue
FdxCmdTxSubQueueWrite	Reads Data from Transmit Buffer Queue

4.3.1 Global Transmitter Functions

4.3.1.1 FdxCmdTxControl

Prototype:

AiReturn FdxCmdTxControl(AiUInt32 ul_Handle,
const TY_FDX_TX_CTRL *px_TxControl);

Purpose:

This function is used to control the transmit operation of the board.

Input:

TY_FDX_TX_MODE_CTRL *px_TxControl

Pointer to a control structure to setup the transmit port start mode and the counter for cyclic transmission.

```
typedef struct {
   TY_FDX_E_TX_START_MODE e_StartMode;
   AiUInt32 ul_Count;
   TY_FDX_IRIG_TIME x_StartTime;
   TY_FDX_E_TX_EXTENDED_STOP_MODE e_ExtendedStopMode;
   AiUInt32 ul_TransmitTime;     /* Time duration in ms */
} TY_FDX_TX_CTRL;
```

TY_FDX_E_TX_START_MODE e_StartMode

Control Parameter for the Transmitter Mode

Constant	*Description*
FDX_STOP	Stop the Transmitter
FDX_START	Start the Transmitter immediately
FDX_START_TRG	Start the Transmitter on external Trigger
FDX_START_TIME	Start on specified Start Time

AiUInt32 ul_Count

This parameter defines the number of times, the AFDX Frame Queue is repeated in Generic Transmission mode. A value of 0 means cyclic transmission.

In Replay Transmission Mode, this parameter is not used.

In Individual Transmission Mode, this parameter is not used.

TY_FDX_IRIG_TIME x_StartTime

Absolute IRIG time to start the transmitter. This structure is only applicable with the Start Mode FDX_START_TIME. The specified start time must be located in future.

TY_FDX_E_TX_EXTENDED_STOP_MODE e_ExtendedStopMode

Control Parameter for the extended Stop Mode. This can be used to stop transmission by a external Trigger Condition or a Time Condition. The Parameter FDX_ESTOP_EXT_TRG_RESTART shall only be used in combination of e_Startmode = FDX_START_TRG

Constant	*Description*
FDX_ESTOP_NOT_USED	Ext. Stop Mode not used
FDX_ESTOP_EXT_TRG	Stop Transmission on external Trigger strobe
FDX_ESTOP_EXT_TRG_RESTART	Stop Transmission on external trigger strobe and allow to continue with the next external trigger strobe. This Parameter shall only be used in combination of e_Startmode = FDX_START_TRG
FDX_ESTOP_TIME	Stop Transmission after a defined time span. The timer starts count down with real starting transmission.

AiUInt32 ul_TransmitTime

Transmission Time Duration for the extended Stop Mode FDX_ESTOP_TIME. The time duration must be defined in ms.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

4.3.1.2 FdxCmdTxModeControl

Prototype:

AiReturn FdxCmdTxModeControl(AiUInt32 ul_Handle,
const TY_FDX_TX_MODE_CTRL *px_TxModeControl);

Purpose:

This function is used to configure the operational mode of the transmit port.

Input:

TY_FDX_TX_MODE_CTRL *px_TxModeControl

Pointer to a control structure to setup the transmit port and also get information about that port.

```
typedef struct {
   AiUInt32 ul_TransmitMode;
   AiUInt32 ul_RerosPortDelay;
} TY_FDX_TX_MODE_CTRL;
```

AiUInt32 ul_TransmitMode

There are four modes of operation for a Transmitter port

Value	*Description*
FDX_TX_GENERIC	Generic Transmit Mode A List of AFDX Frames with additional Header Information (Gap Info etc..) can be sent from a specified queue cyclically and a specific number of times
FDX_TX_REPLAY	Replay Transmit Mode This is used for replaying previously recorded AFDX traffic back to the wire
FDX_TX_INDIVIDUAL	Individual Transmit Mode (Simulation Transmit Mode) Several UDP ports can be defined using Virtual Links, which are configured in a separate command. Each UDP port is defined by it's own parameters. The VL associated traffic shaping is supported
FDX_TX_FIFO	Fifo Transmit Mode One AFDX Frame or a List of AFDX Frames with additional Header Information (IFG Cnt Info etc..) can be sent from a specified queue

Note:	***Replay is not available if the port has been configured for redundant operation***

AiUInt32 ul_RerosPortDelay

Reserved

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.1.3 FdxCmdTxPortInit

Prototype:

AiReturn FdxCmdTxPortInit (AiUInt32 ul_Handle,
const TY_FDX_PORT_INIT_IN *px_PortInitIn,
TY_FDX_PORT_PORT_OUT *px_PortInitOut);

Purpose:

This function is to initialize a port.

For Transmit functionality, the initialized state is as follows:

- No Transmit Queues defined
- No VL created, No UPD Ports created
- FdxCmdTxControl command has no effect

Input:

TY_FDX_PORT_INIT_IN* px_PortInitIn

Pointer to a board control input structure.

```
typedef struct {
   AiUInt32 ul_PortMap;
} TY_FDX_PORT_INIT_IN;
```

AiUInt32 ul_PortMap

This is a user definable byte for identification.

Note: This function can only be done by a privileged user!

Output:

TY_FDX_PORT_INIT_OUT* px_PortInitOut

```
typedef struct {
   AiUInt32 ul_PortConfig;
   AiUInt32 ul_PortUsed;
   AiUInt32 ul_GlobalMemFree;
   AiUInt32 ul_SharedMemFree;
} TY_FDX_PORT_INIT_OUT;
```

AiUInt32 ul_PortConfig

reflects the current port configuration

Value	*Description*
FDX_SINGLE	Single Mode
FDX_REDUNDANT	Redundant Mode

AiUInt32 aul_PortUsed

Each port can be used by different clients. This array over the maximum count of ports per board shows how many clients are using the ports at this time. This information only includes if the port is used or not. For detailed information the function FdxQueryResource(..) can be used

Value	*Description*
0	Port is not used by any client
>0	Number of clients are using this port

AiUInt32 ul_GlobalMemFree

Size of Global Memory (in Bytes) which is not already allocated.

AiUInt32 ul_SharedMemFree

Size of Shared Memory (in Bytes) which is not already allocated.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.1.4 FdxCmdTxStaticRegsControl

Prototype:

AiReturn FdxCmdTxStaticRegsControl (AiUInt32 ul_Handle,
const TY_FDX_TX_STATIC_REGS
px_TxStaticRegs);

Purpose:

This function is used to setup the Static Transmit Registers of a port. This is especially used in **Generic** Transmit Mode, together with corresponding **Payload Generation Modes** (see ***FdxCmdTxQueueWrite*** command). For other Transmit Modes (Replay and Individual) this command has no effect.

Input:

TY_FDX_TX_STATIC_REGS *px_TxStaticRegs

Pointer to a setup structure for Static Transmit Registers

```
typedef struct {
   TY_FDX_TX_STATIC_REGS_MAC x_TxStaticRegsMAC;
   TY_FDX_TX_STATIC_REGS_IP x_TxStaticRegsIP;
   TY_FDX_TX_STATIC_REGS_UDP x_TxStaticRegsUDP;
} TY_FDX_TX_STATIC_REGS;
```

TY_FDX_TX_STATIC_REGS_MAC x_TxStaticRegsMAC

Sub-Structure for MAC Static values

```
typedef struct {
   AiUInt8 uc_MACDest2;       // MAC Destination Byte 2
   AiUInt8 uc_MACDest3;       // MAC Destination Byte 3
   AiUInt8 uc_MACDest4;       // MAC Destination Byte 4
   AiUInt8 uc_MACDest5;       // MAC Destination Byte 5
   AiUInt8 uc_MACSrc0;        // MAC Source Byte 0
   AiUInt8 uc_MACSrc3;        // MAC Source Byte 3
   AiUInt8 uc_MACSrc4;        // MAC Source Byte 4
   AiUInt8 uc_MACSrc5;        // MAC Source Byte 5
   AiUInt16 uw_MACLengthType; // MAC Length/Type
} TY_FDX_TX_STATIC_REGS_MAC;
```

TY_FDX_TX_STATIC_REGS_IP x_TxStaticRegsIP

Sub-Structure for IP Static values

```
typedef struct {
   AiUInt8  uc_IPTypeSrv;     // IP Type of Service
   AiUInt8  uc_IPVersion;     // IP Version Field (Bits 0..3)
   AiUInt8  uc_IPIHL;         // IP IHL (Bits 0..3)
   AiUInt8  uc_IPProtocol;    // IP Protocol = UDP = 0x11
   AiUInt8  uc_IPTTLive;      // IP Time To Live
   AiUInt8  uc_IPCtrl;        // IP Control
   AiUInt16 uw_IPFrag;        // IP Fragment ID
   AiUInt16 uw_IPTotalLength; // IP total Length
   AiUInt16 uw_IPFragOffs;    // IP Fragment. Offset
   AiUInt16 uw_IPHeaderChkSum;   // IP Header Checksum
   AiUInt32 ul_IPDest;        // IP Destination Address
   AiUInt32 ul_IPSrc;         // IP Source Address
} TY_FDX_TX_STATIC_REGS_IP;
```

TY_FDX_TX_STATIC_REGS_UDP x_TxStaticRegsUDP

Sub-Structure for UDP Static values

```
typedef struct {
   AiUInt16 uw_UDPDest;       // UDP Destination Port
   AiUInt16 uw_UDPSrc;        // UDP Source Port
   AiUInt16 uw_UDPLength ;    // UDP Length
   AiUInt8  uc_UDPPayload[22] // 22 Bytes of UDP Payload
} TY_FDX_TX_STATIC_REGS_UDP;
```

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.1.5 FdxCmdTxStatus

Prototype:

AiReturn FdxCmdTxStatus (AiUInt32 ul_Handle,
TY_FDX_TX_STATUS *px_TxStatus);

Purpose:

This function is used to retrieve the transmitter status of the board

Input:

None

Output:

TY_FDX_TX_STATUS *px_TxStatus

Pointer to a transmit port status information structure

```
typedef struct {
   TY_FDX_E_TX_STATUS e_Status;
   AiUInt32 ul_Frames;
   AiUInt32 ul_TransmitMode;
   AiUInt32 ul_FramesPortB;
} TY_FDX_TX_STATUS;
```

TY_FDX_E_TX_STATUS e_Status

Status information of the Transmitter

Value	*Description*
FDX_STAT_STOP	Transmitter Stopped
FDX_STAT_RUN	Transmitter Running
FDX_STAT_ERROR	Transmitter Error
FDX_STAT_WAIT_F_TRIG	Transmitter Waiting for Trigger

AiUInt32 ul_Frames

Total Number of transmitted AFDX Frames. In case of a redundant Port, this counter shows the Frames transmitted on Net A.

AiUInt32 ul_FramesPortB

This counter is only applicable in case of a redundant Port. In that case this counter shows the number of transmitted Frames on Net B.

AiUInt32 ul_TransmitMode

The mode in which the transmitter is set up. (Refer also to function ***FdxCmdTxModeControl*** parameter ul_TransmitMode.*)*

Return Value

Returns FDX_OK on success or a negative error code on error.
Error Codes: FDX_ERR

4.3.1.6 FdxCmdTxTrgLineControl

Prototype:

AiReturn FdxCmdTxTrgLineControl (AiUInt32 ul_Handle,
const TY_FDX_TRG_LINE_CTRL *px_TrgLineCtrl);

Purpose:

This function is used to select the Trigger In- and Output lines for the Transmitter part of the associated port. Trigger lines can be used for starting a transmission, or for output of trigger strobes on special frames (especially in the Generic Transmission Mode).

Input

TY_FDX_TRG_LINE_CTRL *px_TrgLineCtrl

This structure defines the Trigger In- and Output line routing

```
typedef struct {
   AiUInt32 ul_TrgInLine;
   AiUInt32 ul_TrgOutLine;
} TY_FDX_TRG_LINE_CTRL;
```

AiUInt32 ul_TrgInLine

Transmitter Trigger Input Line

AiUInt32 ul_TrgOutLine

Transmitter Trigger Output Line

Values for Trigger Lines:

Value	*Description*
FDX_STROBE_LINE_OFF	Trigger Off
FDX_STROBE_LINE_1	Trigger Line 1
FDX_STROBE_LINE_2	Trigger Line 2
FDX_STROBE_LINE_3	Trigger Line 3
FDX_STROBE_LINE_4	Trigger Line 4
FDX_STROBE_LINE_KEEP	Keep current setting

Return Value

Returns FDX_OK on success or a negative error code on error.
Error Codes: FDX_ERR

4.3.1.7 FdxCmdTxVLControl

Prototype:

AiReturn FdxCmdTxVLControl (AiUInt32 ul_Handle,
const TY_FDX_TX_VL_CONTROL *px_TxVLControl);

Purpose:

This function is used to enable or disable a VL. By default all VL's are enabled. For Individual Transmit Mode each VL must be explicitly created with the function **FdxCmdTxCreateVL** or **FdxCmdTxCreateHiResVL**.

Input:

TY_FDX_TX_VL_CONTROL *px_TxVLControl

Pointer to a setup structure for a Virtual Link

```
typedef struct {
   AiUInt32 ul_VlId;
   AiUInt32 ul_EnableTyp;
} TY_FDX_TX_VL_CONTROL;
```

AiUInt32 ul_VlId

Virtual Link Identifier. A value in a range from 0 to 65535.

AiUInt32 ul_EnableTyp

Value	*Comment*
FDX_ENA	Virtual Link is enabled. All frames defined for VL are transmitted
FDX_DIS	Virtual Link is disabled. All frames for the given VLs are discarded.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.2 Generic and Replay Transmitter Functions

4.3.2.1 FdxCmdTxQueueCreate

Prototype:

```
AiReturn FdxCmdTxQueueCreate ( AiUInt32 ul_Handle,
                               const TY_FDX_TX_QUEUE_SETUP
                               *px_TxQueueCreate,
                               TY_FDX_TX_QUEUE_INFO *px_TxQueueInfo);
```

Purpose:

This function is used to create a queue of AFDX Frames. This Queue is the input for the Generic or Replay Transmit Mode. Only one Transmit Queue can be created for one port.

Input:

TY_FDX_TX_QUEUE_SETUP *px_TxQueueCreate

Pointer to a setup structure for a Transmit queue.

Pointer to a setup structure for a Transmit queue.

```
typedef struct {
   AiUInt32 ul_QueueSize;
} TY_FDX_TX_QUEUE_SETUP;
```

AiUInt32 ul_QueueSize

Defines the size of the Queue in Byte. This means the memory which is allocated in BIU associated memory. If this value is set to zero, a internal default queue size will be selected.

AiUInt32 ul_TimingType

Value	*Description*
FDX_TX_QUEUE_ABSOLUTE	Queue transmission starts at the specified time, provided in *x_StartTime*. The offset to the actual IRIG Time on Board will be calculated internally between *x_StartTime* and the first provided frame to send.

TY_FDX_IRIG_TIME x_StartTime

Transmit start time in absolute timing mode.

Note: ***The parameters*** **uw_TimingType** ***and*** **x_StartTime** ***are only relevant for a reloadable queue, and therefore only in Replay Transmission Mode***

Output:

*TY_FDX_TX_QUEUE_INFO *px_TxQueueInfo*

Pointer to a setup structure for a Transmit Queue information structure

```
typedef struct {
   AiUInt32 ul_QueueSize;
   AiUInt32 ul_QueueBaseAddr;
} TY_FDX_TX_QUEUE_INFO;
```

AiUInt32 ul_QueueSize

This parameter returns the real size of the transmit queue.

AiUInt32 ul_QueueBaseAddr

This parameter returns the Queue Base Address on Target

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.2.2 FdxCmdTxQueueStatus

Prototype:

AiReturn FdxCmdTxQueueStatus(AiUInt32 ul_Handle,
TY_FDX_TX_QUEUE_STATUS *px_TxQueueStatus);

Purpose:

This function is used to retrieve the AFDX Transmit Queue status when in Generic or Replay Transmit Mode.

Input:

None

Output:

TY_FDX_TX_QUEUE_STATUS *px_TxQueueStatus

Pointer to an information structure of the transmitter

```
typedef struct {
   TY_FDX_E_TX_QUE_STATUS e_QueueStatus;
   AiUInt32 ul_BytesReloadable;
   AiUInt32 ul_FramesSent;
   AiUInt32 ul_FramesInCycQueue;
} TY_FDX_TX_QUEUE_STATUS;
```

TY_FDX_E_TX_QUE_STATUS e_TxQueueStatus;

An enumerated value, which describes the state of the generic transmit queue

```
typedef enum fdx_que_status {
   FDX_QUE_EMPTY,
   FDX_QUE_FILLED,
   FDX_QUE_FULL,
   FDX_QUE_SENT,
   FDX_QUE_CYCL_RUN,
   FDX_QUE_CYCL_SENT,
   FDX_QUE_RP_UNDERRUN
} TY_FDX_E_TX_QUE_STATUS;
```

Status:	Description
FDX_QUE_EMPTY*	The transmit queue is just created, but no frame has been entered.
FDX_QUE_FILLED*	The transmit queue is partially filled with frames. The remaining free memory space in the queue is described by the next parameter in this structure
FDX_QUE_FULL*	The transmit queue is filled with frames. There is no more room for data entry.
FDX_QUE_SENT*	All frames ever copied to the transmit queue have been transmitted. No more frames are on any buffers on the hardware.
FDX_QUE_CYCL_RUN	The transmit queue is configured as cyclic, frames are written to the queue and the transmitter is up and running.
FDX_QUE_CYCL_SENT	Reserved

**** These status codes are only applicable if the Replay Transmission Mode***

AiUInt32 ul_BytesReloadable

Remaining space in queue which can be written to the transmit queue.

Note: This value is only valid in Replay Transmission Mode

AiUInt32 ul_FramesSent

Number of frames sent through this queue. For a cyclic queue, the frames of each cycle will be counted.

AiUInt32 ul_FramesInCycQueue

If the queue is configured in cyclic mode, this value shows the number of frames written to the queue.

Note: This value is only valid in Generic Transmission Mode

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.2.3 FdxCmdTxQueueUpdate

AiReturn FdxCmdTxQueueUpdate (AiUInt32 ul_Handle,
const TY_FDX_TX_QUEUE_UPDATE *px_Update,
const void *pv_WriteBuffer);

Purpose:

This function allows update of frames in a generic transmit queue after a frame was already written to the queue. The update is also possible while the transmission is running (on the fly). It is only possible to update AFDX- FRAME data (MAC-Frame) data and not the Fixed Header like described in FdxCmdTxQueueWrite.

Input:

*TY_FDX_TX_QUEUE_UPDATE *px_Update*

A pointer to a structure which describes which frame in the queue and also the data position and length of data which shall be updated within the MAC frame.

```
typedef struct {
   AiUInt32 ul_Index;
   AiUInt32 ul_Offset;
   AiUInt32 ul_Length;
   AiUInt32 ul_SubQueueHandle;
} TY_FDX_TX_QUEUE_UPDATE;
```

AiUInt32 ul_Index

I Index to the frame which shall be updated. This is a counting value starting with 0 over all frames written to the queue with the command FdxCmdTxQueueWrite or FdxCmdTxSubQueueWrite. The first written frame has the index 0. If there are commends inserted to the queue, the commands are also numbered.

If ul_SubQueueHandle is unequal to 0, ul_Index 0 addresses the first transfer of the SubQueuestarting with the byte referenced by ul_Offset.

AiUInt32 ul_Offset

Byte offset within the MAC frame, where update of data shall be started. Offset 0 addresses Byte 0 of the MAC Frame

AiUInt32 ul_Length

Number of Bytes shall be updated starting with the byte referenced by ul_Offset.

AiUInt32 ul_ SubQueueHandle

If ul_SubQueueHandle is unequal to 0 it indicates that the Transfer which shall be controlled is located in a Transmitter SubQueue. A value of 0 indicates the Main TransferQueue Handle

void *pv_WriteBuffer

A pointer to byte buffer, where the new data is located.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.2.4 FdxCmdTxQueueWrite

Prototype:

```
AiReturn FdxCmdTxQueueWrite ( AiUInt32 ul_Handle,
                              AiUInt32 ul_HeaderType,
                              AiUInt32 ul_EntryCount,
                              AiUInt32 ul_WriteBytes,
                              const void *pv_WriteBuffer);
```

Purpose:

This function is used to write Entries to a Transmit Queue from a provided buffer. For this write function the number of Entries and the number of bytes to write needs to be specified. The entries will always be queued at the end of the transmit queue.

All information about VL and port addressing must be defined inside the data buffer inside the frame data. In generic transmit mode there is a limitation for defining only 255 frames with the same VL. If this limitation is exceeded there will be a problem of automatioc Sequence-Numbering of that frames.

Input:

AiUInt32 ul_HeaderType

This parameter defines, the type of the frame header structure.

TransmitMode:	*Value:*	*Description:*
FDX_TX_GENERIC / FDX_TX_INDIVIDUAL	FDX_TX_FRAME_HEADER_GENERIC	Standard Generic Tx Frame, only applicable for Generic Transmit mode. Layout of frame header follows the TY_FDX_TX_FRAME_HEADER structure
FDX_TX_REPLAY	FDX_TX_FRAME_HEADER_REPLAY	Replay Tx Frame, only applicable in Replay Transmit mode. Layout of frame header follows the TY_FDX_FRAME_BUFFER_HEADER Structure, described at the **FdxCmdMonQueueRead** command.
FDX_TX_FIFO	FDX_TX_FRAME_HEADER_REPLAY	FIFO Tx Frame, only applicable in FIFO Transmit mode. Layout of frame header follows the TY_FDX_TX_FIFO_FRAME_HEADER structure

AiUInt32 ul_EntryCount

TransmitMode :	*Value / Description:*
FDX_TX_GENERIC	At the moment only a count of 1 is suported
FDX_TX_REPLAY	Not applicable
FDX_TX_INDIVIDUAL	At the moment only a count of 1 is suported
FDX_TX_FIFO	Number of Entries to write

AiUInt32 ul_WriteBytes

Number of bytes that shall be written to the queue.

*void *pv_WriteBuffer*

Pointer to the data buffer providing the Entries to write. The size of this buffer should correspond to *ul_WriteBytes*.

One Entry specifies one Frame + Header Information. This means one complete MAC frame plus a fixed sized Header. The Header contains information about the manner in which the frame should be sent on the network.

Layout of one Queue Entry:

	Entry Layout
Fixed Header	Fixed Frame Header Layout dependent on *ul_HeaderType* and *uc_FrameType* parameter (see following description)
AFDX Frame	AFDX- FRAME data to transmit (dependent on the Payload Buffer and Payload Generation mode, see description below) (802.3 defines: 64 to 1518 bytes)

For Header Type ***FDX_TX_FRAME_HEADER_REPLAY*** refer to the frame buffer layout described in function **FdxCmdMonQueueRead**.

Note: ***The replay mode does not reproduce any recorded physical error conditions, but is tolerant to protocol errors as well as size violations. A packet will be discarded by the firmware if any of the following error conditions is detected: PHY, PRE, TRI, CRC, IFG, SFD. The following error types are tolerated and will be replayed: IPE, MAE, LNG (up to frame length of 2000 bytes), SHR (from frame length of 40 bytes), VLS, SNE, TNS.***

Note: ***If frames defined with the same VL the number of those frames shall net exceed 255. Otherwise there will be a problem in Sequence Numbering.***

For Header Type ***FDX_TX_FRAME_HEADER_GENERIC*** see following description.

TY_FDX_TX_FRAME_HEADER x_TxFrameHeader

```
typedef struct {
   AiUInt8   uc_FrameType;
   TY_FDX_TX_FRAME_ATTRIB x_FrameAttrib;
   TY_FDX_TX_INSTR_ATTRIB x_InstrAttrib;
} TY_FDX_TX_FRAME_HEADER;
```

Note:	***The FdxInitTxFrameHeader function supports a default initialization of this structure (see this function in the chapter 'Target Independent Administration Functions'***

AiUInt8 uc_FrameType

The Type of the frame:

Value:	***Description:***
FDX_TX_FRAME_STD	Standard Generic Tx Frame
FDX_TX_FRAME_INSTR	Instruction Type

TY_FDX_TX_FRAME_ATTRIB x_FrameAttrib

This structure describes the Frame Attributes in case of ***FDX_TX_FRAME_STD uc_FrameType.***

```
typedef struct {
   AiUInt16 uw_FrameSize;
   AiUInt32 ul_InterFrameGap;
   AiUInt32  ul_PacketGroupWaitTime;
   AiUInt8   uc_PayloadBufferMode;
   AiUInt8   uc_PayloadGenerationMode;
   AiUInt32  ul_BufferQueueHandle;
   AiUInt8   uc_ExternalStrobe;
   AiUInt8   uc_PreambleCount;
   AiUInt32  ul_Skew;
   AiUInt8   uc_NetSelect;
   AiUInt8   uc_FrameStartMode;
   AiUInt32  ul_PhysErrorInjection;
   AiUInt16  uw_SequenceNumberInit;
   AiUInt16  uw_SequenceNumberOffset;
   AiUInt8   uc_TxIntEnable
   AiUInt32  ul_IntIdent
} TY_FDX_TX_FRAME_ATTRIB;
```

AiUInt16 uw_FrameSize;

Total size of the associated frame in Bytes (incl. CRC). Short and Long Frame Error Conditions are possible by setting the corresponding values. AFDX compliant values are 64…1518. For Frame length less than 60 no proper frame transmission is guaranteed.

AiUInt32 ul_InterFrameGap

This value defines the interframe gap between the preceding frame and the current frame with a resolution of 40ns, measured from the end of the last bit fo the preceding frame to the first preamble bit of the actual frame.

To implement a physical gap between the frames, a minimum interframe gap of 120 ns (value = 3) shall be initialized. The maximum provided interframe gap will be up to approx. 655us (14 Bits are used for encoding). If the Packet group Wait Time is used, this field shall be initialized with zero. This Gap is only used if ***uc_FrameStartMode*** is set to FDX_TX_START_FRAME_IFG.

See also the notes for ul_Skew parameter in redundant mode.

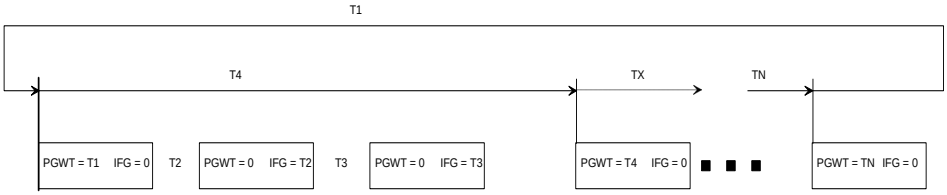
AiUInt32 ul_PacketGroupWaitTime

The Packet Group Wait Time (PGWT) provides the capability to implement a sequencing control for one or a group of frames. Each time, if a frame is processed where the PGWT value is not zero, the BIU- Processor handles the corresponding timing. The PGWT value (20bit) specifies the time from the transmission start point of the last frame where the PGWT value is processed to the start point of the current frame with a resolution of 1us.

At the processing of the first frame after the operation is enabled or after the execution of a 'Wait for Trigger' instruction, the PGWT value is ignored and the frame is transmitted immediately.

This Gap is only used if ***uc_FrameStartMode*** is set to FDX_TX_START_FRAME_PGWT.

For Start ModeFDX_TX_START_FRAME_TRG_D this field describes the delay time between trigger pulse and start of frame in µs.



AiUInt8 uc_PayloadBufferMode

The Payload Buffer Modes (PBMs) can be used to implement dynamic payload for either MAC or UDP- frame for this transmit frame, by using the separate buffer queue. The separate buffer queue will only be used, if the Buffer Queue Header Pointer is appropriate initialized (not zero) and the Payload Generation is disabled. The Buffer Queue itself implements a queue with one or multiple payload buffers, which each provides it's individual payload for the transmit frame. Due to the Buffer Queue function capabilities, different payload buffers can be used for the transmission of the frame. If a Payload Buffer Mode different from **FDX_TX_FRAME_PBM_STD** is used, the data for the frame must be provided in a separate Buffer, allocated via the ***FdxCmdTxBufferQueueAlloc*** function, dependent on the Payload Buffer Mode.

Value:	***Description:***
FDX_TX_FRAME_PBM_STD	Complete frame data is taken from the entry *ul_BufferQueueHandle* must be set to NULL
FDX_TX_FRAME_PBM_MAC Note: This mode cannot be selected if a Payload Generation Mode different from FDX_TX_FRAME_PGM_USER has been selected for this frame.	MAC- Payload is provided in the separate Buffer Queue. That means, the BIU- Processor fetches the Frame Header Words and the first 16 bytes of the frame data out from this entry and switches then to the separate buffer queue. Thus, the complete MAC- Header and the two static bytes of the IP- Header are used from this entry and the rest of the frame payload is used from the separate buffer queue. *ul_BufferQueueHandle* must contain a valid Buffer Queue handle, previously allocated via the function *FdxCmdTxBufferQueueAlloc*.
FDX_TX_FRAME_PBM_UDP Note: This mode cannot be selected if a Payload Generation Mode different from FDX_TX_FRAME_PGM_USER has been selected for this frame.	UDP- Payload is provided in the separate Buffer Queue. That means, the BIU- Processor fetches the Frame Header Words and the first 40 bytes of the frame data out from this entry and switches then to the separate buffer queue. Thus, the complete MAC- Header, the IP- Header and 6 bytes of the UDP- Header are used from this entry and the remainder of the frame payload is used from the separate buffer queue. That means, the 2 bytes of the UDP-Checksum (always zero) and the UDP- payload resides in the separate buffer. *ul_BufferQueueHandle* must contain a valid Buffer Queue handle, previously allocated via the function *FdxCmdTxBufferQueueAlloc*.
FDX_TX_FRAME_PBM_FULL Note: This mode cannot be selected if a Payload Generation Mode different from FDX_TX_FRAME_PGM_USER has been selected for this frame.	The full MAC-Frame is provided in the separate Buffer Queue. That means, the BIU- Processor fetches the Frame Header Words out from this entry and switches then to the separate buffer queue. *ul_BufferQueueHandle* must contain a valid Buffer Queue handle, previously allocated via the function *FdxCmdTxBufferQueueAlloc*.

AiUInt8 uc_PayloadGenerationMode

The Payload Generation Mode (PGM) defines, which contents of the frame data are automatically generated by the MAC- Hardware out of the static Tx data registers (see Command ***FdxCmdTxStaticRegsControl***). The payload generation mode is based on the UDP- protocol within the frame. Therefore, if one of the data reduction modes is selected, the transmit frame must include an UDP- protocol type.

Value:	***Description:***
FDX_TX_FRAME_PGM_USER	Default, no payload generation. All AFDX- frame data for transmission will be provided from this frame entry or partly from the payload buffer queue. In this mode, all MAC-frame data, except the FCS- field, has to be fed into the MAC by the BIU- Processor. Complete frame data must be provided for this frame.
FDX_TX_FRAME_PGM_IP_PART	The MAC- Destination Address Bytes 2..5, the MAC-Source Address Bytes 3…5, the MAC- Length Type Field Byte 0..1, the IP- Version field, the IP- Header Length field, the IP- protocol field and the UDP- checksum field as well as the hole UDP- payload will be generated by the MAC-Hardware by using the Static Transmit Registers. Only the MAC header and IP Header and UDP Header needs to be provided as frame data for this frame.
FDX_TX_FRAME_PGM_IP_FULL	The MAC- Destination Address Bytes 2…5, the MAC-Source Address Bytes 3…5, the MAC- Length Type Field Byte 0...1, the IP- Version field, the hole IP- Header (20 bytes) and the hole UDP- Header as well as the hole UDP-payload will be generated by the MAC- Hardware by using the Static Transmit Registers. Only the MAC header needs to be provided as frame data for this frame.
FDX_TX_FRAME_PGM_IP_PART_TT	Same as FDX_TX_FRAME_PGM_IP_PART, plus UDP-payload data will be filled with the start Timetag, which is repeated every eight bytes.
FDX_TX_FRAME_PGM_IP_FULL_TT	Same as FDX_TX_FRAME_PGM_IP_FULL, plus UDP-payload data will be filled with the start Timetag, which is repeated every eight bytes.

Note: ***This Static Transmit Registers must be setup properly if any frame uses a Payload Generation Mode, different from FDX_TX_FRAME_PGM_USER ! Otherwise the frame data may be invalid.***

The following Table shows the necessary size of one Queue Entry dependent on the Payload Buffer and Payload Generation Modes.

Payload Generation Mode	***Size of Queue Entry***
FDX_TX_FRAME_PGM_USER	sizeof (TY_FDX_TX_FRAME_HEADER) + uw_FrameSize
FDX_TX_FRAME_PGM_IP_PART FDX_TX_FRAME_PGM_IP_PART_TT	sizeof (TY_FDX_TX_FRAME_HEADER) + MAC-Header (14 Bytes) + IP-Header (20 Bytes) + UDP-Header (8 Bytes)
FDX_TX_FRAME_PGM_IP_FULL FDX_TX_FRAME_PGM_IP_FULL_TT	sizeof (TY_FDX_TX_FRAME_HEADER) + MAC-Header (14 Bytes)

The following Table shows the necessary size of one Queue Entry dependent on the Payload Buffer and Payload Generation Modes.

Payload Buffer Mode	*Payload Generation Mode*	*Size of Queue Entry*
FDX_TX_FRAME_PBM_STD	FDX_TX_FRAME_PGM_USER	sizeof (TY_FDX_TX_FRAME_HEADER) + uw_FrameSize
FDX_TX_FRAME_PBM_STD	FDX_TX_FRAME_PGM_IP_PART FDX_TX_FRAME_PGM_IP_PART_TT	sizeof (TY_FDX_TX_FRAME_HEADER) + MAC-Header (14 Bytes) + IP-Header (20 Bytes) + UDP-Header (8 Bytes)
FDX_TX_FRAME_PBM_STD	FDX_TX_FRAME_PGM_IP_FULL FDX_TX_FRAME_PGM_IP_FULL_TT	sizeof (TY_FDX_TX_FRAME_HEADER) + MAC-Header (14 Bytes)
FDX_TX_FRAME_PBM_FULL	FDX_TX_FRAME_PGM_USER	sizeof (TY_FDX_TX_FRAME_HEADER) Note: The full MAC Frame must be provided in separate Buffer.
FDX_TX_FRAME_PBM_MAC	FDX_TX_FRAME_PGM_USER	sizeof (TY_FDX_TX_FRAME_HEADER) + MAC-Header (14 Bytes) + IP-Version (1Byte) + IP-Type of Service (1Byte) Note: Remaining Data must be provided in separate Buffer.
FDX_TX_FRAME_PBM_UDP	FDX_TX_FRAME_PGM_USER	sizeof (TY_FDX_TX_FRAME_HEADER) + MAC-Header (14 Bytes) + IP-Header (20 Bytes) + UDP Source Port (2Bytes) + UDP Destination Port (2Bytes) + UDP Length (2Bytes) Note: Remaining Data must be provided in separate Buffer.

Note: ***For Timetag-Payload-Generation modes the Timetag Format in Payload is started at Byte 44 (2 bytes after UDP-Checksum)***
BYTE 44...47 – Timetag HI
Bit 5... 0 : Seconds of minute
Bit 11... 6 : Minutes of hour
Bit 16...12 : Hours of day
Bit 25...17 : Days of year
Bit 31...26 : reserved (0)
BYTE 48...51 – Timetag LO
Bit 19... 0 : Microseconds of second
Bit 25...20 : Seconds of minute
Bit 31...26 : Minutes of hour

AiUInt32 ul_BufferQueueHandle

In Payload Buffer Mode FDX_TX_FRAME_PBM_FULL, FDX_TX_FRAME_PBM_MAC or FDX_TX_FRAME_PBM_UDP is used for this frame, a valid Buffer Queue Handle must be set. This Buffer Handle is obtained via the function ***FdxCmdTxBufferQueueAlloc***. If Payload Buffer Mode is not used, it should be initialized with NULL. Dependent on the Payload Buffer Mode, the allocated Buffer must contain the corresponding data beginning with MAC payload or UDP payload.

Using Payload Buffer Mode FDX_TX_FRAME_PBM_UDP or FDX_TX_FRAME_PBM_MAC would allow the user to change data associated with this frame during run-time by the ***FdxCmdTxBufferQueueWrite*** and ***FdxCmdTxBufferQueueCtrl*** functions.

AiUInt8 uc_ExternalStrobe

Control assertion of Trigger Strobe if this frame is transmitted. See the ***FdxCmdTxTrgLineCtrl*** for further information about the Trigger Lines.

Value:	Description:
FDX_DIS	Disable Trigger Strobe
FDX_ENA	Assert external Trigger Strobe on transmission of this frame

AiUInt8 uc_PreambleCount

This value defines the number of preamble Bytes sent for this frame

Value:	Description:
FDX_TX_FRAME_PRE_DEF	Send default preamble count of 7 Bytes
All other values from n=1..15	Send n preamble Bytes

AiUInt32 ul_Skew

This parameter defines the transmission skew between the redundant frames on the AFDX- ports. The skew can be programmed with a resolution of 1us. Range is 0...65535. This parameter is only used if uc_NetSelect is FDX_TX_FRAME_DLY_A or FDX_TX_FRAME_DLY_B.

Note:	***This function is only provided in redundant port operation mode.***

Note:	***If the ul_Skew parameter is set and one redundant frame is delayed this time may be added to ul_InterFrameGap and may exceed maximum value of ul_InterFrameGap in the receiver.This means it can result in a higher Interframe Gap Time because the IFG counter for transmit is sterted synchroniously for both networks after both redundant frames are sent..***

AiUInt8 uc_NetSelect

This parameter defines the physical Interface_ID of the MAC, which shall send it's current frame delayed (with ***ul_Skew*** µs) to the alternate port.

