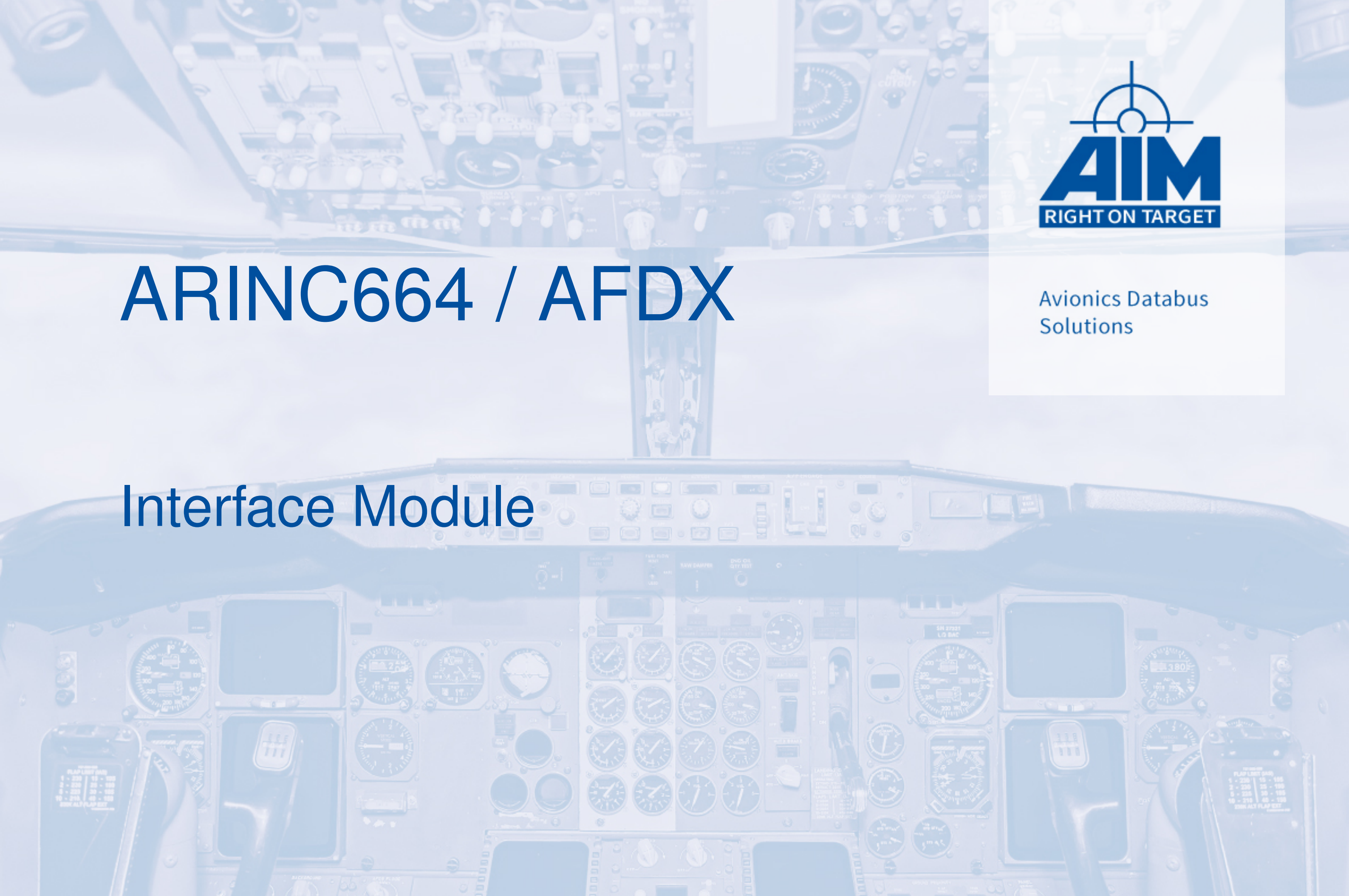


ARINC664 / AFDX

Avionics Databus Solutions

Interface Module

Reference Manual

Version 19-6-0
September, 2023

ARINC664 / AFDX

Interface Module

Reference Manual

Version 19-6-0
September, 2023

AIM NO. 60-15900-36-19-6-0

AIM – Gesellschaft für angewandte Informatik und Mikroelektronik mbH

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

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

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

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

TABLE OF CONTENTS

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. Below you can find a list of actual available and older AFDX Interface Modules.

Actual Standard Modules are:

APE-FDX-2	PCIe Module	(PCIe-based)
AMCX-FDX-2	PMC Module	(PCIe-based)
ASC-FDX-2	USB Module	
AXC-FDX-2	XMC Module	(PCIe-based)
ACE-FDX-2	CPCIe Module	(PCIe-based)

Older known Modules are:

API-FDX-2	PCI Module
AMC-FDX-2	PMC Module
fdXTap	USB Module for monitor only
APU-FDX-2	USB Module
ACC-FDX-2/4	Compact PCI (cPCI) 6U

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. The AIM AFDX Application Interface currently supports all 32-bit and 64-bit Windows® platforms (Win7 and newer).

The AIM AFDX High Level Application Interfaces for PCI and cPCI are 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 capable of interfacing DLLs (32-Bit and 64-Bit). Dynamic Libraries for LINUX (32-Bit and 64-Bit) are also available.

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. ARINC664 Part 7 Specification, published June 2005
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 Application Interfacing

2.1 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

2.2 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.
Error values can be translated into a text description using the function **FdxGetErrorDescription**. 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.

2.3 Necessary Files and Defines

For all platforms two C-syntax header files, **Ai_cdef.h** and **AiFfd_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**.

_AIM_WINDOWS For using the DLL (Windows applications) the aim_fdx.'x'.lib import library must be linked to the application. ('x' reflects actual version of dll)

The calling convention for AIM_FDX Application Interface DLLs is **cdecl**.
The AFDX import library is generated with Microsoft Visual C/C++ compiler and is available for 32-bit and 64 bit applications.

3 Function Reference

This chapter contains a reference for all ARINC 664 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 most functions is called **"ul_Handle"** and determines the ARINC 664 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

3.1 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.

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
FdxSetMDIMode	Switch the physical interface of a port between MDI and MDI-X mode

Table 3.1: Library Administration Functions

3.1.1 Platform Independent Functions

3.1.1.1 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.

3.1.1.2 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.

3.1.1.3 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. This information can be retrieved via the function FdxQueryLoginInfo(...).

```
#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 the resource to log in on the server. This resource can be either a board or a port of a board. The IDs of the available resources can be obtained by a calling the function FdxQueryServerConfig(...).

AiUInt32 ul_Privileges

Defines the access mode and access rights the client has to the selected resource. Currently only one value is possible/supported:

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 a physical 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.

3.1.1.4 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:

Unique handle to the resource, which can be either a board or a physical port.

Output:

None

Return Value:

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

3.1.1.5 FdxGetErrorDescription

Prototype:

```
const AiChar* FdxGetErrorDescription(AiInt32 fdx_error_code);
```

Purpose:

Returns the text description of a given ARINC 664 Library numeric error code to allow users to better understand the meaning of an error.

Input:

`AiInt32 fdx_error_code`

A numeric error code returned from an ARINC 664 Library function. Error codes are typically negative integers.

Output:

None

Return Value:

Returns a string (in the form of a const AiChar*) containing the text description.

3.1.1.6 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 **"Fdx-InstIntHandler"**.

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.

3.1.1.7 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_RX [3)]	Transmit Buffer / Instruction Base Address							
FDX_INT_RX [1)]	Reserved							
FDX_INT_TX [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_RX [3)]	Transmit Buffer / Instruction Base Address							
FDX_INT_RX [1)]	Receive / Monitor Buffer Pointer (Bits 25 - 0)							
FDX_INT_TX [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_RX [3)]	Transmit Buffer / Instruction Base Address							
FDX_INT_RX [1)]	Receive / Monitor Buffer Pointer (Bits 25 - 0)							
FDX_INT_TX [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_RX 3)	Transmit Buffer / Instruction Base Address							
FDX_INT_RX 1)	Receive / Monitor Buffer Pointer (Bits 25 - 0)							
FDX_INT_TX 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_TX3)	Time Tag High		
FDX_INT_RX1)			
FDX_INT_RX2)			

AiUInt32 ul_LWordF

Interrupt Loglist Event, Entry Word 6

TYPE	Bit 31		Bit 0
FDX_INT_TX3)	Time Tag Low		
FDX_INT_RX1)			
FDX_INT_RX2)			

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.

3.1.1.8 FdxQueryResource

Prototype:

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

Purpose:

To get detailed information about a specific 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 that hosts the resource. “local” for resources in the local PC.

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`

ID of the resource to query. Can be obtained by FdxQueryServerConfig(...).

Output:

`void *px_ResourceInfo`

Pointer to structure holding information of a resource. Dependent on the input resource ID the structure is of type TY_FDX_BOARD_RESOURCE or TY_FDX_PORT_RESOURCE.

In case of type TY_FDX_BOARD_RESOURCE:

```
typedef struct{
    AiChar ac_BoardName[MAX_STRING_1];
    AiUInt32 ul_BoardSerialNo;
    AiUInt32 ul_NumOfEthernetPorts;
    AiUInt32 Reserved;
    AiUInt32 ul_GlobalMemSize;
    AiUInt32 ul_SharedMemSize;
```

```
    AiUInt32 Reserved;
} TY_FDX_BOARD_RESOURCE;
```

AiChar ac_BoardName[]

String that contains a human readable description of the board. This description is in the format **[hardware type]-FDX-[variant]-[number of ports]**. With:

hardware type Hardware type of the device. E.g. API, APE, ASC,...

variant <empty>=default ARINC664 protocol support
B=variant with EDE protocol support
BTM=variant with EDE protocol and Time Manager support

number of ports number of physical ports

For example:
ASC-FDX-B2 = ASC device with EDE protocol support and two ports
APE-FDX-2 = APE device with 664 protocol support and two ports

AiUInt32 ul_BoardSerialNo

The Serial Number of the Board

AiUInt32 ul_NumOfEthernetPorts

Number of physical ports available on the board.

AiUInt32 Reserved

AiUInt32 ul_GlobalMemSize

Size of the global memory (firmware interface) in bytes

AiUInt32 ul_SharedMemSize

Size of shared memory in bytes.

AiUInt32 Reserved

void *px_ResourceInfo (In case of type TY_FDX_PORT_RESOURCE)

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

AiChar ac_PortName

Human readable name of the port. This string contains PortX" 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.

AiUInt32 Reserved

AiUInt8 uc_PortNo

Port number. Valid range is 1 (port 1) to 4 (port 4).

AiUInt8 Reserved

Return Value:

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

3.1.1.9 FdxInstallServerConfigCallback

Prototype:

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

Purpose:

This function is used to get notifications when a AIM ARINC664 device is plugged or unplugged from the system during run-time of your application.

Note:
This functionality is only available for APU-FDX-2, ASC-FDX-2 and fdxTap modules

Input:

`AiChar ac_SrvName`

Name of the computer where the function listens for device adding or removal. Has to be set to *'local'*, as functionality is not available for remote servers.

`FDX_SERVER_CALLBACK_FUNC *f_CallbackFunction`

Pointer to a function that is invoked as soon as a device is added or removed to/from the system. The prototype for the callback function is:

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 computer where the device has been added/removed.

`AiUInt32 ul_ChangeType`

Can be used to distinguish if a device has been either added or removed.

Value	Description
FDX_RESOURCE_CHANGE_DELETE	device has been removed
FDX_RESOURCE_CHANGE_ARRIVE	device has been added

`TY_RESOURCE_LIST_ELEMENT *px_ResourceList`

List of all newly added or removed resources. For the layout of this list, see FdxQuery-ServerConfig (3.1.1.10). In case of a device removal, only the member *ul_ResourceID* of each single *TY_RESOURCE_LIST_ELEMENT* structure is valid.

Note:
The callback function must not free any of the resource list entries i.e. call free() on one of them. Also, the resource list must not be accessed any more after completion of the callback routine, as it gets invalid.

Output:

None

Return Value:

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

3.1.1.10 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 contains information about this resource. Please refer to Section 3.1.1.8 "FdxQueryResource", parameter ac_BoardName for more information.

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.

3.1.1.11 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 the number and details of clients that are logged in to a specific resource.

Input:

`AiChar *ac_SrvName`

Address of the server that is hosting the AFDX resource for which the info is to be retrieved..

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 the AFDX board or port resource in question. The function provides information about all clients that are logged in to the resource specified by this ID.
Use FdxQueryServerConfig(...) to obtain the resource IDs of the resources provided by 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 for this array is allocated by the Application Interface Library. It must be freed by the calling application with the function FdxFreeMemory().

```
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 Section 3.1.1.3 "FdxLogin").

AiUInt32 ul_Info

Additional Information for future expansion.

AiUInt32 ul_StructId

Element which identifies the type of this structure (See Section 3.5.2 "FdxCmdFreeMemory")

Return Value:

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

3.1.1.12 FdxCmdPxiGeographicalAddressGet

Prototype:

```
AiReturn FdxCmdPxiGeographicalAddressGet(AiUInt32 ul_Handle,
AiUInt32* ul_location);
```

Purpose:

Queries the current geographical address of a board. This command applies only to boards in PXI systems.

Output:

AiUInt32* ul_location

This geographical address is the slot number of the board (identified by ul_Handle) within the PXI system.

Return Value:

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

3.2 Device Handling and Configuration

3.2.1 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.

3.2.2 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 read and set specific board and port parameters. These are:

- port speed
- single/redundancy mode
- MAC/IP header verification mode
- Discarding of erroneous frames

Input:

AiUInt32 ul_Control

Value	Description
FDX_READ	Read specific board and port parameters
FDX_WRITE	Set specific board and port parameters.