Value:	Description:
FDX_TX_FRAME_DLY_A	Packet on Network A is delayed by the Skew value, related to Network B
FDX_TX_FRAME_DLY_B	Packet on Network B is delayed by the Skew value, related to Network A
FDX_TX_FRAME_BOTH	Packet transmitted on both Networks (Skew=0)
FDX_TX_FRAME_ONLY_A	Packet only transmitted on Network A
FDX_TX_FRAME_ONLY_B	Packet only transmitted on Network B

Note:	***This function is only provided in redundant port operation mode.***

AiUInt8 uc_FrameStartMode

This parameter defines the Frame Start mode for the transmission of the current frame.

Value:	*Description:*
FDX_TX_FRAME_START_IFG	Start transmission of this frame if Interframe GAP time has expired (see *ul_InterFrameGap* parameter)
FDX_TX_FRAME_START_PGWT	Start transmission of this frame if Packet Group Wait Time (PGWT) has expired (see *ul_PacketGroupWaitTIme* parameter)
FDX_TX_FRAME_START_TRG	Start transmission of this frame on external Trigger Strobe. This means, frame transmission is stopped with this frame, until the external Trigger Strobe is given to continue transmission with this frame.
FDX_TX_FRAME_START_TRG_D	Start transmission of this frame on external Trigger Strobe. This is the same start mode as described before for FDX_TX_FRAME_START_TRG but extended by a delay. After the Trigger Strobe was occurred, the frame will be started delayed by the time defined in *ul_PacketGroupWaitTime*.

AiUint32 ul_PhysErrorInjection

This parameter defines physical error injection types. The error injection information can be a combination of the following error types:

Value:	*Description:*
FDX_TX_FRAME_ERR_OFF	No Error Injection enabled
FDX_TX_FRAME_ERR_CRC	CRC Error transmitted with this frame
FDX_TX_FRAME_ERR_ALI	Wrong Byte alignment in transmit frame, which means that an odd number of nibbles will be transmitted. Therefore, this error will also cause a CRC error condition *Note: This Error can not be injected in 1 Gbit/s mode.*
FDX_TX_FRAME_ERR_PRE	Wrong Preamble Sequence transmitted. If this type is selected., the Encoder device substitutes the first nibble of the Start Frame Delimiter with the value '1000' instead of '1001'
FDX_TX_FRAME_ERR_PHY	Physical Symbol Error. During Frame Transmission, the MAC-Encoder device asserts the Tx-Error signal, which forces the physical transceiver to transmit 'HALT' symbols.

AiUint16 uw_SequenceNumberInit

This parameter defines Initial Sequence Number for this frame. First frame transmission starts with this Sequence Number and adds then the Sequence Number Offset ***uc_SequenceNumberOffset*** , as described at the following parameter.

Value:	*Description:*
FDX_TX_FRAME_SEQ_INIT_AUTO	Sequence Number Init value is set by the Driver automatically.
0…255	Set Sequence Number Offset for this frame (e.g. Error Injection Purposes if invalid values used !)
FDX_TX_FRAME_SEQ_OFF	Sequence Number handling for this transfer is switched off and used as normal data byte.

AiUint16 uw_SequenceNumberOffset

This parameter defines the Sequence Number Offset. This field provides the Sequence Number offset, which is added to the Sequence Number after the Frame has been transmitted. The Sequence number will be incremented until the value of 255 and then it wraps around to 1. Thus, the user can initialize a transmission sequence, which implements more than one frame with the same VL. If the transmission sequence implements N packets with the same VL, this field shall be initialized with N to implement proper sequence numbering for each transmitted VL- frame.

If uw_SequenceNumberInit is switched to FDX_TX_FRAME_SEQ_OFF the value for uw_SequenceNumberOffset is obsolete and don't care.

Value:	***Description:***
FDX_TX_FRAME_SEQ_OFFS_AUTO	Sequence Number Offset is set by the Driver automatically.
0…255	Set Sequence Number Offset for this frame (e.g. Error Injection Purposes if invalid values used !)

Note:	***For FDX_TX_FRAME_SEQ_OFFS_AUTO frames defined with the same VL the number of those frames shall net exceed 255. Otherwise there will be a problem in Sequence Numbering.***

AiUInt8 uc_TxIntEnable

This parameter enables signaling by interrupt when this Transfer is transmitted. The interrupt will be report information in the Interrupt loglist

AiUInt32 ul_IntIdent

With this parameter it is possible to define a unique interrupt identifier for this transfer. This identifier will be reported in the Interrupt Loglist

TY_FDX_TX_INSTR_ATTRIB x_TxInstrAttrib

This structure describes the Instruction Attributes in case of ***FDX_TX_FRAME_INSTR uc_FrameType.***

```
typedef struct {
   AiUInt8 uc_Code;
   AiUInt8 uc_Interrupt ;
   AiUInt8 uc_NumOfSubQueues
   AiUInt8 uc_ActivSubQueue
   AiUInt32 aul_SubQueueHandle[FDX_MAX_TX_SUB_QUEUES];
} TY_FDX_TX_INSTR_ATTRIB;
```

AiUInt8 uc_Code

Following Instruction Codes are supported :

Value:	***Description:***
FDX_TX_FRAME_INSTR_NOP	No Operation
FDX_TX_FRAME_INSTR_STOP	Stop Transmission Transmission is stopped if BIU processor runs on this Instruction
FDX_TX_FRAME_INSTR_SYNC	Synchronize BIU Processor waits until Transmit Burst Buffer (between BIU and MAC) is empty
FDX_TX_FRAME_INSTR_CALL	Call a Transmit Sub Queue.
FDX_TX_FRAME_INSTR_ACYC_MARK	Insert a Marker point for execution of Acyclic Instruction. If an Acyclic instruction is defined, it will be scheduled for transmission by reaching this instruction. This marker can be inserted

	several times in a transfer queue or SubQueue.

AiUInt8 uc_Interrupt

Enable/Disable Interrupt on execution of Instruction

AiUInt8 uc_ActivSubQueue

This parameter is only valid for command FDX_TX_FRAME_INSTR_CALL of other commands i is obsolete.

This parameter defines, which Transmit SubQueue of all referenced SubQueues shall be active first with this call.

The paramete must be in a range of 0 up to uc_NumOfSubQueues-1.

AiUInt8 aul_SubQueueHandle[FDX_MAX_TX_SUB_QUEUES]

This parameter is only valid for command FDX_TX_FRAME_INSTR_CALL of other commands i is obsolete.

This is a array of Transmit Sub Queue handles returened form the command 'FdxCmdTxSubQueueCreate'

The size of this array can be up to FDX_MAX_TX_SUB_QUEUES entries.

FDX_MAX_TX_SUB_QUEUES is defined to 8.

Note:	***For the usage of Call instructions to sub Queues in an AFDX conform environment it is recommended to use only one Sub Queue. The reason for this limitation is the problem of Sequence numbering. Only the Sub Queue initialized before start of transmitter is taken in account for correct AFDX Sequence numbering.***

For Header Type ***FDX_TX_FRAME_HEADER_REPLAY (FIFO mode)*** see following description.

TY_FDX_TX_FIFO_FRAME_HEADER

This is a structural description of the frame buffer header

```
typedef struct _fdx_tx_fifo_frame_header {
   AiUInt32 ul_Reserved1;
   AiUInt32 ul_Reserved2;
   AiUInt32 ul_Reserved3;
   AiUInt32 ul_Reserved4;
   AiUInt32 ul_FrameHeaderWord0;
   AiUInt32 ul_FrameHeaderWord1;
   AiUInt32 ul_FrameHeaderWord2;
   AiUInt32 ul_Reserved5;
   AiUInt32 ul_Reserved6;
} TY_FDX_TX_FIFO_FRAME_HEADER;
```

AiUInt16 ul_Reserved1

Reserved

AiUInt16 ul_Reserved2

Reserved

AiUInt16 ul_Reserved3

Reserved

AiUInt16 ul_Reserved4

Reserved

Frame Header Word 0 contains following information

Value	*Bit*	*Description*
Reserved	31..20	reserved
Physical Error Injection	19	CRC error transmitted in this frame
	18	Wrong Byte alignment in transmit frame, which means that an odd number f nibbles will be transmitted. Therefore, this error also cause a CRC error condition.
	17	Wrong Preamble/SFD Sequence transmitted. If this function is enabled, the Encoder device injects an Error within the Start Frame Delimiter. Note: if a Wrong Reamble/SFD Sequence is transmitted, the Physical Receiver devices connected on the network may get some problems to detect the current frame proberly.
	16	Physical Symbol Error. During the Frame transmission in 100Mbit/s operattion mode, the MAC – Encoder deviceasserts the TX-Error signal, which forces the phydical tranceiver to transmit 'HALT'-Symbols. The error type is not available in 10Mbit/s operation mode.
TX-Skew	15…0	Thias 16 bit value defines the skew in microseconds between two redundant frames in transmission. The skwe can be programmed from 0ms up to 65ms, with a resolution of 1 microsenconds. The port, where the delayed frame is transmitted is defined in the Frame Header Word 1.This value is only used, if the Interface is configured in redundant operation mode.

AiUInt32 ul_FrameHeaderWord_1

Frame Header Word 1 contains following information

Value	*Bit*	*Description*
RSF-Port	31..29	This field contains the information about the special transmit functionality for the current frame in redundant operation mode. This functionality allows either to delay the frame transmission on the defined physical port or it allows to suppress the transmission of the frame on the defined physical port. This functionality is only provided in redundant operation mode.
	0 0 0 0 0 1 0 1 0 1 0 1 1 1 0 default	Both packets are sent simultaneously (without skew) Packet on Network A is delayed by the Skew value, related to Network B Packet on Network B is delayed by the Skew value, related to Network A Packet on Network A is suppressed at redundant transmission Packet on Network B is suppressed at redundant transmission reserved
Reserved	28	reserved
IFG-CNT		Interframe gap This 14bit value defines the minimum Interframe gap between the preceding frame and the current frame. The Interframe gap is defined in gap time units (GTU), which are dependent on the transmission speed mode. A GTU is time to transmit four data bits, which means 40ns at 100Mbit/s operation mode and 400ns at 10Mbit/s operation mode. The range of the Interframe Gap value starts from four GTUs up to 16383 GTUs. If less than 24 GTUs are defined, the connected Receive devices on the network may (shall) detect an 'Short Interframe Gap' error condition. The maximum provided Interframe gap will be up to appr. 655us 100Mbit/s operation mode or up to appr. 6,55ms 10Mbit/s operation mode. Please Note, the Interframe gap is only applicable, if two frames are loaded consecutive to the Transmit FIFO. In this case, the Interframe gap applied by the Encoder device.
Reserved	13..11	reserved
Byte Count	10..0	This field contains the number of bytes, which shall be transmitted within the AFDX- MAC- Frame, including the CRC field. Since this value is programmable, Short frame conditions (below 64bytes) as well as Frame too Long conditions (more as 1518 bytes) can be generated. If Byte count values less than 60 bytes are programmed, no proper frame transmission can be guaranteed. Note: Byte Count values less than 40 bytes and more than 2000bytes are not allowed and may be cause unrecoverable transmit operation errors.

AiUInt32 ul_FrameHeaderWord_2

Frame Header Word 2 contains following information

Value	*Bit*	*Description*
Buffer-Sizet	31..16	The Buffer Size field indicates the size of the total AFDX-Buffer entry in bytes, including the buffer header. Due to the burst capabilities of the Hardware, the AFDX- Buffer entry size is aligned to 64 byte blocks.
Reserved	15..11	reserved
ATT	10	Add Transmit Timestamp Logical '1': If this bit is set additional to the FTI, then BIU-Processor adds the corresponding Timestamp of the FTI Event to the Interrupt Loglist Entry
FTI	9	Frame Transmitted Interrupt Logical '1': If this bit is set, the BIU- Processor asserts an interrupt, if the current frame was physically transmitted and the Interrupt Loglist Entry will include the start address of the current frame.
SND	8	Sequence Number Insertion Disable Logical '1': If this bit is set, the AFDX- Sequence Number insertion at the end of the MAC- Frame is suppressed. Logical '0': Default, AFDX- Sequence Number insertion at the end of the MAC- Frame.
SN	7…0	MAC- Frame Sequence Number for this packet. At packet transmission, the Encoder add this sequence number at the end of the AFDX- Frame, if the SND bit is not set.

AiUInt16 ul_Reserved5

Reserved

AiUInt16 ul_Reserved6

Reserved

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.2.5 **FdxCmdTxQueueControl**

AiReturn FdxCmdTxQueueControl (AiUInt32 ul_Handle,
const TY_FDX_TX_QUEUE_CONTROL *px_Update);

Purpose:

This function allows to control parameters of the Fixed Header like described in FdxCmdTxQueueWrite for frames in a generic transmit queue after a frame was already written to the queue. Control is also possible while the transmission is running (on the fly).

Input:

*TY_FDX_TX_QUEUE_CONTROL *px_Control*

A pointer to a control structure which describes which frame in the queue shall be modified and also which parameters shall be modified.

```
typedef struct {
   AiUInt32 ul_SubQueueHandle;
   AiUInt32 ul_Index;
   AiUInt32 ul_ControlType;
   AiUInt32 ul_DisaEna;
   AiUInt32 ul_Size;
   AiUInt32 ul_IFG;
   AiUInt32 ul_PGWT;
   AiUInt32 ul_PError;
   AiUInt32 ul_StartMode;
   AiUInt32 ul_DisaEnaInt;
   AiUInt32 ul_NextSubQueueIndex;
} TY_FDX_TX_QUEUE_CONTROL;
```

AiUInt32 ul_SubQueueHandle

If ul_SubQueueHandle is unequal to 0 it indicates that the Transfer which shall be controlled is located in a Transmitter SubQueue. A value of 0 indicates the Main TransferQueue.

AiUInt32 ul_Index

Index to the frame which shall be controlled with this call. This is a counting value starting with 0 over all frames written to the queue with the command FdxCmdTxQueueWrite. The first written frame has the index 0. If there are commends inserted to the queue, the commands are also numbered.

If ul_SubQueueHandle is unequal to 0, ul_Index 0 addresses the first transfer of the SubQueue

AiUInt32 ul_ControlType

This parameter indicates which values shall be controlled and which following values of this structures must be initialised.

It is possible to control several values at one time by selecting several Control Types by wired or function.

Constant	*Description*
FDX_TX_CTL_ENDIS	Enable or disable a Transfer
FDX_TX_CTL_SIZE	Set a new Frame Size for this Transfer
FDX_TX_CTL_IFG	Modify the Inter Frame Gap for this Transfer
FDX_TX_CTL_PGWT	Modify the Packet Group Wait-Time for this Transfer
FDX_TX_CTL_PERROR	Set the physical Error Injection for this Transfer
FDX_TX_CTL_SMODE	Modify the Start Mode for this Transfer
FDX_TX_CTL_ENDISINT	Enable or Disable Interrupt capability for this Transfer
FDX_TX_CTL_SUBQUE	Switch to an other defined SubQueu for call SubQueueInstruction

AiUInt32 ul_DisaEna

Parameter to enable or disable a Transfer in a Transfer Queue or Sub Transfer Queue.

This parameter must be initialized if FDX_TX_CTL_ENDIS is set to ul_ControlType.

Value	*Comment*
FDX_ENA	Enable Transfer or call instruction
FDX_DIS	Disable Transfer or call instruction

AiUInt32 ul_Size

This parameter must be initialized if FDX_TX_CTL_SIZE is set to ul_ControlType.

Total size of the associated frame in Bytes (incl. CRC). Short and Long Frame Error Conditions are possible by setting the corresponding values. AFDX compliant values are 64…1518. For Frame length less than 60 no proper frame transmission is guaranteed.

The size should only varied in the range from minimum frame length up to the maximum frame length the frame was specified originally with the FdxCmdTxQueueWrite Command. In other cases you will get unpredictable behaviour.

AiUInt32 ul_IFG

This parameter must be initialized if FDX_TX_CTL_IFG is set to ul_ControlType.

This value defines the interframe gap between the preceding frame and the current frame with a resolution of 40ns, measured from the end of the last bit from the preceding frame to the first preamble bit of the actual frame.

To implement a physical gap between the frames, a minimum interframe gap of 120 ns (value = 3) shall be initialized. The maximum provided interframe gap will be up to approx. 655us (14 Bits are used for encoding). If the Packet group Wait Time is used, this field shall be initialized with zero. This Gap is only used if ***uc_FrameStartMode*** is set to FDX_TX_START_FRAME_IFG.

See also the notes for ul_Skew parameter in redundant mode.

AiUInt32 ul_PGW

This parameter must be initialized if FDX_TX_CTL_PGW is set to ul_ControlType.

AiUInt32 ul_PError

This parameter must be initialized if FDX_TX_CTL_PERROR is set to ul_ControlType.

This parameter defines physical error injection types. The error injection information can be a combination of the following error types:

Value:	*Description:*
FDX_TX_FRAME_ERR_OFF	No Error Injection enabled
FDX_TX_FRAME_ERR_CRC	CRC Error transmitted with this frame
FDX_TX_FRAME_ERR_ALI	Wrong Byte alignment in transmit frame, which means that an odd number of nibbles will be transmitted. Therefore, this error will also cause a CRC error condition *Note: This Error can not be injected in 1 Gbit/s mode.*
FDX_TX_FRAME_ERR_PRE	Wrong Preamble Sequence transmitted. If this type is selected., the Encoder device substitutes the first nibble of the Start Frame Delimiter with the value '1000' instead of '1001'
FDX_TX_FRAME_ERR_PHY	Physical Symbol Error. During Frame Transmission, the MAC-Encoder device asserts the Tx-Error signal, which forces the physical transceiver to transmit 'HALT' symbols.

AiUInt32 ul_StartMode

This parameter defines the Frame Start mode for the transmission of the current frame.

This parameter must be initialized if FDX_TX_CTL_SMODE is set to ul_ControlType.

Value:	*Description:*
FDX_TX_FRAME_START_IFG	Start transmission of this frame if Interframe GAP time has expired (see *ul_InterFrameGap* parameter)
FDX_TX_FRAME_START_PGWT	Start transmission of this frame if Packet Group Wait Time (PGWT) has expired (see *ul_PacketGroupWaitTIme* parameter)
FDX_TX_FRAME_START_TRG	Start transmission of this frame on external Trigger Strobe. This means, frame transmission is stopped with this frame, until the external Trigger Strobe is given to continue transmission with this frame.

AiUInt32 ul_DisaEnaInt

This switch disables or reenables interrupt generation for Transfer.

This parameter must be initialized if FDX_TX_CTL_ENDISINT is set to ul_ControlType.

Note: Interrupt enable is only possible if interrupt generation for this transfer was enabled by writing the Transfer with FdxCmdTxQueueWrite or FdxCmdTxSubQueueWrite and interrupt was disable before.

Value	*Comment*
FDX_ENA	Enable Interrupt
FDX_DIS	Disable Disable

AiUInt32 ul_NextSubQueueIndex

This parameter must be initialized if FDX_TX_CTL_SUBQUE is set to ul_ControlType.

This parameter is only applicable for a call to SubQueue Instruction and can only combined with enabling or disabling a SubQueue call instruction

The value specifies the index of the next defined SubQueue which shall be called from this instruction.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.2.6 FdxCmdTxQueueAcyclic

Prototype:

AiReturn FdxCmdTxQueueAcyclic (AiUInt32 ul_Handle,
AiUInt32 ul_WriteBytes,
const void *pv_WriteBuffer);

Purpose:

This function is used to insert an acyclic frame in a configured stream of generic transfers. This frame will be sent immediate when an acyclic marker in the generic transfer list is detected for one time. The acyclic marker must be set by writing the command to the transmit queue or sub queue

Input:

AiUInt32 ul_WriteBytes

Number of bytes that shall be written to the queue.

void *pv_WriteBuffer

Pointer to the data buffer providing the Entries to write. The size of this buffer should correspond to *ul_WriteBytes*.

One Entry specifies one Frame + Header Information. This means one complete MAC frame plus a fixed sized Header. The Header contains information about the manner in which the frame should be sent on the network.

Layout of one Queue Entry:

	Entry Layout
Fixed Header	Fixed Frame Header Layout dependent on *ul_HeaderType* and *uc_FrameType* parameter (see following description)
AFDX Frame	AFDX- FRAME data to transmit (dependent on the Payload Buffer and Payload Generation mode, see description below) (802.3 defines: 64 to 1518 bytes)

TY_FDX_TX_FRAME_HEADER x_TxFrameHeader

```
typedef struct {
   TY_FDX_TX_FRAME_ATTRIB x_FrameAttrib;
} TY_FDX_TX_ACYCLIC_FRAME_HEADER;
```

Note:	***The FdxInitTxFrameHeader function supports a default initialization of this structure (see this function in the chapter 'Target Independent Administration Functions'***

TY_FDX_TX_FRAME_ATTRIB x_FrameAttrib

This structure describes the Frame Attributes inside the fixed frame header. For all details how to setup this fixed frame header please refer to the function 'FdxCmdTxQueueWrite'

Note:	***Acyclic inserted frames are not taken in account for correct AFDX Sequence numbering.***

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.3 Individual (UDP Port oriented)Transmitter Functions

4.3.3.1 FdxCmdTxCreateVL

Prototype:

AiReturn FdxCmdTxCreateVL(AiUInt32 ul_Handle,
const TY_FDX_TRANSMIT_VL *px_TransmitVL);

Purpose:

This function is used to setup a Virtual Link to transmit data. This is only a description of the Virtual Link specific parameters, which must be known by the system for transmitting. This function can be used only when the transmitter is not running.

Input:

TY_FDX_TRANSMIT_VL *px_TransmitVL

A pointer to a structure which describes the Virtual Link related parameters for the transmitter.

```
typedef struct {
   AiUInt32 ul_VlId;
   AiUInt32 ul_SubVls;
   AiUInt32 ul_Bag;
   AiUInt32 ul_NetSelect;
   AiUInt32 ul_MaxFrameLength;
   AiUInt32 ul_FrameBufferSize;
   AiUInt32 ul_MACSourceLSLW;
   AiUInt32 ul_MACSourceMSLW;
   AiUInt32 ul_skew;
} TY_FDX_TRANSMIT_VL;
```

AiUInt32 ul_VLId

Virtual Link Identifier. A value in a range from 0 to 65535. This value is part of the MAC destination address.

AiUInt32 ul_SubVls

Number of Sub VLs associated to this VL. This Value must be in a range from 1 to 4.

AiUInt32 ul_Bag

Specifies the Bandwidth Allocation Gap (BAG)for this Virtual Link in milliseconds.

Possible Values for the BAG are 1 to 128ms defined as 2^k[in ms], where k can have a range from integer 0 to 7.(Source [4])

The bag limits the bandwidth of a VL. Refer also to ul_SamplingRate defined with ***FdxCmdTxUDPCreatePort.***

AiUInt32 ul_MaxFrameLength

Specifies the maximum length of one MAC-Frames which can by this VL.

AiUInt32 ul_FrameBufferSize

Specifies the size of the VL buffer in bytes. If this value is set to zero, the onboard target software will set it to a default value. The buffer is part of global (BIU) memory.

AiUInt32 ul_MACSourceLSLW

Least Significant Longword of the MAC Source Address

aa:bb:cc:dd:ee:ff

Bit 31-24	Bit 23-16	Bit 15-8	Bit 7-0
cc	dd	ee	ff

AiUInt32 ul_MACSourceMSLW

Most Significant Longword of the MAC Source Address

aa:bb:cc:dd:ee:ff

Bit 31-24	Bit 23-16	Bit 15-8	Bit 7-0
0 (reserved)	0 (reserved)	aa	bb

AiUInt32 ul_NetSelect

Value:	Description:
FDX_TX_FRAME_DLY_A	Packet on Network A is delayed by the Skew value, related to Network B
FDX_TX_FRAME_DLY_B	Packet on Network B is delayed by the Skew value, related to Network A
FDX_TX_FRAME_BOTH	Packet transmitted on both Networks (Skew=0)
FDX_TX_FRAME_ONLY_A	Packet only transmitted on Network A
FDX_TX_FRAME_ONLY_B	Packet only transmitted on Network B

Note: This function is only provided in redundant port operation mode.

AiUInt32 ul_skew

This parameter defines the transmission skew between the redundant frames on the AFDX-ports. The skew can be programmed with a resolution of 1µs. Range is 0...65535. This parameter is only used if uc_NetSelect is FDX_TX_FRAME_DLY_A or FDX_TX_FRAME_DLY_B.

Note: This function is only provided in redundant port operation mode.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.3.2 FdxCmdTxCreateHiResVL

Prototype:

AiReturn FdxCmdTxCreateHiresVL(AiUInt32 ul_Handle,
const TY_FDX_TRANSMIT_VL *px_TransmitVL);

Purpose:

This function is used to setup a Virtual Link to transmit data. This is only a description of the Virtual Link specific parameters, which must be known by the system for transmitting. This function can be used only when the transmitter is not running. This function is used to specify a VL with a high resolution bag.

Input:

TY_FDX_TRANSMIT_VL *px_TransmitVL

A pointer to a structure which describes the Virtual Link related parameters for the transmitter.

```
typedef struct {
   AiUInt32 ul_VlId;
   AiUInt32 ul_SubVls;
   AiUInt32 ul_Bag;
   AiUInt32 ul_NetSelect;
   AiUInt32 ul_MaxFrameLength;
   AiUInt32 ul_FrameBufferSize;
   AiUInt32 ul_MACSourceLSLW;
   AiUInt32 ul_MACSourceMSLW;
   AiUInt32 ul_skew;
} TY_FDX_TRANSMIT_VL;
```

AiUInt32 ul_VLId

Virtual Link Identifier. A value in a range from 0 to 65535. This value is part of the MAC destination address.

AiUInt32 ul_SubVls

Number of Sub VLs associated to this VL. This Value must be in a range from 1 to 4.

AiUInt32 ul_Bag

Specifies the Bandwidth Allocation Gap (BAG) for this Virtual Link in microseconds.

Can be adjusted in 500 microsecond steps with a maximum of 128000 microseconds. Values that are not multiples of 500 are not allowed and will lead to undefined/platform dependent behaviour. The bag limits the bandwidth of a VL.

Note: ARINC664 conform values for the BAG are 1 to 128ms defined as 2^k [in ms], where k can have a range

AiUInt32 ul_MaxFrameLength

Specifies the maximum length of one MAC-Frames which can by this VL.

AiUInt32 ul_FrameBufferSize

Specifies the size of the VL buffer in bytes. If this value is set to zero, the onboard target software will set it to a default value. The buffer is part of global (BIU) memory.

AiUInt32 ul_MACSourceLSLW

Least Significant Longword of the MAC Source Address

aa:bb:cc:dd:ee:ff

Bit 31-24	Bit 23-16	Bit 15-8	Bit 7-0
Cc	dd	ee	ff

AiUInt32 ul_MACSourceMSLW

Most Significant Longword of the MAC Source Address

aa:bb:cc:dd:ee:ff

Bit 31-24	Bit 23-16	Bit 15-8	Bit 7-0
0 (reserved)	0 (reserved)	aa	bb

AiUInt32 ul_NetSelect

Value:	Description:
FDX_TX_FRAME_DLY_A	Packet on Network A is delayed by the Skew value, related to Network B
FDX_TX_FRAME_DLY_B	Packet on Network B is delayed by the Skew value, related to Network A
FDX_TX_FRAME_BOTH	Packet transmitted on both Networks (Skew=0)
FDX_TX_FRAME_ONLY_A	Packet only transmitted on Network A
FDX_TX_FRAME_ONLY_B	Packet only transmitted on Network B

Note: This function is only provided in redundant port operation mode.

AiUInt32 ul_skew

This parameter defines the transmission skew between the redundant frames on the AFDX-ports. The skew can be programmed with a resolution of 1µs. Range is 0...65535. This parameter is only used if uc_NetSelect is FDX_TX_FRAME_DLY_A or FDX_TX_FRAME_DLY_B.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.3.3 FdxCmdTxSAPBlockWrite

Prototype:

AiReturn FdxCmdTxSAPBlockWrite (AiUInt32 ul_Handle,
const TY_FDX_SAP_BLOCK_WRITE_IN
****px_SapBlockWriteIn,***
TY_FDX_SAP_BLOCK_WRITE_OUT
****px_SapBlockWriteOut);***

Purpose:

This function is used to write a pure message to one or more SAP ports. If the data size is not applicable for the data size associated to this port this function will return an error.

This function can be used if the transmitter is running or not running.

Input:

TY_FDX_SAP_BLOCK_WRITE_IN
****px_SapBlockWriteIn***

```
typedef struct {
   AiUInt32    ul_MsgCount;
   TY_FDX_SAP_BLOCK_WRITE_IN_MSG* px_SapBlockWriteMsgArray;
} TY_FDX_SAP_BLOCK_WRITE_IN;
```

AiUInt32 ul_MsgCount

Specifies the number of messages to be written.

TY_FDX_SAP_BLOCK_WRITE_IN_MSG
****px_SapBlockWriteMsgArray***

Pointer to an array structures. Each structure describes the message to be written to a single UDP transmission port. The array contains ul_MsgCount elements.

```
typedef struct {
   AiUInt32    ul_UdpHandle;
   AiUInt32    ul_ByteCount;
   AiUInt32    ul_UdpDst;
   AiUInt32    ul_IpDst;
   void        *pv_Data;
} TY_FDX_SAP_BLOCK_WRITE_IN_MSG;
```

AiUInt32 ul_UdpHandle

The handle of the UDP port to which the message shall be written. This may be a handle to either a Queuing or Sampling UDP port.

AiUInt32 ul_ByteCount

Number of bytes to write to this SAP port. The value must be equal or smaller than ul_MaxMessageSize defined with FdxCmdTxSAPCreatePort().

AiUInt32 ul_UdpDst

The UDP destination port for the message

AiUInt32 ul_IpDst

The IP destination of the message

void *pv_Data

Pointer to a buffer containing the data to write.

Output:

*TY_FDX_SAP_BLOCK_WRITE_OUT *px_SapBlockWriteOut*

```
typedef struct {
   AiReturn st_GlobalResultCode;
   AiUInt32 ul_MsgCount;
   TY_FDX_SAP_BLOCK_WRITE_OUT_RESULT *px_SapBlockWriteResultArray;
} TY_FDX_SAP_BLOCK_WRITE_OUT;
```

AiReturn st_GlobalResultCode

Specifies the overall result of the block write operation.

Value	*Description*
FDX_OK	The block operation completed successfully. All messages were successfully written to the respective UDP ports.
FDX_ERR	At least one of the individual writes to the UDP ports has failed. The st_ResultCode entries in the output array should be checked for identification of which message(s) have failed.

TY_FDX_SAP_BLOCK_WRITE_OUT_RESULT *px_SapBlockWriteResultArray

Pointer to an array of structures. Each structure specifies the result of an individual SAP write operation. The array contains ul_MsgCount elements.

```
typedef struct {
   AiUInt32    ul_UdpHandle;
   AiUInt32    ul_BytesWritten;
   AiReturn    st_ResultCode;
} TY_FDX_UDP_BLOCK_WRITE_OUT;
```

AiUInt32 ul_UdpHandle

The handle of the associated SAP port.

AiUInt32 ul_BytesWritten

Number of bytes actually written. Might be smaller than ul_ByteCount if SAP buffer is full. (ul_NumBufMessages defined with FdxCmdTxSAPCreatePort)

AiReturn st_ResultCode

The result of the individual write operation. FDX_OK on success or a negative error code if an error occurs.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.3.4 FdxCmdTxSAPCreatePort

Prototype:

AiReturn FdxCmdTxSAPCreatePort (AiUInt32 ul_Handle,
const TY_FDX_TX_SAP_CREATE_IN *px_TxSapCreateIn,
TY_FDX_TX_SAP_CREATE_OUT *px_TxSapCreateOut);

Purpose:

Creates an SAP type UDP port to transmit data from a specified UDP connectionless port. Default settings after creation of SAP port are:

- UDP port enabled
- Error injection: OFF
- Skew (redundant mode only): 0 usec.

To change settings of the SAP port the function ***FdxCmdTxUDPControl*** can be used. This function can be used only, if transmitter is not running.

Input:

TY_FDX_TX_SAP_CREATE_IN *px_SapCreateIn

Pointer to a structure, which describes the SAP port.

```
typedef struct {
   AiUInt32 ul_UdpSrc;
   AiUInt32 ul_IpSrc;
   AiUInt32 ul_VlId;
   AiUInt32 ul_SubVlId;
   AiUInt32 ul_UdpNumBufMessages;
   AiUInt32 ul_UdpMaxMessageSize;
}TY_FDX_TX_SAP_CREATE_IN;
```

AiUInt32 ul_UdpSrc

The UDP port number which is to be used in the UDP Source field of datagrams sent from this port.

AiUInt32 ul_IpSrc

The IP Source address which is to be used in the IP Source address filed of packets sent from this port.

AiUInt32 ul_VlId

The AFDX Tx VL over which frames originating from this port shall be transmitted.

AiUInt32 ul_SubVlId;

Sub Virtual Link Identifier (Sub VLs are only relevant in Tx Mode). This value must be in a range from 1 to 4. If Sub VLs are not used, the Sub VL Id equals to 1.

Note:	***Must be consistent with parameter ul_SubVls of function FdxCmdTxCreateVL or FdxCmdTxCreateHiResVL.***

AiUInt32 ul_UdpNumBufMessages

Number of messages which can be stored by the onboard Target software.

Size of the local Buffer which should be created by the onboard Target software to store data of the created SAP port can be calculated by ul_UdpNumBufMessages * ul_UdpMaxMessageSize.

If this value is set to zero, the onboard target software will set to a default value.

AiUInt32 ul_UdpMaxMessageSize;

Maximum size of a message to send.

Output:

TY_FDX_TX_SAP_CREATE_OUT *px_TxSapCreateOut

Pointer to a structure, which describes the SAP port.

```
typedef struct {
   AiUInt32 ul_UdpHandle;
}TY_FDX_TX_SAP_CREATE_OUT;
```

AiUInt32 ul_UdpHandle

Handle to the SAP port.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.3.5 FdxCmdTxSAPWrite

Prototype:

AiReturn FdxCmdTxSAPWrite (AiUInt32 ul_Handle,
const TY_FDX_SAP_WRITE_IN *px_TxSapWriteIn,
TY_FDX_SAP_WRITE_OUT *px_TxSapWriteOut);

Purpose:

This function is used to write a pure message to a SAP type port. The port must be an SAP type Transmit port.

This function can be used if the transmitter is running or not running.

Input:

TY_FDX_SAP_WRITE_IN *px_TxSapWriteIn

```
typedef struct {
   AiUInt32 ul_UdpHandle;
   AiUInt32 ul_UdpDst;
   AiUInt32 ul_IpDst;
   AiUInt32 ul_ByteCount;
   void     *pv_Data;
} TY_FDX_SAP_WRITE_IN;
```

AiUInt32 ul_UdpHandle

See description of FdxCmdTxSAPCreatePort.

AiUInt32 ul_UdpDst

The UDP destination port for the message.

AiUInt32 ul_IpDst

The IP destination of the message

AiUInt32 ul_ByteCount

Number of bytes to write to this SAP port. The value must be equal or smaller than ul_MaxMessageSize defined with FdxCmdTxSAPCreatePort().

void *pv_Data

Pointer to a buffer containing the data to write.

Output:

TY_FDX_SAP_WRITE_OUT *px_TxSapWriteOut

```
typedef struct {
   AiUInt32 ul_BytesWritten;
} TY_FDX_SAP_WRITE_OUT;
```

AiUInt32 ul_BytesWritten

Number of bytes actually written. Might be smaller than ul_ByteCount if SAP buffer is full. (ul_NumBufMessages defined with FdxCmdTxSAPCreatePort)

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.3.6 FdxCmdTxUDPBlockWrite

Prototype:

AiReturn FdxCmdTxUDPBlockWrite (const AiUInt32 ul_Handle,
const TY_FDX_UDP_BLOCK_WRITE_IN
****px_UdpBlockWriteIn,***
TY_FDX_UDP_BLOCK_WRITE_OUT
****px_UdpBlockWriteOut);***

Purpose:

This function is used to write a pure message to one or more UDP ports. The ports can be a sampling or a queuing port AFDX Communication port. If the data size is not applicable for the data size associated to this port, this function will return an error.

For sampling ports this function initializes / modifies data contents.

For queuing ports a transmission is initiated when data is written to a UDP port if the transmitter is running.

This function can be used if the transmitter is running or not running.

For sampling ports this function should be called before the port is started to initialize the data contents of the sampling port.

Input:

TY_FDX_UDP_BLOCK_WRITE_IN
****px_UdpBlockWriteIn***

```
typedef struct {
   AiUInt32 ul_MsgCount;
   TY_FDX_UDP_BLOCK_WRITE_IN_MSG* px_UdpBlockWriteMsgArray;
} TY_FDX_UDP_BLOCK_WRITE_IN;
```

AiUInt32 ul_MsgCount

Specifies the number of messages to be written.

TY_FDX_UDP_BLOCK_WRITE_IN_MSG
****px_UdpBlockWriteMsgArray***

Pointer to an array structures. Each structure describes the message to be written to a single UDP transmission port. The array contains ul_MsgCount elements.

```
typedef struct {
   AiUInt32    ul_UdpHandle;
   AiUInt32    ul_ByteCount;
   void       *pv_Data;
} TY_FDX_UDP_BLOCK_WRITE_IN_MSG;
```

AiUInt32 ul_UdpHandle

The handle of the UDP port to which the message shall be written. This may be a handle to either a Queuing or Sampling UDP port.

AiUInt32 ul_ByteCount

Number of bytes to write to this UDP port. The value must be equal or smaller than ul_MaxMessageSize defined with FdxCmdTxUDPCreatePort().

Port Type	*Comment*
Sampling	The value does not influence transmitted frame size. When the number is smaller than ul_MaxMessageSize only first part of UDP buffer is updated.
Queuing	The value is equivalent to transmitted UDP message size.

void *pv_Data

Pointer to a buffer containing the data to write.

Output:

TY_FDX_UDP_BLOCK_WRITE_OUT *px_UdpBlockWriteOut

```
typedef struct {
   AiReturn st_GlobalResultCode;
   AiUInt32 ul_MsgCount;
   TY_FDX_UDP_BLOCK_WRITE_OUT_RESULT* px_UdpBlockWriteResultArray;
} TY_FDX_UDP_BLOCK_WRITE_OUT;
```

AiReturn st_GlobalResultCode

Specifies the overall result of the block write operation.

Value	Description
FDX_OK	The block operation completed successfully. All messages were successfully written to the respective UDP ports.
FDX_ERR	At least one of the individual writes to the UDP ports has failed. The st_ResultCode entries in the output array should be checked for identification of which message(s) have failed.

TY_FDX_UDP_BLOCK_WRITE_OUT_RESULT *px_UdpBlockWriteOut

Pointer to an array of structures. Each structure specifies the result of an individual UDP write operation. The array contains ul_MsgCount elements.

```
typedef struct {
   AiUInt32    ul_UdpHandle;
   AiUInt32    ul_BytesWritten;
   AiReturn    st_ResultCode;
} TY_FDX_UDP_BLOCK_WRITE_OUT;
```

AiUInt32 ul_UdpHandle

The handle of the associated UDP port. This may be a handle to either a Sampling or Queuing port.

AiUInt32 ul_BytesWritten

Number of bytes actually written. Might be smaller than ul_ByteCount if UDP buffer is full. (Queuing ports ul_NumBufMessages defined with FdxCmdTxUDPCreatePort)

AiReturn st_ResultCode

The result of the individual write operation. FDX_OK on success or a negative error code if an error occurs.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.3.7 FdxCmdTxUDPChgSrcPort

Prototype:

AiReturn FdxCmdTxUDPChgSrcPort (AiUInt32 ul_Handle,
AiUInt32 ul_UdpHandle,
AiUInt32 ul_UdpSrc);

Purpose:

This function is used to change source of a defined UDP queuing port. This function can be used if transmitter is running.

Input:

AiUInt32 ul_UdpHandle

The handle of the UDP port to which the message shall be written. This must be a handle to a Queuing UDP port.

AiUInt32 UdpSrc

New source number for UDP port.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.3.8 FdxCmdTxUDPControl

Prototype:

AiReturn FdxCmdTxUDPControl (AiUInt32 ul_Handle,
AiUInt32 ul_UdpHandle,
const TY_FDX_TX_UDP_CONTROL
****px_TxUdpControl);***

Purpose:

This function is used to enable or disable a UDP Port, manage error-injection purposes and modify Network and Skew parameters.

Input:

AiUInt32 ul_UdpHandle

See description of ***FdxCmdTxUDPCreatePort*** and *FdxCmdTxSAPCreatePort*.

TY_FDX_TX_UDP_CONTROL *px_TxUdpControl

Pointer to a setup structure for a Transmit UDP Port

```
typedef struct {
   AiUInt32 ul_EnableTyp;
   AiUInt32 ul_NetSelect;
   AiUInt32 ul_Skew;
   AiUInt32 ul_ErrorInjectionCount;
   AiUInt32 ul ErrorInjectionTyp;
   AiUInt32 ul TxIntEnable;
   AiUInt32 ul Reserved1;
   AiUInt32 ul_Reserved2;

} TY_FDX_TX_UDP_CONTROL;
```

AiUInt32 ul_EnableTyp

Value	*Comment*
FDX_ENA	UDP Port is enabled. All frames defined for this port are transmitted
FDX_DIS	UDP Port is disabled. All frames for the given UDP Port are discarded.

AiUInt32 ul_NetSelect

Value:	*Description:*
FDX_TX_FRAME_DLY_A	Packet on Network A is delayed by the Skew value, related to Network B
FDX_TX_FRAME_DLY_B	Packet on Network B is delayed by the Skew value, related to Network A
FDX_TX_FRAME_BOTH	Packet transmitted on both Networks (Skew=0)
FDX_TX_FRAME_ONLY_A	Packet only transmitted on Network A
FDX_TX_FRAME_ONLY_B	Packet only transmitted on Network B
FDX_TX_FRAME_VL_DEFAULT	Use Net preselected setting of the VL

Note: This function is only provided in redundant port operation mode.

AiUInt32 ul_Skew

This parameter defines the transmission skew between the redundant frames on the AFDX-ports. The skew can be programmed with a resolution of 1us.

Selectable Range is 0..65535us.

A value of 0 uses the preselected setting of the VL.

This parameter is only used if ul_NetSelect is FDX_TX_FRAME_DLY_A or FDX_TX_FRAME_DLY_B.

Note: This function is only provided in redundant port operation mode.

AiUInt32 ul_ErrorInjectionCount

This parameter controls the physical error injection which is selected with ul_ErrorInjectionTyp.

Value	*Comment*
0	Cyclic error injection.
>0	Number of erroneous MAC Frames to be injected on selected UDP Port.
0xFFFF.FFFF	Reserved

AiUInt32 ul_ErrorInjectionTyp

Refer to parameter ul_PhysErrorInjection of function ***FdxCmdTxQueueWrite.***

AiUInt32 ul_TxIntEnable;

Value	*Comment*
0	No Interrupt will be creaed on send.
FDX_ENA	An Interrupt will be created on Frame send.
3	An Interrupt will be created on Frame send, extended with TimeTag.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.3.9 FdxCmdTxUDPCreatePort

Prototype:

AiReturn FdxCmdTxUDPCreatePort (AiUInt32 ul_Handle,
const TY_FDX_UDP_DESCRIPTION *px_UdpDescription,
AiUInt32 *pul_UdpHandle);

Purpose:

This function is used to create a port to transmit data from a specified UDP connection oriented port. Default settings after creation of the UDP port are:

- UDP port enabled
- Error injection: OFF
- Skew (redundant mode only): 0 usec.

To change the settings of the UDP port the function ***FdxCmdTxUDPControl*** can be used. This function can be used only if transmitter is not running.

Input:

TY_FDX_UDP_DESCRIPTION *px_UdpDescription

Pointer to a structure, which describes the UDP connection.

```
typedef struct {
   AiUInt32 ul_PortType;
   TY_FDX_QUINTUPLET x_Quint;
   AiUInt32 ul_SubVlId;
   AiUInt32 ul_UdpNumBufMessages;
   AiUInt32 ul_UdpMaxMessageSize;
   AiUInt32 ul_UdpSamplingRate;
}TY_FDX_UDP_DESCRIPTION

typedef struct _quintuplet {
   AiUInt32 ul_UdpSrc;
   AiUInt32 ul_IpSrc
   AiUInt32 ul_VlId;
   AiUInt32 ul_IpDst;
   AiUInt32 ul_UdpDst;
} TY_FDX_QUINTUPLET;
```

AiUInt32 ul_PortType

Type of the port connection

Value	*Description*
FDX_UDP_SAMPLING	Port is a sampling port. Each Message is represented by one MAC Frame. The size of the messages is fix.
FDX_UDP_QUEUING	Port is a queuing port: Each Message can be represented by one or more MAC Frame. Reassemble will be done in the IP layer. A Message can have a size up to 8kByte.

struct TY_FDX_QUINTUPLET

This structure describes a full identification of the communication.

AiUInt32 x_Quint.ul_UdpSrc

UDP port-number of the source UDP port.

AiUInt32 x_Quint.ul_IpSrc

IP address of the source partition.

AiUInt32 x_Quint.ul_VlId

Virtual Link Identifier

AiUInt32 x_Quint.ul_IpDst

Destination IP address

AiUInt32 x_Quint.ul_UdpDst

Destination UDP port

AiUInt32 ul_SubVlId;

Sub Virtual Link Identifier (Sub VLs are only relevant in Tx Mode). This value must be in a range from 1 to 4. If Sub VLs are not used, the Sub VL Id equals to 1.

Note: Must be consistent with parameter ul_SubVls of function FdxCmdTxCreateVL or FdxCmdTxCreateHiResVL.

AiUInt32 ul_UdpNumBufMessages

Number of messages which can be stored by the onboard Target software.

Size of the local Buffer which should be created by the onboard Target software to store data of the created UDP port can be calculated by ul_UdpNumBufMessages * ul_UdpMaxMessageSize.

New!

For sampling ports the number of messages shall normally be set to 1. But it is also possible for sampling ports to define an buffer input queue.

This queue can internally processed either as FIFO or as a cyclic queue. To decide wich mode shall be used set Bit 31 of ul_UdpNumBufMessages to

'0' for FIFO or

'1 for cyclic.

For queuing ports an adequate buffer depth have to be provided.

If this value is set to zero, the onboard target software will set to a default value.

AiUInt32 ul_UdpMaxMessageSize;

Maximum size of a message to send.

For a sampling port, this is the fix size of the sampling message, which means the size without the header overhead (MAC, IP and UDP).

For a queuing port this is the maximum size of the variable length message.

Port Type	***Value***
Sampling	Must be equal or smaller than ul_MaxFrameLength – (MAC, IP and UDP) defined in function ***FdxCmdTxCreateVL*** or ***FdxCmdTxCreateHiResVL.***
Queuing	Must be equal or smaller than 8Kbytes (max. UDP message size)

AiUInt32 ul_UdpSamplingRate;

Specifies the sampling rate for this frame on the AFDX bus in milliseconds. This means a repetition rate the data shall be sent cyclic. Specify value starting with 1msec in multiples of 1msec. Refer also to parameter ul_Bag of function ***FdxCmdTxCreateVL*** or ***FdxCmdTxCreateHiResVL*** which restrict usage of sampling rate.

Note: This parameter is only applicable for a sampling port.

Note: Maximum number of different sampling rates for each physical port is 512.

Example:

VL-Load:

$$\text{VL-Bag} * \left(\frac{1}{\text{Sampling Rate UDP Port 1}} + \frac{1}{\text{Sampling Rate UDP Port 2}} + \ldots\right) \leq 1$$

If above calculated value is bigger than 1 VL is overloaded and sampling rates can not be achieved.

Queuing ports are additional load to the VL and are only transmitted if bandwidth is not fully exhausted by sampling ports.

Output:

AiUInt32 *pul_UdpHandle

Handle to the UDP port.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.3.10 **FdxCmdTxUDPDestroyPort**

Prototype:

AiReturn FdxCmdTxUDPDestroyPort (AiUInt32 ul_Handle,
const AiUInt32 ul_UdpHandle);

Purpose:

This function is used to destroy the UDP transmit port, if it is no longer in use. (This function is the opposite of function to *FdxCmdTxUDPCreatePort* and *FdxCmdTxSAPCreatePort.)* This function can be used only when transmitter is not running.

Input:

AiUInt32 ul_UdpHandle

See description of *FdxCmdTxUDPCreatePort* and *FdxCmdTxSAPCreatePort*.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.3.11 FdxCmdTxUDPWrite

Prototype:

```
AiReturn FdxCmdTxUDPWrite (  AiUInt32 ul_Handle,
                             const AiUInt32 ul_UdpHandle
                             const AiUInt32 ul_ByteCount
                             const void *pv_Data
                             AiUInt32 *pul_BytesWritten);
```

Purpose:

This function is used to write a pure message to a UDP port. This port can be a sampling or a queuing port. If the data size is not applicable for the data size associated to this port, this function will return an error.

For sampling ports this function initializes / modifies data contents.

For queuing ports a transmission is initiated when data is written to a UDP port.

This function can be used if the transmitter is running or not running.

For sampling ports this function should be called before the port is started to initialize the data contents of the sampling port.

Input:

AiUInt32 ul_UdpHandle

See description of *FdxCmdTxUDPCreatePort*.

AiUInt32 ul_ByteCount

Number of bytes to write to this UDP port. The value must be equal to or smaller than ul_MaxMessageSize defined with ***FdxCmdTxUDPCreatePort()***.

Port Type	*Comment*
Sampling	The value does not influence transmitted frame size. When the number is smaller than ul_MaxMessageSize only the first part of the UDP buffer is updated.
Queuing	The value is equivalent to transmitted UDP message size.

void *pv_Data

Pointer to a buffer containing the data to write.

Output:

AiUInt32 *pul_BytesWritten

Number of bytes actually written. Might be smaller than ul_ByteCount if UDP buffer is full. (Queuing ports ul_NumBufMessages defined with FdxCmdTxUDPCreatePort)

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.3.12 FdxCmdTxUDPGetStatus

Prototype:

AiReturn FdxCmdTxUDPGetStatus (AiUInt32 ul_Handle,
const AiUInt32 ul_UdpHandle,
TY_FDX_TX_UDP_STATUS *px_UdpTxStatus);

Purpose:

This function is used to retrieve the status of the specified UDP transmit port.

Input:

AiUInt32 ul_UdpHandle

See description of *FdxCmdTxUDPCreatePort*.

Output:

TY_FDX_TX_UDP_STATUS *px_UdpTxStatus

```
typedef struct {
   AiUInt32 ul_MsgCount;
   AiUInt32 ul_CurrentIndex;
   AiUInt32 ul_IndexCycleCount;
}TY_FDX_TX_UDP_STATUS;
```

AiUInt32 ul_MsgCount;

Counter of messages sent through this UDP port since transmitter was started.

AiUInt32 ul_CurrentIndex;

New!

This parameter is only valid for Udp Sampling Ports which have assigned an input queue.

The parameter shows the index in the indexed input queue which will be written next to the firmware queue. The buffer before ul_CurrentIndex is already sent.

AiUInt32 ul_IndexCycleCount t;

New!

This parameter is only valid for Udp Sampling Ports which have assigned an input queue.

The parameter counts the cycles of processing for the input queue. Counting by switching the index back from last index to 0.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.3.13 FdxCmdTxUDPWriteIndexed

Prototype:

AiReturn FdxCmdTxUDPWriteIndexed (const AiUInt32 ul_Handle, const TY_FDX_UDP_INDEXED_WRITE_IN * px_UdpWriteIndexedIn, TY_FDX_UDP_INDEXED_WRITE_OUT * px_UdpWriteIndexedOut);

Purpose:

This function is used to write a pure message to a defined queue of an UDP Sampling port which has assotiated a input Queue. This message can be addressed by index

Input:

TY_FDX_UDP_INDEXED_WRITE_IN *px_UdpWriteIndexedIn

Pointer to an input structure for information about buffers, shall be written by index

```
typedef struct {
   AiUInt32 ul_UdpHandle;
   AiUInt32 ul_MsgCount;
   TY_FDX_UDP_INDEXED_WRITE_IN_MSG* px_UdpIndexedWriteMsgArray;
} TY_FDX_UDP_INDEXED_WRITE_IN;
```

AiUInt32 ul_UdpHandle;

Handle to an UPD Port where the following data shall be written. See description of *FdxCmdTxUDPCreatePort*.

AiUInt32 ul_MsgCount;

Count of messages which follows described by the following structure.

TY_FDX_UDP_INDEXED_WRITE_IN_MSG* px_UdpIndexedWriteMsgArray;

Start Pointer to an array of messages description blocks described by the following structure. This array must be provided by user in a length which is calculated by sizeof (TY_FDX_UDP_INDEXED_WRITE_IN_MSG) * ul_MsgCount.

```
typedef struct {
   AiUInt32 ul_Index;
   AiUInt32 ul_ByteCount;
   void     *pv_Data;
} TY_FDX_UDP_INDEXED_WRITE_IN_MSG;
```

AiUInt32 ul_Index

Index to the buffer structure of the UDP Sampling port where data shall be written. The Index must be in a range of 0 to ul_UdpNumBufMessages -1 of the dedicated Udp Sampling port.

AiUInt32 ul_ByteCount

Number of bytes which shall be written to the buffer of the UDP port

void *pv_Data

Pointer to a data buffer, where the input data is stored. Here in the structer is only a pointer to the data. The array must be provided by user.

Output:

TY_FDX_UDP_INDEXED_WRITE_OUT *px_UdpWriteIndexedOut

Pointer to an output structure which gives information about success of write for each buffer, sorted by index.

```
typedef struct {
   AiReturn st_GlobalResultCode;
   AiUInt32 ul_MsgCount;
   TY_FDX_UDP_INDEXED_WRITE_OUT_RESULT* px_UdpIndexedWriteResultArray;
} TY_FDX_UDP_INDEXED_WRITE_OUT;
```

AiUInt32 st_GlobalResultCode;

Global Result over all write actions of all indexed buffers. All results from writing data to a buffer are accumulated here. If this value is FDX_OK, all following results are also FDX_OK

AiUInt32 ul_MsgCount;

Count of acknowledge messages which follows described by the following structure.

TY_FDX_UDP_INDEXED_WRITE_OUT_RESULT* px_UdpIndexedWriteResultArray;

Start Pointer to an array of acknowledge messages description blocks described by the following structure. This array must be provided by user in a length which is calculated by sizeof (TY_FDX_UDP_INDEXED_WRITE_OUT_RESULT) * ul_MsgCount.

```
typedef struct {
   AiUInt32 ul_Index;
   AiReturn st_ResultCode;
} TY_FDX_UDP_INDEXED_WRITE_OUT_RESULT;
```

AiUInt32 ul_Index

Index to the buffer structure of the UDP Sampling port where the result comes from.

AiUInt32 st_ResultCode

Result code for the Buffer described by ul_Index. All these values are accumulated in st_GlobalResultCode

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.3.14 FdxCmdTxVLWrite

Prototype:

AiReturn FdxCmdTxVLWrite (AiUInt32 ul_Handle,
const TY_FDX_TX_VL_WRITE_IN *px_TxVLWriteIn,
TY_FDX_TX_VL_WRITE_OUT *px_TxVLWriteOut);

Purpose:

This function is used to write data directly to a virtual link buffer. The Virtual link has to be defined using function ***FdxCmdTxCreateVL*** or ***FdxCmdTxCreateHiResVL***.

Input:

TY_FDX_TX_VL_WRITE_IN *px_TxVLWriteIn

Pointer to a setup structure for a Virtual Link

```
typedef struct {
   AiUInt32 ul_VlId;
   AiUInt32 ul_SubVlId;
   AiUInt32 ul_ByteCount;
   const void *pv_Data;
} TY_FDX_TX_VL_WRITE_IN;
```

AiUInt32 ul_VLId

Virtual Link Identifier. A value in a range from 0 to 65535.

AiUInt32 ul_SubVlId;

Sub Virtual Link Identifier (Sub VLs are only relevant in Tx Mode). This value must be in a range from 1 to 4. If Sub VLs are not used, the Sub VL Id equals to 1.

AiUInt32 ul_ByteCount

Number of bytes which shall be written to this VL.

void *pv_Data

Pointer to a buffer containing the data to write. The size of this buffer should correspond to *ul_ByteCount*.

Output:

TY_FDX_TX_VL_WRITE_OUT *px_TxVLWriteOut

Pointer to a structure containing information about data written to specified Virtual Link

```
typedef struct {
   AiUInt32 ul_BytesWritten;
} TY_FDX_TX_VL_WRITE_OUT;
```

AiUInt32 ul_BytesWritten

Number of bytes actually written. Might be smaller than ul_ByteCount.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

Return Value:	ul_BytesWritten	Description:
FDX_OK	ul_ByteCount	Frame is sent to BIU. **Note:** Even if ul_ByteCount of the frame exceeds VL associated MaxFrameLength which was defined with FdxCmdTxCreateVL the frame is sent to BIU if the data buffer(*) size is big enough.
FDX_OK	0	Data buffer(*) in BIU was full, not able to take this frame
FDX_ERR	0	Vl is not initialized, or does not exist
FDX_ERR	0	SubVl not initialized

(*) Data buffer size is defined with function ***FdxCmdTxCreateVL***, parameter ***ul_FrameBufferSize.***

4.3.3.15 FdxCmdTxVLWriteEx

Prototype:

AiReturn FdxCmdTxVLWriteEx (AiUInt32 ul_Handle,
const TY_FDX_TX_VL_WRITE_IN_EX *px_TxVLWriteInEx,
TY_FDX_TX_VL_WRITE_OUT_EX *px_TxVLWriteOutEx);

Purpose:

This function is used to write data directly to a virtual link buffer. The Virtual link has to be defined using function ***FdxCmdTxCreateVL*** or ***FdxCmdTxCreateHiResVL***. There are extended frame control possibilities compared to function ***FdxCmdTxVLWrite***.

Input:

TY_FDX_TX_VL_WRITE_IN_EX *px_TxVLWriteInEx

Pointer to a setup structure for a Virtual Link

```
typedef struct {
   AiUInt32 ul_FrameCount;
   TY_FDX_TX_VL_WRITE_FRAME_IN* px_TxVLWriteFrameArray;
} TY_FDX_TX_VL_WRITE_IN_EX;
```

AiUInt32 ul_FrameCount

Number of frames to write.

TY_FDX_TX_VL_WRITE_FRAME_IN* px_TxVLWriteFrameArray

```
typedef struct {
   TY_FDX_TX_VL_WRITE_FRAME_INFO*   x_FrameInfo;
   AiIInt8 *pv_Data;
} TY_FDX_TX_VL_WRITE_FRAME_IN;
```

TY_FDX_TX_VL_WRITE_FRAME_INFO x_FrameInfo

```
typedef struct {
   AiUInt32 ul_VlId;
   AiUInt32 ul_SubVlId;
   AiUInt32 ul_FrameSize;
   AiUInt32 ul_InterFrameGap;
   AiUInt32 ul_Skew;
   AiUInt32 ul_PhysErrorInjection;
   AiUInt32 ul_ExternalStrobe;
   AiUInt32 ul_PreambleCount;
   AiUInt32 ul_NetSelect;
   AiUInt32 ul_InterruptControl;
   AiUInt32 ul_SequenceNumberControl;
   AiUInt32 ul_Reserved;
} TY_FDX_TX_VL_WRITE_FAME_INFO;
```

AiUInt32 ul_VLId

Virtual Link Identifier. A value in a range from 0 to 65535.

AiUInt32 ul_SubVlId;

Sub Virtual Link Identifier (Sub VLs are only relevant in Tx Mode). This value must be in a range from 1 to 4. If Sub VLs are not used, the Sub VL Id equals to 1.

AiUInt32 ul_FrameSize

Number of bytes which shall be written to this VL.

AiUInt32 ul_InterFrameGap

This value defines the interframe gap between the preceding frame and the current frame with a resolution of 40ns, measured from the end of the last bit fo the preceding frame to the first preamble bit of the actual frame.

To implement a physical gap between the frames, a minimum interframe gap of 120 ns (value = 3) shall be initialized. The maximum provided interframe gap will be up to approx. 655us (14 Bits are used for encoding). If the Packet group Wait Time is used, this field shall be initialized with zero. This Gap is only used if uc_FrameStartMode is set to FDX_TX_START_FRAME_IFG.

See also the notes for ul_Skew parameter in redundant mode.

AiUInt32 ul_Skew

This parameter defines the transmission skew between the redundant frames on the AFDX-ports. The skew can be programmed with a resolution of 1us. Range is 0...65535. This parameter is only used if ul_NetSelect is FDX_TX_FRAME_DLY_A or FDX_TX_FRAME_DLY_B.

Note:	***This function is only provided in redundant port operation mode.***

Note:	***If the ul_Skew parameter is set and one redundant frame is delayed this time may be added to ul_InterFrameGap and may exceed maximum value of ul_InterFrameGap in the receiver.This means it can result in a higher Interframe Gap Time because the IFG counter for transmit is sterted synchroniously for both networks after both redundant frames are sent..***

AiUInt32 ul_PhysErrorInjection

This parameter defines physical error injection types. The error injection information can be a combination of the following error types:

Value:	***Description:***
FDX_TX_FRAME_ERR_OFF	No Error Injection enabled
FDX_TX_FRAME_ERR_CRC	CRC Error transmitted with this frame
FDX_TX_FRAME_ERR_ALI	Wrong Byte alignment in transmit frame, which means that an odd number of nibbles will be transmitted. Therefore, this error will also cause a CRC error condition *Note: This Error can not be injected in 1 Gbit/s mode.*
FDX_TX_FRAME_ERR_PRE	Wrong Preamble Sequence transmitted. If this type is selected., the Encoder device substitutes the first nibble of the Start Frame Delimiter with the value '1000' instead of '1001'
FDX_TX_FRAME_ERR_PHY	Physical Symbol Error. During Frame Transmission, the MAC-Encoder device asserts the Tx-Error signal, which forces the physical transceiver to transmit 'HALT' symbols.

AiUInt32 ul_ExternalStrobe

Control assertion of Trigger Strobe if this frame is transmitted. See the FdxCmdTxTrgLineCtrl for further information about the Trigger Lines.

Value:	***Description:***
FDX_DIS	Disable Trigger Strobe
FDX_ENA	Assert external Trigger Strobe on transmission of this frame

AiUInt32 ul_PreambleCount

This value defines the number of preamble Bytes sent for this frame

Value:	*Description:*
FDX_TX_FRAME_PRE_DEF	Send default preamble count of 7 Bytes
All other values from n=1..15	Send n preamble Bytes

AiUInt32 ul_NetSelect

This parameter defines the physical Interface_ID of the MAC, which shall send it's current frame delayed (with ul_Skew µs) to the alternate port.

Value:	*Description:*
FDX_TX_FRAME_DLY_A	Packet on Network A is delayed by the Skew value, related to Network B
FDX_TX_FRAME_DLY_B	Packet on Network B is delayed by the Skew value, related to Network A
FDX_TX_FRAME_BOTH	Packet transmitted on both Networks (Skew=0)
FDX_TX_FRAME_ONLY_A	Packet only transmitted on Network A
FDX_TX_FRAME_ONLY_B	Packet only transmitted on Network B

Note: This function is only provided in redundant port operation mode.

AiUInt32 _InterruptControl

Enable/Disable Interrupt on execution of Instruction

Value:	*Description:*
FDX_DIS	Disable Interrupt on execution of Instruction
FDX_ENA	Enable Interrupt on execution of Instruction. Interrupt is asserted after data packet was processed.

AiUInt32 ul_SequenceNumberControl

This parameter controls the handling of the Sequence Number for the transmit packets.

Value:	*Description:*
FDX_TX_FRAME_SN_DEF	Default. The Sequence Number incrementation is controlled by the VL Descriptor. The value of the Sequence Number is incremented by 1 compared to the previous transmitted packet.
FDX_TX_FRAME_SN_INC2	The value of the Sequence Number is incremented by 2 compared to the previous transmitted packet.
FDX_TX_FRAME_SN_INC3	The value of the Sequence Number is incremented by 3 compared to the previous transmitted packet.
FDX_TX_FRAME_SN_NO	The value of the Sequence Number of the MDSN (Mode Dependent Sequence Number) is inserted into the transmitted packet.

AiUInt32 ul_Reserved

MDSN.

Sequence Number of frame for ul_SequenceNumberControl mode FDX_TX_FAME_SN_NO.

AiUInt8 *pv_Data

Pointer to a buffer containing the data to write. The size of this buffer should correspond to *ul_FrameSize*.

Output:

*TY_FDX_TX_VL_WRITE_OUT_EX *px_TxVLWriteOutEx*

Pointer to a structure containing information about data written to specified Virtual Link

```
typedef struct {
   AiUInt32 ul_FramesWritten;
   TY_FDX_TX_VL_WRITE_OUT_FAME_INFO *px_TxVLWriteFrameInfoArray;
} TY_FDX_TX_VL_WRITE_OUT;
```

AiUInt32 ul_FramesWritten

Number of frames actually written. Might be smaller than ul_FrameCount.

```
typedef struct {
   AiUInt32 ul_Status;
} TY_FDX_TX_VL_WRITE_OUT_FRAME_INFO;
```

AiUInt32 ul_Status

Status of write operation.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

Return Value:	*ul_FramesWritten*	*Description:*
FDX_OK	ul_FrameCount	Frame is sent to BIU.
FDX_OK	< ul_FrameCount	Not all frames written to BIU
FDX_ERR	0	Vl is not initialized, or does not exist
FDX_ERR	0	SubVl not initialized

(*) Data buffer size is defined with function ***FdxCmdTxCreateVL*** or ***FdxCmdTxCreateHiResVL***, parameter ***ul_FrameBufferSize.***

4.3.4 Data Buffer Functions

4.3.4.1 FdxCmdTxBufferQueueAlloc

Prototype:

AiReturn FdxCmdTxBufferQueueAlloc (AiUInt32 ul_Handle,
TY_FDX_TX_BUF_QUEUE_DESC *px_TxBufferQueueDesc,
TY_FDX_FW_BUF_HDL *px_TxBufferQueueHandle,
TY_FDX_TX_BUF_QUEUE_INFO *px_TxBufferQueueInfo)

Purpose:

This function allocates a Transmit Buffer Queue in the BIU associated memory (Global Ram) to use it for commands which need a special associated buffer queue (e.g. *FdxCmdTxQueueWrite* for special Payload Buffer Modes). The Buffer Queue provides administration and handling of a single or multiple Payload Buffers, which are organised in a wrap around manner by the queue.

Input:

TY_FDX_TX_BUF_QUEUE_DESC *px_TxBufferQueueDesc

Structure which describes the Buffer Queue Parameter

```
typedef struct {
   AiUInt32 ul_MaxTransfers;
   AiUInt32 ul_BuffersInQueue;
   AiUInt32 ul_BufferSize;
   AiUInt32 ul_BufferQueueMode;
   AiUInt32 ul_BufferIndex;
   AiUInt32 ul_BufferPayloadMode;
      } TY_FDX_TX_BUF_QUEUE_DESC;
```

AiUInt32 ul_MaxTransfers

This parameter describes the maximum number of Transfers which share this Buffer Queue.

It is importand, that all transfers sharing this Buffer Queue are set up with the same Payload Buffer Mode.

AiUInt32 ul_BuffersInQueue

This parameter describes the number of Buffers in the Queue.

The size of each Buffer is given by the ul_BufferSize parameter. Therefore the total size of allocated Buffer Memory is equal to ul_BufferSize * ul_BuffersInQueue plus an internal overhead. The maximum number of buffers in a queue is 128. A value of zero is invalid for this parameter. This number schould be given in power of 2 value. If not it will be aligned to the next possible value (1, 2, 4, 8 … … 128).

AiUInt32 ul_BufferSize

This parameter describes the size in Bytes for one Buffer in the Buffer Queue. The maximum size for a buffer can be up to 2044 Bytes. Minimum size is 64. The value can be given here as a Byte value, but it will be aligned to internally 64 Byte.

AiUInt32 ul_BufferQueueMode

This parameter defines the Buffer Queue Mode for the Queue.

Value	*Description*
FDX_TX_BUF_QUEUE_CYC	The Queue works is cyclic mode. This means, each time the associated frame is transmitted, the internal buffer index is incremented, for using the next buffer in the queue for the next transmission of the frame. If end of Buffer Queue is reached, a wrap around is performed.
FDX_TX_BUF_QUEUE_SNG	The Queue works in single mode. This means, each time the associated frame is transmitted, the internal buffer index is incremented, for using the next buffer in the queue for the next transmission of the frame. If end of Buffer Queue is reached, a wrap around is performed and the last Buffer in the queue will be used for all following frames.
FDX_TX_BUF_QUEUE_HOST	The Queue works in host controlled mode. This means, the Buffer from the current Internal Buffer Index (after allocation, this is Index 0) will be used for transmission until it is changed via the host by using the *FdxCmdTxBufferQueueCtrl* command.

The Buffer Queue Mode can be changed during operation by using the *FdxCmdTxBufferQueueCtrl* command.

AiUInt32 ul_BufferIndex

This parameter describes initial Buffer Index of the Queue. Therefore the value must be in the range between 0 (first buffer in queue) and the "*ul_BuffersInQueue* – 1" , given at this function.

AiUInt32 ul_BufferPayloadMode;

It is importand, that all transfers sharing this Buffer Queue are set up with the same Payload Buffer Mode so assingne the Buffer Queue with the corresponding Mode to check within the Transfer setup the correct mode.

See the following table for the possible Values

Value:	*Description:*
FDX_TX_FRAME_PBM_MAC	MAC- Payload is provided in the separate Buffer Queue. That means, the BIU- Processor fetches the Frame Header Words and the first 16 bytes of the frame data out from this entry and switches then to the separate buffer queue. Thus, the complete MAC- Header and the two static bytes of the IP- Header are used from this entry and the rest of the frame payload is used from the separate buffer queue.
FDX_TX_FRAME_PBM_UDP	UDP- Payload is provided in the separate Buffer Queue. That means, the BIU- Processor fetches the Frame Header Words and the first 40 bytes of the frame data out from this entry and switches then to the separate buffer queue. Thus, the complete MAC- Header, the IP- Header and 6 bytes of the UDP- Header are used from this entry and the remainder of the frame payload is used from the separate buffer queue. That means, the 2 bytes of the UDP-Checksum (always zero) and the UDP- payload resides in the separate buffer.
FDX_TX_FRAME_PBM_FULL	The full MAC-Frame is provided in the separate Buffer Queue. That means, the BIU- Processor fetches the Frame Header Words out from this entry and switches then to the separate buffer queue.

Output:

*TY_FDX_FW_BUF_HDL *px_TxBufferQueueHandle*

```
typedef struct {
   AiUInt32 ul_Handle;
} TY_FDX_FW_BUF_HDL;
```

AiUInt32 ul_Handle

Handle to this buffer queue. Must be used with all following commands which want to use this buffer queue.

*TY_FDX_TX_BUF_QUEUE_INFO *px_TxBufferQueueInfo*

```
typedef struct {
   AiUInt32 ul_BuffersInQueue;
   AiUInt32 ul_BufferSize;
   AiAddr pv_BufferQueueStart;
} TY_FDX_TX_BUF_QUEUE_INFO;
```

AiUInt32 ul_BuffersInQueue

Effectively number of buffers allocated for the Queue. This can differ from the requested number of buffers if that was not conform to the rules of Buffer Queues

AiUInt32 ul_BufferSize

Effectively number of bytes allocated for one buffers in the Queue. This can differ from the requested number of bytes if that was not conform to the rules of Buffer Queues

AiAddr pv_BufferQueueStart

Reserved for internal use.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.4.2 FdxCmdTxBufferQueueFree

Prototype:

AiReturn FdxCmdTxBufferQueueFree (AiUInt32 ul_Handle,
TY_FDX_FW_BUF_HDL x_TxBufferQueueHandle);

Purpose:

This function frees a Transmit Buffer Queue in the BIU associated memory (Global Ram).

Input:

TY_FDX_FW_BUF_HDL x_TxBufferQueueHandle

Handle to the buffer queue which shall be freed. After the Handle has been freed it should not be used for further function calls. This may cause unpredictable results.

(See description of *FdxCmdTxBufferQueueAlloc*).

Output:

None

Return Value

Returns FssDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.4.3 FdxCmdTxBufferQueueRead

Prototype:

AiReturn FdxCmdTxBufferQueueRead (AiUInt32 ul_Handle,
TY_FDX_FW_BUF_HDL x_TxBufferQueueHandle,
AiUInt32 ul_StartIndex,
AiUInt32 ul_StartByte,
AiUInt32 ul_BytesToRead,
void *pv_Data,
AiUInt32 *pul_BytesRead);

Purpose:

This function reads buffer contents from a Transmit Buffer Queue in the BIU associated memory (Global Ram).

Input:

TY_FDX_FW_BUF_HDL x_TxBufferQueueHandle

Handle to the buffer queue, this command is applied to.

(See description of *FdxCmdTxBufferQueueAlloc*).

AiUInt32 ul_StartIndex

Start Index of the buffer queue in BIU associated memory which describes the start location to read data from. This value must not exceed the maximum number of buffers in the queue. Therefore the value must be in the range between 0 (first buffer in queue) and the " *ul_BuffersInQueue* – 1" parameter , given at the *FdxCmdTxBufferQueueAlloc* function. A value of FDX_TX_BUF_QUEUE_ACT reads the data from the current Buffer Index.

AiUInt32 ul_StartByte

Offset to the first byte to read inside the selected buffer. Using this option you have the possibility to read only a part of a message.

AiUInt32 ul_BytesToRead

Number of bytes which shall be read from the BIU associated memory. Since the Buffer Queue allocates multiple buffers continuously, it is possible to read more than one buffer of the queue with one call, by setting this parameter to the corresponding number of bytes.

Output:

void *pv_Data

Pointer to a data buffer to write the read the data to. This pointer must point to a host allocated memory location, big enough to store *ul_BytesToRead* Bytes.

AiUInt32 *pul_BytesRead

Number of bytes, definitely read form the BIU associated memory.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.4.4 **FdxCmdTxBufferQueueWrite**

Prototype:

AiReturn FdxCmdTxBufferQueueWrite (AiUInt32 ul_Handle,
TY_FDX_FW_BUF_HDL x_TxBufferQueueHandle,
AiUInt32 ul_StartIndex,
AiUInt32 ul_StartByte,
AiUInt32 ul_BytesToWrite,
void *pv_Data,
AiUInt32 *pul_BytesWritten);

Purpose:

This function writes data to a Buffer Queue in the BIU associated memory (Global Ram).

A write to a buffer of a queue which is used by an active transmitter may produce inconsistent data for one transfer.

Input:

TY_FDX_FW_BUF_HDL x_BufferHandle

Handle to the buffer queue, this command is applied to.

(See description of *FdxCmdTxBufferQueueAlloc*).