TY_FDX_BOARD_CTRL_IN* px_BoardControlIN

```
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[FDX_MAX_BOARD_PORTS]

Used for setting the configuration of each of the logical ports of the device. Available modes are:

Value	Description
FDX_SINGLE	The logical port is mapped to one physical port.
FDX_REDUNDANT	The logical port is mapped to two physical ports which are operated in redundant mode (e.g. all frames are transmitted on both physical ports). Only the first logical port can be used in this mode. The ARINC664 redundancy management algorithm (RMA) can be activated/deactivated by FdxCmdRxVlControl.
FDX_TAP	The logical port is mapped to one physical port. Each frame received on one port will be forwarded to the second port (Rxa→Txb, Rxb→Txa).In this mode it is not possible to send any frames but received frames can be captured and analyzed.

Note: This configuration shall only be changed if the ports are not in use

AiUInt32 aul_PortSpeed[FDX_MAX_BOARD_PORTS]

Value	Description
FDX_10MBIT	Network bit rate 10Mbit
FDX_100MBIT	Network bit rate 100Mbit
FDX_1000MBIT	Network bit rate 1Gbit (only APE/ACE/AXC/AMCX-FDX)
FDX_AUTO_100MBIT or FDX_AUTO_10MBIT	Port mode is set to auto negotiation. Port speed will be adapted to the maximum port speed of the other side of the connection. Note: Not supported on ASC-FDX-2

Note:
All existing ports must be set to same speed value.

Note:
When setting speed, ports will loose their physical links and it may take several seconds to re-establish them. It is recommended to check output parameter 'aul_GoodLink' periodically for link establishing being completed.

AiUInt32 aul_ExpertMode[FDX_MAX_BOARD_PORTS]

Option to configure the port error reporting. Default value is FDX_DISA_GHOST. It can be set to one of the following options:

Value	Description
FDX_DISA_GHOST	Disable Ghost Frames. Received frames containing more than four physical errors or no valid SFD (start frame delimiter) are filtered out during capture and are not visible to the user.
FDX_EXPERT_MODE	All received frames are displayed to the user. Received frames containing errors are not filtered out and the error information is visible to the user.

Note:
Not supported on ASC-FDX-2

AiUInt32 ul_RxVeriMode

Value to configure the receive verification mode. The AIM A664 board has the capability to check specified fields inside the MAC and IP header against constant values. If a received frame violates these fields, a MAC or IP Error will be reported with this frame.

Value	Description
FDX_BOARD_VERIFICATION_TYPE_DEFAULT	default verification is used.Independent of the board type the default is always FDX_BOARD_VERIFICATION_TYPE_A664
FDX_BOARD_VERIFICATION_TYPE_AFDX	AFDX specific verification is used.
FDX_BOARD_VERIFICATION_TYPE_A664	ARINC664 specific verification is used.
FDX_BOARD_VERIFICATION_TYPE_BOEING	Boeing specific verification is used. This mode is equal to A664
FDX_BOARD_VERIFICATION_TYPE_DEFINED	Verification registers have to be defined by customer by parameters aul_RxVeriData and aul_RxVeriMask

AiUInt32 aul_RxVeriData[8]

Compare values for receive frame check. Each element consists of 4 bytes. The first byte of the frame corresponds to the LSB of aul_RxVeriData[0], the second byte of the frame to the second lowest byte of aul_RxVeriData[0] and so on. 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[8]

Mask values for receive frame check. Each element consists of 4 bytes. The first byte of the frame corresponds to the LSB of aul_RxVeriMask[0], the second byte of the frame to the second lowest byte of aul_RxVeriMask[0] and so on. 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 is ignored.

The parameters aul_RxVeriData and aul_RxVeriMask are only used when verification mode FDX_BOARD_VERIFICATION_TYPE_DEFINED is used. For this case they must be set to proper values.

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 or set to FDX_BOARD_VERIFICATION_TYPE_A664 / FDX_BOARD_VERIFICATION_TYPE_BOEING. It also demonstrates how to set ul_RxVeriMode and aul_RxVeriMask when ul_RxVeriMode is set to FDX_BOARD_VERIFICATION_TYPE_DEFINED.

	FDX_BOARD_VERIFICATION_TYPE_								
	aul_RxVeriData		aul_RxVeriMask		Hex or Bin		Field Identification		
Byte	AFDX	A664 /BOEING	AFDX	A664 /BOEING	AFDX	A664 / BOEING			
0	03	03	FF	FF	03	03	Constant	MAC Dest	MAC Header
1	00	00	FF	FF	00	00			
2	00	00	FF	FF	00	00			
3	00	00	FF	FF	00	00			
4	00	00	00	00	XX	XX	VL		
5	00	00	00	00	XX	XX			
6	02	02	FF	FF	02	02	Const	MAC Source	
7	00	00	FF	FF	00	00			
8	00	00	FF	FF	00	00			
9	00	00	F0	00	X0	XX	Network ID		
10	00	00	00	00	XX	XX	Equipment ID		
11	00	00	1F	1F	XXX00000b	XXX00000b	Interface ID		
12	08	08	FF	FF	08	08	Type		
13	00	00	FF	FF	00	00			
14	45	45	FF	FF	45	45	Version / IHL		IP Header
15	00	00	FF	FF	00	00	Type of Service		
16	00	00	00	00	XX	XX	Total Length		
17	00	00	00	00	XX	XX			
18	00	00	00	00	XX	XX	Fragment ID		
19	00	00	00	00	XX	XX			
20	00	00	80	80	0XXXXXXXb	0XXXXXXXb	Control Flag /		
21	00	00	00	00	XX	XX	Frag Offset		
22	01	01	FF	FF	01	01	Time to live		
23	00	00	00	00	XX	XX	Protocol		
24	00	00	00	00	XX	XX	Header		
25	00	00	00	00	XX	XX	Checksum		
26	0A	0A	FF	FF	0A	0A	Constant	IP Source	
27	00	00	F0	00	0X	XX	Network ID		
28	00	00	00	00	XX	XX	Equipment ID		
29	00	00	E0	00	000XXXXXb	XX	Partition ID		
30	00	00	00	00	XX	XX		IP Dest	
31	00	00	00	00	XX	XX			

Table 3.2: Rx Verification Data and Mask

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 speed

Value	Description
FDX_100MBIT	Network bit rate 100Mbit
FDX_10MBIT	Network bit rate 10Mbit
FDX_1000MBIT	Network bit rate 1Gbit
FDX_AUTO_100MBIT	Auto negotiated rate is 100Mbit bit
FDX_AUTO_10MBIT	Auto negotiated rate is 10Mbit bit
FDX_AUTO_ERROR	Auto negotiation failed

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.

Note:
Not supported on ASC-FDX-2

AiUInt32 ul_SharedMemFree

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

Note:
Not supported on ASC-FDX-2

AiUInt32 ul_RxVeriMode

Rx error verification mode which is used by the Board

Note:
Not supported on ASC-FDX-2

Return Value:

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

3.2.3 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 time of the on-board clock of your device. This time is mainly used for time stamping of received frames.
The function can also be used to configure this clock for synchronization to an external, IRIG-B compliant time signal.

Input:

`AiUInt32 ul_Control`

Command code that specifies the action to be performed when calling the function:

Value	Description
FDX_IRIG_READ	Read the current time of the on-board clock
FDX_IRIG_WRITE	Set the time of the on-board clock. Not applicable if clock is configured for external synchronization.
FDX_IRIG_EXTERN	Synchronize the on-board clock to an external IRIG-B time signal. This is useful to synchronize several boards to a common time source.
FDX_IRIG_INTERN	On-board clock uses own, internally generated time. This is the default.

Note:
API-FDX, AMC-FDX and APU-FDX boards can not explicitly be set to external synchronization. They are able to detect external IRIG-B time signals and will automatically switch to the external synchronization mode.

Note:
On ASC-FDX boards, synchronization to external IRIG-B time signals can take up to 6.5 seconds. Setting the time using FDX_IRIG_WRITE while in internal mode is done immediately.

Note:
On API/AMC/APU/APE/ACE/AXC/AMCX boards, setting of time by command code FDX_IRIG_WRITE can take up to three seconds until it actually takes effect. Hence subsequent calls to FdxCmdIrigTimeControl using command code FDX_IRIG_READ will yield invalid times during this period. Capturing of frames during this period will lead to leaps in time when referring to the receive time stamps of frames.

Input/Output:

TY_FDX_IRIG_TIME *px_IrigTime

Pointer to a structure that holds the time to set or the current on-board clock time when reading.

```
typedef struct {
   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;       /* Currently not used */
} TY_FDX_IRIG_TIME;
```

Note:
On API/AMC/APU/APE/ACE/AXC/AMCX boards the ul_MilliSec, ul_MicroSec, ul_NanoSec values are ignored when setting the time and are set to 0 instead.

Note:
There are constraints concerning the resolution when reading the current time on different board types. These are:
API/AMC/APU-FDX: millisecond resolution
APE/ACE/AXC/AMCX-FDX: 100 nanosecond resolution
ASC-FDX: microsecond resolution

Output:

AiUInt32 *pul_Mode

Reflects the actual mode the on-board clock is set to.

Value	Description
FDX_IRIG_EXTERN	Clock is synchronized to external IRIG time signal.
FDX_IRIG_INTERN	Clock runs with internally generated time.
FDX_IRIG_EXTERN_NOT_SYNC[1)]	Clock runs in external synchronization mode but got no valid external signal.
FDX_IRIG_INTERN_NOT_SYNC[1)]	Clock runs with internally generated time but is currently not valid as clock is being adjusted due to a new time setting.

[1)] These modes are only available on APE-FDX, AMCX-FDX, AXC-FDX, ACE-FDX and also ASC-FDX boards.

Return Value:

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

3.2.4 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.

3.2.5 FdxReadBSPVersion

Prototype:

```
AiReturn FdxReadBSPVersion(AiUInt32 ul_Handle,
                           TY_FDX_BSP_VERSION  *px_BspVersion);
```

Purpose:

Note: This function is deprecated and will always set parameter ul_BspCompability to value FDX_BSP_NOT_COMPATIBLE. Use FdxVersionGet and FdxVersionGetAll to get information about software versions.

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

Since the function is deprecated, the returned compability status is now always FDX_BSP_NOT_COMPATIBLE.

Return Value:

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

3.2.6 FdxVersionGetAll

Prototype:

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

Purpose:

This function reads the version information of all available components for a specific board.

Input:

AiUInt32 ul_Count

Maximum number of versions to read. The ax_Versions array must be able to hold at least ul_Count elements.

Output:

TY_VER_INFO ax_Versions[]

Array of TY_VER_INFO elements to store the version information. The size of the array must be greater than or equal to ul_Count elements. The array must be allocated/deallocated by the application.

Each array element holds the version information for one specific component.