AiUInt32 ul_StartIndex

Start Index of the buffer queue in BIU associated memory which describes the start location to write data to. This value must not exceed the maximum number of buffers in the queue. Therefore the value must be in the range between 0 (first buffer in queue) and the " *ul_BuffersInQueue* – 1" parameter , given at the *FdxCmdTxBufferQueueAlloc* function. A value of FDX_TX_BUF_QUEUE_ACT write the data beginning with the current Buffer Index.

AiUInt32 ul_StartByte

Offset to the first byte to write inside the selected buffer. Using this option you have the possibility to write only a part of a message.

AiUInt32 ul_BytesToWrite

Number of bytes which shall be written to the BIU associated memory. Since the Buffer Queue allocates multiple buffers continuously, it is possible to write more than one buffer of the queue with one call, by setting this parameter to the corresponding number of bytes.

Output:

*void *pv_Data*

Pointer to a data buffer where the data to write is provided. This pointer must point to a host allocated memory location, providing enough memory to write *ul_BytesToWrite* Bytes from.

*AiUInt32 *pul_BytesWritten*

Number of bytes, definitely written to the BIU associated memory.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.4.5 FdxCmdTxBufferQueueCtrl

Prototype:

AiReturn FdxCmdTxBufferQueueCtrl (AiUInt32 ul_Handle,
TY_FDX_FW_BUF_HDL x_TxBufferQueueHandle,
TY_FDX_TX_BUF_QUEUE_CTRL
****px_TxBufferQueueCtrl,***
TY_FDX_TX_BUF_QUEUE_DESC
****px_TxBufferQueueDesc);***

Purpose:

This function controls a Transmit Buffer Queue Attributes. It is intended to provide the user necessary control over the Buffer Queue function in order to change the Queue Mode or the current Index of the Queue or both. It can furthermore used to retrieve Information about the current Queue Parameter.

Input:

TY_FDX_FW_BUF_HDL x_BufferHandle

Handle to the buffer queue, this command is applied to.

(See description of *FdxCmdTxBufferQueueAlloc*).

TY_FDX_TX_BUF_QUEUE_CTRL
****px_TxBufferQueueCtrl***

Structure which describes the Buffer Queue Control Parameter

```
typedef struct {
   AiUInt32 ul_BufferQueueMode;
   AiUInt32 ul_BufferQueueIndex;
} TY_FDX_TX_BUF_QUEUE_DESC;
```

AiUInt32 ul_BufferQueueMode

This parameter defines the Buffer Queue Mode for the Queue, which can be changed

Value	*Description*
FDX_TX_BUF_QUEUE_CYC	See FdxCmdTxBufferQueueAlloc function
FDX_TX_BUF_QUEUE_SNG	See FdxCmdTxBufferQueueAlloc function
FDX_TX_BUF_QUEUE_HOST	See FdxCmdTxBufferQueueAlloc function
FDX_TX_BUF_QUEUE_KEEP	Buffer Queue Mode is not changed

AiUInt32 ul_BufferIndex

This parameter allows to change the Internal Buffer Index in order to force use of another buffer of the buffer queue with the next associated frame. The value for this Index must be in the range between 0 (first buffer in queue) and the

"*ul_BuffersInQueue* – 1" parameter , given at the *FdxCmdTxBufferQueueAlloc* function. A value of FDX_TX_BUF_QUEUE_KEEP will not modify the current internal Buffer Index.

Output:

TY_FDX_TX_BUF_QUEUE_DESC *px_TxBufferQueueDesc

Structure which describes the Buffer Queue Parameter

```
typedef struct {
   AiUInt32 ul_BuffersInQueue;
   AiUInt32 ul_BufferSize;
   AiUInt32 ul_BufferQueueMode;
   AiUInt32 ul_BufferIndex;
} TY_FDX_TX_BUF_QUEUE_DESC;
```

A detailed description of this structure is found at the *FdxCmdTxBufferQueueAlloc* function.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.5 Generic Transmitter Sub-Queue-Functions

4.3.5.1 FdxCmdTxSubQueueCreate

Prototype:

AiReturn FdxCmdTxSubQueueCreate (AiUInt32 ul_Handle,
const TY_FDX_TX_SUB_QUEUE_CREATE_IN
****px_TxSubQueueCreateIn,***
TY_FDX_TX_SUB_QUEUE_CREATE_OUT
****px_TxSubQueueCreateOut);***

Purpose:

This function is used to create a queue of AFDX Frames which can be used as a Sub Queue. The Sub Queue will be called from the main Queue by a call instruction. After transmission of the Sub Queue execution will be return to the next transfer or command in the main queue right after the call to the sub queue.

Input:

Const TY_FDX_TX_SUB_QUEUE_CREATE_IN *px_TxSubQueueCreateIn

```
typedef struct {
   AiUInt32 ul_QueueSize;
} TY_FDX_TX_SUB_QUEUE_CREATE_IN;
```

AiUInt32 ul_QueueSize

Specifies the size of the Queue in Byte. This means the memory which is allocated in BIU associated memory. If this value is set to zero, an internal default queue size will be selected.

In most cases one frame or instruction needs 64 Bytes of memory in the transmit queue. The buffer for real frame data will be allocated in memory by the SubQueueWrite command.

Output:

TY_FDX_TX_SUB_QUEUE_CREATE_OUT *px_TxSubQueueCreateOut

```
typedef struct {
   AiUInt32 ul_SubQueueHandle;
} TY_FDX_TX_SUB_QUEUE_CREATE_OUT;
```

AiUInt32 ul_SubQueueHandle

A returned handle for all further access to this Sub Transmit queue.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.5.2 FdxCmdTxSubQueueDelete

Prototype:

AiReturn FdxCmdTxSubQueueDelete (AiUInt32 ul_Handle,
AiUInt32 ul_SubQueueHandle);

Purpose:

In difference to the main Transmit queue the Sub-Queue must be deleted under control of the user.

By deleting this queue it should be guarantied that the Sub-Queue is not longer in use.

Input:

AiUInt32 ul_SubQueueHandle

Handle to a defined Transmit Sub Queue.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.5.3 FdxCmdTxSubQueueWrite

Prototype:

```
AiReturn FdxCmdTxSubQueueWrite (    AiUInt32 ul_Handle,
                                    AiUInt32 ul_SubQueueHandle,
                                    AiUInt32 ul_EntryCount,
                                    AiUInt32 ul_WriteBytes,
                                    const void *pv_WriteBuffer);
```

Purpose:

This function is used to write one Entry to a Transmit Sub Queue from a provided buffer. For this write Function the number of bytes to write needs to be specified. The entry will always be queued at the end of the transmit sub queue.

Input:

AiUInt32 ul_SubQueueHandle

Handle to a defined Transmit Sub Queue.

AiUInt32 ul_EntryCount

Number of Entries to write.

AiUInt32 ul_WriteBytes

Number of bytes that shall be written to the queue.

void *pv_WriteBuffer

Pointer to the data buffer providing the Entries to write. The size of this buffer should correspond to *ul_WriteBytes*.

One Entry specifies one Frame + Header Information. This means one complete MAC frame plus a fixed sized Header. The Header contains information about the manner in which the frame should be sent on the network.

Layout of one Queue Entry:

	Entry Layout
Fixed Header	Fixed Frame Header Layout dependent on *ul_HeaderType* and *uc_FrameType* parameter (see following description)
AFDX Frame	AFDX- FRAME data to transmit (dependent on the Payload Buffer and Payload Generation mode, see description below) (802.3 defines: 64 to 1518 bytes)

TY_FDX_TX_FRAME_HEADER x_TxFrameHeader

```
typedef struct {
   AiUInt8   uc_FrameType;
   TY_FDX_TX_FRAME_ATTRIB x_FrameAttrib;
   TY_FDX_TX_INSTR_ATTRIB x_InstrAttrib;
} TY_FDX_TX_FRAME_HEADER;
```

Note:	***The FdxInitTxFrameHeader function supports a default initialization of this structure (see this function in the chapter 'Target Independent Administration Functions'***

For detailed description of the x_TxFrameHeader Structure refer to description of command FdxCmdTxQueueWrite.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.3.6 Theorie of Generic Transmitter

On the following page you can find a schematic which shows the layout of memory organisaton in BIU related Global RAM for the generic transmitter part.

Basically each transmit channel has a Transmit Queue which is organized as a contoguous part of memory used as a cyclic queue. This memory can contain Frames to transmit and also Instructions. Instructions are commands like described in the Reference Manual to control transmission in several ways. Transmit Frames can be provided either directly in the Transmit Queue, which means the memory of the Transmit queue is used to store the Transmit Frame Header followed by the Frame Data for transmission. Or the Frame data can be provided in a Buffer Queue which is controlled by some entries in the Frame Header.

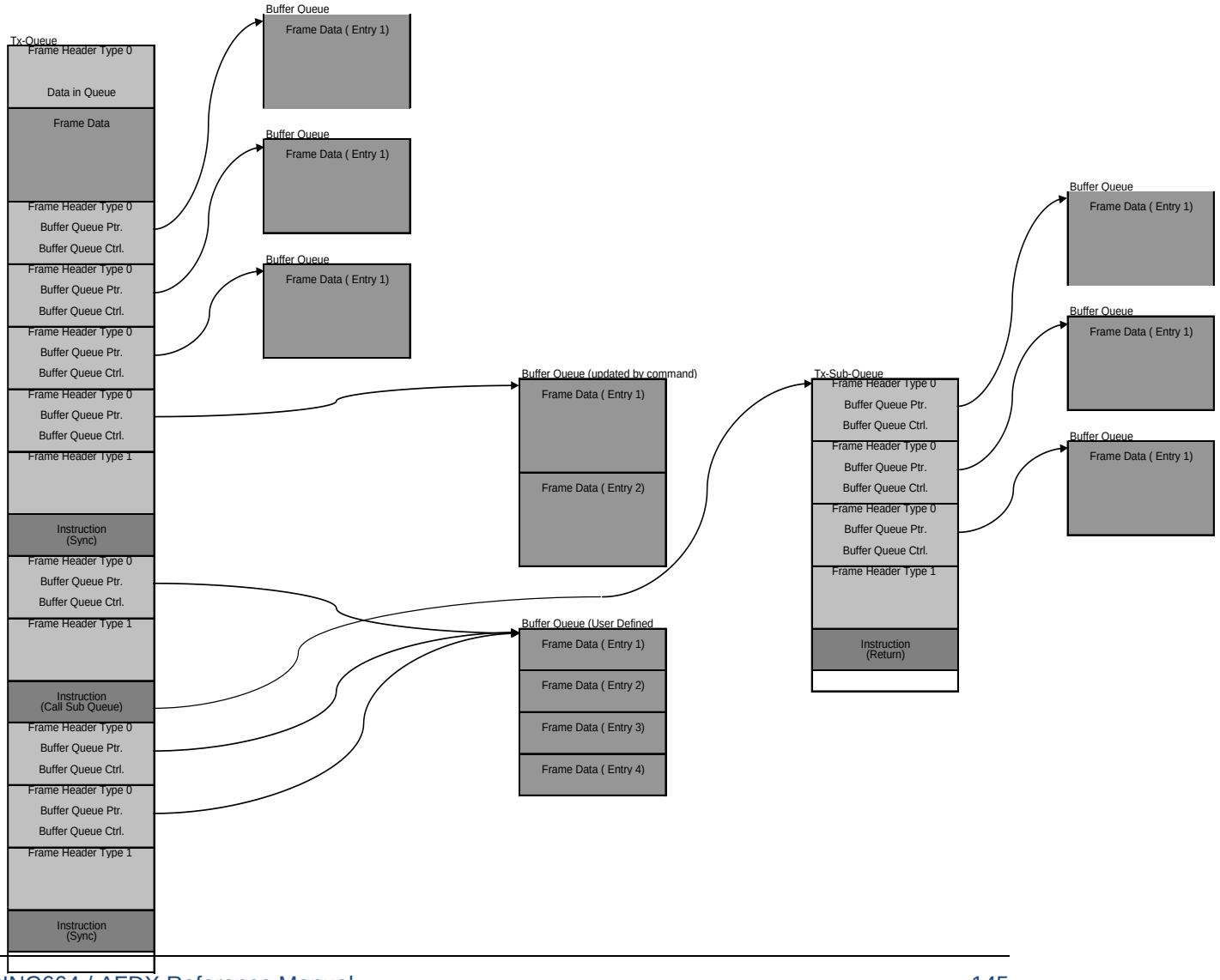
Normally a Transmit Frame is stored in the Transmit Queue with its Frame Header and a reference to a Buffer Queue with the size of one buffer. If the command

‘FdxCmdTxQueueUpdate’ is used for this Transmit Frame the Buffer Queue will be extended automatically to a size of two buffers. This is to guarantee consistent frame data on transmission. This means, that a frame can not be updated by the update function while it is actually transmitted by the BIU transmitter functionality.

Additionally a Buffer Queue with up to 128 buffers can be defined by user. This Buffer Queue can be shared by a arbitrary number of transfers. The number must be defined at allocation time of the buffer queue. The administration of assignement of transfers and buffer queues works internally in the boards Target Software.

With a special instruction inside the Transmit Queue a Sub Transmit Queue can be called. This Sub Transmit Queue has the same properties as the main Transmit Queue. After execution of this Sub Transmit Queue processing will return to the next instruciotn or transmission after the call instruction.

In the following schematic you can see a layout of memory organisaton in BIU related Global RAM.


Tx-Queue
Frame Header Type 0
Data in Queue
Frame Data
Frame Header Type 0
Buffer Queue Ptr.
Buffer Queue Ctrl.
Frame Header Type 0
Buffer Queue Ptr.
Buffer Queue Ctrl.
Frame Header Type 0
Buffer Queue Ptr.
Buffer Queue Ctrl.
Frame Header Type 0
Buffer Queue Ptr.
Buffer Queue Ctrl.
Frame Header Type 1
Instruction
(Sync)
Frame Header Type 0
Buffer Queue Ptr.
Buffer Queue Ctrl.
Frame Header Type 1
Instruction
(Call Sub Queue)
Frame Header Type 0
Buffer Queue Ptr.
Buffer Queue Ctrl.
Frame Header Type 0
Buffer Queue Ptr.
Buffer Queue Ctrl.
Frame Header Type 1
Instruction
(Sync)
Buffer Queue
Frame Data (Entry 1)
Buffer Queue
Frame Data (Entry 1)
Buffer Queue
Frame Data (Entry 1)
Buffer Queue (updated by command)
Frame Data (Entry 1)
Frame Data (Entry 2)
Buffer Queue (User Defined
Frame Data (Entry 1)
Frame Data (Entry 2)
Frame Data (Entry 3)
Frame Data (Entry 4)
Tx-Sub-Queue
Frame Header Type 0
Buffer Queue Ptr.
Buffer Queue Ctrl.
Frame Header Type 0
Buffer Queue Ptr.
Buffer Queue Ctrl.
Frame Header Type 0
Buffer Queue Ptr.
Buffer Queue Ctrl.
Frame Header Type 1
Instruction
(Return)
Buffer Queue
Frame Data (Entry 1)
Buffer Queue
Frame Data (Entry 1)
Buffer Queue
Frame Data (Entry 1)

THIS PAGE INTENTIONALLY LEFT BLANK

4.4 Receiver Functions

The following section describes functions to use the receiver part of the FDX-2/4 board. The functions are separated into three sections. The first section describes the general set up functions. The second section describes functions to get information and data on a Virtual Link-or UDP-Port-based view. The third section describes commands to monitor a continuous data stream.

The Handle input parameter to the following functions must be a port related handle.

Table 4.4-I: Receiver Functions

Function	*Description*
Global Receiver Functions	
FdxCmdRxPortInit	Initializes receiver on this port
FdxCmdRxModeControl	Defines the Mode of the receiver
FdxCmdRxControl	Starts and stops the receiver
FdxCmdRxStatus	Obtains status information about the receiver
FdxCmdRxGlobalStatistics	Obtains global statistics about the bus load
FdxCmdRxVLControl	Controls settings for each Virtual Link
FdxCmdRxVLControlEx	Controls extended settings for each Virtual Link
FdxCmdRxVLGetActivity	Obtains Activity information of one Virtual Link
FdxCmdRxTrgLineControl	Controls Receiver associated Trigger Lines
FdxCmdRxVlSetHwFilter	Initialize Hw VL-Filter (APX-GNET only)
VL-Oriented Receiver Functions	
FdxCmdRxUDPCreatePort	Creates an AFDX Comm-type connection oriented port
FdxCmdRxUDPChgDestPort	Change destination of an UDP port
FdxCmdRxSAPCreatePort	Creates a SAP type connectionless port
FdxCmdRxUDPDestroyPort	Destroys a UDP connection oriented port
FdxCmdRxUDPRead	Reads data from an UDP port
FdxCmdRXUDPGetStatus	Obtains the Status of an UDP port
FdxCmdRxUDPBlockRead	Reads data from one or several UDP ports
FdxCmdRxUDPControl	Allows a host interrupt on UDP frame reception
FdxCmdRxSAPRead	Reads data from a SAP port
FdxCmdRxSAPBlockRead	Reads data from one or several SAP ports
Chronologic Receiver Operation (Monitor) Functions	
FdxCmdMonCaptureControl	Defines the capturing mode
FdxCmdMonTCBSetup	Defines a Trigger Control Block
FdxCmdMonTrgWordIni	Initializes the Monitor Trigger Word
FdxCmdMonTrgIndexWordIni	Initializes the Monitor Trigger Index Word
FdxCmdMonTrgIndexWordIniVL	Initializes the VL specific Monitor Trigger Index Word
FdxCmdMonGetStatus	Obtains the Status of a Monitor port
FdxCmdMonQueueControl	Creates a Queue, associated with the Monitor
FdxCmdMonQueueRead	Reads data from a Monitor Data Queue
FdxCmdMonQueueSeek	Sets the internal Read index to a Monitor Data Queue
FdxCmdMonQueueTell	Gets the internal Read index to a Monitor Data Queue
FdxCmdMonQueueStatus	Gets status information of the Monitor Data Queue
Continuous Capture Second Edition Functions	
FdxCmdMonContCapControl	Defines the capturing mode special for CCSE
FdxCmdMonContCapProvideMemory	Provide capture data to system driver resp. board
FdxCmdMonContCapInvalidateMemory	Invalidate provided capture data to system driver resp. board
FdxCmdMonContCapForceDataTransfer	Force System to pass databuffer to Host and use next

4.4.1 Global Receiver Commands

4.4.1.1 FdxCmdRxControl

Prototype:

AiReturn FdxCmdRxControl(AiUInt32 ul_Handle,
const TY_FDX_RX_CTRL *px_RxControl);

Purpose:

This function is used to control the receive operation of the board.

Input:

TY_FDX_RX_CTRL *px_RxControl

Pointer to a control structure to start the receive port

```
typedef struct {
     AiUInt32 ul_StartMode;
     AiUInt32 ul_GlobalStatisticReset;
  } TY_FDX_RX_CTRL;
```

AiUInt32 ul_StartMode

Control Parameter for the Receiver Mode

Value	*Description*
FDX_STOP	Stop the Receiver
FDX_START	Start the Receiver

AiUInt32 ul_ GlobalStatisticReset

This parameter is to control resetting of the statistic parameters in the target, described in the output section. See also ***FdxCmdRxGlobalStatistics*** command.

Value	*Description*
FDX_RX_GS_RES_NO_CNT	Reset nothing
FDX_RX_GS_RES_ALL_CNT	Reset all counts
FDX_RX_GS_RES_ERR_CNT	Reset only the Error related Counters

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.1.2 FdxCmdRxGlobalStatistics

Prototype:

```
AiReturn FdxCmdRxGlobalStatistics(      AiUInt32 ul_Handle,
                                        AiUint32 ul_Control,
                                        TY_FDX_RX_GLOB_STAT *px_GlobalStatistic,
                                        TY_FDX_RX_GLOB_STAT *px_GlobalStatisticPortB );
```

Purpose:

Get the busload measurement values to give an indicator about the full bus traffic over all Virtual Links.

Input:

AiUint32 ul_Control

This parameter is to control resetting of the statistic parameters in the target, described in the output section.

A Reset always means 'Reset after read'.

Value	*Description*
FDX_RX_GS_RES_NO_CNT	Reset nothing
FDX_RX_GS_RES_ALL_CNT	Reset all counts
FDX_RX_GS_RES_ERR_CNT	Reset only the Error related Counters

Output:

*TY_FDX_RX_GLOB_STAT *px_GlobalStat*

The members of this structure give an overview of the received data and the actual busload.

```
typedef struct {
   AiUInt32 ul_TotalByteCount;
   AiUInt32 ul_FrameGoodCount;
   AiUInt32 ul_FrameErrorCount;
   AiUInt32 ul_BytesPerSecond;
   AiUInt32 ul_FramesPerSecond;
   TY_FDX_RX_GLOB_STAT_ERR  x_StatErr;
   TY_FDX_RX_GLOB_STAT_SIZE x_StatSize;
}TY_FDX_RX_GLOB_STAT;
```

AiUInt32 ul_TotalByteCount;

Counter of total Bytes received, since start or the last counter reset.

AiUInt32 ul_FrameGoodCount;

Counter of error free Frames received, since start or the last counter reset.

AiUInt32 ul_FrameErrorCount;

Counter of all erroneous Frames received, since start or the last counter reset.

AiUInt32 ul_BytesPerSecond;

Bytes received per second. This value will be updated by the Target Software in a fixed time interval.

AiUInt32 ul_FramesPerSecond;

Frames received per second. This value will be updated by the Target Software in a fixed time interval.

TY_FDX_RX_GLOB_STAT_ERR x_StatErr

This structure contains extended count information about erroneous received frames since start or the last counter reset.

```
typedef struct {
   AiUInt32 ul_PhysErrorCount;
   AiUInt32 ul_PreamErrorCount;
   AiUInt32 ul_UnaligErrorCount;
   AiUInt32 ul_CRCErrorCount;
   AiUInt32 ul_IFGErrorCount;
   AiUInt32 ul_IPErrorCount;
   AiUInt32 ul_MACErrorCount;
   AiUInt32 ul_NoSfdErrorCount;
   AiUInt32 ul_VLLenErrorCount;
   AiUInt32 ul_SNIntegrityErrorCount;
   AiUInt32 ul_TrafShapingViolationCount;
} TY_FDX_RX_GLOB_STAT_ERR;
```

AiUInt32 ul_PhysErrorCount

Number of physical errors detected

AiUInt32 ul_PreamErrorCount

Number of preamble errors detected

AiUInt32 ul_UnaligErrorCount

Number of unaligned frame errors detected

AiUInt32 ul_CRCErrorCount

Number of CRC errors detected

AiUInt32 ul_IFGErrorCount

Number of Interframe Gap (IFG) errors detected

AiUInt32 ul_IPErrorCount

Number of IP static header fields errors detected

AiUInt32 ul_MACErrorCount

Number MAC static header fields errors detected

AiUInt32 ul_NoSfdErrorCount

Number Frames detected with no without Start Frame Delimiter

AiUInt32 ul_VLLenErrorCount[3]

Number of VL specific frame length error detected

ul_SNIntegrityErrorCount[3]

Number of frames detected with sequence number integrity error.

AiUInt32 ul_TrafShapingViolationCount [2]

Traffic shaping violation detected

[1]: only applicable in redundant operation

[2]: only applicable if VL Descriptor setup and Traffic Shaping verification enabled

[3]:only applicable if VL Descriptor setup

TY_FDX_RX_GLOB_STAT_SIZE x_StatSize

This structure contains extended count information about the size of received frames since start or the last counter reset.

```
typedef struct {
   AiUInt32 ul_MAC1Short;
   AiUInt32 ul_MAC64To127Count;
   AiUInt32 ul_MAC128To255Count;
   AiUInt32 ul_MAC256To511Count;
   AiUInt32 ul_MAC512To1023Count;
   AiUInt32 ul_MAC1024To1518Count;
   AiUInt32 ul_MACLong;
} TY_FDX_RX_GLOB_STAT_SIZE;
```

AiUInt32 ul_MAC1Short

Number of frames, with length from 1…63 Bytes

AiUInt32 ul_MAC64To127Count

Number of frames, with length from 64…127 Bytes

AiUInt32 ul_MAC128To255Count

Number of frames, with length from 128…255 Bytes

AiUInt32 ul_MAC256To511Count

Number of frames, with length from 256…511 Bytes

AiUInt32 ul_MAC512To1023Count

Number of frames, with length from 512…1023 Bytes

AiUInt32 ul_MAC1024To1518Count

Number of frames, with length from 1024…1518 Bytes

AiUInt32 ul_MACLong

Number of frames, with length > 1518 Bytes

*TY_FDX_RX_GLOB_STAT *px_GlobalStatPortB*

This pointer is only necessary if the Handle references a Port configured in redundant Mode. In that Case this structure reports the same information as *px_GlobalStat for the redundant Port.

In case of single configuration it is possible to pass NULL for this parameter.

Return Value

Returns FDX_OK on success or a negative error code on error.
Error Codes: FDX_ERR

4.4.1.3 FdxCmdRxModeControl

Prototype:

```
AiReturn FdxCmdRxModeControl(  AiUInt32 ul_Handle,
                               const TY_FDX_RX_MODE_CTRL_IN
                               *px_RxModeControlIn,
                               TY_FDX_RX_MODE_CTRL_OUT
                               *px_RxModeControlOut);
```

Purpose :

This function is used to configure the operational mode of the receive port.

Input:

TY_FDX_RX_MODE_CTRL_IN
****px_RxModeControlIn***

Pointer to a control structure to setup the receive port and also get information about that port.

```
typedef struct {
   AiUInt32 ul_ReceiveMode
   AiUInt32 ul_DefaultPayloadMode;
   AiUInt32 ul_DefaultCronoMode;
   AiUInt32 ul_GlbMonBufferSize;
   AiUInt32 ul_RerosDefaultOutputPortmap;
} TY_FDX_RX_MODE_CTRL_IN;
```

AiUInt32 ul_ReceiveMode

There are two modes of operation for a Receiver port

1. **Chronological monitoring**
   Data of all enabled Virtual Links are stored in one large chronological buffer.
2. **Virtual Link oriented reception**
   Each enabled Virtual Link has its own buffer, where only the data of this VL will be stored. All VL's are disabled by default. Use ***FdxCmdRxVLControl*** to enable VL's of interest.

Value	*Description*
FDX_RX_CHRONO	Chronological monitoring
FDX_RX_VL	VL oriented storage

AiUInt32 ul_DefaultPayloadMode

Defines the Payload Mode, for data flow reduction. So you can specify up to which level the data shall be stored in the data buffer.

In this mode you can see the traffic on the bus without monitoring and transferring the full data.

Value	Description
FDX_PAYLOAD_FULL	Frames will be stored with full payload in the Monitor buffer. This means the full Ethernet (MAC) frame is available for the application.
FDX_PAYLOAD_IP_EXT	Only the Frame status header, the MAC- /IP- frame headers plus 20 bytes of the IP-Payload will be stored in the corresponding data buffer.
FDX_PAYLOAD_IP	Only the Frame status header, the MAC- /IP- frame headers will be stored in the corresponding data buffer.
FDX_PAYLOAD_MAC	Only the Frame status header, the MAC- frame headers will be stored in the corresponding data buffer.
GNET_PORT_PAYLOAD_FRH64[1)]	Only Frame Header and 64 Byte of Frame Data are stored to the Receive Buffer. (MAC and IP Header + 30 Bytes of data).
GNET_PORT_PAYLOAD_FRH32[1)]	Only Frame Header and 32 Byte of Frame Data are stored to the Receive Buffer. (MAC Header + 18 MAC-Payload).

Note: ***This parameter is not used for ul_ReceiveMode = FDX_RX_VL. Then always ul_DefaultPayloadMode = FDX_PAYLOAD_FULL is used.***
The GNET_PORT_ parameters are only valid for APX-GNET boards.

[1)] These Parameters for GNET_PORT_PAYLOAD_... must be selected wired ored to the other standard payload modes of can be selected instead of FDX_PAYLOAD_FULL for the APX-GNET 2/4 board to select the dedicated Port Data Store Mode.

AiUInt32 ul_DefaultChronoMode

There are three possible default Modes, if **FDX_RX_CHRONO** has been selected for the Receive Mode. If **FDX_RX_VL** has been selected for the Receive Mode, **FDX_RX_DEFAULT_ENA_CNT** is used, which means, that Global Statistics and VL oriented counters are available.

Value	Description
FDX_RX_DEFAULT_ENA_CNT	All Virtual Links are disabled for capturing. VL oriented counters will be updated. This mode is helpful to see any activity on the bus, without monitoring any data.
FDX_RX_DEFAULT_ MON_ ENA_ALL	All Virtual Links are enabled for capturing. That means, all incoming frames are stored in the Global Monitor Buffer, defined with the parameters above. In parallel the VL oriented counters will be updated.
FDX_RX_DEFAULT_MON_ ENA_GOOD	All Virtual Links are enabled for monitoring. That means, only good (error free frames) are stored in the Global Monitor Buffer, defined with the parameters above. In parallel the VL oriented counters will be updated.

Note: ***This parameter is not used for ul_ReceiveMode = FDX_RX_VL. Then always ul_DefaultChronoMode = FDX_RX_DEFAULT_ENA_CNT is used.***

AiUInt32 ul_GlbMonBufferSize

Size of the global Monitor Memory buffer for this port resource. This value must be specified in Bytes. This parameter also reflects the size of the Monitor Queue.

This parameter is only relevant if the chronological monitoring mode is selected. The parameter is not used for ul_ReceiveMode = FDX_RX_VL.

If this value is set to 0, the onboard target software will be set to a default value.

AiUInt32 ul_ RerosDefaultOutputPortmap

Reserved.

Output:

TY_FDX_RX_MODE_CTRL_OUT *px_RxModeControlOut

```
typedef struct {
   AiUInt32 ul_GlbMonBufferSize;
} TY_FDX_RX_MODE_CTRL_OUT;
```

AiUInt32 ul_GlbMonBufferSize

The adjusted global Monitor Memory buffer size is reported in this output parameter. For VL oriented mode, this value will be 0.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.1.4 FdxCmdRxPortInit

Prototype:

AiReturn FdxCmdRxPortInit (AiUInt32 ul_Handle,
const TY_FDX_PORT_INIT_IN *px_PortInitIln,
TY_FDX_PORT_INIT_OUT *px_PortInitOut);

Purpose:

This function is for initializing the port.

For Receive functionality, the initialized state is as follows:

- Global Statistics available
- All Virtual Links, enabled for Activity information
- Chronological Receive Mode, No VLs enabled for capturing
- No Trigger Control Block Processing Enabled

Input:

TY_FDX_PORT_INIT_IN* px_PortInitIN

Pointer to a board control input structure.

```
typedef struct {
   AiUInt32 ul_PortMap;
} TY_FDX_PORT_INIT_IN;
```

AiUInt32 ul_PortMap

This is a user definable byte for identification. This identification will occur later in the received data (respective Frames) to identify the port.

Note: This function can only be performed by a privileged user!

Output:

TY_FDX_PORT_INIT_OUT* px_PortInitOut

```
typedef struct {
   AiUInt32 ul_PortConfig;
   AiUInt32 ul_PortUsed;
   AiUInt32 ul_GlobalMemFree;
   AiUInt32 ul_SharedMemFree;
} TY_FDX_PORT_INIT_OUT;
```

AiUInt32 ul_PortConfig

Reflects the current port configuration

Value	Description
FDX_SINGLE	Single Mode
FDX_REDUNDANT	Redundant Mode

AiUInt32 ul_PortUsed

Each port can be used by different clients. This array over the maximum count of ports per board shows how many clients are using the ports at this time. This information only includes if the port is used or not. For detailed information the function FdxQueryResource(..) can be used

Value	Description
0	Port is not used by any client
>0	Number of clients are using this port

AiUInt32 ul_GlobalMemFree

Size of Global Memory (in Bytes) which is not already allocated.

AiUInt32 ul_SharedMemFree

Size of Shared Memory (in Bytes) which is not already allocated.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.1.5 FdxCmdRxStatus

Prototype:

AiReturn FdxCmdRxStatus (AiUInt32 ul_Handle,
TY_FDX_RX_STATUS *px_RxStatus);

Purpose:

This function is used to retrieve the receiver status of the board.

Input:

None

Output:

TY_FDX_RX_STATUS *px_RxStatus

```
typedef {
   AiUInt32 ul_Status;
   AiUInt32 ul_Info;
} TY_FDX_RX_STATUS;
```

AiUInt32 ul_Status

Status information of the Receiver:

Value	*Description*
FDX_STAT_STOP	Receiver Stopped
FDX_STAT_RUN	Receiver Running
FDX_STAT_ERROR	Receiver Error

AiUInt32 ul_Info

Additional Receiver Status Information

Value	*Description*
FDX_RX_INFO_OVERLOAD	Receiver Overload! Firmware was not able to capture all data.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.1.6 FdxCmdRxTrgLineControl

Prototype:

AiReturn FdxCmdRxTrgLineControl (AiUInt32 ul_Handle,
const TY_FDX_TRG_LINE_CTRL *px_TrgLineCtrl);

Purpose:

This function is used to select the Trigger In- and Output lines for the Receiver part of the associated port.

Input

TY_FDX_TRG_LINE_CTRL *px_TrgLineCtrl

This structure defines the Trigger In- and Output line routing

```
typedef struct {
   AiUInt32 ul_TrgInLine;
   AiUInt32 ul_TrgOutLine;
} TY_FDX_TRG_LINE_CTRL;
```

AiUInt32 ul_TrgInLine

Receive Trigger Input Line

AiUInt32 ul_TrgOutLine

Receive Trigger Output Line

Values for Trigger Lines:

Value	*Description*
FDX_STROBE_LINE_OFF	Trigger Off
FDX_STROBE_LINE_1	Trigger Line 1
FDX_STROBE_LINE_2	Trigger Line 2
FDX_STROBE_LINE_3	Trigger Line 3
FDX_STROBE_LINE_4	Trigger Line 4
FDX_STROBE_LINE_KEEP	Keep current setting

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.1.7 **FdxCmdRxVLControl**

Prototype:

```
AiReturn FdxCmdRxVLControl(  AiUInt32 ul_Handle,
                             const TY_FDX_RX_VL_CTRL
                             *px_VLControl,
                             const TY_FDX_RX_VL_DESCRIPTION
                             *px_VlDescription);
```

Purpose:

This function is used to control the Virtual link specific setting for a receive port. The settings for a VL can be equivalent to a VL capture filter. That means, only data of a VL which is enabled for capturing, will be stored in the associated buffer.

Input:

*TY_FDX_RX_VL_CTRL *px_VLControl*

A pointer to a structure which describes the Virtual Link related parameters for the receiver.

```
typedef struct {
   AiUInt32 ul_VLId;
   AiUInt32 ul_VLRange;
   AiUInt32 ul_EnableMode;
   AiUInt32 ul_PayloadMode;
   AiUInt32 ul_TCBIndex;
} TY_FDX_RX_VL_CTRL;
```

AiUInt32 ul_VLId

Virtual Link Identifier. A value in a range from 0 to 65535. This value is part of the MAC destination address.

A range of VLs to which the TY_FDX_RX_VL_CTRL settings are applied can be given by setting the Start VL Identifier into the ***ul_VLId***, and the number of VLs (starting at the VL Id in the ***ul_VLId*** parameter) into the ***ul_VLRange*** parameter. This is not allowed together with ***ul_EnableMode*** = FDX_RX_VL_ENA_EXT of the TY_FDX_RX_VL_CTRL settings !

AiUInt32 ul_VLRange

Number of VLs which are affected by the TY_FDX_RX_VL_CTRL settings, beginning with the VL set in the ***ul_VLId*** parameter.

AiUInt32 ul_EnableMode

Mode which is used for the given VL.

Value	*Comment*
FDX_RX_VL_DIS	Virtual Link is disabled. All Frames for the given VLs are discarded. No VL specific counters and global statistic counters are updated.
FDX_RX_VL_ENA_STAT	Virtual Link is enabled for global statistic counting only
FDX_RX_VL_ENA_CNT	Virtual Link is enabled for global statistic and VL specific counting. This mode is helpful to see any activity on the bus, without storing any frames
FDX_RX_VL_ENA_MON_ALL	Same as FDX_RX_VL_ENA_MON plus the capability to store erroneous frames as well. This mode is only applicable if the receive port works in chronological oriented receive mode (see *FdxCmdModeControl* command) and the port is configured for non-redundant operation.
FDX_RX_VL_ENA_MON_GOOD	Virtual Link is enabled for reception of frames. This means, all incoming frames for the given VL are stored in the Global Monitor Buffer, defined with the parameters above. In parallel the VL specific counters and global statistic counters are updated. The processing of Trigger Control Blocks for the given VL is enabled (see *FdxCmdMonTCBSetup* and related commands) . Erroneous frames are not stored. This mode is only applicable if the receive port works in chronological oriented receive mode (see *FdxCmdModeControl* command)
FDX_RX_VL_ENA_EXT	Virtual Link is enabled for extended operation. Frames are stored, dependent on the receive mode set via the *FdxCmdModeControl* command. If the receiver is running in VL oriented reception mode, frames of the given VL are stored in an individual buffer, with the size given by the *ul_VLBufSize* parameter. When the receiver is in chronological mode, the frames of the given VL are stored in a Global Monitor buffer. In this mode, the verification mode and the extended filtering can be applied to the given VL if necessary.

Note: If port oriented functions used only ul_EnableMode = FDX_RX_VL_ENA_EXT is possible.

VL Mode versus **Receiver Mode** Overview

VL Mode	*Chronological Rx Mode*	*VL oriented Rx Mode*
FDX_RX_VL_DIS	Yes	No
FDX_RX_VL_ENA_STAT	Yes	No
FDX_RX_VL_ENA_CNT	Yes	No
FDX_RX_VL_ENA_MON	Yes	No
FDX_RX_VL_ENA_MON_GOOD	Yes (if non-redundant)	No
FDX_RX_VL_ENA_EXT	Yes	Yes

AiUInt32 ul_PayloadMode

Defines the Payload Mode, for data flow reduction. So you can specify up to which level the data shall be stored in the data buffer. With this function you can define this level for each VL.

Using this mode you can see the traffic on the bus without monitoring and transferring the full data load.

Value	*Description*
FDX_PAYLOAD_FULL	Frames will be stored with full payload in the Monitor buffer. This means the full Ethernet (MAC) frame is available for the application.
FDX_PAYLOAD_IP_EXT	Only the Frame status header, the MAC- /IP- frame headers plus 20 bytes of the IP-Payload will be stored in the corresponding data buffer.
FDX_PAYLOAD_IP	Only the Frame status header, the MAC- /IP- frame headers will be stored in the corresponding data buffer.
FDX_PAYLOAD_MAC	Only the Frame status header, the MAC- frame headers will be stored in the corresponding data buffer.
FDX_PAYLOAD_DEFAULT	use default Value

Note: ***If port oriented functions used only ul_PayloadMode = FDX_PAYLOAD_FULL is possible.***

AiUInt32 ul_TCBIndex

This value defines the Trigger Control Block (TCB) Index which is written to the Monitor Trigger Index Word if a frame has been received for the given VL. A value of FFh disables any modification of the Trigger Index Word if a frame for the given VL is received.

A value of 0, disables the complete Trigger Control Block Processing with reception of the given VL.

Valid range is 0...FDh and FFh (FEh reserved for internal use, ASP).

See also ***FdxCmdMonTrgIndexWordIniVL*** function.

Note: ***If port oriented functions used this parameter is not available.***

*TY_FDX_RX_VL_DESCRIPTION *px_VlDescription*

```
typedef struct {
   AiUInt32 ul_VerificationMode;
   AiUInt32 ul_Bag;
   AiUInt32 ul_Jitter;
   AiUInt32 ul_MaxFrameLength;
   AiUInt32 ul_MaxSkew;
   AiUInt32 ul_VLBufSize;
   TY_FDX_RX_VL_EXT_FLT x_VLExtendedFilter;
   AiUInt32 ul_MinFrameLength;
} TY_FDX_RX_VL_DESCRIPTION;
```

Note: ***Structure TY_FDX_RX_VL_DESCRIPTION is used only if parameter ul_EnableMode is FDX_RX_VL_ENA_EXT, pointer can be set to NULL otherwise.***

AiUInt32 ul_VerificationMode

Control parameter to set up verification mode for the given VL. This parameter is only applicable if the VL is set to **FDX_RX_VL_ENA_EXT** mode.

Value	*Description*	*Comment*	*R*	*S*
FDX_RX_VL_CHECK_DISA	Verification disabled			
FDX_RX_VL_CHECK_ENA_DEFAULT	Default Setting	This is a combination of the following values, dependant on if the Port is **R**edundant or **S**ingle. See the columns 'R' and 'S' for the specification of this value.	default redundant	default single
FDX_RX_VL_CHECK_REDMAM	Redundancy Management	Enable Redundancy Management like described in AFDX End System Specification [2]	✓	
FDX_RX_VL_CHECK_TRAFIC	Traffic shaping Verification	Enable Traffic Shaping Verification like described in AFDX Switch Specification [3]		✓
FDX_RX_VL_CHECK_FRAMESIZE	Vl specific Framesize Check	Maximum frame size for the given VL is checked.	✓	✓
FDX_RX_VL_CHECK_SNINTEG	Sequence Number Integrity check	Sequence numbering of the incoming frames are checked	✓	
FDX_RX_VL_CHECK_INVPAC	Invalid Packet processing	All Packets, also the erroneous, will be passed through to the buffer		

Note: If a verification mode is enabled, the structure px_VLDescription must be properly setup.

AiUInt32 ul_Bag

Bandwidth Allocation Gap (BAG) for the defined Virtual Link.

As shown below, this parameter is interpreted differently depending on the value of Bit 31.

Bit 31	*Bit 30-0*
1	BAG in Microseconds

Bit 31	*Bit 30-0*
0	BAG in Milliseconds

If specifying the range in milliseconds, ranges from 1 to 1000 are allowed

If specifying the range in microseconds, setting can be adjusted in 500 microsecond steps with a maximum value of 1000000. Values that are not multiples of 500 are not allowed and will lead to undefined/platform dependent behaviour.

Incoming frames that violate the BAG setting will be discarded if FDX_RX_VL_CHECK_TRAFIC is set in *ul_VerificationMode*. If FDX_RX_VL_CHECK_INVPAC is set also in *ul_VerificationMode* then frames which violate the bag are marked as erroneous but forwarded to the application (see TRS-bit in *uw_ErrorField* of **FdxCmdMonQueueRead**).

Note: The byte based policing algorithm is used here. So the received frame sizes are used to calculate the account function for the VL.

Note: ARINC664 conform values for the BAG are 1 to 128ms defined as 2^k [in ms], where k can have a range from integer 0 to 7

AiUInt32 ul_Jitter

Maximum allowed Jitter Value in µs, for the given Virtual Link.

Possible Range for the Jitter 1 to 65535 µs.

AiUInt32 ul_MaxFrameLength

Maximum Frame Length of a MAC Frame specified for this Virtual Link.

AiUInt32 ul_MinFrameLength

Minimum Frame Length of a MAC Frame specified for this Virtual Link.

AiUInt32 ul_MaxSkew

The maximum time difference between the arrival time of redundant frame with the same sequence number in µs. Possible Range for the MaxSkew 0 to 65535 µs.

AiUInt32 ul_VLBufSize

Size of the local Buffer which should be created by the onboard Target software to store data of the selected VL.

This parameter is only applicable if the receive port works in VL oriented mode (see ***FdxCmdModeControl*** command)

If this value is set to 0, the onboard target software will set to a default value.

TY_FDX_RX_VL_EXT_FLT x_VLExtendedFilter

By defining this structure, an extended, second level, frame Filter for each Virtual Link can be applied.

This extended filter is a generic filter to mask and compare four bytes of the data stream. These four bytes can be located on any position in the frame, specified by the filter position. The following figure shows the mechanism of this filter

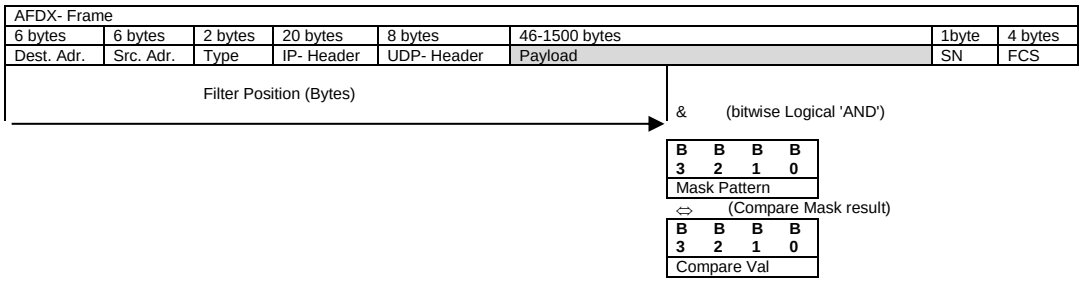


Figure 4.4.1.6-1: Mechanism of second level Filter

```
typedef struct {
   AiUInt32 ul_FilterMode
   AiUInt32 ul_FilterPosition;
   AiUInt32 ul_FilterMask;
   AiUInt32 ul_FilterData;
} TY_FDX_VL_EXT_FLT;
```

AiUInt32 ul_FilterMode

Filter Mode of the second level filter

Value	***Description***
FDX_DIS	Disable filtering
FDX_RX_VL_FLT_ENA	Enable filtering, frame is stored if the filter condition matches
FDX_RX_VL_FLT_ENA_INV	Enable filtering, frame is stored if the filter condition does not match

AiUInt32 ul_FilterPosition

Filter position offset to the start of the AFDX frame, where the value shall be compared.

AiUInt32 ul_FilterMask

Filter Mask to mask the bits of four consecutive bytes for comparing with the filter data. If bit is set (1) the according bit in filter data is relevant. If bit is not set (0) the according bit in filter data is don't care.

AiUInt32 ul_FilterData

Filter Data to compare with the result of masking.

(Example: Checking for Udp-Destination = 10 decimal.

The following settings can be used:

ul_FilterPosition = 36 (Udp-Destination)

ul_FilterMask = FFFF0000hex (mask out UDP-length)

ul_FilterData = 000A0000hex (check for UDP-destination 10decimal))

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.1.8 FdxCmdRxVLControlEx

Prototype:

AiReturn FdxCmdRxVLControEx(AiUInt32 ul_Handle,
const TY_FDX_RX_VL_CTRL_EX
****px_VLControlEx);***

Purpose:

This function is to control the extended Virtual link specific setting for a receive port. This command can only be applied if the corresponding VL has been previously setup with the ***FdxCmdRxVLControl*** command and the VL is operating in ***FDX_RX_VL_ENA_EXT*** mode.

Input:

TY_FDX_RX_VL_CTRL_EX *px_VLControlEx

A pointer to a structure which describes the extended Virtual Link related parameters for the receiver.

```
typedef struct {
   AiUInt32 ul_VLId;
   AiUInt32 ul_RmdIpp;
   AiUInt32 ul_Ctrl;
}TY_FDX_RX_VL_CTRL_EX;
```

AiUInt32 ul_VLId

Virtual Link Identifier. A value in a range from 0 to 65535. This value is part of the MAC destination address.

A range of VLs to which the TY_FDX_RX_VL_CTRL settings are applied can be given by setting the Start VL Identifier into the LSW, and the End VL Identifier of the range into the MSW of the ***ul_VLId*** parameter.

AiUInt32 ul_RmdIpp

This parameter is no longer used in this function

AiUInt32 ul_Ctrl

This parameter allows the user to define miscellaneous control settings for VL related reception.

Following functional groups can be controlled. The modes within one group are exclusive. But all group modes can be combined.

Frame related interrupt control

Mode	*Description*
FDX_RX_VL_NOINT	No interrupt on frame reception
FDX_RX_VL_INT	Interrupt if frame received
FDX_RX_VL_INT_ERR	Interrupt if error on received frame

Frame related output strobe generation

Mode	Description
FDX_RX_VL_NOS	No strobe
FDX_RX_VL_STR	strobe trigger output on frame reception
FDX_RX_VL_EST	strobe trigger output on erroneous frame reception

VL Buffer Store mode (only applicable in VL oriented Rx mode)

Mode	Description
FDX_RX_VL_CYC	Cyclic Data Storage

VL Buffer related Interrupt control (only applicable in VL oriented Rx mode)

Mode	Description
FDX_RX_VL_NOBI	No interrupt
FDX_RX_VL_BFI	Interrupt on Buffer Full
FDX_RX_VL_HFI	Interrupt on Half Buffer Full
FDX_RX_VL_QFI	Interrupt on Quarter Buffer Full

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.1.9 FdxCmdRxVLGetActivity

Prototype:

AiReturn FdxCmdRxVLGetActivity (AiUInt32 ul_Handle,
const TY_FDX_RX_VL_ACTIVITY_IN *px_VLActivityIn,
TY_FDX_RX_VL_ACTIVITY_OUT *px_VLActivityOut);

Purpose:

This function is used to obtain the activity status for all active or a specific Virtual Link(s). The memory which is needed for this list has to be provided by the application.

Input:

TY_FDX_RX_VL_ACTIVITY_IN* px_VLActivityIn

Structure, containing control parameters for the VL Activity command.

```
typedef struct {
   AiUInt32 ul_Mode;
   AiUInt32 ul_VLId;
   AiUInt32 ul_MaxReadBytes;
} TY_FDX_RX_VL_ACTIVITY_IN;
```

AiUInt32 ul_Mode

Following modes are supported :

Mode	Description
FDX_RX_VL_ACT_ALL	Gets Activity Information of all active VLs
FDX_RX_VL_ACT_VL	Get Activity Information for a specific VL
FDX_RX_VL_ACT_CNT	Get count of current active VLs

AiUInt32 ul_VLId

Virtual Link Identifier. A value in a range from 0 to 65535. This value is part of the MAC destination address. The parameter is only applicable in case of FDX_RX_VL_ACT_VL mode (see above).

AiUInt32 ul_MaxReadBytes

Maximum number of bytes which can be written to the provided data buffer.

Output:

TY_FDX_RX_VL_ACTIVITY_OUT* px_VLActivityOut

Structure, containing the VL Activity Information.

```
typedef struct {
   AiUInt32 ul_NumOfActivVL;
   AiUInt32 ul_ActivOvfl;
   TY_FDX_RX_VL_ACTIVITY *pax _VLActivity;
   AiUInt32 ul_EntriesRead;
} TY_FDX_RX_VL_ACTIVITY_OUT;
```

AiUInt32 ul_NumOfActivVL

The number of VLs which have received data. This is also the size of the following array.

AiUInt32 ul_ActivOvfl

VL Activity Overflow indication. A value not equal zero reports an overflow of the VL Activity List on the related port. Due to the limitation of the memory of the port, used for VL Activity information, this memory may be exhausted in case of too much different VLs received on this port. The current limit is at 877 different which VL can be handled by the activity processing of the port.

TY_FDX_RX_VL_ACTIVITY *pax _VLActivity

Pointer to a array of structured elements. Each element has the following structure The memory of this array has to be provided by calling function.

```
typedef struct {

   AiUInt32 ul_VLIdent;
   AiUInt32 ul_EnableMode;
   AiUInt32 ul_VerificationMode;
   AiUInt32 ul_PayloadMode;
   AiUInt32 ul_VLErrorOccurrenceA;
   AiUInt32 ul_FrameCountA;
   AiUInt32 ul_ErrorCountA;
   AiUInt32 ul_FramesPerSecondA;
   AiUInt32 ul_VLErrorOccurrenceB;
   AiUInt32 ul_FrameCountB;
   AiUInt32 ul_ErrorCountB;
   AiUInt32 ul_FramesPerSecondB;
   AiUInt32 ul_FramesRmDiscardedA;
   AiUInt32 ul_FramesRmDiscardedB;
} TY_FDX_RX_VL_ACTIVITY;
```

AiUInt32 ul_VLIdent

Virtual Link Identifier

AiUInt32 ul_EnableMode

AiUInt32 ul_PayloadMode

AiUInt32 ul_VerificationMode

Verification Mode which is selected for the specified Virtual Link. For description see the specification of the command FdxCmdRxFLControl(...).

AiUInt32 ul_PayloadMode

This parameter reflects the current Enable, Verification and Payload mode of the given VL (see ***FdxCmdRxVLControl*** for detailed explanation of these parameter).

AiUInt32 ul_VLErrorOccurrenceA/B[1]

This bit-oriented information is a cumulative list of error types which have occurred for the Virtual –Link.

The Library function ***FdxTranslateErrorWord*** translates the given Error Word into a zero terminated string, containing '/' separated Error abbreviations (see Abbreviation Column).

Note: Error Types may occur as combinations of the following constants.

Error Type Constant	*Error Description*	*Abbreviation*
FDX_RX_ERROR GNET_RX_ERROR	Wrong physical Symbol during frame reception.	PHY
FDX_PREAMBLE_ERROR	Wrong Preamble/Start Frame Delimiter received.	PRE
FDX_TRIP_NIBBLE_ERROR	Unaligned Frame length received	TRI
FDX_CRC_ERROR GNET_CRC_ERROR	MAC CRC Error.	CRC
FDX_SHORG_IFG_ERROR GNET_SHORG_IFG_ERROR	Short Interframe Gap Error (<960ns)	IFG
FDX_IP_ERROR GNET_IP_ERROR	AFDX IP Framing Error (AFDX-IP frame specific settings violated).	IPE
FDX_MAC_ERROR GNET_MAC_ERROR	AFDX MAC Framing Error (AFDX-MAC frame specific settings violated).	MAE
FDX_NO_VALID_SFD	Frame without valid Start Frame Delimiter received	SFD
FDX_LONG_FRAME_ERROR GNET_LONG_FRAME_ERROR	Long Frame Received (> 1518 Bytes)	LNG
FDX_SHORT_FRAME_ERROR GNET_SHORT_FRAME_ERROR	Short Frame Received (< 64 Bytes)	SHR
FDX_VL_FRAME_SIZE_ERROR GNET_VL_FRAME_SIZE_ERROR	VL specific Frame size Violation	VLS
FDX_SEQUENCE_NO_ERROR GNET_SEQUENCE_NO_ERROR	Sequence No. Mismatch	SNE
FDX_TRAFFIC_SHAP_ERROR GNET_TRAFFIC_SHAP_ERROR	Traffic Shaping Violation	TRS

Note: It is strictly recommended to use the GNET_ defines for the APX-GNET board because the defines are slightly different to the FDX_ defines. The use of wrong defines can cause unpredictable results.

AiUInt32 ul_FrameCount A/B[1]

Counter of valid frames received for that Virtual Link

AiUInt32 ul_ErrorCount A/B[1]

Counter of erroneous frames received for that Virtual Link

AiUInt32 ul_FramesPerSecond A/B[1]

Frames received per second for this VL. This value is updated by the onboard Target software in a fixed interval.

AiUInt32 ul_FramesRmDiscarded A/B[2]

Each frame received redundant by this port must be discarded by the receiver. This counter counts all these frames.

[1]: The "B"-variables are only applicable, if the Handle used for this function references a redundant configured port. Then the VL-specific counter and error information for both ports is provided.

[2]: This counter is only applicable, if the Handle used for this function references a redundant configured port.

AiUInt32 ul_EntriesRead

Number of VL-entries of type TY_FDX_RX_VL_ACTIVITY written to the provided buffer. If the provided buffer (indicated by variable ul_MaxReadBytes) is too small to contain the information for all active VLs this number might be smaller than number of active VLs (ul_NumOfActivVL).

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.1.10 FdxCmdRxVLSetHwFilter

Prototype:

AiReturn FdxCmdRxVLSetHwFilter (AiUInt32 ul_Handle,
const TY_FDX_RX_VL_SET_HW_FILTER_IN *px_VLHwFilterIn,
TY_FDX_RX_VL_SET_HW_FILTER_OUT *px_VLHwFilterOut);

Purpose:

This function is used to set up a VL based Filter in Hardware which filters the frames directly at the frontend to reduce date which must be processed by the onboard firmware. This is a function directly implemented in hardware and for this the Vl Hardware Filters are limited to dfew blocks founded to limited hardware resources. The number of available Vl Hardware Filter blocks are different to the different boards.

Note: This function is only available for APX-GNET 2/4 boards

Input:

TY_FDX_RX_VL_HW_FILTER_IN *px_VLHwFilterIn

Structure, containing control and setup information for the VL hardware filters.

```
typedef struct {
   AiUInt32 ul_Index;
   TY_FDX_RX_HW_VL_FILTER x_VlHwFilter;
} TY_FDX_RX_VL_SET_HW_FILTER_IN;
```

AiUInt32 ul_Index

Index over the number of Filters to specify in a Ragne form 0 to ul_NumOfPossibleFilters -1.

TY_FDX_RX_HW_VL_FILTER x_VlHwFilter

Structure which describes the function of the VL Hardware Filter.

```
typedef struct {
   AiUInt32 ul_Ena;
   AiUInt32 ul_VLRangeMin;
   AiUInt32 ul_VLRangeMax;
   AiUInt32 ul_VLCompareLogic;
   AiUInt32 ul_AcceptErrors;
   AiUInt32 ul_HwTriggerEna;
} TY_FDX_RX_HW_VL_FILTER;
```

AiUInt32 ul_Ena

Specifies if this Hardware Filter shall ba active or not.

Value	***Description***
FDX_ENA	Enables this Hardware Filter
FDX_DIS	Disables this Hardware Filter

AiUInt32 ul_VLRangeMin

Minimum Limit of the Virtual Link Identifier Range which shall be filterds. The value must be in a range from 0 to 65535 and must be lower or equat to ul_VLRangeMax.

AiUInt32 ul_VLRangeMax

Maximum Limit of the Virtual Link Identifier Range which shall be filterds. The value must be in a range from 0 to 65535 and must be greater or equat to ul_VLRangeMin.

AiUInt32 ul_VLCompareLogic;

Specifies how the VL compare range shall be used

Value	*Description*
FDX_RX_VL_HWF_INSIDE	Pass all frames through which are inside the range defined by ul_VlRangeMin and ul_VlRangeMax.
FDX_RX_VL_HWF_OUTSIDE	Pass all frames through which are not inside the range defined by ul_VlRangeMin and ul_VlRangeMax.

AiUInt32 ul_AcceptErrors

Specifies if erroneous frames shall be recorded or discarded.

Value	*Description*
FDX_OFF	All erroneous frames are discarded by the hardware.
FDX_ON	All erroneous frames are passed through to the upper following instances.

AiUInt32 ul_HwTriggerEna

Specifies if the hardware related Trigger Control Block shall be evaluate this frame for Trigger capability or not

Value	*Description*
FDX_OFF	Frame will be only passed to the receive buffer if accepted by the filter.
FDX_ON	Frame will be passed to the related Hardware Trigger Control Blockfilter to evaluate the trigger condition.

Output:

*TY_FDX_RX_VL_HW_FILTER_OUT *px_VLHwFilterOut*

Structure, containing the information about the VL hardware filters.

```
typedef struct {
   AiUInt32 ul_Ena;
   AiUInt32 ul_NumOfPossibleFilters;
} TY_FDX_RX_VL_SET_HW_FILTER_OUT;
```

AiUInt32 ul_Ena

Gives a feedback if this Vl Hardware Filter is enabled or not.

AiUInt32 ul_NumOfPossibleFilters

Returns the number of Filters which ara possible to set up for this port.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.2 VL oriented Receiver Functions

These functions are only applicable, if the receive port is switched to Virtual Link oriented mode.

4.4.2.1 FdxCmdRxSAPBlockRead

Prototype:

AiReturn FdxCmdRxSAPBlockRead (AiUInt32 ul_Handle,
const TY_FDX_SAP_BLOCK_READ_IN
****px_SapBlockReadIn,***
TY_FDX_SAP_BLOCK_READ_OUT *px_SapBlockReadOut);

Purpose:

This function reads data from one or several SAP connectionless ports.

Input:

TY_FDX_SAP_BLOCK_READ_IN *px_SapBlockReadIn

Pointer to an array of structures. Each structure describes an individual read operation for a single SAP port. The array contains ul_PortCount elements.

```
typedef struct {
  AiUInt32 ul_PortCount;
  TY_FDX_BLOCK_READ_IN_PORT* px_SapBlockReadInPortArray;
}TY_FDX_SAP_BLOCK_READ_IN;
```

AiUInt32 ul_PortCount

Specifies the number of SAP ports which shall be read from.

TY_FDX_BLOCK_READ_IN_PORT *px_SapBlockReadInPortArray

```
typedef struct {
   AiUInt32 ul_UdpHandle;
   AiUInt32 ul_MsgCount;
} TY_FDX_BLOCK_READ_IN_PORT;
```

AiUInt32 ul_UdpHandle

The handle of the SAP port to read messages from.

AiUInt32 ul_MsgCount

Number of Messages to read.

Note: At this time, only reading of 1 message per port is supported. Hence, you have to set this parameter to 1! To read more messages of one port, add another entry to input array px_SapBlockReadInPortArray with same ul_UdpHandle.

Output:

TY_FDX_SAP_BLOCK_READ_OUT *px_SapBlockReadOut

A pointer to an array of structures. Each structure contains result information about an individual SAP read operation. The array contains ul_PortCount elements.

```
typedef struct {
  AiUInt32  ul_PortCount;
  TY_FDX_BLOCK_READ_OUT_PORT* px_SapBlockReadOutPortArray;
} TY_FDX_SAP_BLOCK_READ_OUT;
```

Note: The maximum size of the number of bytes to be returned is limited. The limit is system dependent. If the maximum was exceeded the ul_PortCount of the output structure will be less than the ul_PortCount of the input structure.

ul_PortCount

Number of SAP ports of which data has been read from.

TY_FDX_BLOCK_READ_OUT_PORT *px_SapBlockReadOutPortArray

```
typedef struct {
   AiUInt32  ul_UdpHandle;
   AiReturn  st_ResultCode;
   AiUInt32  ul_MsgRead;
   void      *pv_ReadBuffer;
} TY_FDX_BLOCK_READ_OUT_PORT;
```

ul_UdpHandle

The handle of the associated SAP port.

st_ResultCode

The result of the individual read operation for the associated SAP port.

ul_MsgRead

Number of Messages actually read (0 or 1).

Note: For APE/ACE/AXC/AMCX-FDX boards, this field has to be initialized with the size of pv_ReadBuffer in bytes. Otherwise no message will be read from the requested port!

void *pv_ReadBuffer

Pointer to the buffer where data to be read should be stored. The size required for the buffer can be calculated as:

required buffer size = ul_UdpMaxMessageSize + sizeof(TY_SAP_BUFFER_HEADER).

The ul_UdpMaxMessageSize is defined with function FdxCmdRxSAPCreatePort.

One entry specifies one message. For special system information and administration a fix sized header is preceeding the message.

For layout of such an entry refer to definition in function FdxCmdRxSAPRead.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.2.2 FdxCmdRxSAPCreatePort

Prototype:

AiReturn FdxCmdRxSAPCreatePort (AiUInt32 ul_Handle,
const TY_FDX_RX_SAP_CREATE_IN
****px_SapCreateIn,***
TY_FDX_RX_SAP_CREATE_OUT
****px_SapCreateOut);***

Purpose:

Creates a port to receive data from a specified SAP connectionless port. This function can be used only, if receiver is not running. This function assumes that ***FdxCmdRxVLControl*** was previously called in order to enable a VL for reception. On this VL this function creates an AFDX SAP Rx-port.

Input:

TY_FDX_RX_SAP_CREATE_IN *px_SapCreateIn

Pointer to a structure, which describes the SAP port.

```
typedef struct {
   AiUInt32 ul_UdpDst;
   AiUInt32 ul_IpDst;
   AiUInt32 ul_VlId;
   AiUInt32 ul_UdpNumBufMessages;
   AiUInt32 ul_UdpMaxMessageSize;
}TY_FDX_RX_SAP_CREATE_IN;
```

AiUInt32 ul_UdpDst

The UDP destination address of this port.

AiUInt32 ul_IpDst

The IP destination address of this port.

AiUInt32 ul_VlId

The AFDX VL to which this port is associated.

AiUInt32 ul_UdpNumBufMessages

Number of messages which should be stored by the onboard Target software.

Size of the local Buffer which should be created by the onboard Target software to store data of the created UDP port can be calculated by ul_UdpNumBufMessages * ul_UdpMaxMessageSize.

AiUInt32 ul_UdpMaxMessageSize;

This is the maximum size of the variable length message which can be received by this port. (0..ize ≤ 8kBytes)

Note: ***If received message exceeds maximum size this message will be cut off and only ul_UdpMaxMessageSize bytes will be saved.***

Output:

TY_FDX_RX_SAP_CREATE_OUT *px_SapCreateOut

Pointer to a structure, which describes the SAP port.

```
typedef struct {
   AiUInt32 ul_UdpHandle;
}TY_FDX_RX_SAP_CREATE_OUT;
```

AiUInt32 ul_UdpHandle

Handle to the SAP port.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.2.3 FdxCmdRxSAPRead

Prototype:

AiReturn FdxCmdRxSAPRead (AiUInt32 ul_Handle,
const TY_FDX_RX_SAP_READ_IN *px_SapReadIn,
TY_FDX_RX_SAP_READ_OUT *px_SapReadOut);

Purpose:

This function reads data from an SAP Connectionless port.

Input:

TY_FDX_RX_SAP_READ_IN *px_SapReadIn

Pointer to a SAP port definition structure.

```
typedef struct {
   AiUInt32 ul_UdpHandle;
   AiUInt32 ul_MsgCount;
} TY_FDX_RX_SAP_READ_IN;
```

AiUInt32 ul_UdpHandle

See description of FdxCmdRxSAPCreatePort.

AiUInt32 ul_MsgCount

Number of Messages to read. This means the newest ul_MsgCount Entries.

Output:

TY_FDX_RX_SAP_READ_OUT *px_SapReadOut

Pointer to a structure which contains SAP port data.

```
typedef struct {
   AiUInt32 ul_MsgRead;
   void     *pv_ReadBuffer;
} TY_FDX_RX_SAP_READ_OUT;
```

AiUInt32 ul_MsgRead

Number of Messages actually read.

void *pv_ReadBuffer

Pointer to the data buffer the Entries should be stored. Required size of buffer can be calculated: ul_UdpMaxMessageSize * ul_MsgCount.

The ul_UdpMaxMessageSize is defined with function ***FdxCmdRxSAPCreatePort***.

One Entry specifies one Message, which means one complete SAP port message. For special system information and administration a Fix sized Header is preceded.

The following picture shows the schematic of such an entry.

	SAP Message Buffer Layout							
	31	24	23	16	15	8	7	0
SAP Header	Time Tag High							
	Time Tag Low							
	Message Size							
	IP Source Address							
	UDP Source Port							
	Reserved							
SAP Frame	Received SAP- FRAME (sampling message: up to SAP payload (1 – 1471 bytes queuing message: up to 8Kbytes)							

Figure 4.4.2.3-1: RX SAP Message Buffer Layout

TY_SAP_BUFFER_HEADER

This is a structural description of the SAP header

```
typedef struct sap_buffer_Header {
        TY_FDX_FW_IRIG_TIME   x_FwTimeTag;
        AiUInt32              ul_BufferSize;
        AiUInt32              ul_IPSrc;
        AiUInt32              ul_UDPSrcPort;
        AiUInt32              ul_Reserved;
} TY_SAP_BUFFER_HEADER;
```

TY_FDX_FW_IRIG_TIME x_FwIrigTime

The Firmware IRIG Time Tag information is from last received message. In case of a queuing port where the messages can be fragmented it is the Time Tag of the last received fragment.

```
typedef struct {
   AiUInt32 ul_TtHigh;
   AiUInt32 ul_TtLow;
} TY_FDX_FW_IRIG_TIME;
```

AiUInt32 ul_TtHigh;

Timetag word in firmware format. The higher part of the time tag, containing the minutes of hour, hours of day and day of year.

For further description see Firmware specification.

AiUInt32 ul_TtLow;

Timetag word in firmware format. The lower part of the time tag, containing the Microseconds of second, seconds of minutes and minutes of hour.

4.4.3 To get a ‘C’ structured information of the Time Tag you can use the functions FdxCmdFreeMemory

Prototype:

AiReturn FdxCmdFreeMemory(void * vp_MemPointer, AiUInt32 ul_StructId);

Purpose:

This function releases memory (previously allocated by other Application Interface Library Functions) in a proper manner.

Input:

void * vp_MemPointer

A pointer to an information list or an information array.

- If a pointer to an information list element, this must be the pointer to the first entry of the information list. The function will release the memory of each element in that list.
- If a pointer to an information array, this must be the pointer to the start of the array.

The application programmer should take care to insure that all memory allocated by an Application Interface Library function is freed prior to termination of the application.

AiUInt32 ul_StructId

Identification of the type of memory to be freed.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

FdxFwIrig2StructIrig ().

AiUInt32 ul_MsgSize;

SAP payload size in bytes of received Frame.

AiUInt32 ul_IPSrc

The IP source address of the received message

AiUInt32 ul_UdpSrc

The UDP source port of the received message

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.3.1 FdxCmdRxUDPBlockRead

Prototype:

AiReturn FdxCmdRxUDPBlockRead (const AiUInt32 ul_Handle,
const TY_FDX_UDP_BLOCK_READ_IN
****px_UdpBlockReadIn,***
TY_FDX_UDP_BLOCK_READ_OUT
****px_UdpBlockReadOut);***

Purpose:

This function reads data from one or several UDP connection oriented ports.

Input:

TY_FDX_UDP_BLOCK_READ_IN *px_UdpBlockReadIn

Pointer to a structure, which describes the configuration of one or more UDP ports.

```
typedef struct {
   AiUInt32 ul_PortCount;
   TY_FDX_BLOCK_READ_IN_PORT* px_UdpBlockReadInPortArray;
} TY_FDX_UDP_BLOCK_READ_IN;
```

AiUInt32 ul_PortCount

Specifies the number of UDP ports which shall be read from.

TY_FDX_BLOCK_READ_IN_PORT *px_UdpBlockReadInPortArray

Pointer to an array of structures. Each structure describes an individual read operation for a single UDP port. The array contains ul_PortCount elements.

```
typedef struct {
   AiUInt32 ul_UdpHandle;
   AiUInt32 ul_MsgCount;
}TY_FDX_UDP_BLOCK_READ_IN;
```

AiUInt32 ul_UdpHandle

The handle of the UDP port to which the message(s) will be written. This can be a handle to either a Sampling or Queuing port.

AiUInt32 ul_MsgCount

Number of Messages to read.

Note: At this time, only reading of 1 message per port is supported. Hence, you have to set this parameter to 1! To read more messages of one port, add another entry to input array px_UdpBlockReadInPortArray with same ul_UdpHandle.

Output:

TY_FDX_UDP_BLOCK_READ_OUT *px_UdpBlockReadOut

A pointer to an array of structures. Each structure contains result information about an individual UDP read operation. The array contains ul_PortCount elements.

```
typedef struct {
 AiUInt32  ul_PortCount;
 TY_FDX_BLOCK_READ_OUT_PORT* px_UdpBlockReadOutPortArray;
}TY_FDX_UDP_BLOCK_READ_OUT;
```

Note: The maximum size of the number of bytes to be returned is limited. The limit is system dependent. If the maximum was exceeded the ul_PortCount of the output structure will be less than the ul_PortCount of the input structure.

ul_PortCount

Number of UDP ports of which data are read from.

TY_FDX_BLOCK_READ_OUT_PORT *px_UdpBlockReadOutPortArray

```
typedef struct {
   AiUInt32  ul_UdpHandle;
   AiReturn  st_ResultCode;
   AiUInt32  ul_MsgRead;
   void      *pv_ReadBuffer;
} TY_FDX_BLOCK_READ_OUT_PORT;
```

ul_UdpHandle

The handle of the associated UDP port. This may be a handle to either a Sampling or Queuing port.

st_ResultCode

The result of the individual read operation for the associated UDP port.

ul_MsgRead

Number of Messages actually read (0 or 1).

Note: For APE/ACE/AXC/AMCX-FDX boards, this field has to be initialized with the size of pv_ReadBuffer in bytes. Otherwise no message will be read from the requested port!

void *pv_ReadBuffer

Pointer to the data buffer the Entries should be stored. Required size of buffer can be calculated: ul_UdpMaxMessageSize + sizeof(TY_FDX_UDP_HEADER).

The ul_UdpMaxMessageSize is defined with function FdxCmdRxUDPCreatePort.

One Entry specifies one Message, which means one complete sampling or queuing message. For special system information and administration a Fix sized Header is preceded.

For layout of such an entry refer to definition in function FdxCmdRxUDPRead.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.3.2 FdxCmdRxUDPControl

Prototype:

AiReturn FdxCmdRxUDPControl (AiUInt32 ul_Handle,
const AiUInt32 ul_UdpHandle,
const TY_FDX_RX_UDP_CONTROL
****px_UdpControl);***

Purpose:

This function allows to configure several settings of a UDP Receive Port.

Input:

AiUInt32 ul_UdpHandle

The handle of the associated UDP port.

TY_FDX_RX_UDP_CONTROL *px_UdpControl

Pointer to a UDP control structure.

```
typedef struct {
   AiUInt32 ul_NetSelect;
   AiUInt32 ul_InterruptControl;
} TY_FDX_RX_UDP_CONTROL;
```

AiUInt32 ul_NetSelect

Specifies the network to which the Rx UDP port is bound.

Value	*Description*
FDX_RX_UDP_BOTH	The port will receive messages from both networks
FDX_RX_UDP_ONLY_A	The port will receive messages from only network A
FDX_RX_UDP_ONLY_B	The port will receive messages from only network B

AiUInt32 ul_InterruptControl

This parameter allows to get a user interrupt on reception of each message on a UDP-Port.

Value	*Description*
FDX_RX_UDP_NOINT	No interrupt
FDX_RX_UDP_INT	Interrupt
FDX_RX_UDP_UDF	Not implemented

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.Error Codes:

FDX_ERR

4.4.3.3 FdxCmdRxUDPChgDestPort

Prototype:

AiReturn FdxCmdRxUDPChgDestPort (AiUInt32 ul_Handle,
AiUInt32 ul_UdpHandle,
AiUInt32 ul_UdpDst);

Purpose:

This function changes the UDP destination of a UDP queuing port. It can be used when receiver is running.

Input:

AiUInt32 ul_UdpHandle

The handle of the associated UDP port. This must be a handle to a Queuing port.

AiUInt32 ul_UdpDst

Destination UDP port. Part of the UDP quintuplet.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.
Error Codes: FDX_ERR

4.4.3.4 FdxCmdRxUDPCreatePort

Prototype:

AiReturn FdxCmdRxUDPCreatePort (AiUInt32 ul_Handle,
const TY_FDX_UDP_DESCRIPTION
****px_UdpDescription,***
AiUInt32 *pul_UdpHandle);

Purpose:

This function creates a port to receive data from a specified UDP connection oriented port. This function can be used only if the receiver is not running. This function assumes that ***FdxCmdRxVLControl*** was previously called in order to enable a VL for reception. On this VL this function creates an FDX communication Rx-port.