```
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;
```

AiUInt32 ul_VersionType

The version type number identifies the component the version refers to. The following enum type describes all possible values. Please note that not each version type is available on every platform.

```
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_FIRMWARE_PACKAGE_VER = 23,
    __AI_MAX_VERSIONS        /*! Indicates last entry. */
} TY_E_VERSION_ID;

#define AI_MAX_VERSIONS __AI_MAX_VERSIONS
```

Value	Description
AI_SYS_DRV_VER	Version information of the device driver on host system
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 information of BIU with ID x
AI_MAIN_LCA_VER	LCA version information of the main board
AI_IO_LCA_VER_BIUx	LCA version information of the BIU with ID x
AI_PCI_LCA_VER	PCI LCA version information
AI_TCP_VER	TCP version information
AI_BOOTSTRAP_VER	Bootstrap version information
AI_MONITOR_VER	Onboard Monitor Software version information
AI_TARGET_OS_VER	Onboard Target Operating System version information
AI_FPGA_VER	FPGA version information
AI_SUBDLLx_VER	Version Information of Sub-DLLx
AI_FIRMWARE_PACKAGE_VER	Firmware package version information (Note: This version is supported only for ASC-FDX boards, it is equal to BSP version.)

AiChar ac_Description[AI_DESCRIPTION_STRINGL]

Zero-terminated ASCII string that contains the name of the component. This string's size is limited to AI_DESCRIPTION_STRINGL.

AiUInt32 ul_MajorVer

Major version as a decimal number.

AiUInt32 ul_MinorVer

Minor version as a decimal number.

AiUInt32 ul_PatchVersion

Patch version as a decimal number.

AiUInt32 ul_BuildNr

Build number as a decimal number.

AiChar ac_FullVersion [AI_VERSION_STRINGL]

Zero-terminated ASCII version string. It contains the version in following format: major.minor.patch-'build'-'version extension'. 'build' and 'version extension' may be omitted. This string's size is limited to 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

Number of actually returned version information structures in array ax_Versions. The first ul_Count elements contain valid version informations.
This value may be less than or equal to the input parameter ul_Count.

AiUInt32 ul_MaxCount

Number of available version elements for the device. If this value is higher than the number of returned elements ul_Count there are more version elements available. To get all version elements read again with an increased ax_Versions array.

Return Value:

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

3.2.7 FdxVersionGet

Prototype:

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

Purpose:

This function is used to read the version information of a specific component.

Input:

TY_E_VERSION_ID eId

Version identifier of the component. For detailed description see FdxVersionGetAll.

Output:

TY_VER_INFO* px_Version

Pointer to structure that holds the version information. This memory must be allocated/deallocated by the application. For detailed description see FdxVersionGetAll.

Return Value:

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

3.2.8 FdxGetMDIMode

Prototype:

```
AiReturn FdxGetMDIMode(AiUInt32 port_handle,
                       FDX_MDI_MODE* mode);
```

Purpose:

Query the current MDI mode setting of a port.

Note:
This command is only available on APE/ACE/AXC/AMCX and ASC-FDX cards.

Input:

`AiUInt32 port_handle`

Handle to the port of which MDI mode shall be queried. Returned by FdxLogin (3.1.1.3)

`FDX_MDI_MODE* mode`

Returns the current MDI mode setting of a port. Please refer to detailed description in function FdxSetMDIMode (3.2.9).

Value	Description
FDX_MDI_AUTO	The pin assignment will be established automatically depending on the setting used by the remote station. This is the default mode
FDX_MDI_STRAIGHT	Pins 1 & 2 are assigned to transmitting. Pins 3 & 6 are assigned to receiving.
FDX_MDI_X	Pins 1 & 2 are assigned to receiving. Pins 3 & 6 are assigned fto transmitting.

Output:

None.

Return Value:

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

3.2.9 FdxSetMDIMode

Prototype:

```
AiReturn FdxSetMDIMode(AiUInt32 port_handle,
                       FDX_MDI_MODE mode);
```

Purpose:

Switch the physical interface of ports between MDI and MDI-X mode.

Note:
This command is only available on APE/ACE/AXC/AMCX and ASC-FDX cards.

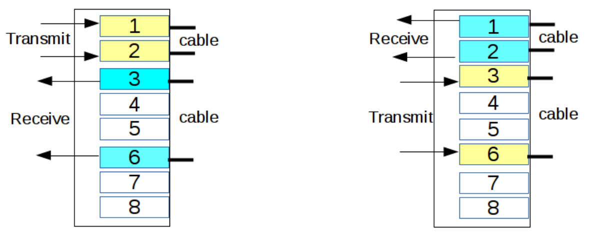
Input:

AiUInt32 port_handle

Handle to the port to configure the physical interface of. Returned by FdxLogin (3.1.1.3)

FDX_MDI_MODE mode

Medium-dependent interface setting. This refers to the pin assignment of the ARINC664 interface and determines what type of cable has to be used in order to connect the device to a remote station. If the interface of your device and of the remote station are set to same mode, then a cross-over cable has to be used. If they use different modes, a straight-through cable is needed. For example assuming your device is set to MDI and the remote station is set to MDI-X (usually a switch), a straight-through cable has to be attached to get a valid link.



Value	Description
FDX_MDI_AUTO	The pin assignment will be established automatically depending on the setting used by the remote station. This is the default setting
FDX_MDI_STRAIGHT	Pins 1 & 2 are assigned to transmitting. Pins 3 & 6 are assigned to receiving.
FDX_MDI_X	Pins 1 & 2 are assigned to receiving. Pins 3 & 6 are assigned fto transmitting.

Output:

None.

Return Value:

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

3.2.10 FdxCmdContolDiscreteIo

Prototype:

```
AiReturn FdxCmdContolDiscreteIo (AiUInt32 ul_Handle,
                                  TY_FDX_DISCRETE_IO_CONTROL_IN *px_DIoControl,
                                  TY_FDX_DISCRETE_IO_CONTROL_OUT *px_DIoStatus);
```

Purpose:

This function is used to control input and output discrete digital I/O lines of the AxE-FDX generation of cards. This includes APE/ACE/AXC/AMCX cards. Each of these cards provide 4 independent I/O pins. See corresponding HW Manual for pin out of connector.

Note:
This command is only available on APE/ACE/AXC/AMCX cards.

Input:

TY_FDX_DISCRETE_IO_CONTROL_IN *px_DIoControl

Input structure to specify the control and input parameters. The parameters ul_OutputValue and ul_OutputMask are only valid for control value DIO_WRITE or DIO_SETCONFIG.

```
typedef struct _gnet_discrete_io_control_in
{
    TY_FDX_E_DISCRETE_IO_MODE e_DIoControl;
    AiUInt32 ul_OutputValue;
    AiUInt32 ul_OutputMask;
} TY_FDX_DISCRETE_IO_CONTROL_IN;
```

TY_FDX_E_DIISCRETE_IIO_MODE e_DIIoConttroll

This is the control value to specify what to do by this command call

```
typedef enum gnet_e_discrete_io_mode {
    DIO_READ = 0,
    DIO_WRITE,
    DIO_GETCONFIG,
    DIO_SETCONFIG,
    DIO_RESET
} TY_FDX_E_DISCRETE_IO_MODE;
```

Value	Description	Comment
DIO_READ	Read discrete IO Values	
DIO_WRITE	Write discrete IO Values	
DIO_GETCONFIG	Get Channel configuration	Get configuration of I/O pins
DIO_SETCONFIG	Set Channel configuration	Set configuration of I/O pins
DIO_RESET		Not implemented

AiiUIIntt32 ull_OuttputtVallue

The contents of this value depends on the selected input control. For all cases only bit 0 .. 3 are valid.

In case of input control DIO_WRITE:

The bit value describes the values for the discrete outputs which shall be written. If bit is 1 value will be set to '1'. If bit is 0 value will be set o '0'
In case of input control DIO_SETCONFIG:

The bit value describes the configuration for the bit according the following table.

Bit Value	Description	Comment
0_b	I/O pin is input	(default)
1_b	I/O pin is output	

AiiUIIntt32 ull_OuttputtMask

Describes a Mask value to define which output bit shall be modified or left unchanged with the control DIO_WRITE. Only bits 0..3 of this mask are respected. If bit is 1 this value will be modified. If bit is 0 value will be unchanged.

Output:

TY_FDX_DISCRETE_IO_CONTROL_OUT *px_DIoStatus

This output structure for the command which reports different information dependant on the input control value.