Input:

TY_FDX_UDP_DESCRIPTION *px_UdpDescription

Pointer to a structure, which describes the UDP connection.

```
typedef struct {
   AiUInt32 ul_PortType;
   struct _quintuplet {
      AiUInt32 ul_UdpSrc;
      AiUInt32 ul_IpSrc
      AiUInt32 ul_VlId;
      AiUInt32 ul_IpDst;
      AiUInt32 ul_UdpDst;
   }x_Quint;
   AiUInt32 ul_SubVlId;
   AiUInt32 ul_UdpNumBufMessages;
   AiUInt32 ul_UdpMaxMessageSize;
   AiUInt32 ul_UdpSamplingRate;
}TY_FDX_UDP_DESCRIPTION;
```

AiUInt32 ul_PortType

Type of the port connection

Value	*Description*
FDX_UDP_SAMPLING	Port is a sampling port. Each Message is represented by one MAC Frame. The size of the messages is fixed.
FDX_UDP_QUEUING	Port is a queuing port: Each Message can be represented by one or more MAC Frame. Reassembling will be done in the IP layer. A Message can have a size up to 8kByte.

struct _quintuplet

This structure provides a full identification of the communication.

AiUInt32 x_Quint.ul_UdpSrc

UDP port-number of the source UDP port.

AiUInt32 x_Quint.ul_IpSrc

IP address of the source partition.

AiUInt32 x_Quint.ul_VlId

Virtual Link Identifier

AiUInt32 x_Quint.ul_IpDst

Destination IP address

AiUInt32 x_Quint.ul_UdpDst

Destination UDP port

AiUInt32 uw_SubVlId;

Not relevant in Rx Mode.

AiUInt32 ul_UdpNumBufMessages

Number of messages which should be stored by the onboard Target software.

Size of the local Buffer which should be created by the onboard Target software to store data of the created UDP port can be calculated by ul_UdpNumBufMessages * ul_UdpMaxMessageSize.

For sampling ports the number of messages has to be set to 1.

For queuing ports an adequate buffer depth has to be provided.

If this value is set to zero, the onboard target software will be set to a default value.

AiUInt32 ul_UdpMaxMessageSize;

Maximum size of a message to receive.

For a sampling port, this is the fixed size of the sampling message, which means the size without the header overhead (MAC, IP and UDP).

For a queuing port this is the maximum size of the variable length message.

(Queuing: 0..ize ≤ 8kBytes, Sampling: ul_UdpMaxMessageSize)

Port Type	*Value*
Sampling	0..1471 bytes
Queuing	0..8 kBytes

Note: ***If received message exceeds maximum size this message will be cut off and only ul_UdpMaxMessageSize bytes will be saved.***

AiUInt32 ul_UdpSamplingRate;

Not relevant in Rx Mode.

Output:

AiUInt32 *pul_UdpHandle

Handle to the UDP port.

Return Value

Returns FDX_OK on success or a negative error code on error.
Error Codes: FDX_ERR

4.4.3.5 **FdxCmdRxUDPDestroyPort**

Prototype:

AiReturn FdxCmdRxUDPDestroyPort (AiUInt32 ul_Handle,
const AiUInt32 ul_UdpHandle);

Purpose:

This function is used to destroy the UDP receive port, if it is no longer in use. (This function is the opposite function to *FdxCmdRxUDPCreatePort* and *FdxCmdRxSAPCreatePort*). This function can be used only when the receiver is not running.

Input:

AiUInt32 ul_UdpHandle

See description of *FdxCmdRxUDPCreatePort*.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.3.6 FdxCmdRxUDPGetStatus

Prototype:

AiReturn FdxCmdRxUDPGetStatus (AiUInt32 ul_Handle,
const AiUInt32 ul_UdpHandle,
TY_FDX_RX_UDP_STATUS *px_UdpRxStatus);

Purpose:

This function reads the status of a UDP connection oriented port.

Input:

AiUInt32 ul_UdpHandle

See description of *FdxCmdRxUDPCreatePort* and *FdxCmdRxSAPCreatePort*.

Output:

TY_FDX_RX_UDP_STATUS *px_UdpRxStatus

Pointer to a structure, which contains status information of the selected UDP port.

```
typedef struct {
   AiUInt32 ul_MsgCount;
   AiUInt32 ul_MsgErrorCount;
} TY_FDX_RX_UDP_STATUS;
```

AiUInt32 ul_MsgCount

Total number of Messages written to the UDP port buffer since last start of receiver.

AiUInt32 ul_MsgErrorCount

Total number of erroneous Messages written to the UDP port buffer since last start of receiver.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.3.7 **FdxCmdRxUDPRead**

Prototype:

```
AiReturn FdxCmdRxUDPRead (  AiUInt32 ul_Handle,
                            const AiUInt32 ul_UdpHandle,
                            AiUInt32 ul_MsgCount,
                            AiUInt32 *pul_MsgRead,
                            void *pv_ReadBuffer);
```

Purpose:

This function reads data from a UDP connection oriented port.

Input:

AiUInt32 ul_UdpHandle

See description of *FdxCmdRxUDPCreatePort*.

AiUInt32 ul_MsgCount

Number of Messages to read. This means the newest *ul_MsgCount* Entries. For a sampling port there is a maximum of one message to read.

Note: For sampling ports the number of messages to read shall be always 1.

Output:

AiUInt32 *pul_MsgRead

Number of Messages actually read.

void *pv_ReadBuffer

Pointer to the data buffer the Entries should be stored. Required size of buffer can be calculated: ul_UdpMaxMessageSize * ul_MsgCount.

The ul_UdpMaxMessageSize is defined with function ***FdxCmdRxUDPCreatePort***.

One Entry specifies one Message, which means one complete sampling or queuing message. For special system information and administration a Fixed sized Header is preceded.

The following figure shows the schematic of such an entry.

Figure 4.4.2.8-1: RX UDP Message Buffer Layout

	UDP Message Buffer Layout							
	31	24	23	16	15	8	7	0
UDP Header	Time Tag High							
	Time Tag Low							
	Message Size							
	Error Information							
UDP Frame	Received UDP- FRAME (sampling message: up to UDP payload (1 – 1471 bytes queuing message: up to 8Kbytes)							

TY_FDX_UDP_HEADER

This is a structural description of the UDP header

```
typedef struct _fdx_udp_header {
   TY_FDX_FW_IRIG_TIME x_FwIrigTime;
   AiUInt32 ul_MsgSize;
   AiUInt32 ul_ErrorInfo;
} TY_FDX_UDP_HEADER;
```

TY_FDX_FW_IRIG_TIME x_FwIrigTime

The Firmware IRIG Time Tag information is from the last received message. For a queuing port where the messages can be fragmented, it is the Time Tag of the last received fragment.

```
typedef struct {
   AiUInt32 ul_TtHigh;
   AiUInt32 ul_TtLow;
} TY_FDX_FW_IRIG_TIME;
```

AiUInt32 ul_TtHigh;

Timetag word in firmware format. The higher part of the time tag, contains the minutes of hour, hours of day and day of year.

For further description see Firmware specification.

AiUInt32 ul_TtLow;

Timetag word in firmware format. The lower part of the time tag, contains the Microseconds of second, seconds of minutes and minutes of hour.

4.4.4 To get a ‘C’ structured information of the Time Tag you can use the functions FdxCmdFreeMemory

Prototype:

AiReturn FdxCmdFreeMemory(void * vp_MemPointer, AiUInt32 ul_StructId);

Purpose:

This function releases memory (previously allocated by other Application Interface Library Functions) in a proper manner.

Input:

*void * vp_MemPointer*

A pointer to an information list or an information array.

- If a pointer to an information list element, this must be the pointer to the first entry of the information list. The function will release the memory of each element in that list.
- If a pointer to an information array, this must be the pointer to the start of the array.

The application programmer should take care to insure that all memory allocated by an Application Interface Library function is freed prior to termination of the application.

AiUInt32 ul_StructId

Identification of the type of memory to be freed.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

FdxFwIrig2StructIrig ().

AiUInt32 ul_MsgSize;

UDP payload size in bytes of received Frame.

AiUInt32 ul_ErrorInfo;

Bit 31-16	*Bit15-0*
Error information as available in FdxCmdMonQueueRead variable uw_ErrorField	Queuing ports additional error information: IP_REASS_ERROR_SYNC IP_REASS_ERROR_ORDER IP_REASS_ERROR_FRAQ IP_REASS_ERROR_SIZE IP_REASS_ERROR_BUF

Return Value

Returns FDX_OK on success or a negative error code on error.
Error Codes: FDX_ERR

4.4.5 Chronological Monitor

These functions are only applicable, if the receive port is switched to chronological mode.

4.4.5.1 FdxCmdMonCaptureControl

Prototype:

AiReturn FdxCmdMonCaptureControl (AiUInt32 ul_Handle,
const TY_FDX_MON_CAP_MODE *px_MonCapMode);

Purpose:

This function is to control the capture mode of the Monitor.

Input:

TY_FDX_MON_CAP_MODE *px_MonCapMode

Pointer to a Capture Mode Setup structure

```
typedef struct {
   AiUInt32 ul_CaptureMode;
   AiUInt32 ul_TriggerPosition;
   AiUInt32 ul_Strobe;
} TY_FDX_MON_CAP_MODE;
```

AiUInt32 ul_CaptureMode

There are several capture modes which can be selected by the user

Mode	*Value*	*Description*
Single	FDX_MON_SINGLE	The Monitor buffer in the target will be filled one time. Then the capturing stops. The data will be available after first frame captured.
Continuous	FDX_MON_CONTINUOUS	The Monitor buffer in the target will be filled in a cyclic manner. This means. if the buffer is filled up one time it will be filled again. The oldest frame will be overwritten by the newest frame. The data will be available after first frame captured.
Continuous_2 [1)]	FDX_MON_CONTINUOUS_SE	The Monitor buffer is organized in the same way as in the continuous mode. The data will be available at the interface in prior provided memory buffers. It will be transferred directly by DMA to the host buffer.
Selective	FDX_MON_SELECTIVE	Frames are stored to the buffer which match a Start Trigger Condition which can be forced by using TCBs (Trigger Control Blocks) to control the frames which are captured (e.g. capture only if an external event has become true, capture only erroneous frames or frames which have a specific bit pattern set ….)
Selective Continuous [1)]	FDX_MON_CONT_SELECTIVE	Same mode as described for Selectiv but in Continuous Capture Mode.
Selective Continuous_2 [1)]	FDX_MON_CONT_SE_SELECTIVE	Same mode as described for Selectiv but in Continuous Capture Second Edition Mode.
Recording	FDX_MON_RECORDING	The Monitor buffer is organized in the same way as in the continuous mode. The data will not be available at the interface. It will be written directly to a file.

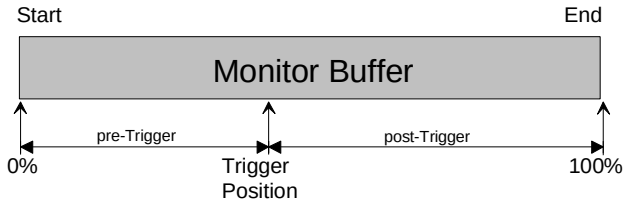
Mode	Value	Description
		Note: Recording to a file is not available for AVC-FDX VxWorks applications, however, recorded data can be used for real-time playback.
Selective Recording [1)]	FDX_MON_RECORDING_SELECTIVE	Same mode as described for Selectiv but with capabilities of Recording Mode.

1) At the moment this Mode is not available for APM-FDX and fdXTap.

AiUInt32 ul_TriggerPosition

This is a value between 0 and 100 %.

The trigger position is only relevant in single mode. It indicates the position in the Monitor buffer where the trigger event shall be located. This is used to balance the pre- and the post-trigger memory.



AiUInt32 ul_Strobe

Value	Description
FDX_MON_STROBE_DIS	Discrete Output Strobe disabled
FDX_MON_STROBE_STOP	Discrete Output Strobe is set if Capturing is stopped (Monitor Capture Buffer is full) in Standard or Selective Capture mode. In Continuous or Recording mode, the Discrete Output Strobe is set, after Half Monitor Buffer Size has been filled., and it is repeated after Half Buffer Size has been captured.
FDX_MON_STROBE_START	Discrete Output Strobe is set once if Capturing is started (In Standard, Continuous or Recording Mode). In Selective Capture Mode, the Strobe is asserted if the Capturing is restarted.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.
Error Codes: FDX_ERR

4.4.5.2 FdxCmdMonGetStatus

Prototype:

AiReturn FdxCmdMonGetStatus (AiUInt32 ul_Handle
TY_FDX_E_MON_STATUS *pe_MonStatus,
TY_FDX_MON_REC_STATUS *px_MonRecStatus);

Purpose:

This function returns the status of the selected receiver monitor port.

Input:

None

Output:

TY_FDX_E_MON_STATUS *pe_MonStatus

A pointer to an enumerated variable which indicates the status of the Monitor

```
typedef enum _mon_status {
  FDX_MON_OFF,
  FDX_MON_WAIT_FOR_TRIGGER,
  FDX_MON_TRIGGERED,
  FDX_MON_STOPPED,
  FDX_MON_FULL,
  FDX_MON_OVERLOAD
} TY_FDX_E_MON_STATUS;
```

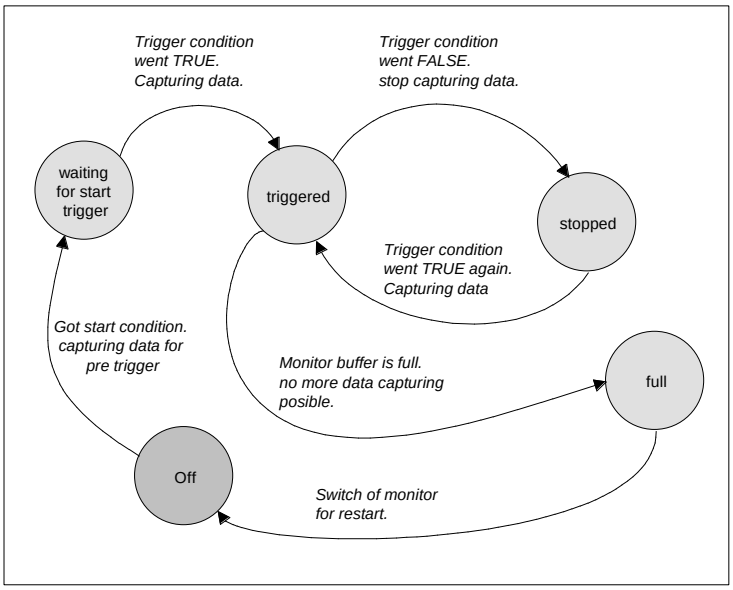
The following states are defined:

Status:	*Description*
FDX_MON_OFF	Monitor is not running. Captured data from a previous capturing session in the buffer is available.
FDX_MON_WAIT_FOR_TRIGGER	Monitor is waiting for Start Trigger. Data capturing is active
FDX_MON_TRIGGERED	Start Trigger occurred. Data will be captured.
FDX_MON_STOPPED	Capturing is stopped (e.g. if Selective Capturing mode has been selected and Start Trigger Condition has become false)
FDX_MON_FULL	The monitor buffer is full and no more data will be captured
FDX_MON_OVERLOAD	Onboard firware was not able to store data to the associated buffers. This can happen in continuous capture mode if traffic is to fast.

Note: In the mode continuous capturing, the state FDX_MON_FULL never will be reached.

Figure 4.4.3.2-1 shows the chronological monitor states and the junction between the states. The state Off can be reached from each other state by intervention of the user.

Figure 4.4.3.2-1: States of the Chronological Monitor



*TY_FDX_MON_REC_STATUS *px_MonRecStatus*

Status of the continuous capturing mode.

```
typedef struct {
   AiUInt32 ul_FramesCaptured;
   Ai_UInt64_Union u64_BytesRecorded;
} TY_FDX_MON_REC_STATUS;
```

AiUInt32 ul_FramesCaptured

Total number of frames captured since start.

Ai_UInt64_Union u64_BytesRecorded

Reserved.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.5.3 FdxCmdMonQueueControl

Prototype:

AiReturn FdxCmdMonQueueControl (AiUInt32 ul_Handle,
const TY_FDX_MON_QUEUE_CTRL_IN
****px_QueueCtrlIn,***
TY_FDX_MON_QUEUE_CTRL_OUT
****px_QueueCtrlOut);***

Purpose:

This function controls creation or deletion of queues, associated with the chronological monitor. The function returns a queue ID which must be used for all queue related functions.

Operating mode must be defined by using ***FdxCmdRxModeControl*** and ***FdxCmdCaptureControl*** before using this function. Dependant on the capture mode, the Frame data will be buffered in a second level queue of the onboard Target-Software (FDX_MON_CONTINUOUS and FDX_MON_RECORDING). In other cases the frames will be read directly from the receive buffer.

Input:

TY_FDX_MON_QUEUE_CTRL_IN *px_QueueCtrlIn

Pointer to structure of output data

```
typedef struct {
   AiUInt32 ul_QueueCtrl;
   AiUInt32 ul_QueueId;
   AiChar ac_RecordingFileName[AI_FDX_MAX_PATH];
   AiUInt32 ul_MaxFileSize;
} TY_FDX_MON_QUEUE_CTRL_IN;
```

AiUInt32 ul_QueueCtrl

Queue Control Code

Value:	*Description:*
FDX_MON_QUEUE_CREATE	Create queue, associated chronological buffer
FDX_MON_QUEUE_DELETE	Delete Queue

AiUInt32 ul_QueueId

Queue Identifier to delete (only needed if the Queue Control Code is set to **FDX_MON_QUEUE_CREATE**

AiChar ac_RecordingFileName

Name of recording file. May include also drive and path information.

Only relevant for recording mode and if the Queue Control Code is set to **FDX_MON_QUEUE_CREATE**)

Note :	for remote recording: The recording file is generated always locally on the server (where the fdx board is present). So drive and path information is related to the server.

AiUInt32 ul_MaxFileSize

Specifies the maximum file size in MBytes.

Output:

TY_FDX_MON_QUEUE_CTRL_OUT *px_QueueCtrlOut

Pointer to structure of output data

```
typedef struct {
   AiUInt32 ul_QueueId;
} TY_FDX_MON_QUEUE_CTRL_OUT;
```

AiUInt32 ul_QueueId

Queue Identifier. Valid Queue Identifier

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.5.4 FdxCmdMonQueueRead

Prototype:

AiReturn FdxCmdMonQueueRead (AiUInt32 ul_Handle,
AiUInt32 ul_QueueId,
const TY_FDX_MON_QUEUE_READ_IN
****px_QueueReadIn,***
TY_FDX_MON_QUEUE_READ_OUT
****px_QueueReadOut);***

Purpose:

This function is used to read Entries of a chronological capturing queue to a provided buffer. For this read function only the number of Entries to be read is specified. The reading start position is administrated internal to the target. The start position can by identified and modified by the functions *FdxCmdMonQueueTell(...)* and *FdxCmdMonQueueSeek(...).* Additional processing, for accessing the Frame Header data via the ***TY_FDX_FRAME_BUFFER_HEADER*** structure, must be done via ***FdxProcessMonQueue*** function**.**

Note: Data, read via ***FdxCmdMonQueueRead,*** can be directly used for Replay via the ***FdxCmdTxQueueWrite*** function, if the corresponding Transmit Port has been configured for Replay.

Input:

AiUInt32 ul_QueueId

Queue Identifier. Valid Queue Identifiers are obtained via the ***FdxCmdMonQueueControl*** command.

TY_FDX_MON_QUEUE_READ_IN *px_QueueReadIn

Pointer to Queue Read command parameter

```
typedef struct {
   AiUInt32 ul_EntryCount;
   AiUInt32 ul_ReadQualifier;
   AiUInt32 ul_MaxReadBytes;
} TY_FDX_MON_QUEUE_READ_IN;
```

AiUInt32 ul_EntryCount,

Number of Entries to read.

AiUInt32 ul_ReadQualifier

This is a qualifier, indicating which part of the Entry should be read.

Value:	*Description:*
FDX_MON_READ_FULL	Read the full entry
FDX_MON_READ_HEADER	Read only the fixed header
FDX_MON_READ_DATA	Read only the data, starting with the MAC Frame

AiUInt32 ul_MaxReadBytes,

The maximum number of bytes that can be written to the provided data buffer.

Output:

*TY_FDX_MON_QUEUE_READ_OUT *px_QueueReadOut*

Pointer to output data structure.

```
typedef struct {
   AiUInt32 ul_EntryRead;
   AiUInt32 ul_BytesRead;
   AiUInt32 ul_Synchronized;
   AiUInt64 vpv_ReadBuffer;
} TY_FDX_MON_QUEUE_READ_OUT;
```

AiUInt32 ul_EntyRead

Number of Entries actually read.

AiUInt32 ul_BytesRead

Number of Bytes actually read.

AiUInt32 ul_Synchronized

Capture mode FDX_MON_SINGLE, FDX_MON_SELECTIVE:

Signals the number of overwrites which have occurred in the monitor queue. Frames will then be read starting at the current location of the monitor queue write pointer used by the firmware.

For buffer overflow, existing data in the buffer is overwritten. Pointers of the data buffer are then re-synchronized and ul_Synchronized is increased by one (loss of data). Set to 0 when receiver is started.

Capture mode FDX_MON_SINGLE, FDX_MON_CONTINUOUS

A value greater than zero indicates an error:

Value:	*Description:*
FDX_SLQ_HW_ERROR	Second level queue hardware error.
FDX_SLQ_OVERFLOW_ERROR	Second level queue overflow error.
FDX_SLQ_UNKNOWN_STATUS_ERROR	Second level queue unknown status error.

*void *pv_ReadBuffer*

Pointer to the data buffer where the Entries should be stored. The size of this buffer should correspond to *ul_MaxReadBytes*.

One Entry specifies one Message, which means one complete MAC frame transferred on the network. For special system information and administration a Fixed sized Header is preceded.

Figure 4.4.3.4-1 shows the schematic of such an entry.

Figure 4.4.3.4-1: Monitor Queue Entry Structure of a standard AFDX Frame (API/AMC/APU-FDX)

Monitor Buffer Entry Layout (AFDX)									
Word		31	24	23	16	15	8	7	0
Fixed Entry Header	0	Previous Buffer Start Pointer - pointer to previous frame							
	1	**Next Buffer Start Pointer** - pointer to the next frame							
	2	Buffer Entry Control Word (BECW)							
	3	Reserved							
	4	**Frame Header Word 0 (FHW0)** - Bitmap of frame errors and VL ID							
	5	**Frame Header Word 1 (FHW1)** - Network ID, IFG info, bytes received							
	6	**Frame Header Word 2 (FHW2)** - Sequence Number and frame size (minus CRC)							
	7	Time Tag High Word (TTHW)							
	8	Time Tag Low Word (TTLW)							
AFDX Frame	Received AFDX- FRAME (802.3 defines: 64 to 1518 bytes but it may be less or more at Frame size violation case)								

Figure 4.4.3.4-1: Monitor Buffer Entry Layout (APX-GNET and APE-FDX)

Monitor Buffer Entry Layout (GNET)									
Word		31	24	23	16	15	8	7	0
Fixed Entry Header	0	Buffer Entry Control Word (BECW)							
	1	Buffer Entry Size Word (BESW)							
	2	**Buffer Header Type Word (BHTW)**							
	3	**Next Buffer Start Pointer** – pointer to the next frame							
	4	**Frame Header Word 0 (FHW0)** – MAC ID, bitmap of frame errors and VL ID							
	5	**Frame Header Word 1 (FHW1)** - IFG info, frame bytes received							
	6	Time Tag High Word (TTHW)							
	7	Time Tag Low Word (TTLW)							
AFDX Frame	Received AFDX- FRAME (802.3 defines: 64 to 1518 bytes but it may be less or more at Frame size violation case)								

TY_FDX_FRAME_BUFFER_HEADER

This is a structural description of the frame buffer header

```
typedef struct _fdx_frame_buffer_header {
   AiUInt32 ul_Prev;
   AiUInt32 ul_Next;
   AiUInt32 ul_EntryControl;
   AiUInt32 ul_Reserved;
   TY_FDX_FRAME_HEADER_INFO x_FrameHeaderInfo;
   TY_FDX_FW_IRIG_TIME x_FwIrigTime;
} TY_FDX_FRAME_BUFFER_HEADER;
```

AiUInt32 ul_Prev;

A pointer to the previous Frame.

If this frame is the first in a List, this pointer points to itself.

AiUInt32 ul_Next;

A pointer to the next Frame.

If this frame is the last in a list and not already updated, it points to itself

AiUInt32 ul_EntryControl

Additional Information about the captured frame

Value	Bit	Description
Buffer Type	31..30	Buffer Entry Type 0: Default Receive Buffer Entry 3: APE-FDX /APX-GNET Buffer Entry other values: Reserved
Payload store mode	29..27	Which payload store mode is selected for this frame. 0: Received frame completely stored 1: Frame Header, MAC,IP Header and 20 Bytes if IP Payload stored 2: Frame Header, MAC,IP Header stored 3: Frame Header and MAC Header stored other values: Reserved
Start Trigger Flag	26	Captured asserted Start Trigger condition
	25..24	Reserved
AFDX Port Map ID	23..16	User defined Port Number which is associated to the physical port(PortMapID see FdxCmdRxPortInit).
Reserved	15..14	
Speed Mode	13	0: 100 Mbit/sec 1: 10 Mbit/sec
Traffic shaping checked	12	The VL specific Traffic shaping verification was applied to this Frame.
Sequence No checked	11	The sequence number integrity check was applied to this Frame.
Frame size checked	10	The VL specific Frame size check was applied to this Frame.
Reserved	9...0	

TY_FDX_FRAME_HEADER_INFO x_FrameHeaderInfo;

This structure contains information about the received frame. This structure is actually only valid for standard AFDX frame.

```
typedef struct {
   // Frame Header Word 0:
   AiUInt16 uw_VlId;
   AiUInt16 uw_ErrorField;

   // Frame Header Word 1:
   AiUInt32 ul_FrameHeaderWord_1;

   // Frame Header Word 2:
   AiUInt8  uc_SequenceNr;
   AiUInt8  uc_Reserved1;
   AiUInt16 uw_BufferSize;
   } TY_FDX_FRAME_HEADER_INFO;
```

Frame Header Word 0 contains following information

AiUInt16 uw_VlId

Virtual Link Identifier, associated with the frame

AiUInt16 uw_ErrorField

This bit-oriented information is a cumulative list of error types which have occurred.

The Library function ***FdxTranslateErrorWord*** translates the given Error Word into a zero terminated string, containing '/' separated Error abbreviations (see Abbreviation Column)

Bit	*Error Type*	*Abbreviation*
0	Wrong physical Symbol during frame reception.	PHY
1	Wrong Preamble/Start Frame Delimiter received.	PRE
2	Unaligned Frame length received (Triple Nibble Error).	TRI
3	MAC CRC Error.	CRC
4	Short Interframe Gap Error (<960ns)	IFG
5	AFDX IP Framing Error (AFDX-IP frame specific settings violated).	IPE
6	AFDX MAC Framing Error (AFDX-MAC frame specific settings violated).	MAE
7	Frame without valid Start Frame Delimiter received	SFD
8	Long Frame Received (> 1518 Bytes)	LNG
9	Short Frame Received (< 64 Bytes)	SHR
10	VL specific Frame size Violation	VLS
11	Sequence No. mismatch	SNE
12	not used	RS2
13	Traffic Shaping Violation	TRS
Rest	not used	

AiUInt32 ul_FrameHeaderWord_1

Frame Header Word 1 contains following information

Value	*Bit*	*Description*
Network ID	31..29	The network on which the Frame was received 001: Frame received Network A 010: Frame received Network B others: reserved
Interframe Gap High	28	A indication that the gap between two frames was greater than 655,36 µs
Interframe Gap Count	27..14	Counter of the interframe gap with a resolution of 40ns, if the gap is lower than 655,36 µs. (Example: A 'Interframe Gap Count' – value of 5 is a gap time of 5*40ns = 200ns)
Reserved	13..11	
Byte Count	10..0	Byte count of the received AFDX / MAC – Frame (incl. CRC Bytes).

Frame Header Word 2 contains following information

AiUInt8 uc_SequenceNr

Sequence number (copied from AFDX frame)

AiUInt8 uc_Reserved1

Reserved

AiUInt16 uw_BufferSize

Size of the frame entry (Header + Data) in Bytes (Note: CRC Bytes are not stored)

TY_FDX_FW_IRIG_TIME x_FwIrigTime

The Firmware IRIG time tagged to this frame. The reference point for the Time Tag is the start of the preamble, transferred in front of this frame.

```
typedef struct {
   AiUInt32 ul_TtHigh;
```

```
   AiUInt32 ul_TtLow;
} TY_FDX_FW_IRIG_TIME;
```

AiUInt32 ul_TtHigh;

Timetag word in firmware format. The higher part of the time tag, contains the minutes of hour, hours of day and day of year.

For further description see Firmware specification.

AiUInt32 ul_TtLow;

Timetag word in firmware format. The lower part of the time tag, contains the Microseconds of second, seconds of minutes and minutes of hour.

4.4.6 To get a 'C' structured information of the Time Tag you can use the functions FdxCmdFreeMemory

Prototype:

AiReturn FdxCmdFreeMemory(void * vp_MemPointer, AiUInt32 ul_StructId);

Purpose:

This function releases memory (previously allocated by other Application Interface Library Functions) in a proper manner.

Input:

void * vp_MemPointer

A pointer to an information list or an information array.

- If a pointer to an information list element, this must be the pointer to the first entry of the information list. The function will release the memory of each element in that list.
- If a pointer to an information array, this must be the pointer to the start of the array.

The application programmer should take care to insure that all memory allocated by an Application Interface Library function is freed prior to termination of the application.

AiUInt32 ul_StructId

Identification of the type of memory to be freed.

Output:

None

Return Value

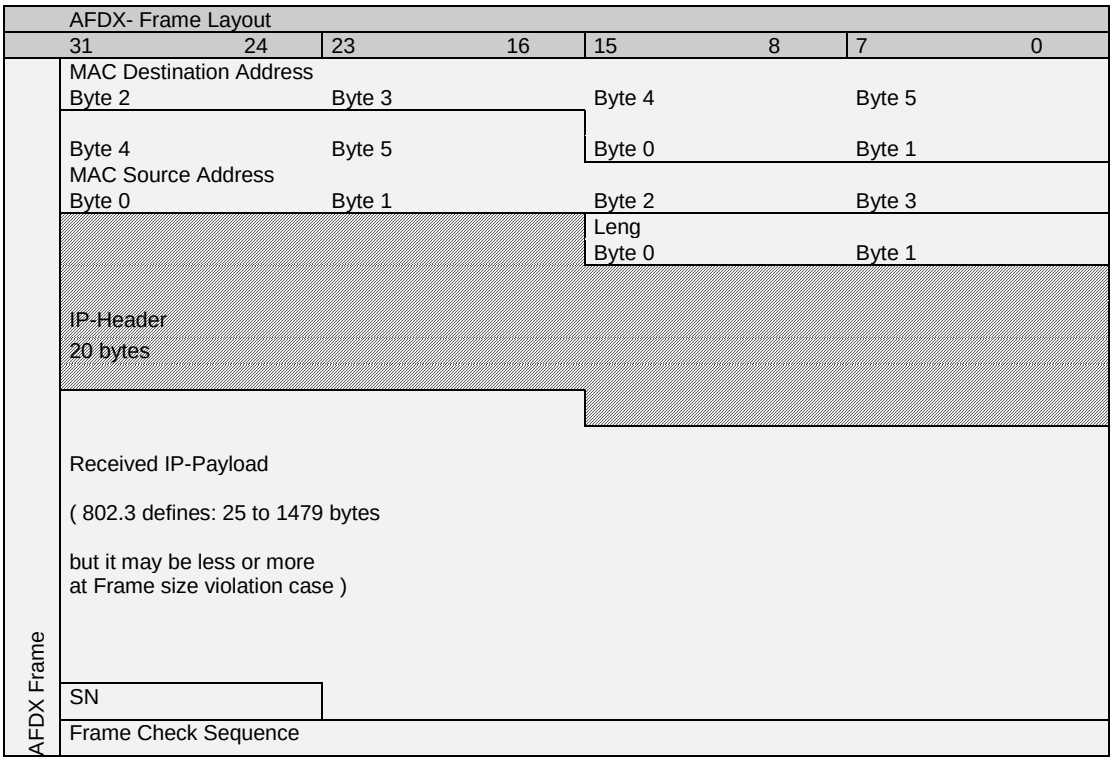
Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

FdxFwIrig2StructIrig ().

Figure 4.4.3.4-2 shows a more detailed structure of an AFDX Frame.

Figure 4.4.3.4-2: AFDX Frame Layout



Return Value

Returns FDX_OK on success or a negative error code on error.
Error Codes: FDX_ERR

4.4.6.1 **FdxCmdMonQueueSeek**

Prototype:

AiReturn FdxCmdMonQueueSeek (AiUInt32 ul_Handle,
AiUInt32 ul_QueueId,
const TY_FDX_MON_QUEUE_SEEK_IN *px_QueueSeekIn,
TY_FDX_MON_QUEUE_SEEK_OUT *px_QueueSeekOut);

Purpose:

This function sets the read index to the data queue to read queue entries from a specified location.

This function is only applicable if the capture control is set to single (non-continuous).

Input:

AiUInt32 ul_QueueId

Queue Identifier. Valid Queue Identifiers are get via the ***FdxCmdMonQueueControl*** command.

TY_FDX_MON_QUEUE_SEEK_IN *px_QueueSeekIn

Pointer to seek command parameter control structure

```
typedef struct {
   AiInt32   l_SeekOffset;
   AiUInt32 ul_SeekOrigin;
} TY_FDX_MON_QUEUE_SEEK_IN;
```

AiInt32 l_SeekOffset

Seek offset from the specified Seek Origin. The offset is specified in full message.

AiUInt32 ul_SeekOrigin

The following Values are specified:

Define	***Description***
FDX_SEEK_SET	Seek from the beginning of the queue to the offset position.
FDX_SEEK_END	Seek from the end of the queue to the offset position.
FDX_SEEK_CUR	Seek from the current internal read pointer of the queue to the offset position.
FDX_SEEK_TRG	Seek from the Trigger of the queue to the offset position

Output:

*TY_FDX_MON_QUEUE_SEEK_OUT *px_QueueSeekOut*

Pointer to structure of output data

```
typedef struct {
   AiUInt32 ul_ByteOffset;
} TY_FDX_MON_QUEUE_SEEK_OUT;
```

AiUInt32 ul_ByteOffset

Byte offset to the seek target position calculated from start of the queue. This value can be used to calculate memory buffer sizes for reading buffer entries.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.6.2 **FdxCmdMonQueueTell**

Prototype:

AiReturn FdxCmdMonQueueTell (AiUInt32 ul_Handle,
AiUInt32 ul_QueueId,
TY_FDX_MON_QUEUE_TELL_OUT
****px_QueueTellOut);***

Purpose:

This function gets the actual read index to the data queue where the next queue read will occur.

This function is only applicable if the capture control is set to single (non-continuous).

Input:

AiUInt32 ul_QueueId

Queue Identifier. Valid Queue Identifiers are get via the ***FdxCmdMonQueueControl*** command.

Output:

TY_FDX_MON_QUEUE_TELL *px_QueueTellOut

Pointer to structure of output data

```
typedef struct {
   AiUInt32 ul_Position;
   AiUInt32 ul_ByteOffset;
} TY_FDX_MON_QUEUE_TELL_OUT;
```

AiUInt32 ul_Position

Internal read index position. This index can be modified by *FdxCmdMonQueueSeek(...)*.

AiUInt32 ul_ByteOffset

Byte offset to the seek target position calculated from start of the queue. This value can be used to calculate memory buffer sizes for reading buffer entries.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.6.3 FdxCmdMonQueueStatus

Prototype:

AiReturn FdxCmdMonQueueStatus (AiUInt32 ul_Handle,
AiUInt32 ul_QueueId,
TY_FDX_MON_QUEUE_ STATUS_OUT
****px_QueueStatusOut);***

Purpose:

This function shows the status for a monitor capture queue of a receiver port.

This function is applicable for capture control modes single and continuous (no recording).

Input:

AiUInt32 ul_QueueId

Queue Identifier. Valid Queue Identifiers are get via the ***FdxCmdMonQueueControl*** command.

Output:

TY_FDX_MON_QUEUE_STATUS_OUT
****px_QueueStatusOut***

Pointer to structure of output data

```
typedef struct {
   TY_FDX_E_MON_STATUS e_Status;
   AiUInt32 ul_FramesToRead;
   AiUInt32 ul_BytesToRead;
} TY_FDX_MON_QUEUE_STATUS_OUT;
```

TY_FDX_E_MON_STATUS e_Status

Reflects the actual status of the Monitor Queue.

Define	*Description*
FDX_MON_STAT_EMPTY	The Queue is empty
FDX_MON_STAT_ FILLED	There are frames to read
FDX_MON_STAT_ FULL	The queue is full. In case of a continuous capture queue data shall be read immediately to prevent overflow.
FDX_MON_STAT_ OVERFLOW	This is only applicable for continuous capture mode. The internal receiver task was not able to write available data to the queue.
FDX_MON_STAT_ ERROR	Any kind of Capture queue Error.

AiUInt32 ul_ FramesToRead

Returns number of Frames actually in Queue to read out.

AiUInt32 ul_ BytesToRead

Returns number of Bytes needed to read out the actually number of Frames in Queue. This value can be higher than the real number of frame data, because each frame buffer is internally aligned to 64 Byte.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.6.4 FdxCmdMonTCBSetup

Prototype:

AiReturn FdxCmdMonTCBSetup (AiUInt32 ul_Handle,
AiUInt32 ul_TCBIndex,
const TY_FDX_MON_TCB_SET
****px_MonTCBSet);***

Purpose:

This function is used to setup one Monitor Trigger Control Block (TCB), specifying a trigger event as trigger condition. See Figure 4.4.3.7-1 , which shows the dependencies and functionality of the trigger engine. This function is also used to setup a Hardware Trigger Control Block for APX-GNET 2/4 Board. For this case some limitations mut be cared fore. The main limitation is, that there can only ba one hardware TCB per port

Input:

AiUInt32 ul_TCBIndex

Index of the Monitor Trigger Control Block to setup. Valid range is 1...253 (FDh).
For HW TCB ul_TCBIndex can only be 1.