```
    typedef struct _gnet_discrete_io_control_out
{
    AiUInt32 ul_Value;
    TY_FDX_IRIG_TIME x_SetTime;
} TY_FDX_DISCRETE_IO_CONTROL_OUT;
```

AiiUIIntt32 ull_Vallue

The contents of this output value depends on the selected input control and has therefore different values. For all cases only bit 0 .. 3 are valid.

In case of input control DIO_READ:

This value returns the actual state of the discrete digital IO lines. For input signals this value returns the state of the physical input signal. For output signals the actual set state is read back form the hardware

In case of input control DIO_WRITE:

Same information as for input control DIO_READ. The Values are read back after setting the new values.

In case of input control DIO_GETCONFIG:

Returns the configuration of the discrete digital IO lines where each digital IO line is described by one bit of the 32 Bit output value. See following table for information of the returned value.

Bit Value	Description	Comment
0_b	I/O pin is input	(default)
1_b	I/O pin is output	

TY_FDX_IIRIIG_TIIME x_SettTiime

This structure is only used if the input Control DIO_WRITE is used. It describes the time at which the output values are set. The time is determined by the built-in target software immediately after the output signal is changed. If no output signal is changed, the returned time is 0.

```
typedef struct _fdx_irig_time {
    AiInt32 l_Sign; /* sign 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 resolution 100ns*/
    AiUInt32 ul_Info;
} TY_FDX_IRIG_TIME;
```

Return Value:

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

3.3 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/ASC-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.

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

Table 3.3: Transmitter Functions

3.3.1 Global Transmitter Functions

3.3.1.1 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 a particular port.

Input:

TY_FDX_TX_MODE_CTRL *px_TxControl

Pointer to the structure that contains parameters for controlling 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 transmission

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

Note:
For ASC-FDX-2 FDX_START_TIME is not yet supported

AiUInt32 ul_Count

Only valid in transmit mode FDX_TX_GENERIC. Number of times the user defined frame sequence is sent. A value of 0 means the frame sequence is repeated endlessly..

TY_FDX_IRIG_TIME x_StartTime

Time when the transmission is started. (Absolute IRIG time). This structure is only applicable with the start mode FDX_START_TIME. The specified start time must be located in the 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.

Constant	Description
FDX_ESTOP_NOT_USED	Extended stop mode not used
FDX_ESTOP_EXT_TRG	Stop transmission when external trigger signal is detected
FDX_ESTOP_EXT_TRG_RESTART	Suspend transmission when external trigger signal is detected until the trigger signal is detected once again. This setting must only be used in combination with e_Startmode = FDX_START_TRG
FDX_ESTOP_TIME	Stop transmission after a time span. The time span is defined with ul_TransmitTime. Count down of this expiration time starts when transmission is actually started.

Note:
For ASC-FDX-2 FDX_ESTOP_TIME is not yet supported

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.

3.3.1.2 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 the structure that contains the port settings.

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

`AiUInt32 ul_TransmitMode`

There are five different transmit modes for a particular port.

Value	Description
FDX_TX_GENERIC	Generic Transmit Mode Allows to define a custom sequence of MAC frames to transmit to the network. Transmission is hardware scheduled, so this mode is applicable when needing a very precise timing for the sequence. Frame sequence can be automatically repeated an arbitrary number of times or even endlessly.
FDX_TX_REPLAY[1)2)]	Replay Transmit Mode This mode is used for replaying previously recorded ARIN664 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 its own parameters. The VL associated traffic shaping is supported
FDX_TX_FIFO[3)]	Fifo Transmit Mode Application can dynamically send frames to the network by writing to the transmit queue. Transmission is scheduled by application. This mode is applicable to construct flexible frame sequences on application side e.g. in request/response protocols.
FDX_TX_REROS [2)]	Rerouting Transmit Mode Transmit port is used for rerouting frames received by receiver port in REROS setup. For more details (See Section 3.6 "Reros Functions") FdxCmdRerosVLReroute.

Note:
1. If the board is configured in redundant operation, mode Replay is not available for the ports.
2. Replay and Reros modes are currently only available for API-, AMC- and APU-FDX-2.

AiUInt32 ul_RerosPortDelay

To guarantee a constant delay time from one port to other in Reros rerouting setup the delay time in milliseconds can be specified here.

Output:

None

Return Value:

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

3.3.1.3 FdxCmdTxPortInit

Prototype:

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

Purpose:

This function is used to reset the transmit functionality of the port to an initial state. The initial state is as follows:

- Transmitter is stopped
- 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 identification number. Only lowest 8 bits of the 32bit ul_PortMap value are used. This value is only used in reros mode, (See Section 3.6.1 "FdxCmdRerosVLReroute") FdxCmdRerosVLReroute.

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`

Number of logins that were performed on this port.

`AiUInt32 ul_GlobalMemFree`

Size of Global Memory (in Bytes) which is not already allocated. Only valid for AMC/API/APU-FDX cards.

`AiUInt32 ul_SharedMemFree`

Size of Shared Memory (in Bytes) which is not already allocated. Only valid for AMC/API/APU-FDX cards.

Return Value:

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

3.3.1.4 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 used in **generic** transmit mode, together with corresponding **payload generation modes** (See Section 3.3.2.4 "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

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

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

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.

3.3.1.5 FdxCmdTxStatus

Prototype:

```
AiReturn FdxCmdTxStatus(AiUInt32 ul_Handle,
                        TY_FDX_TX_STATUS* px_TxStatus);
```

Purpose:

This function is used to get the transmitter status of a certain port. It is useful for getting transmitter mode and number of transmitted frames.

Input:

None

Output:

TY_FDX_TX_STATUS* px_TxStatus

```
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

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 i.e. transmitter was started, but no frame has been sent yet.

AiUInt32 ul_Frames

Counter of transmitted frames. If the board is operated in redundant mode this counter shows the frames transmitted on network A.

AiUInt32 ul_FramesPortB

This counter is only applicable if the board is operated in redundant mode. In that case this counter shows the number of transmitted frames on network B.

AiUInt32 ul_TransmitMode

Current mode of the transmitter. This mode is selected with function FdxCmdTx-ModeControl. Refer to parameter ul_TransmitMode of that function, for possible values.

Return Value:

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

3.3.1.6 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.

3.3.1.7 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 **FdxCmdTxCreate-HiResVL**.

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.

3.3.2 Generic and Replay Transmitter Functions

3.3.2.1 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 applicable for creating a transmit queue that can be used for sending ARINC664 frames. Creation of such a queue is necessary in FDX_TX_GENERIC, FDX_TX_REPLAY and FDX_TX_FIFO modes as selected with FdxCmdTxModeControl function. Only one transmit queue per port can be created.

Input:

TY_FDX_TX_QUEUE_SETUP *px_TxQueueCreate

Pointer to the structure that contains the transmit queue settings.

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

AiUInt32 ul_QueueSize

Defines the queue size in byte. If this value is set to zero, an internal default queue size will be selected.

Output:

TY_FDX_TX_QUEUE_INFO *px_TxQueueInfo

Returns detailed parameters of the created transmit queue

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

AiUInt32 ul_QueueSize

This parameter returns the actual allocated size of the transmit queue in bytes.

`AiUInt32 ul_QueueBaseAddr`

Only valid for API/AMC/APU devices. 0 for ASC device

Return Value:

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

3.3.2.2 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.

3.3.2.3 FdxCmdTxQueueUpdate

Prototype:

```
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 FdxCmdTx-SubQueueWrite. 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 Transfer-Queue 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.

3.3.2.4 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 frame entries to a transmit queue. The new entries will always be added to the end of the transmit queue.
One entry consists of the actual MAC frame to send and a transmit header. The transmit header layout and fields depend on the transmit mode selected with FdxCmdTxModeControl.
The transmit header sets specific send options for the MAC frame.

Layout of one frame entry:

	Entry Layout
Transmit Header	Fixed Frame Header Layout dependent on ul_HeaderType and uc_FrameType parameter (see following description)
MAC Frame	ARINC664-FRAME data to transmit (dependent on the Payload Buffer and Payload Generation mode, see description below) (802.3 defines: 64 to 1518 bytes)

Note:
If frames defined with the same VL, the number of those frames shall not exceed 255. Otherwise automatic sequence numbering does not work.

Input:

`AiUInt32 ul_HeaderType`

This parameter defines the type of the transmit header and has to match the transmit mode selected with FdxCmdTxModeControl.

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, refer to chapter Section 3.3.2.5.1 "FDX_TX_FRAME_HEADER_GENERIC"
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 chapter Section 3.4.3.4 "FdxCmdMonQueueRead"
FDX_TX_FIFO	FDX_TX_FRAME_ HEADER_FIFO	FIFO Tx frame, only applicable in FIFO transmit mode. Layout of frame header follows the TY_FDX_TX_FRAME_HEADER_FIFO structure, refer to chapter Section 3.3.2.5.2 "FDX_TX_FRAME_HEADER_FIFO"

AiUInt32 ul_EntryCount

Number of frames to write to the transmit queue with this function call. There are constraints about this parameter dependent on the transmission mode:

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

AiUInt32 ul_WriteBytes

Size of pv_WriteBuffer in Bytes.

void *pv_WriteBuffer

Data buffer that contains the frame entries to write. The size of this buffer must equal ul_WriteBytes.

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.

Output:

None

Return Value:

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

3.3.2.5 Frame Header Definitions

3.3.2.5.1 FDX_TX_FRAME_HEADER_GENERIC

TY_FDX_TX_FRAME_HEADER x_TxFrameHeader

This C structure represents the generic frame header with all its fields:

```
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 entry that is written to the transmit queue:

Value:	Description:
FDX_TX_FRAME_STD	Standard ARINC664 frame to be sent
FDX_TX_FRAME_INSTR	Not a frame to be sent, but a special instruction **Note:** Not yet supported on ASC-FDX-2

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. ARINC664 compliant values are 64..1518. For frame length less than 60 no proper frame transmission is guaranteed.

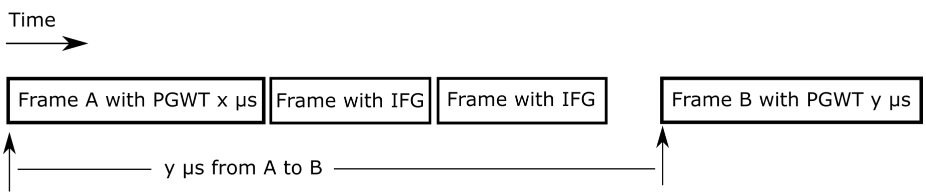
AiUInt32 ul_InterFrameGap

Transmission delay between start of this frame and end of preceding frame. One unit corresponds to 4 bit times of the current network speed. This is 40ns at 100Mbit/s operation mode and 400ns at 10Mbit/s operation mode. Maximum value for this setting is 16383, meaning up to approx. 655µs at 100Mbit/s operation mode or up to approx. 6,55ms at 10Mbit/s operation mode. If set to less than 24, the connected receive devices on the network may detect an 'Short interframe gap' error condition. This setting is only used if uc_FrameStartMode is set to FDX_TX_START_FRAME_IFG. If the packet group wait time is used, this field shall be initialized with zero. See also the notes for ul_Skew parameter in redundant mode.

AiUInt32 ul_PacketGroupWaitTime

The packet group wait time (PGWT) can be used to group frames and control the sending of these groups.

It is only used for frames that have their uc_FrameStartMode header field set to FDX_TX_START_FRAME_PGWT.
The PGWT value defines the time between the start point of the previous frame, that is set to FDX_TX_START_FRAME_PGWT, and the start point of the current frame. Resolution of this value is 1µs and the maximum possible value is 1.048.576µs.



The very first frame in a sequence that is set to FDX_TX_START_FRAME_PGWT will be transmitted immediately as there is no reference point. For Start Mode FDX_TX_START_FRAME_TRG_D this field describes the delay time between trigger pulse and start of frame in µs.

AiUInt8 uc_PayloadBufferMode

Prerequisites: To use payload buffer modes other than FDX_TX_FRAME_PBM_STD the uc_PayloadGenerationMode must be set to FDX_TX_FRAME_PGM_USER.
For payload buffer modes other than FDX_TX_FRAME_PBM_STD a separate buffer queue needs to be provided through the APIs FdxCmdTxBufferQueue..(). Either MAC, UDP or both header can be generated with data from the separate buffer queue.

Value:	Description:
FDX_TX_FRAME_PBM_STD	There is no payload generation. The complete frame data is taken from pv_WriteBuffer. ul_BufferQueueHandle must be set to NULL
FDX_TX_FRAME_PBM_MAC	MAC payload is provided in the separate buffer queue. The complete MAC header and the two static bytes of the IP header are taken from pv_WriteBuffer and the rest of the frame payload is taken from the separate buffer queue. ul_BufferQueueHandle must contain a valid buffer queue handle, previously allocated through the API FdxCmdTxBufferQueueAlloc. **Note:** Only available for API-, AMC- and APU-FDX-2
FDX_TX_FRAME_PBM_UDP	UDP payload is provided in the separate buffer queue. The complete MAC header, the IP header and 6 bytes of the UDP header are taken from pv_WriteBuffer and the remainder of the frame payload are taken from the separate buffer queue. ul_BufferQueueHandle must contain a valid buffer queue handle, previously allocated through the API FdxCmdTxBufferQueueAlloc. **Note:** Only available for API-, AMC- and APU-FDX-2
FDX_TX_FRAME_PBM_FULL	The full MAC frame is provided in the separate buffer queue. ul_BufferQueueHandle must contain a valid buffer queue handle, previously allocated through the API FdxCmdTxBufferQueueAlloc. **Note:** Only available for API-, AMC- and APU-FDX-2

The following table shows the necessary size of one queue entry dependent on the payload buffer modes.

Payload buffer mode	Size of queue entry
FDX_TX_FRAME_PBM_STD	sizeof (TY_FDX_TX_FRAME_HEADER) + uw_FrameSize
FDX_TX_FRAME_PBM_FULL	sizeof (TY_FDX_TX_FRAME_HEADER) Note: The full MAC Frame must be provided in separate Buffer.
FDX_TX_FRAME_PBM_MAC	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	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.

`AiUInt8 uc_PayloadGenerationMode`

Prerequisites: To use payload generation modes other than FDX_TX_FRAME_PGM_USER the uc_PayloadBufferMode must be set to FDX_TX_FRAME_PBM_STD.
The payload generation mode (PGM) defines, which content of the frame data is automatically generated with data from Tx static registers (see command Fdx-CmdTxStaticRegsControl).

For payload generation modes other than FDX_TX_FRAME_PGM_USER the IP-protocol must be set to UDP. Possible values are:

- FDX_TX_FRAME_PGM_USER
- FDX_TX_FRAME_PGM_IP_PART
- FDX_TX_FRAME_PGM_IP_PART_TT

- FDX_TX_FRAME_PGM_IP_ FULL
- FDX_TX_FRAME_PGM_IP_FULL_TT

Payload generation mode **FDX_TX_FRAME_PGM_USER:**
This is the default value. The payload will not be generated. All AFDX frame data for transmission will be taken from the pv_WriteBuffer. Complete frame data must be provided for this frame.

Payload generation mode **FDX_TX_FRAME_PGM_IP_PART:**
The following highlighted bytes (in network byte order) of the AFDX frame will be generated with data from the static transmit registers. (Each cell represents one byte except for the UDP payload).