TY_FDX_TRG_TCB_SET *p x_MonTCBSet

Structure to setup a Trigger Control Block.

```
typedef struct {
   AiUInt32 ul_TrgType;
   TY_FDX_MON_TCB_SET_GEN x_GenTrg;
   TY_FDX_MON_TCB_SET_ERR x_ErrTrg;
   AiUInt32 ul_NextTrueIndex;
   AiUInt32 ul_NextFalseIndex;
   AiUInt32 ul_TriggerBits;
   AiUint32 ul_TCBEx;
   TY_FDX_IRIG_TIME x_TriggerTime;
   AiUInt32 ul_TimeDuration;              /* Time duration in ms */
   AiUInt32 ul_TriggerVl;                 /* VL for specified Trigger */
   } TY_FDX_MON_TCB_SET;
```

AiUInt32 ul_TrgType

Trigger type, which shall be evaluated for this Trigger Control Block.

(The following Trigger Types can be combined in order to logically OR the Trigger Types for one TCB)

Value	*Description*	*HW*
FDX_TRG_ERROR	Trigger on Error Error Specification given by *x_ErrTrg* structure	☺
FDX_TRG_EXTERNAL	Trigger on External strobe.	☺
FDX_TRG_GENERIC	Generic Data pattern to evaluate, described by the *x_GenTrg* structure	☺
FDX_TRG_VL	Trigger on reception of a VL = Trigger on any received frame	☺
FDX_TRG_VL_DEFINED [1)]	Trigger on reception of a frame with a defined VL. The VL identifier must be specified in the parameter ul_TCBEx.	☺
FDX_TRG_TIME_ABSOLUTE	Trigger if received Time Tag is equal or newer than the appointed absolute Trigger Time	
FDX_TRG_TIME_DURATION	Trigger on reception of a frame if the Time Duration is expired, since the TCB becomes active.	
GNET_TRG_HW_BASED	Special trigger mode for APX-GNET. All trigger favilities are transferred to hardware. This mode must be combined with one of the above marked Trigger Modes. This is only cabaple for TCB No 1 bacause for this mode only one TCB is available. By setting up TCB 1 in this way the The Trigger Logic of the APX-GNET Board will automatically set up for this mode.	

[1)] Only applicable for APX-GNET

TY_FDX_MON_TCB_SET_GEN x_GenTrg

structure, which contains generic trigger specification

```
typedef struct {
   AiUInt32 ul_GenTrgType;
   AiUInt32 ul_GenBytePos;
   AiUInt32 ul_GenTrigMask;
   AiUInt32 ul_GenTrigComp;
} TY_FDX_MON_TCB_SET_GEN;
```

AiUInt32 ul_GenTrgType

Defines the generic trigger type

Value	*Description*
FDX_TRG_TCB_GEN_STD	Trigger if Frame Data at *ul_GenBytePos*, masked with *ul_GenTrigMask* is equal to *ul_GenTrigComp*
FDX_TRG_TCB_GEN_INV	Trigger if Frame Data at *ul_GenBytePos*, masked with *ul_GenTrigMask* is *not* equal to *ul_GenTrigComp*

AiUInt32 ul_GenBytePos

Defines the generic trigger type byte position. See also description and figure of parameter x_VLExtendedFilter of function ***FdxCmdRxVLControl***.

AiUInt32 ul_GenTrigMask

Defines the generic trigger type mask. See also description and figure of parameter x_VLExtendedFilter of function ***FdxCmdRxVLControl***.

AiUInt32 ul_GenTrigComp

Defines the generic trigger type compare value.

TY_FDX_MON_TCB_SET_ERR x_ErrTrg

structure, which contains error trigger specification

```
typedef struct {
   AiUInt32 ul_ErrType;
}TY_FDX_MON_TCB_SET_ERR;
```

AiUInt32 ul_ErrType

Describes the error trigger condition (see following Error Type constants, which can be combined in order allow Error Trigger condition on multiple errors)

Error Type Constant	*Error Description*	*Abbreviation*
FDX_RX_ERROR GNET_RX_ERROR	Wrong physical Symbol during frame reception.	PHY
FDX_PREAMBLE_ERROR	Wrong Preamble/Start Frame Delimiter received.	PRE
FDX_TRIP_NIBBLE_ERROR	Unaligned Frame length received	TRI
FDX_CRC_ERROR GNET_CRC_ERROR	MAC CRC Error.	CRC
FDX_SHORG_IFG_ERROR GNET_SHORG_IFG_ERROR	Short Interframe Gap Error (<960ns)	IFG
FDX_IP_ERROR GNET_IP_ERROR	AFDX IP Framing Error (AFDX-IP frame specific settings violated).	IPE
FDX_MAC_ERROR GNET_MAC_ERROR	AFDX MAC Framing Error (AFDX-MAC frame specific settings violated).	MAE
FDX_NO_VALID_SFD	Frame without valid Start Frame Delimiter received	SFD
FDX_LONG_FRAME_ERROR GNET_LONG_FRAME_ERROR	Long Frame Received (> 1518 Bytes)	LNG
FDX_SHORT_FRAME_ERROR GNET_SHORT_FRAME_ERROR	Short Frame Received (< 64 Bytes)	SHR
FDX_VL_FRAME_SIZE_ERROR GNET_VL_FRAME_SIZE_ERROR	VL specific Frame size Violation	VLS
FDX_SEQUENCE_NO_ERROR GNET_SEQUENCE_NO_ERROR	Sequence No. Mismatch	SNE
FDX_TRAFFIC_SHAP_ERROR GNET_TRAFFIC_SHAP_ERROR	Traffic Shaping Violation	TRS

Note: It is strictly recommended to use the GNET_ defines for the APX-GNET board because the defines are slightly different to the FDX_ defines. The use of wrong defines can cause unpredictable results.

AiUInt32 ul_NextTrueIndex

Index of the next TCB to be evaluated after the condition for this TCB is **true**.

AiUInt32 ul_NextFalseIndex

Index of the next TCB to be evaluated after the condition for this TCB is **false**

AiUInt32 ul_TriggerBits

Bit	*Description*
0 – 7	Trigger Bits, which are cleared if TCB evaluation becomes true
8-15	Trigger Bits, which are set if TCB evaluation becomes true
16-31	Reserved

AiUInt32 ul_TCBEx

Defines extended parameter of the TCB

Bit	*Description*

0	0: disable external strobe 1: assert external strobe if TCB evaluation is true
1	0: disable interrupt 1: assert interrupt if TCB evaluation is true
2-3	Reserved
4	Only applicable for GNET_TRG_HW_BASED mode 0: **OR** conditions of hardware trigger block 1: **AND** conditions of hardware trigger block
5-31	Reserved

TY_FDX_IRIG_TIME x_TriggerTime

Absolute Trigger Time in IRIG format. Trigger becomes true, if time of received frame is newer or equal to this time.

AiUInt32 ul_TimeDuration

Time Duration in milliseconds [ms]. Time starts running when the TCB becomes active.

AiUint32 ul_TriggerVl

Specify the identifier of the VL for trigger type FDX_TRG_VL_DEFINED. This is only valid for APX-GNET,

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.
Error Codes: FDX_ERR

Figure 4.4.3.7-1 shows the dependencies of the trigger engine and the related commands for setting up the corresponding items.

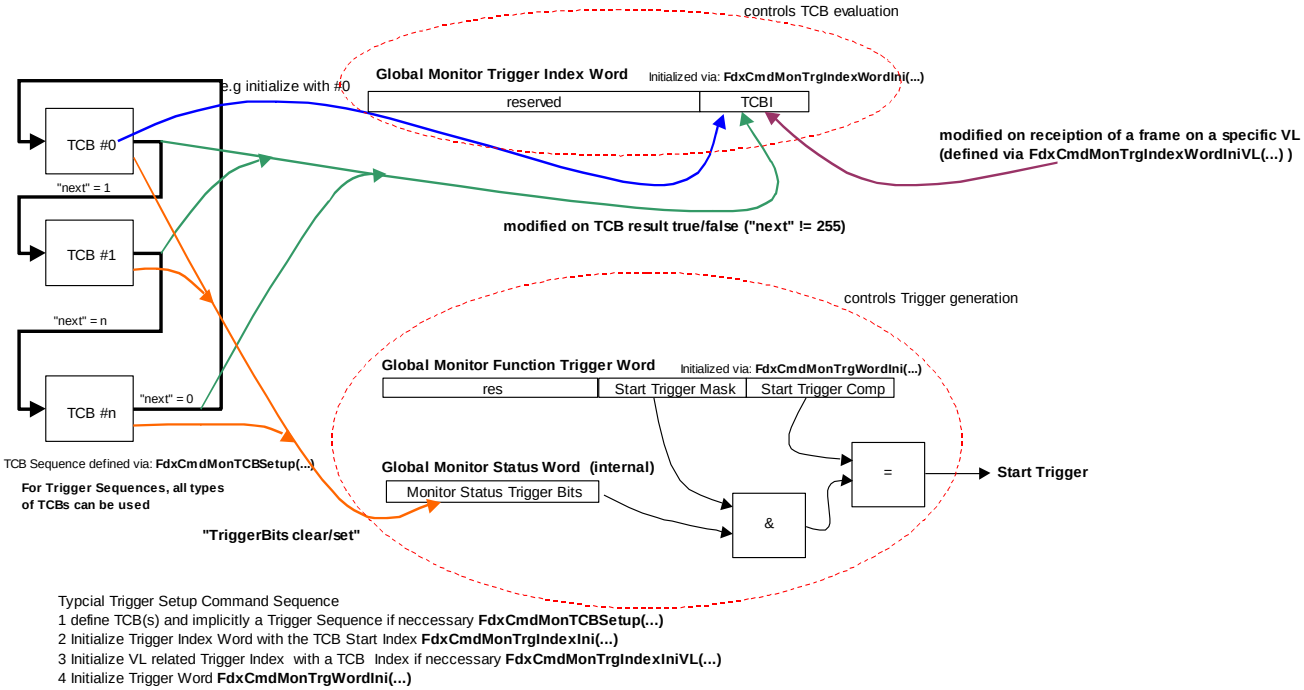


Figure 4.4.3.7-1: Trigger Engine Dependencies

4.4.6.5 **FdxCmdMonTrgIndexWordIni**

Prototype:

AiReturn FdxCmdMonTrgIndexWordIni (AiUInt32 ul_Handle
AiUInt32 ul_TCBIndexIni);

Purpose:

This function initializes the Trigger Index Word with the given Trigger Control Block Index value. See Figure 8 for a diagram of the dependencies and functionality of the trigger engine.

Input:

AiUInt32 ul_TCBIndexIni

This value defines the initial Trigger Control Block (TCB) Index, which is processed by the firmware after Start of Receiver Operation. A value of zero disables any TCB processing by the firmware. See also the FdxCmdMonTCBSetup command ul_NextTrueIndex/ul_NextFalseIndex parameter of the *TY_FDX_MON_TCB_SET* structure.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.6.6 **FdxCmdMonTrgIndexWordIniVL**

Prototype:

AiReturn FdxCmdMonTrgIndexWordIniVL (AiUInt32 ul_Handle
AiUInt32 ul_VLId,
AiUInt32 ul_TCBIndex);

Purpose:

This function initializes the Trigger Index Word with the given Trigger Control Block Index value. See Figure 4.4.3.7-1 for a diagram of the dependencies and functionality of the trigger engine.

Input:

AiUInt32 ul_VLIdr

Defines the associated VL identifier.

AiUInt32 ul_TCBIndex

This value defines the Trigger Control Block (TCB) Index which is written to the Monitor Trigger Index Word if a frame has been received for the given VL. A value of FFh disables any modification of the Trigger Index Word if a frame for the given VL is received.

A value of 0, disables the complete Trigger Control Block Processing with reception of the given VL.

Valid range is 0...FDh and FFh (FEh reserved for internal use, ASP).

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.6.7 **FdxCmdMonTrgWordIni**

Prototype:

AiReturn FdxCmdMonTrgWordIni (AiUInt32 ul_Handle
const TY_FDX_MON_TRG_WORD_INI
****px_MonTrgWordIni);***

Purpose:

This function initializes the Monitor Trigger Word. See Figure 4.4.3.7-1 which shows the dependencies and functionality of the trigger engine.

Input:

TY_FDX_MON_TRG_WORD_INI *px_MonTrgWordIn

Pointer to a structure which contains, the Trigger Mask- and Trigger Compare Values for Rx Start- and Stop Trigger. See also the FdxCmdMonTCBSetup command *ul_TriggerBits* parameter of the *TY_FDX_MON_TCB_SET* structure.

```
typedef {
   AiUInt32 ul_StartTriggerMask;
   AiUInt32 ul_StartTriggerComp;
   AiUInt32 ul_StopTriggerMask;
   AiUInt32 ul_StopTriggerComp;
} TY_FDX_MON_TRG_WORD_INI;
```

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.7 Continuous Capture Second Edition Functions

The following functions for Continuous Capture Second Edition (CCSE) are designed for a more performant way of continuously capturing data from an AIM AFDX board. The intention and gloabal different design difference to other continous capture mode is, that memory can provided at the user interface to the onboard Target software. The regulare handling assumes, that several buffers will be provided at start time. So in case of occurrence of date, the Target software can directly write tha data to the provided buffer. If a data buffer is filled and available for the user he will be informed by a callback. At this callback several actions can be scheduled to compute the received data but the essention action would be to provide a next buffer to use for the Target software. It is also possible to read in parallel the status of the receiver to get information about received data.

At the moment these functionality is only available on special parts of the supported platforms. Have a look to the following table to get an overview.

Table of Availability

	API-FDX	***AMC-FDX***	***APM-FDX***	***APX-GNET***	***fdXTap***	***APU-FDX***
Windows	•	•		•		
Linux	•	•				•
VxWorks (VME)						

4.4.7.1 FdxCmdMonQueueContCapControl

Prototype:

AiReturn FdxCmdMonQueueContCapControl (AiUInt32 ul_Handle,
const TY_FDX_MON_QUEUE_CONTCAP_CTRL_IN
****px_MonQueueContCapControl,***
TY_FDX_MON_QUEUE_CTRL_OUT *px_QueueCtrlOut);

Purpose:

This function is used to control the extended continuous capture mode of the Monitor.

This function controls creation or deletion of queues, associated with the chronological monitor. The function returns a queue ID which must be used for all continuous capture Queue related functions.

Operating mode must be defined by using ***FdxCmdRxModeControl*** and ***FdxCmdMonCaptureControl*** before using this function. This function is only valid if capture mode is set to FDX_MON_CONTINUOUS_SE by the function ***FdxCmdMonCaptureControl***. For this case capture data will be captured in BIU related memory buffer, which is associated to generate data transfer to the host, every quarter buffer is filled. Additionally user defined condition for data transfer (Timeout, Trigger or Host command) can be defined. By occurrence of one of these user defined conditions data transmission will be forced for a provided data buffer, regardless of bulk of available data. Before Receiver is started User memory for data storage must be provided by function ***FdxCmdMonContCapProvideMemory*** to prevent lost of data. The user will be informed about available data with the associated call-back function provided with the call of this control function.

Input:

TY_FDX_MON_QUEUE_CONTCAP_CTRL_IN *px_MonQueueContCapControl

Pointer to a Continuous Capture Setup structure with following members.

```
typedef AiReturn  (*TY_FUNC_PTR_FDX_CONT_CAP_CALLBACK)(
            AiUInt32 ul_Handle,
            TY_FDX_MON_CONTCAP_PROVIDE_MEM x_ContCapBufferInfo      ,
            AiUInt32 ul_Info);

typedef struct {
   AiUInt32 ul_QueueCtrl;
   AiUInt32 ul_QueueId;
   AiUInt32 ul_QueueMemorySize;
   AiUInt32 ul_Timeout;
   AiUInt32 ul_TriggerTCBIndex;
   TY_FUNC_PTR_FDX_CONT_CAP_CALLBACK pf_CaptureCallback;
} TY_FDX_MON_QUEUE_CONTCAP_CTRL_IN;
```

AiUInt32 ul_QueueCtrl

Queue Control Code

Value:	*Description:*
FDX_MON_CC_QUEUE_CREATE	Create queue, associated chronological buffer
FDX_MON_CC_QUEUE_DELETE	Delete Queue

AiUInt32 ul_QueueId

Queue Identifier to delete (only needed if the Queue Control Code is set to **FDX_MON_CC_QUEUE_DELETE** .

AiUInt32 ul_QueueMemorySize

Memory which shall be used for the internal global Monitor Buffer for the BIU firmware in Global RAM. Setting this value to 0 will force to use an internal defined value for this size.

AiUInt32 ul_Timeout

Defines a time out value in milliseconds to force data transmission to the host. This value shall be in a range form 1 to 0xFFFFFFFF ms. For a value higher than 10ms you would get a high-performance by setting a value which is dividable by 10.

A value of 0 will result in no Timeout what means received data will be transferred if a quarter of the internal global Monitor Buffer for the BIU firmware is filled. This is a quarter of ul_GlobMonSize which is retuned by this function.

AiUInt32 ul_TriggerTCBIndex

Number of a Trigger Control Block which shall force data transmission to the host. This assumes that this Trigger Control Block is correctly set up before referencing here. Valid numbers for Trigger control blocks are in range from 1 to 253.

A value of 0 indicates no usage of a Trigger Control Block.

TY_FUNC_PTR_FDX_CONT_CAP_CALLBACK pf_CaptureCallback

Callback function provided by the user for signaling completion of reception of data for one provided buffer. This function will be called after received date is transferred by DMA from the board to the host memory.

If this function gets called and this buffer was not marked with parameter FDX_CC_MEM_REUSABLE, the user should provide a new buffer to the board with function ***FdxCmdMonContCapProvideMemory***.

The following parameters will be passed to that call back function.

AiUInt32 ul_Handle

Id which was generated as identifier for this Queue. See output parameter of this function ***FdxCmddMonQueueContinuousCaptureControl***.

TY_FDX_ MON_CONTCAP_PROVIDE_MEM x_ContCapBufferInfo

This structure is a copy of the information to the memory buffer which was provided with the function ***FdxCmdMonContCapProvideMemory***. All the information will be passed through for identification of the buffer. Excepting the parameter ul_BufferSize. This will be modified with the real size of data copied to this buffer.

AiUint32 ul_Info

This parameter gives additional information about the Data Transfer or the buffer.

Value:	***Description:***
FDX_MON_CC_QUEUE_INF_NO	No additional information
FDX_MON_CC_QUEUE_INF_TRUNC	Data is truncated and must be followed by a next buffer.
FDX_MON_CC_QUEUE_INF_CONTINU	Data is a continuation of a preceded buffer which was truncated.
FDX_MON_CC_QUEUE_INF_CONTRUN	This is a combination of the values FDX_MON_CC_QUEUE_INF_TRUNC and FDX_MON_CC_QUEUE_INF_CONTINU which means this is a fragment of a buffer preceded with a buffer before and followed by the next buffer.

The information values FDX_MON_CC_QUEUE_INF_TRUNC, FDX_MON_CC_QUEUE_ INF_CONTINU and FDX_MON_CC_QUEUE_INF_CONTRUN will be used if a provided buffer is too small for transferring all data which is available on the board. In this case the data will be truncated and transferred with the next available buffer.

Output:

*TY_FDX_MON_QUEUE_CTRL_OUT *px_QueueCtrlOut*

Pointer to structure of output data

```
typedef struct {
   AiUInt32 ul_QueueId;
   AiUInt32 ul_GlobMonSize;
} TY_FDX_MON_QUEUE_CTRL_OUT;
```

AiUInt32 ul_QueueId

Valid Queue Identifier.

AiUInt32 ul_GlobMonSize

Size of the global Monitor Buffer for the BIU firmware in Global RAM. This value should be taken in account for providing receive memory buffers.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.7.2 FdxCmdMonContCapProvideMemory

Prototype:

AiReturn FdxCmdMonContCapProvideMemory (AiUInt32 ul_Handle,
const TY_FDX_MON_CONTCAP_PROVIDE_MEM
****px_MonContCapProvideMem);***

Purpose:

This function is used to provide user memory for enhanced continuous capture mode. To prevent loss of received data some rules for setting up this memory shall be considered.

If expecting huge data traffic the best performance will be achieved by setting up buffers which have a quarter of the used global Monitor Buffer for the BIU firmware in Global RAM.

The provided buffers shall have in a minimum a half of the used global Monitor Buffer for the BIU firmware in Global RAM. Better is to provide more

Input:

TY_FDX_MON_CONTCAP_PROVIDE_MEM
****px_MonContCapProvideMem***

Pointer to a Continuous Capture Buffer Setup structure to provide Receiver Memory.

```
typedef struct {
   AiUInt32 ul_QueueId;
   AiUInt32 ul_MemoryQalifier;
   AiUInt32 ul_UserIdent;
   AiAddr pv_Memory;
   AiUInt32 ul_BufferSize;
} TY_FDX_MON_CONTCAP_PROVIDE_MEM;
```

AiUInt32 ul_QueueId

Queue Identifier which is returned on creation of the queue to identify the Queue.

AiUInt32 ul_MemoryQalifier

This parameter is to set information for the handling of the memory buffer.

Value:	*Description:*
FDX_CC_MEM_DEFAULT	Normal use of the Data Buffer which means after writing to the Data Buffer it will no longer be used by the Library.
FDX_CC_MEM_REUSABLE	After writing to the Data Buffer the user must be copy the buffer because this buffer will be kept in a cyclic queue of buffers.

AiUInt32 ul_UserIdent

A User Identifier which can be set by user for easy buffer identification.

AiAddr pv_Memory

Pointer to the start of the memory buffer in host environment.

AiUInt32 ul_BufferSize

Size of the provided Memory buffer in Byte. The buffers shall have in a minimum a quarter of the used global Monitor Buffer for the BIU firmware in Global RAM. The size of this global Monitor Buffer is returned by function ***FdxCmdMonQueueContCapControl***.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.7.3 FdxCmdMonContCapInvalidateMemory

Prototype:

AiReturn FdxCmdMonContCapInvalidateMemory (AiUInt32 ul_Handle,
const TY_FDX_MON_CONTCAP_PROVIDE_MEM
****px_MonContCapInvalidateMem);***

Purpose:

This function is used to cancel provided memory which is no longer needed. This is to prevent accidentally write to memory which is already freed on host. So this function shall be called before freeing memory which was not written by the target and is not longer used.

Input:

TY_FDX_MON_CONTCAP_PROVIDE_MEM *px_MonContCapInvalidateMem

Pointer to a Continuous Capture Buffer Setup structure to invalidate a memory block. Here the same structure is used as for function **FdxCmdMonContCapProvideMemory** but some of the parameters are not used for invalidation.

```
typedef struct {
   AiUInt32 ul_QueueId;
   AiUInt32 ul_MemoryQalifier;
   AiUInt32 ul_UserIdent;
   AiAddr pv_Memory;
   AiUInt32 ul_BufferSize;
} TY_FDX_MON_CONTCAP_PROVIDE_MEM;
```

AiUInt32 ul_QueueId

Queue Identifier which you got on creation of the queue to identify the Queue.

AiUInt32 ul_MemoryQalifier

Not needed for this call.

AiUInt32 ul_UserIdent

A User Identifier which can be set by user for easy buffer identification.

AiAddr pv_Memory

Pointer to the start of the memory buffer in host environment

AiUInt32 ul_ul_BufferSize

Not needed for this call.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.4.7.4 FdxCmdMonContCapForceDateTransfer

Prototype:

AiReturn FdxCmdMonContCapForceDateTransfer (AiUInt32 ul_Handle, const TY_FDX_MON_CONTCAP_FORCE_TRANSFER *px_MonContCapInvalidateMem);

Purpose:

This function is used to force transfer of available received data from API-FDX-2 board memory to the previously provided receiver memory. This function will initiate transfer of all received and not transferred data up to receiving this command.

Input:

TY_FDX_MON_CONTCAP_FORCE_TRANSFER *px_MonContCapForceTansfer

Pointer to a Command structure.

```
typedef struct {
   AiUInt32 ul_QueueId;
} TY_FDX_MON_CONTCAP_FORCE_TRANSFER;
```

AiUInt32 ul_QueueId

Queue Identifier which you got on creation of the queue to identify the Queue.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.5 Target Independent Administration Functions

This section contains a collection of useful functions often used by writing a program which uses the Application Interface Library of the FDX board or functions, necessary to handle some cases of programming.

Table 4.5-I: Target Independent Administration Functions

Function	*Description*
FdxCmdFreeMemory	Frees memory, allocated by the Library, in the proper manner
FdxFwIrig2StructIrig	Converts an IRIG time in the format used by the Firmware to a structured format.
FdxStructIrig2FwIrig	Converts an IRIG time in the structured format to the format used by the Firmware.
FdxAddIrigStructIrig	Adds two IRIG time structures
FdxSubIrigStructIrig	Subtracts two IRIG time structures
FdxTranslateErrorWord	Translates a FW encoded Error Word for Error Information on Receiver Side
FdxInitTxFrameHeader	Supports a default initialization of a Transmit Header Structure, needed in Generic Transmit Mode
FdxProcessMonQueue	Processes data read via FdxCmdMonQueueRead.

4.5.1 FdxAddIrigStructIrig and FdxSubIrigStructIrig

Prototype:

TY_FDX_IRIG_TIME FdxAddIrigStructIrig (const TY_FDX_IRIG_TIME *px_IrigTimeA
const TY_FDX_IRIG_TIME *px_IrigTimeB);

TY_FDX_IRIG_TIME FdxSubIrigStructIrig (const TY_FDX_IRIG_TIME *px_IrigTimeA
const TY_FDX_IRIG_TIME *px_IrigTimeB);

Purpose:

These two functions are used to calculate time tag sums and differences.

Result = IRIG Time A + IRIG Time B (add) or
Result = IRIG Time A - IRIG Time B (sub).
(Calculates with 366 Days / Year)

Input:

TY_FDX_IRIG_TIME *px_IrigTimeA

Format see FdxFwIrig2StructIrig function above.

TY_FDX_IRIG_TIME *px_IrigTimeB

Format see FdxFwIrig2StructIrig function above.

Output:

None

Return Value

TY_FDX_IRIG_TIME. The result of the IRIG time calculation. Format can be absolute or relative (see definition of TY_FDX_IRIG_TIME above)

Px_IrigTimeA	Operation	Px_IrigTimeB	Result	Function
Absolute	Add	Absolute	Relative	FdxAddIrigStructIrig
Absolute	Add	Relative	Absolute	FdxAddIrigStructIrig
Relative	Add	Absolute	Absolute	FdxAddIrigStructIrig
Relative	Add	Relative	Relative	FdxAddIrigStructIrig
Absolute	Sub	Absolute	Relative	FdxSubIrigStructIrig
Absolute	Sub	Relative	Absolute	FdxSubIrigStructIrig
Relative	Sub	Absolute	Absolute	FdxSubIrigStructIrig
Relative	Sub	Relative	Relative	FdxSubIrigStructIrig

4.5.2 FdxCmdFreeMemory

Prototype:

AiReturn FdxCmdFreeMemory(void * vp_MemPointer, AiUInt32 ul_StructId);

Purpose:

This function releases memory (previously allocated by other Application Interface Library Functions) in a proper manner.

Input:

void * vp_MemPointer

A pointer to an information list or an information array.

- If a pointer to an information list element, this must be the pointer to the first entry of the information list. The function will release the memory of each element in that list.
- If a pointer to an information array, this must be the pointer to the start of the array.

The application programmer should take care to insure that all memory allocated by an Application Interface Library function is freed prior to termination of the application.

AiUInt32 ul_StructId

Identification of the type of memory to be freed.

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.5.3 FdxFwIrig2StructIrig

Prototype:

AiReturn FdxFwIrig2StructIrig (const TY_FDX_FW_IRIG_TIME *px_FwIrigTime,
TY_FDX_IRIG_TIME *px_IrigTime)

Purpose:

This function can be used to convert an IRIG time in the format used by the Firmware to a structured format. This function is helpful since the format of the IRIG time used by the firmware is different to the format often used in the Application Library.

Input:

TY_FDX_FW_IRIG_TIME *px_FwIrigTime

```
typedef struct {
   AiUInt32 ul_TtHigh;
   AiUInt32 ul_TtLow;
} TY_FDX_FW_IRIG_TIME;
```

AiUInt32 ul_TtHigh;

Timetag word in firmware format. The higher part of the time tag, containing the minutes of hour, hours of day and day of year.

Firmware Timetag High				
31 24	**23 ..... 20**	**19 ..... 11**	**10 ..... 6**	**5 ..... 0**
Reserved	Sub-micro seconds [1)]	Days of year 1 to 365	Hours of day 0 to 23	Minutes of hour 0 to 59

[1)] Only valid for APX-GNET in steps of 100ns.

Firmware Timetag Low		
31 26	**25 ..... 20**	**19 ..... 0**
Minutes of hour 0 to 59	Seconds of minute 0 to 59	Microseconds of second 0 to 999.999

AiUInt32 ul_TtLow;

Timetag word in firmware format. The lower part of the time tag, containing the Microseconds of second, seconds of minutes and minutes of hour.

Output:

*TY_FDX_IRIG_TIME *px_IrigTime*

IRIG Timecode Library structure

```
typedef struct {
   AiInt32 l_Sign;         // sign (0=absolute, 1=relative positive,
                           // -1=relative negative
                           // only needed for calculation)
                           // absolute=Irig format (day 1..366)
                           // relative=No Irig format (day 0..365)
   AiUInt32 ul_Hour;       // 0..23
   AiUInt32 ul_Min;        // 0..59
   AiUInt32 ul_Second;     // 0..59
   AiUInt32 ul_Day;        // 1..366
   AiUInt32 ul_MilliSec;   // 0..999
   AiUInt32 ul_MicroSec;   // 0..999
   AiUInt32 ul_NanoSec;    /* 0..900 in steps of 100 */
   AiUInt32 ul_Info;
} TY_FDX_IRIG_TIME;
```

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.5.4 FdxInitTxFrameHeader

Prototype:

AiReturn FdxInitTxFrameHeader (TY_FDX_TX_FRAME_HEADER *px_TxFrameHeader)

Purpose:

This function initializes a Transmit Frame Header Structure for Standard Frame (No Instruction Type). This structure is used for defining a Generic Transmit Queue entry with the ***FdxCmdTxQueueWrite*** function.

Input:

TY_FDX_TX_FRAME_HEADER
****px_TxFrameHeader***

```
typedef struct {
   AiUInt8   uc_FrameType;
   TY_FDX_TX_FRAME_ATTRIB x_FrameAttrib;
   TY_FDX_TX_INSTR_ATTRIB x_InstrAttrib;
} TY_FDX_TX_FRAME_HEADER;


typedef struct {
   AiUInt16  uw_FrameSize;
   AiUInt32  ul_InterFrameGap;
   AiUInt32  ul_PacketGroupWaitTime;
   AiUInt8   uc_PayloadBufferMode;
   AiUInt8   uc_PayloadGenerationMode;
   AiUInt32  ul_BufferQueueHandle;
   AiUInt8   uc_ExternalStrobe;
   AiUInt8   uc_PreambleCount;
   AiUInt32  ul_Skew;
   AiUInt8   uc_NetSelect;
   AiUInt8   uc_FrameStartMode;
   AiUInt32  ul_PhysErrorInjection;
   AiUInt16  uw_SequenceNumberInit;
   AiUInt16  uw_SequenceNumberOffset;
} TY_FDX_TX_FRAME_ATTRIB;
```

Pointer to structure, which holds the Transmit Frame Header Information. See ***FdxCmdTxQueueWrite*** function. This structure is initialized as follows:

```
x_FrameAttrib.uc_FrameType = FDX_TX_FRAME_STD;
x_FrameAttrib.uc_NetSelect = FDX_TX_FRAME_BOTH;
x_FrameAttrib.uc_ExternalStrobe = FDX_DIS;
x_FrameAttrib.uc_FrameStartMode = FDX_TX_FRAME_START_IFG;
x_FrameAttrib.uc_PayloadBufferMode = FDX_TX_FRAME_PBM_STD;
x_FrameAttrib.uc_PayloadGenerationMode = FDX_TX_FRAME_PGM_USER;
x_FrameAttrib.uc_PreambleCount = FDX_TX_FRAME_PRE_DEF;
x_FrameAttrib.ul_BufferQueueHandle = 0;
x_FrameAttrib.ul_InterFrameGap = 25;           // (1us)
x_FrameAttrib.ul_PacketGroupWaitTime = 1000;   // (1ms)
x_FrameAttrib.ul_PhysErrorInjection = FDX_TX_FRAME_ERR_OFF;
x_FrameAttrib.ul_Skew = 0;
x_FrameAttrib.uw_FrameSize = 0;
x_FrameAttrib.uw_SequenceNumberInit = FDX_TX_FRAME_SEQ_INIT_AUTO;
x_FrameAttrib.uw_SequenceNumberOffset = FDX_TX_FRAME_SEQ_OFFS_AUTO;
```

Output:

TY_FDX_TX_FRAME_HEADER
****px_TxFrameHeader***

Initialized structure (see above)

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.5.5 FdxProcessMonQueue

Prototype:

AiReturn FdxProcessMonQueue (AiUInt32 ul_ReadQualifier, AiUInt32 ul_Entries, void* pv_MonQueueData)

Purpose:

This function processes data, read via ***FdxCmdMonQueueRead*** in order to use the ***TY_FDX_FRAME_BUFFER_HEADER*** structure for access to the Frame Header Information. The processing mainly affects platform dependent variable interpretation / structure access (Little/Big Endian) of the Frame Header Information. Frame Data remains unchanged. This function is typically used in conjunction with ***FdxCmdMonQueueRead*** function.

Note: Data, read via ***FdxCmdMonQueueRead*** (without processing via this function) can be directly used for Replay via the ***FdxCmdTxQueueWrite*** function, if the corresponding Transmit Port has been configured for Replay. This avoids any data processing when using previously captured traffic for Replay.

Input:

ul_ReadQualifier

This parameter must match the ***ul_ReadQualifier*** parameter of previous ***FdxCmdMonQueueRead*** function calls, in order to tell this function which type of Queue Data is passed via ***pv_MonQueueData*** parameter.

Value:	*Description:*
FDX_MON_READ_FULL	***pv_MonQueueData*** points to a sequence of one or more Frame Header + Data information, but only Frame Headers are processed
FDX_MON_READ_HEADER	***pv_MonQueueData*** points to a sequence of one or more Frame Header only information

ul_Entries

This parameter must match the ***ul_EntryRead*** parameter of previous ***FdxCmdMonQueueRead*** function calls, in order to tell the function how many Entries (Header + Data or Header only) are passed via ***pv_MonQueueData*** parameter.

Output:

*void *pv_MonQueueData*

Processed Monitor Queue Data

Return Value

Returns FDX_OK on success or a negative error code on error.
Error Codes: FDX_ERR

4.5.6 FdxStructIrig2FwIrig

Prototype:

void FdxStructIrig2FwIrig (const TY_FDX_IRIG_TIME *px_IrigTime, TY_FDX_FW_IRIG_TIME *px_FwIrigTime);

Purpose:

This function can be used to convert an IRIG time in the structured format to the format used by the Firmware. This function is helpful since the format of the IRIG time used by the firmware is different to the format often used in the Application Library.

Input:

*TY_FDX_IRIG_TIME *px_IrigTime*

IRIG Timecode Library structure

```
typedef struct {
   AiInt32 l_Sign;         // sign (0=absolute, 1=relative positive,
                           // -1=relative negative
                           // only needed for calculation)
                           // absolute=Irig format (day 1..366)
                           // relative=No Irig format (day 0..365)
   AiUInt32 ul_Hour;       // 0..23
   AiUInt32 ul_Min;        // 0..59
   AiUInt32 ul_Second;     // 0..59
   AiUInt32 ul_Day;        // 1..366
   AiUInt32 ul_MilliSec;   // 0..999
   AiUInt32 ul_MicroSec;   // 0..999
   AiUInt32 ul_NanoSec;    /* 0..900 in steps of 100 */
   AiUInt32 ul_Info;
} TY_FDX_IRIG_TIME;
```

The l_sign is not relevant for this function, only for calculation (see below).

Output:

*TY_FDX_FW_IRIG_TIME *px_FwIrigTime*

IRIG Timecode Firmware format structure

```
typedef struct {
   AiUInt32 ul_TtHigh;
   AiUInt32 ul_TtLow;
} TY_FDX_FW_IRIG_TIME;
```

Format see FdxFwIrig2StructIrig function above.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.5.7 FdxTranslateErrorWord

Prototype:

AiReturn FdxTranslateErrorWord (AiUInt16 uw_ErrWord,
AiChar* puc_ErrStr,
AiUInt8 uc_ErrStrSize)

Purpose:

This function translates a Firmware specific Error Report Word into a string, containing 3-character wide abbreviations for the corresponding errors.

Input:

AiUInt16 uw_ErrorWord

Firmware Error Word with following layout. The constants are representing bit position within the Error Word.