Note:
Not yet supported on ASC-FDX-2

Byte 0:	Byte 1:	Byte 2:	Byte 3:
Destination Mac address	Destination Mac address	Destination Mac address	Destination Mac address
Destination Mac address	Destination Mac address	Source Mac address	Source Mac address
Source Mac address	Source Mac address	Source Mac address	Source Mac address
Ether Type	Ether Type	IP Version | IP Length	IP Type Of Service
IP Total Length	IP Total Length	IP Identifier	IP Identifier
IP Flag | IP Fragment Offset	IP Flag | IP Fragment Offset	IP Time To Live	IP Protocol
IP Header Checksum	IP Header Checksum	Source IP	Source IP
Source IP	Source IP	Destination IP	Destination IP
Destination IP	Destination IP	UDP Source port	UDP Source port
UDP Destination port	UDP Destination port	UDP Length	UDP Length
UDP Checksum	UDP Checksum	UDP payload (22 bytes)	

Payload generation mode **FDX_TX_FRAME_PGM_IP_PART_TT:**
Same as payload generation mode FDX_TX_FRAME_PGM_IP_PART with the addition that the UDP payload will be filled with the start timetag, which is repeated every eight bytes.

Note:
Not yet supported on ASC-FDX-2

Payload generation mode **FDX_TX_FRAME_PGM_IP_ FULL:**
The following highlighted bytes (in network byte order) of the AFDX frame will be generated with data from the static transmit registers. (Each cell represents one byte except for the UDP payload).

Note:
Not supported on ASC-FDX-2

Byte 0:	Byte 1:	Byte 2:	Byte 3:
Destination Mac address	Destination Mac address	Destination Mac address	Destination Mac address
Destination Mac address	Destination Mac address	Source Mac address	Source Mac address
Source Mac address	Source Mac address	Source Mac address	Source Mac address
Ether Type	Ether Type	IP Version | IP Length	IP Type Of Service
IP Total Length	IP Total Length	IP Identifier	IP Identifier
IP Flag | IP Fragment Offset	IP Flag | IP Fragment Offset	IP Time To Live	IP Protocol
IP Header Checksum	IP Header Checksum	Source IP	Source IP
Source IP	Source IP	Destination IP	Destination IP
Destination IP	Destination IP	UDP Source port	UDP Source port
UDP Destination port	UDP Destination port	UDP Length	UDP Length
UDP Checksum	UDP Checksum	UDP payload (22 bytes)	

Payload generation mode **FDX_TX_FRAME_PGM_IP_FULL_TT:** Same as payload generation mode
FDX_TX_FRAME_PGM_IP_ FULL with the addition that the UDP payload will be filled with the start timetag, which is repeated every eight bytes.

Note:
Not yet supported on ASC-FDX-2

Note:
This static transmit registers must be setup properly for payload generation modes other than 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 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)

Note:
For timetag payload generation modes FDX_TX_FRAME_PGM_IP_PART_TT and FDX_TX_FRAME_PGM_IP_FULL_TT the timetag in the payload starts at Byte 44 (2 bytes after UDP checksum)

BYTE 44...47 – Timetag high
Bit 0... 5 : Seconds of minute
Bit 6...11 : Minutes of hour
Bit 12...16 : Hours of day
Bit 17...25 : Days of year
Bit 26...29 : with APE-FDX boards this holds the nanoseconds in steps of 100ns else reserved
Bit 30...31 : reserved (0)

BYTE 48...51 – Timetag low
Bit 0...19 : Microseconds of second
Bit 20...25 : Seconds of minute
Bit 26...31 : Minutes of hour

AiUInt32 ul_BufferQueueHandle

If 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 FdxCmdTxBuffer-QueueAlloc. 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 enables the user to change data associated with this frame during run time by the **FdxCmdTxBuffer-QueueWrite** and **FdxCmdTxBufferQueueCtrl** functions.

AiUInt8 uc_ExternalStrobe

Control assertion of trigger strobe if this frame is transmitted. See the **FdxCmdTx-TrgLineCtrl** 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

Number of preamble bytes that will precede the frame. Setting this value to 0 will result in 7 preamble bytes, which is the default as specified in IEEE 802.3.

Value:	Description:
FDX_TX_FRAME_PRE_DEF	Send default preamble count of 7 bytes
1..15	Number of preamble bytes to send

AiUInt32 ul_Skew

This value defines the skew in microseconds between the transmission of two redundant frames. The skew can be programmed in a range from 0 µs to 65,535 µs with a resolution of 1 µs. The uc_NetSelect parameter defines the port on which the frame is delayed. This value is only used if the card is configured for redundant operation and uc_NetSelect is defined as FDX_TX_FRAME_DLY_A or FDX_TX_FRAME_DLY_B.

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 the maximum value of ul_InterFrameGap in the receiver. Therefore a higher interframe gap time may result because the IFG counter for transmit is started synchronously for both networks after both redundant frames are sent.

AiUInt8 uc_NetSelect

This parameter is used to define the physical Interface of the MAC on which redundant frames will be sent. In case of delayed sending, the delay is defined with the skew value (see parameter ul_Skew above).

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:
The parameters ul_Skew and ul_NetSelect are only relevant in redundant port operation mode.

AiUInt8 uc_FrameStartMode

This parameter defines the 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	This setting has the same effect as the start mode described above (FDX_TX_FRAME_START_TRG).

AiUint32 ul_PhysErrorInjection

Can be used to send frames with physical errors. A combination of the following error types is allowed:

Value:	Description:
FDX_TX_FRAME_ERR_OFF	Error injection is disabled
FDX_TX_FRAME_ERR_CRC	Frame will be sent with wrong MAC CRC field value.
FDX_TX_FRAME_ERR_ALI	An alignment error will be induced. This means 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, i.e. a wrong start frame delimiter (SFD) is transmitted with this frame. The first nibble of the start frame delimiter is substituted with value '1000' instead of '1001' **Note:** If a wrong SFD sequence is transmitted, the physical receiver devices connected on the network might have difficulties to detect the current frame properly.
FDX_TX_FRAME_ERR_PHY	Frame will be sent with a physical symbol error (PHY). This means the transceiver will transmit 'HALT' symbols. This error type is only available in 100Mbit/s operation mode.

AiUint16 uw_SequenceNumberInit

Can be used to send frames with a specific sequence number (SN). The parameter sets the SN that is used when the frame is sent for the first time, on the very first send cycle after starting transmission.
See next parameter **uw_SequenceNumberOffset** for information on how to configure SN for this frame in following send cycles.

Value:	Description:
FDX_TX_FRAME_SEQ_INIT_AUTO	Sends the frame with the current SN of the VL on which the frame is sent. The first frame on a VL is sent with SN 0.
0..255	Sets first SN of this frame to this value.
FDX_TX_FRAME_SEQ_OFF	SN is not generated automatically but taken from the payload. **Note:** The **uw_SequenceNumberOffset** parameter has no effect in this case.

AiUint16 uw_SequenceNumberOffset

After each frame is sent, this offset value is added to the SN of the frame. On the next cycle, the frame is sent with this new SN.

Value:	Description:
FDX_TX_FRAME_SEQ_OFFS_AUTO	Automatically generates a valid SN sequence for the VL on which the frame is sent. (SN: 0, 1, 2, 3, ..., 254, 255, 1, 2, 3, ...) **Note:** Only works for a maximum of 255 frames on the same VL.
0..255	This value is added to the SN of the frame to create the SN for the next transmission.

AiUInt8 uc_TxIntEnable

When set to true, an event is signalled each time this frame is transmitted. The function FdxInstIntHandler can be used to configure a handler for these events.

> **Note:**
> Not yet supported on ASC-FDX-2

> **Note:**
> On API/AMC/APU-FDX cards, you have to set this parameter to 2, if you need a valid time stamp of the event. This refers to fields ul_LWordE and ul_LWordF of structure TY_FDX_INTR_LOGLIST_ENTRY, which you are provided in the event handler.

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 encounters 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 when this instruction is reached. This marker can be inserted several times in a transfer queue or sub queue.

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.
This parameter defines, which transmit sub queue of all referenced sub queues shall be activated first with this call.
The parameter 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.
This is an array of transmit subqueue handles returned form the command 'Fdx-CmdTxSubQueueCreate' 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 subqueues in an ARINC664 conform environment, it is recommended to use only one subqueue. The reason for this limitation is the problem of sequence numbering. Only the subqueue initialized before start of transmitter is taken in account for correct ARINC664 sequence numbering.