Error Type Constant	*Error Description*	*Abbreviation*
FDX_RX_ERROR GNET_RX_ERROR	Wrong physical Symbol during frame reception.	PHY
FDX_PREAMBLE_ERROR	Wrong Preamble/Start Frame Delimiter received.	PRE
FDX_TRIP_NIBBLE_ERROR	Unaligned Frame length received	TRI
FDX_CRC_ERROR GNET_CRC_ERROR	MAC CRC Error.	CRC
FDX_SHORG_IFG_ERROR GNET_SHORG_IFG_ERROR	Short Interframe Gap Error (<960ns)	IFG
FDX_IP_ERROR GNET_IP_ERROR	AFDX IP Framing Error (AFDX-IP frame specific settings violated).	IPE
FDX_MAC_ERROR GNET_MAC_ERROR	AFDX MAC Framing Error (AFDX-MAC frame specific settings violated).	MAE
FDX_NO_VALID_SFD	Frame without valid Start Frame Delimiter received	SFD
FDX_LONG_FRAME_ERROR GNET_LONG_FRAME_ERROR	Long Frame Received (> 1518 Bytes)	LNG
FDX_SHORT_FRAME_ERROR GNET_SHORT_FRAME_ERROR	Short Frame Received (< 64 Bytes)	SHR
FDX_VL_FRAME_SIZE_ERROR GNET_VL_FRAME_SIZE_ERROR	VL specific Frame size Violation	VLS
FDX_SEQUENCE_NO_ERROR GNET_SEQUENCE_NO_ERROR	Sequence No. Mismatch	SNE
FDX_TRAFFIC_SHAP_ERROR GNET_TRAFFIC_SHAP_ERROR	Traffic Shaping Violation	TRS

Note: It is strictly recommended to use the GNET_ defines for the APX-GNET board because the defines are slightly different to the FDX_ defines. The use of wrong defines can cause unpredictable results.

AiChar puc_ErrStr*

Pointer to Array of characters, which will hold the error string after successful translation. The Error String will contain the error abbreviations for the occurred errors, separated by a forward slash **'/'** .

AiUInt8 uc_ErrStrSize

Size in bytes of the array of characters, which will hold the error string after successful translation.

Output:

AiChar* puc_ErrStr

Error String, written to the ***puc_ErrStr*** Pointer.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.5.8 GNetTranslateErrorWord

Prototype:

AiReturn GNetTranslateErrorWord (AiUInt16 uw_ErrWord,
AiChar* puc_ErrStr,
AiUInt8 uc_ErrStrSize);

Purpose:

This function translates a Firmware specific Error Report Word into a string, containing 3-character wide abbreviations for the corresponding errors.

This function is special to use with APX-GNET. For detailed description see function FdxTranslateErrorWord

4.5.9 FdxCreateRecIndex

Prototype:

AiReturn FdxCreateRecIndex(const AiChar *ac_RecordingFile,
const AiChar *ac_IndexFile,
PFN_FDXCREATEINDEX_CALLBACK
pfnCreateIndexCallback,
AiUInt32 ul_Granularity)

Purpose:

This function creates an index that corresponds to a recording file. The index can later be used to locate frames more easily. The index can either be written into a separate file or appended to the recording file.

Writing an index can take a lot of time. To visualize the progress and to give the user the the abilitity to abort the process, a callback function can be applied.

This function can also be performed on a recording file, that already contains index informations from a previous call to this function.

Input:

AiChar *ac_RecordingFile

The name of the recording file produced by FdxCmdMonQueueControl.

Note for remote recording:
To use this function direct access to the recording file is required. This can be achived by mounting the directory as network drive (Operating System functionality). The recording file is generated always locally on the server (where the fdx board is present). The FdxCreateRecIndex function does not use functionality of ANS to access the recording file.

AiChar *ac_IndexFile

The name of the index file to be produced. If ac_IndexFile is identical to ac_RecordingFile, the index table will be written to the end of the recording file.

PFN_FDXCREATEINDEX_CALLBACK
pfnCreateIndexCallback

```
typedef AiBoolean (*PFN_FDXCREATEINDEX_CALLBACK)
    (AiUInt32 ul_Percent, const AiChar *ac_RecordingFile,
      const AiChar *ac_IndexFile);
```

The address of a function which allows to display the index creation process or to abort the index creation. Returns TRUE on User-Abort, FALSE otherwise. Called whenever the percentage changes and the percentage is a multiple of ul_Granularity.

The recording and index filename will be passed over to the callback function. These informations can be used to identify the caller in multithreaded applications.

Special:

NULL no callback function available

Sample:

```
AiBoolean CreateIndexCallback(AiUInt32 ul_Percent,
      const AiChar *ac_RecordingFile,
      const AiChar *ac_IndexFile)
{
   printf("\rProgress: %d %%", ul_Percent);
   return kbhit();
}
```

If a user abort is requested and the index is written into the recording file, the already written index informations will be removed. In case of a different index file, the index file would be deleted.

ul_Granularity

Defines the granularity how often the percentage should be displayed. Valid values are 1..100. This allows to display the progress less often in case the display would have any negative impact on the recording performance.

E.g. a granularity of 5 would display the percentages 0, 5, 10, 15,….

Output:

none

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

FDX_ERROR_REC_FILE_CORRUPT
The recording file is corrupt. An illegal frame size has been encountered.

FDX_ERROR_CREATE_REC_INDEX_USER_ABORT
The user has aborted the index creation.

4.5.10 FdxSkipRecFileHeader

Prototype:

AiReturn FdxSkipRecFileHeader(AiHandle h_RecFile,
Ai_UInt64_Union *pu64_IndexPointer,
AiUInt32 *pul_HeaderSize)

Purpose:

This function helps you to move the filepointer forward to the recorded frames. Since the recording file header contains variable length data, skipping the header may be an awful job. It also returns the pointer to the index table within the file and the length of the file header.

Input:

AiHandle h_RecordingFile

A handle to a recording file. (For example the recording file produced by FdxCmdMonQueueControl.)

Output:

Ai_UInt64_Union *pu64_IndexPointer

The address of a 64 bit variable, that receives the file position of the index table. If this value is 0, the recording file does not contain an index table.

AiUInt32 *pul_HeaderSize

The address of a variable, that receives the length of the recording file header.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

THIS PAGE INTENTIONALLY LEFT BLANK

4.6 Reros Functions

4.6.1 FdxCmdRerosVLReroute

Prototype:

AiReturn FdxCmdRerosVLReroute(AiUInt32 ul_HandleRxPort,
const TY_FDX_REROS_VL_REROUTE_IN
****px_RerosVLRerouteIn);***

Purpose:

This API function is for setup and control of the routing of AFDX/ARINC664 frames, based on their Virtual Link (VL) number. The function requires a Port Handle to a Physical (Receive) Port, which is used for reception of all frames. Frames can be rerouted to one or more Physical (Transmit) Ports, which are identified via so called Port Maps. The Port Map numbers can be given by the user application at the **FdxCmdTxPortInit** and **FdxCmdRxPortInit** API functions.

For an AFDX/ARINC664 interface boards equipped with two Physical Ports, the frames can be rerouted to both of these Physical Ports.

The associated Receive and Transmit Ports must be configured for REROS mode (see **FdxCmdTxModeControl** and **FdxCmdRxModeControl** API function).

The **FdxCmdRxModeControl** function also supports the definition of a default Output (Transmit) Port Map, which is used for routing all frames (regardless of the VL number) to the associated Transmit Port. If this default Output Port Map is set to a value, which isn't assigned by the user application yet, no frames will be rerouted per default. So following scenarios can be easily implemented:

- Rerouting of all frames per default, selective disabling of VLs (and re-enabling)
- No rerouting of frames per default, selective enabling of VLs (and re-disabling)

The **FdxCmdTxModeControl** function supports the configuration of a constant delay for the rerouting operation via the **ul_RerosPortDelay** parameter. The delay can be given in 1ms steps. The delay is maintained by the on-board processing to eliminate any jitter during the re-routing. A value of 0 causes a rerouting "as fast as possible" but the re-routing processing time is dependent on the performed operations (e.g. modification rules, see below) any may introduce a jitter.

Input:

AiUInt32 ul_HandleRxPort

The Port Handle of the associated Receiver Port needs to be given here. The Port Handle is returned at the **FdxLogin** function via the ***pul_Handle** parameter.

const TY_FDX_REROS_VL_REROUTE_IN *px_RerosVLRerouteIn

This data structure configures the rerouting of frames for a given VL number. If multiple VLs are to be configured, the function has to be called for every VL number which is to be configured. The VL number is given via the **ul_VLId** function parameter.

```
typedef struct _fdx_reros_vl_reroute_in
{
   AiUInt32 ul_VLId;
   AiUInt32 ul_TxPortMapsArrSize;
   AiUInt32 * pul_TxPortMaps;
} TY_FDX_REROS_VL_REROUTE_IN;
```

AiUInt32 ul_VLId

Virtual Link number (0….65535)

AiUInt32 ul_TxPortMapsArrSize

Number of Tx Port Maps entries given by the following parameter (0…2)

Note: A value of 0 (=passing no Port Map numbers) suppresses the routing of the given VL number.

AiUInt32 * pul_TxPortMaps

Pointer to an array of Port Map numbers associated with the Physical (Transmit) Port.

Note: The array size has to match with the number given via the previous parameter !

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.6.2 FdxCmdRerosParamCreate

Prototype:

AiReturn FdxCmdRerosParamCreate(AiUInt32 ul_HandleRxPort,
const TY_FDX_REROS_PARAMCREATE_IN *px_RerosParamCreateIn,
AiUInt32 * pul_ParamHandle);

Purpose:

This function is for setting up a modification rule for AFDX/ARINC664 frame data (Header and Payload), which is applied as part of the REROS functionality. Multiple modification rules can be configured by calling this function multiple times. The modification rule is applied before the frame is rerouted to the configured Transmit Port Map (see **FdxCmdRerosVLReroute** function), hence the modified frame will be re-transmitted on all associated Ports.

The modification rule is based on the definition of a so called “Parameter”, which identifies a max. 64-Bit wide field inside ADFX/ARiNC664 frame data (Header and Payload). This function defines such a “Parameter” and instantiates the necessary data structures on the interface board. Parameter types other than ‘rpfRaw’ format type see below allow a modification based on different engineering unit formats. E.g modification of AFDX payload (resp. UDP payload) data can be setup to be done in engineering unit format. Modification of the AFDX Frame Header (MAC, IP and UDP Headers) may be done via ‘rpfRaw’ formatted parameter types e.g. to easily access and modify a single bit or a bit field inside these headers.

Input:

AiUInt32 ul_HandleRxPort

The Port Handle of the associated Receiver Port needs to be given here. The Port Handle is returned at the **FdxLogin** function via the ***pul_Handle** parameter.

*const TY_FDX_REROS_PARAMCREATE_IN *px_RerosParamCreateIn*

```
/* FdxCmdRerosParamCreate */
typedef struct _fdx_reros_paramcreate_in
{
   TY_FDX_REROS_PARAM_SOURCE           x_Source;
   TY_FDX_REROS_PARAM_FORMAT           x_Format;
   TY_FDX_E_REROS_PARAM_CONTROL_MODE   e_ControlMode;
} TY_FDX_REROS_PARAMCREATE_IN;
```

TY_FDX_REROS_PARAM_SOURCE x_Source;

```
typedef struct _fdx_reros_param_source
{
   TY_FDX_QUINTUPLET x_Quint;
   AiUInt32 ul_BytePos;
   AiUInt32 ul_BitPos;
   AiUInt32 ul_BitLen;
   AiUInt32 ul_ExtFlags;
   AiUInt32 ul_ExtMultiPurpose;
} TY_FDX_REROS_PARAM_SOURCE;
```

TY_FDX_QUINTUPLET x_Quint;

The Quintuplet identifies the frame to be modified via its

VL Number, Source/Destination IP Address, Source/Destination UDP Port Number

Note: The Quintuplet normally identifies a unique AFDX message by explicitly stating VL Number, IP Source/Destination Addresses and UDP Source/Destination Port numbers. However each IP Address and UDP Port Number setting of 'x_Quint' also accept 0xFFFFFFFF as a "wildcard",e.g. in order to apply a modification to all messages on a given VL regardless of the IP Addresses and UDP Port numbers (e.g. all set to 0xFFFFFFFF).

AiUInt32 ul_BytePos;

Defines the Parameter Byte Position in the frame.

The Range for the Byte position is 0…<FrameSize-2>. The max. frame size is dependent on the associated VL and may vary at run time. Hence there is no cross check behind this function against any VL specific maximum frame sizes!

Notes:
Byte Position 0 = First Byte of the Frame, part of the MAC Destination Address Byte Position <FrameSize-2> = Last Payload Byte of the Frame, AFDX Sequence at Byte position <FrameSize-1> Number is excluded and cannot be modified via a Parameter !

AiUInt32 ul_BitPos;

Defines the Parameter starting Bit Position in the given Byte Position. Therefore the Range is 0….7.

AiUInt32 ul_BitLen;

defines the Parameter Lengths in Bits. The maximum length of the bit field is 64. Minimum length is 1.

AiUInt32 ul_ExtFlags;

This parameter can be used for multi-purpose settings.

In case of Boeing EDE environment it is used to indicate that the message is an EDE message.

For this case set this parameter to `FDX_REROS_EXT_ISEDE`

AiUInt32 ul_ExtMultiPurpose;

This parameter can be used for multi-purpose settings.

In case of Boeing EDE environment it is used to indicate that the message is an EDE message.
For this case Set this parameter to the source ID of the EDE message.

TY_FDX_REROS_PARAM_FORMAT x_Format;

```
typedef struct _fdx_reros_param_format
{
   TY_FDX_E_REROS_PARAM_FORMAT e_FormatType;
   AiDouble d_Scale;       /* scaling and offset is NOT applied for
                              rpfRaw-Types */
   AiDouble d_Offset;
} TY_FDX_REROS_PARAM_FORMAT;
```

TY_FDX_E_REROS_PARAM_FORMAT e_FormatType;

The format type defines the parameter de/encoding. Please note that single and double precision float parameter types do not allow other bit lengths than the ones shown in the table below since these are pre-defined standardized formats (IEEE754) ! The parameter format type also defines the conversion of the data values for limits and triggers passed for the modification control functions (see below function FdxCmdRerosParamControl……() .

```
typedef enum rerosParamFormat
{
   rpfRaw = 0,          /* 1...64 bit */
   rpfBCD,              /* 1...64 bit */
   rpfScaledSigned,     /* 1...64 bit */
   rpfScaledUnsigned,   /* 1...64 bit */
   rpfFloatSinglePrec,  /* 32 bit */
   rpfFloatDoublePrec   /* 64 bit */
} TY_FDX_E_REROS_PARAM_FORMAT;
```

Value:	***Description:***
rpfRaw	No special data encoding, variable size,
rpfBCD	Encoding of the configured bit field in 4-Bit BCD digits (0...9) or less (1,3,7) for the remainder bits. variable size, E.g. A 10-Bit field offers a range from 0...399, BCD encoded In addition the ‚d_Scale' and ‚d_Offset' values (see below) value = Offset+ Scale*<BCD decimal value>
rpfScaledSigned	2's complement binary encoding (1Bit Sign, <ul_BitLen-1> Magnitude), variable size. In addition the ‚d_Scale' and ‚d_Offset' values (see below) value = Offset+ Scale*<2's complement decimal value>
rpfScaledUnsigned	Unsigned binary encoding (<ul_BitLen> Magnitude), variable size. In addition the ‚d_Scale' and ‚d_Offset' values (see below) value = Offset+ Scale*<decimal value>
rpfFloatSinglePrec	IEEE754 Single precision float parameter* (1Bit sign, 8Bit exponent, 23Bit Mantissa), fixed size 32Bit
rpfFloatDoublePrec	IEEE754 Double precision float parameter* (1Bit sign, 11Bit exponent, 52Bit Mantissa), fixed size 64Bit

* Scaling and Offset can be applied to the Single and Double Precision Float formats.

AiDouble d_Scale;

A scaling value which is applied to the converted raw value by multiplying the converted value (see table above) with ‘d_Scale’.

AiDouble d_Offset;

An offset value which is applied to the converted raw value by adding ‘d_Offset’) to the converted and scaled value (see table above).

TY_FDX_E_REROS_PARAM_CONTROL_MODE e_ControlMode;

The control mode allows to configure the data modification operation for the given parameter, if it is controlled interactively by the application (see FdxCmdRerosParamControlInteractive()) or automatically by the board in accordance with the corresponding setup ((see FdxCmdRerosParamControlAutomatic().

```
typedef enum rerosParamControlMode
{
   rpcmAutomatic,
   rpcmInteractive
} TY_FDX_E_REROS_PARAM_CONTROL_MODE;
```

Value:	***Description:***
rpcmAutomatic	Automatic Parameter Modification
rpcmInteractive	Interactive Parameter Modification

Output:

AiUInt32 * pul_ParamHandle

The returned Parameter Handle can be used to control the associated modification rule functionality and to retrieve the status of the modification (see the following functions)

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.6.3 FdxCmdRerosParamStatus

Prototype:

AiReturn FdxCmdRerosParamStatus(AiUInt32 ul_HandleRxPort,
AiUInt32 ul_ParamHandle,
TY_FDX_REROS_PARAMSTATUS_OUT *
px_RerosParamStatusOut);

Purpose:

This function is to retrieve the status of a previously setup modification rule, see **FdxCmdRerosParamCreate** function.

Input:

AiUInt32 ul_HandleRxPort

The Port Handle of the associated Receiver Port needs to be given here. The Port Handle is returned at the **FdxLogin** function via the ***pul_Handle** parameter.

AiUInt32 ul_ParamHandle

The Parameter Handle identifies a previously setup modification rule via the **FdxCmdRerosParamCreate** function, which returns the Parameter Handle value.

Output:

*TY_FDX_REROS_PARAMSTATUS_OUT * px_RerosParamStatusOut*

```
/* FdxCmdRerosParamStatus */
typedef struct _fdx_reros_paramstatus_out
{
   TY_FDX_E_REROS_PARAM_CONTROL_MODE e_ControlMode;
   TY_FDX_E_REROS_TRIGGER_STATE      e_TriggerState;
   TY_FDX_E_REROS_MODIFICATION_STATE e_ModificationState;
   TY_FDX_REROS_PARAMERRINJ          x_ErrInj;
   AiUInt32 ul_ParamInRawLo;  /* raw input value */
   AiUInt32 ul_ParamInRawHi;
   AiDouble d_ParamInEU;      /* EU input value */
   AiUInt32 ul_ParamOutRawLo; /* raw output value */
   AiUInt32 ul_ParamOutRawHi;
   AiDouble d_ParamOutEU;     /* EU output value */
   AiUInt32 ul_ParamCnt;      /* count how many times was parameter
received
                                 (until   receiver started) */
   AiUInt32 ul_ParamRerosCnt; /* count modifications on parameter since
modification
                                 got active */
   AiUInt32 ul_ParamRerosAutoModTimeMs;   /* Time since modification start
for
                                           Automatic mode */
} TY_FDX_REROS_PARAMSTATUS_OUT;
```

TY_FDX_E_REROS_PARAM_CONTROL_MODE e_ControlMode

Returns the parameter control mode for the given parameter.

see FdxCmdRerosParamCreate() above.

```
typedef enum rerosParamControlMode
{
   rpcmAutomatic,
   rpcmInteractive
} TY_FDX_E_REROS_PARAM_CONTROL_MODE;
```

Value:	Description:
rpcmAutomatic	Automatic Parameter Modification
rpcmInteractive	Interactive Parameter Modification

TY_FDX_E_REROS_TRIGGER_STATE e_TriggerState

Shows the trigger state of a given parameter.

```
typedef enum rerosTriggerState
{
  rtsDisabled,
  rtsIsWaitingForStartTrigger,
  rtsIsWaitingForStopTrigger
} TY_FDX_E_REROS_TRIGGER_STATE;
```

Value:	Description:
rtsDisabled	No Trigger for an Automatic modification setup, see FdxCmdRerosParamControlAutomatic() and FdxCmdRerosParamCreate).
rtsIsWaitingForStartTrigger	Waiting for Start Trigger for an Automatic modification, see FdxCmdRerosParamControlAutomatic()
rtsIsWaitingForStopTrigger	Waiting for Stop Trigger for an Automatic modification, see FdxCmdRerosParamControlAutomatic()

TY_FDX_E_REROS_MODIFICATION_STATE e_ModificationState

See FdxCmdRerosParamControlAutomatic() and FdxCmdRerosParamControlInteractive() functions for the setup of the modification functions.

```
typedef enum rerosParamModificationState
{
   rpmsIsNotModified,
   rpmsIsLimitingToMin,
   rpmsIsLimitingToMax,
   rpmsIsModifiedByDynamicFunction,
   rpmsIsModifiedInteractive
} TY_FDX_E_REROS_MODIFICATION_STATE;

rpmsIsNotModified
rpmsIsLimitingToMin
rpmsIsLimitingToMax
rpmsIsModifiedByDynamicFunction
rpmsIsModifiedInteractive
```

Value:	Description:
rpmsIsNotModified	Parameter hasn't been modified yet
rpmsIsLimitingToMin	Parameter has been modified to its min. Limit
rpmsIsLimitingToMax	Parameter has been modified to its max. Limit
rpmsIsModifiedByDynamicFunction	Parameter has been modified by a dynamic function
rpmsIsModifiedInteractive	Parameter has been modified interactively

TY_FDX_REROS_PARAMERRINJ x_ErrInj

This parameter defines mode of physical error injection. These are the same modes as for standard transmit functions. For more details please refer to description of function **FdxCmdTxQueueWrite**

```
typedef struct _fdx_reros_paramerrinj
{
   AiUInt32 ul_ErrInjection;
} TY_FDX_REROS_PARAMERRINJ;
```

AiUInt32 ul_ErrInjection

Parameter for physical error injection For more details please refer to function FdxCmdTxQueueWrite.

AiUInt32 ul_ParamInRawLo

Returns the associated parameter raw input/receive value (before modification), lower 32-Bit part

AiUInt32 ul_ParamInRawHi

Returns the associated parameter raw input/receive value (before modification), higher 32-Bit part

AiDouble d_ParamInEU

Returns the associated parameter converted input/receive (engineering unit) value (before modification),

AiUInt32 ul_ParamOutRawLo

Returns the associated parameter raw output/transmit value (after modification), lower 32-Bit part

AiUInt32 ul_ParamOutRawHi

Returns the associated parameter raw output/transmit value (after modification), higher 32-Bit part

AiDouble d_ParamOutEU

Returns the associated parameter converted (engineering unit) output/transmit value (after modification),

AiUInt32 ul_ParamCnt

Returns the number of occurrences of the associated parameter (resp: receive count) since REROS was active

AiUInt32 ul_ParamRerosCnt

Returns the number of occurrences of the associated parameter (resp: modification count) since modification became active.

AiUInt32 ul_ParamRerosAutoModTimeMs

Elapsed time since modification of associated parameter became active

AiChar* puc_ErrStr

Error String, written to the ***puc_ErrStr*** Pointer.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.6.4 FdxCmdRerosParamControlAutomatic

Prototype:

AiReturn FdxCmdRerosParamControlAutomatic(AiUInt32 ul_HandleRxPort,
AiUInt32 ul_ParamHandle,
const TY_FDX_REROS_PARAM_CONTROL_AUTOMATIC_IN
****px_RerosParamControlAutomaticIn);***

Purpose:

This function supports a setup of an automatic control of a previously setup parameter modification rule, see **FdxCmdRerosParamCreate** function**.**

Input:

AiUInt32 ul_HandleRxPort

The Port Handle of the associated Receiver Port needs to be given here. The Port Handle is returned at the **FdxLogin** function via the ***pul_Handle** parameter.

AiUInt32 ul_ParamHandle

The Parameter Handle identifies a previously setup modification rule via the **FdxCmdRerosParamCreate** function, which returns the Parameter Handle value.

*const TY_FDX_REROS_PARAM_CONTROL_AUTOMATIC_IN *px_RerosParamControlAutomaticIn*

All values are necessary for automatic control of a parameter are summarised in the input structure described here.

```
/* FdxCmdRerosParamControlAutomatic */
typedef struct _fdx_reros_param_control_automatic_in
{  /*--- control which feature shall be applied ---*/
   AiBool32 b_ApplyDynFunction;
   AiBool32 b_ApplyStartTrigger;
   AiBool32 b_ApplyStopTrigger;
   AiBool32 b_ApplyMinLimit;
   AiBool32 b_ApplyMaxLimit;
   AiBool32 b_ApplyErrInjection;
   /*--- dynamic function ---*/
   TY_FDX_REROS_PARAMDYN_FUNCTION x_DynFunc;
   /*--- start trigger ------*/
   TY_FDX_REROS_PARAMTRIG_START x_StartTrig;
   /*--- stop trigger -------*/
   TY_FDX_REROS_PARAMTRIG_STOP x_StopTrig;
   /*--- limiter ------------*/
   AiDouble d_MinLimit;
   AiDouble d_MaxLimit;
   /*--- error injection ----*/
   TY_FDX_REROS_PARAMERRINJ x_ErrInj;
} TY_FDX_REROS_PARAM_CONTROL_AUTOMATIC_IN;
```

AiBool32 b_ApplyDynFunction

Enable/Disable a Dynamic Function the associated parameter modification (see ‘x_DynFunc’ below)

AiBool32 b_ApplyStartTrigger

Enable/Disable a StartTrigger condition or the associated parameter modification (see ‘x_StartTrig’ below)

AiBool32 b_ApplyStopTrigger

Enable/Disable a StopTrigger condition for the associated parameter modification
(see 'x_StopTrig' below)

AiBool32 b_ApplyMinLimit

Enable/Disable a min Value for the associated parameter modification
(see 'd_MinLimit' below)

AiBool32 b_ApplyMaxLimit

Enable/Disable a max Value for the associated parameter modification
(see 'd_MinLimit' below)

AiBool32 b_ApplyErrInjection

Enable/Disable a error injection for the associated parameter modification
(see 'x_ErrInj' below)

TY_FDX_REROS_PARAMDYN_FUNCTION x_DynFunc

Definition of a dynamic function applied to the defined parameter. The function is defined like:

$$y(t, x) = c1 + c2 * x + C3 * t$$

Coefficients c1, c2, c3 are given in the Engineering Units of the associated parameter.

t is the time in steps of 10 milliseconds in the range from start time.

x is the received value of associated parameter value (before modification)

```
typedef struct _fdx_reros_paramdyn_function
{  /* y(t, x) = c1 + c2*x +c3*t */
   AiDouble d_c1;
   AiDouble d_c2;
   AiDouble d_c3;
} TY_FDX_REROS_PARAMDYN_FUNCTION;
```

TY_FDX_REROS_PARAMTRIG_START x_StartTrig

Definition of a start trigger condition after which the change of the associated parameter is to be executed. The Start Trigger condition can be derived from another already defined Parameter (see FdxCmdRerosParamCreate() function) and its values.

```
typedef struct _fdx_reros_paramtrig_start
{
  TY_FDX_E_REROS_PARAMTRIG_START_MODE e_Mode;
  AiUInt32 ul_InternalTrgSrcParamHandle; /* Reference to Parameter
                                            for internal Trigger
                                            start modes */
  AiDouble d_Value;  /* used for Equal/NotEqual/GreaterThan/LessThan
                        comparison */
  AiDouble d_MinVal; /* used for InRange/OutOfRange comparisons */
  AiDouble d_MaxVal;
} TY_FDX_REROS_PARAMTRIG_START;
```

TY_FDX_E_REROS_PARAMTRIG_START_MODE e_Mode

This mode defines the Start mode for the Start Trigger condition in sense of:

```
typedef enum rerosParamTriggerStartMode
{
   rtstartmInternalEqual = 0,
   rtstartmInternalNotEqual,
   rtstartmInternalGreaterThan,
   rtstartmInternalLessThan,
   rtstartmInternalInRange,
   rtstartmInternalOutOfRange,
   rtstartmExternal,
   rtstartmDisabled, /* used for "manual" triggersequence:
                        disabled...immediate */
   rtstartmImmediate
} TY_FDX_E_REROS_PARAMTRIG_START_MODE;
```

"Issue Start Trigger if value of associated parameter ('ul_InternalTrgSrcParamHandle') is"

Value:	*Description:*
rtstartmInternalEqual	associated parameter = ‚d_Value'
rtstartmInternalNotEqual	associated parameter ≠ ‚d_Value'
rtstartmInternalGreaterThan	associated parameter > ‚d_Value"
rtstartmInternalLessThan	associated parameter < ‚d_Value"
rtstartmInternalInRange	‚d_MinVal' ≤ associated parameter ≤ ‚d_MaxVal'
rtstartmInternalOutOfRange	‚d_MinVal' > associated parameter > ‚d_MaxVal'
rtstartmExternal	Start Trigger is issued by external Trigger (on HW Trigger Input)
rtstartmDisabled	Start Trigger is disabled
rtstartmImmediate	Start Trigger issued right with calling this function

AiUInt32 ul_InternalTrgSrcParamHandle

Handle to a Parameter which triggers the Modification of the given Parameter (‘ul_ParamHandle’). Both are to be defined via FdxCmdRerosParamCreate() function.

AiDouble d_Value

Engineering Unit value for compare functions (eq, neq, gt, lt), see above

AiDouble d_MinVal

Engineering Unit value for compare functions (in range, out of range), Min Value

AiDouble d_MaxVal

Engineering Unit value for compare functions (in range, out of range), Max Value

TY_FDX_REROS_PARAMTRIG_STOP x_StopTrig

Defines a Stop Trigger Condition for the modification of the associated parameter.

```
typedef struct _fdx_reros_paramtrig_stop
{
   TY_FDX_E_REROS_PARAMTRIG_STOP_MODE e_Mode;
   AiUInt32 ul_MaxFrameCnt;
   AiUInt32 ul_MaxTimeMs;  /* Time in ms should be multiple of 10ms*/
   AiBool32 b_Retrigger;
} TY_FDX_REROS_PARAMTRIG_STOP;
```

TY_FDX_E_REROS_PARAMTRIG_STOP_MODE e_Mode

```
typedef enum rerosParamTriggerStopMode
{
   rtstopmMaxFrameCnt = 0,
   rtstopmMaxTime,
   rtstopmExternal,
   rtstopmDisabled, /* used for "manual" triggersequence:
                       disabled....immediate */
   rtstopmImmediate
} TY_FDX_E_REROS_PARAMTRIG_STOP_MODE;
```

Value:	*Description:*
rtstopmMaxFrameCnt	Stop Trigger is issued if ‚ul_MaxFrameCnt' reached
rtstopmMaxTime	Stop Trigger is issued if ‚ul_MaxTime' has elapsed
rtstopmExternal	Stop Trigger is issued by external Trigger (on HW Trigger Input)
rtstopmDisabled	Stop Trigger not active
rtstopmImmediate	Stop Trigger issued right with calling this function

AiUInt32 ul_MaxFrameCnt

Maximum number of Frames which are carrying the associated parameter after the modification (started by Start Trigger) will be stopped.

AiUInt32 ul_MaxTimeMs

Maximum time in milliseconds after the modification (started by Start Trigger) will be stopped.

AiBool32 b_Retrigger

Enable/Disable ReTriggering by a Start Trigger condition for the associated parameter.

AiDouble d_MinLimit

Engineering Unit Value for Min value

AiDouble d_MaxLimit

Engineering Unit Value for the Max value

TY_FDX_REROS_PARAMERRINJ x_ErrInj

This parameter defines mode of physical error injection. These are the same modes as for standard transmit functions. For more details please refer to description of function **FdxCmdTxQueueWrite**

```
typedef struct _fdx_reros_paramerrinj
{
   AiUInt32 ul_ErrInjection;
} TY_FDX_REROS_PARAMERRINJ;
```

AiUInt32 ul_ErrInjection

Parameter for physical error injection For more details please refer to function FdxCmdTxQueueWrite.

AiChar* puc_ErrStr

Error String, written to the ***puc_ErrStr*** Pointer.

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

4.6.5 FdxCmdRerosParamControlInteractive

Prototype:

AiReturn FdxCmdRerosParamControlInteractive(AiUInt32 ul_HandleRxPort,
AiUInt32 ul_ParamHandle,
const
TY_FDX_REROS_PARAM_CONTROL_INTERACTIVE_IN
****px_RerosParamControlInteractiveIn);***

Purpose:

This function is for an interactive control of a previously setup modification rule, see **FdxCmdRerosParamCreate** function**.**

This function allows to write/overwrite frame data during the rerouting process from application level in different modes (modify data only once or permanently).

Input:

AiUInt32 ul_HandleRxPort

The Port Handle of the associated Receiver Port needs to be given here. The Port Handle is returned at the **FdxLogin** function via the ***pul_Handle** parameter.

AiUInt32 ul_ParamHandle

The Parameter Handle identifies a previously setup modification rule via the **FdxCmdRerosParamCreate** function, which returns the Parameter Handle value.

*const TY_FDX_REROS_PARAM_CONTROL_INTERACTIVE_IN *px_RerosParamControlInteractiveIn*

In this structure the values here are summarised to control a parameter manually.

```
/* FdxCmdRerosParamControlInteractive */
typedef struct _fdx_reros_param_control_interactive_in
{
   TY_FDX_E_REROS_PARAM_CONTROL_INTERACTIVE_MODE e_Mode;
   AiDouble d_ParamOutEU;
   AiUInt32 ul_ParamOutRawLo; /* raw output value */
   AiUInt32 ul_ParamOutRawHi;
} TY_FDX_REROS_PARAM_CONTROL_INTERACTIVE_IN;
```

TY_FDX_E_REROS_PARAM_CONTROL_INTERACTIVE_MODE e_Mode

```
typedef enum rerosParamControlInteractiveMode
{
   rimApplyEU_Once,
   rimApplyEU_Permanent,
   rimApplyRaw_Once,
   rimApplyRaw_Permanent,
   rimStop /* use to stop permanent mode */
} TY_FDX_E_REROS_PARAM_CONTROL_INTERACTIVE_MODE;
```

Value:	Description:
rimApplyEU_Once	Modification of Engineering Unit value is applied only once to associated parameter
rimApplyEU_Permanent	Modification of Engineering Unit value is applied permanently to associated parameter
rimApplyRaw_Once	Modification of raw value is applied only once to associated parameter
rimApplyRaw_Permanent	Modification of raw value is applied permanently to associated parameter
rimStop	Modification is stopped (e.g. in case it was invoked permanently)

AiDouble d_ParamOutEU

Engineering Unit value which replaces the associated Parameter's received value

AiUInt32 ul_ParamOutRawLo

Raw value which replaces the associated Parameter's received value (Lo Part)

AiUInt32 ul_ParamOutRawHi

Raw value which replaces the associated Parameter's received value (Hi Part)

Output:

None

Return Value

Returns FDX_OK on success or a negative error code on error.

Error Codes: FDX_ERR

5. NOTES

5.1 Abbreviations

µsec	microseconds
AFDX	Avionic Full Duplex Switched Ethernet
ARINC	Aeronautical Radio, Incorporated
ARM	Advanced RISC Machine
ASCII	American Standard Code for Information Exchange
ASP	Application Support Processor
ASP	Application Support Processor
BAG	Bandwidth Allocation Gap
BIP	Bus Interface Unit Processor
BIT	Built IN Test
BIU	Bus Interface Unit
BSP	Board Support Package
DCT	Dynamic Counter Table
E/S	End System
FCS	Frame Check Sequence
FIFO	First in - First out
FS	Frame Size
GTM	Generic Transmit Mode
GTU	Gap Time Unit
I/O	Input / Output
IC	Integrity Checking
ID	Identifier
IFG	Inter-frame Gap
IP	Internet Protocol
IPP	process invalid frames
IRIG B	Inter Range Instrumentation Group, Time Code Format Type B
LCA	Xilinx Logic Cell Array (Field Programmable Logic)
LSB	Least Significant Byte
MAC	Medium Access Controller
Mbps	Mega bits per second
MCFL	Maximum Consecutive Frames Lost
MSB	Most Significant Byte
ns	nanoseconds
OIN	Open Information Network
OSI	Open System Interconnect
PBI	Physical Bus Interface
PC	Personal Computer.
PCI	Peripheral Component Interconnect
PGWT	Packet group wait time
PMC	PCI Mezzanine Card
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer.
RM	Redundancy Management
RMA	Redundancy Management Algorithm
RP(M)	Replay Mode
S/Q	Sampling & Queuing
SAP	Service Access Point
SCB	System Control Block
SFD	Start Frame Delimiter
SN	Sequence Number
STM	Simulator Transmit Mode

TAP	Test Access Point
TBD	To be defined
TCB	Monitor Trigger Control Block
TFTP	Trivial File Transfer Protocol
TS	Traffic Shaping
UDP	User Datagram Protocol
VL	Virtual Link
VME	Versatile Bus Modular European (computer bus)

5.2 Definition of Terms

address quintuplet	the address of an AFDX Comm port which consists of UDP Source/Destination, IP Source/Destination, and MAC Destination address (VL)
Big Endian	a system of memory addressing in which numbers that occupy more than one byte in memory are stored "big end first" with the uppermost 8 bits at the lowest address.
Channel	Two physical AFDX ports
Driver Command	command used by the AIM target s/w to control the AIM device
FLASH	page oriented electrical erasable and programmable memory
function	a self-contained block of code with a specific purpose that returns a single value.
Interframe Gap	Gap between the end of the preceding frame and the current frame.
interrupt	a signal from a device attached to a computer or from a program within the computer that causes the main program that operates the computer (the operating system) to stop and figure out what to do next
Little Endian	a system of memory addressing in which numbers that occupy more than one byte in memory are stored "little end first" with the lowest 8 bits at the lowest address.
multicast	Multicast is communication between a single sender and multiple receivers on a network.
Packet Group Wait Time	The time from the transmission start point of the last frame to the start point of the current frame with a resolution of 1us.
Port	One physical AFDX Port
Strobe	a strobe is a signal that is sent that validates data or other signals on adjacent parallel lines
Target	Refers to the software/communication active on the target device
unicast	Unicast is communication between a single sender and a single receiver over a network.