3.3.2.5.2 FDX_TX_FRAME_HEADER_FIFO

TY_FDX_TX_FRAME_HEADER_FIFO

This C structure represents the FIFO frame header with all its fields.

```
typedef struct _fdx_tx_frame_header_fifo {
        AiUInt32 reserved1;
        AiUInt32 reserved2;
        AiUInt32 reserved3;
        AiUInt32 reserved4;
        TY_TX_FRAME_HEADER_FIFO_WORD_0 frameHeaderWord0;
        TY_TX_FRAME_HEADER_FIFO_WORD_1 frameHeaderWord1;
        TY_TX_FRAME_HEADER_FIFO_WORD_2 frameHeaderWord2;
        AiUInt32 reserved5;
        AiUInt32 reserved6;
    } TY_FDX_TX_FRAME_HEADER_FIFO;
```

AiUInt32 reserved1

Reserved

AiUInt32 reserved2

Reserved

AiUInt32 reserved3

Reserved

AiUInt32 reserved4

Reserved

TY_TX_FRAME_HEADER_FIFO_WORD_0 frameHeaderWord_0

Frame header word 0 contains following information

```
typedef union _tx_frame_header_fifo_word_0 {
        AiUInt32 ul_Value;
        struct {
                AiUInt32 skew : 16;
                AiUInt32 pse : 1;
                AiUInt32 sfd : 1;
                AiUInt32 tne : 1;
                AiUInt32 crce : 1;
                AiUInt32 pre : 4;
                AiUInt32 res_1 : 4;
                AiUInt32 ttag : 1;
                AiUInt32 str : 1;
                AiUInt32 res_2 : 2;
         } bits;
```

```
} TY_TX_FRAME_HEADER_FIFO_WORD_0;
```

Value	Bit Description	
res_2	31..30	Reserved
str	29	If set to 1, trigger pulse on an output line when frame is transmitted.
ttag	28	If set to 1, insert timetag in payload Note: Not yet supported on ASC-FDX-2
res_1	27..24	Reserved
pre	23..20	Number of preamble bytes that will precede the frame. Setting this value to 0 will result in 7 preamble bytes, which is the default as specified in IEEE 802.3
crce	19	If set to 1, frame will be sent with wrong MAC CRC field value
tne	18	If set to 1, a triple nibble error will be induced. That is a wrong byte alignment in frame, which means that an odd number of nibbles will be transmitted. Therefore, this error also causes a CRC error condition.
sfd	17	If set to 1, a wrong start frame delimiter (SFD) is transmitted with this frame **Note:** if a wrong SFD sequence is transmitted, the physical receiver devices connected on the network might have difficulties to detect the current frame properly.
pse	16	If set to 1, the frame will be sent with a physical symbol error (PHY). This means the transceiver will transmit 'HALT' symbols. The error type is only available in 100Mbit/s operation mode.
skew	15...0	This 16 bit value defines the skew in microseconds between the transmission of two redundant frames. The skew can be programmed in a range from 0 µs to 65,535 µs with a resolution of 1 µs. The uc_NetSelect parameter defines the port on which the frame is delayed. This value is only used if the card is configured for redundant operation and MacPortIdent is defined as FDX_TX_FIFO_FRAME_DLY_A or FDX_TX_FIFO_FRAME_DLY_B.

`TY_TX_FRAME_HEADER_FIFO_WORD_1 frameHeaderWord_1`

Frame header word 1 contains following information

```
typedef union _tx_frame_header_fifo_word_1 {
        AiUInt32 ul_Value;
        struct {
                AiUInt32 ByteCount : 11;
                AiUInt32 Reserved1 : 3;
                AiUInt32 IfgCount : 14;
                AiUInt32 Reserved2 : 1;
                AiUInt32 MacPortIdent : 3;
         } fields;
} TY_TX_FRAME_HEADER_FIFO_WORD_1;
```

<table>
<tr><th>Value</th><th>Bit</th><th colspan="2">Description</th></tr>
<tr><td rowspan="7">MacPortIdent</td><td rowspan="7">31..29</td><td colspan="2">This parameter is used to define the physical Interface of
the MAC on which redundant frames will be sent. In case of
delayed sending, the delay is defined with the skew value
(see parameter MacPortIdent in frameHeaderWord_0).
The following table shows the options available:</td></tr>
<tr><td>Value:</td><td>Description:</td></tr>
<tr><td>FDX_TX_FIFO_FRAME_DLY_A</td><td>Packet on network A is delayed by the skew value with respect to network B</td></tr>
<tr><td>FDX_TX_FIFO_FRAME_DLY_B</td><td>Packet on network B is delayed by the skew value with respect to network A</td></tr>
<tr><td>FDX_TX_FIFO_FRAME_BOTH</td><td>Packet transmitted on both networks (skew = 0)</td></tr>
<tr><td>FDX_TX_FIFO_FRAME_SUPPRESS_B</td><td>Transmission suppressed on network B</td></tr>
<tr><td>FDX_TX_FIFO_FRAME_SUPPRESS_A</td><td>Transmission suppressed on network A</td></tr>
<tr><td>Reserved2</td><td>28</td><td colspan="2">Reserved</td></tr>
<tr><td>IfgCount</td><td>25..12</td><td colspan="2">Transmission delay between start of this frame and end of preceding frame.
One unit corresponds to 4 bit times of the current network speed.
This is 40ns at 100Mbit/s operation mode and 400ns at 10Mbit/s operation mode.
Maximum value for this setting is 16383,
meaning up to approx. 655µs at 100Mbit/s operation mode or
or up to approx. 6,55ms at 10Mbit/s operation mode.
If set to less than 24, the connected receive devices on the network
may detect an 'Short interframe gap' error condition.
Please note that this setting is only precise
if two frames are loaded consecutively to the transmit FIFO.
Note: Not yet supported on ASC-FDX-2</td></tr>
<tr><td>Reserved1</td><td>11</td><td colspan="2">reserved</td></tr>
<tr><td rowspan="2">ByteCount</td><td rowspan="2">10..0</td><td colspan="2">MAC frame length in bytes.
This field contains the number of bytes, which shall be transmitted
within the ARINC664 MAC frame, including the CRC field. Since this
value is programmable the short frame conditions (below 64bytes)
as well as the frame too long conditions (more as 1518 bytes) can
be generated. For byte count values less than 60 bytes no proper frame
transmission can be guaranteed.</td></tr>
<tr><td colspan="2">Note: Byte count values less than 40 bytes and more than 2000
bytes are not allowed and may be cause unrecoverable transmit operation errors.</td></tr>
</table>

TY_TX_FRAME_HEADER_FIFO_WORD_2 frameHeaderWord_2

Frame header word 2 contains following information

```
typedef union _tx_frame_header_fifo_word_2 {
        AiUInt32 ul_Value;
        struct {
                AiUInt32 SequenceNumber : 8;
                AiUInt32 UseFrameDataForSequenceNumberInsertion : 1;
                AiUInt32 FrameTransmitInterrupt : 1;
                AiUInt32 AddTTag : 1;
                AiUInt32 Reserved : 5;
                AiUInt32 BufferSize     : 16;
        } fields;
} TY_TX_FRAME_HEADER_FIFO_WORD_2;
```

Value	Bit	Description
BufferSize	31..16	Size of the whole FIFO frame data in bytes. Including TY_FDX_FRAME_HEADER_INFO header and MAC frame. Must be aligned to a 64 byte boundary on API/AMC/APU-FDX boards.
Reserved	15..11	Reserved
AddTTag	10	Timestamp of frame transmission is attached to interrupt information if this flag is set. Only valid if FrameTransmitInterrupt is set.
FrameTransmitInterrupt	9	Set to initiate a transmit interrupt on frame transmit time
UseFrameDataForSequenceNumberInsertion	8	Sequence Number from Sequence Number field is not attached to frame payload if this flag is set. Sequence Number Byte is taken from user data
SequenceNumber	7..0	ARINC664 Sequence Number to send with this frame

AiUInt32 reserved5

Reserved

AiUInt32 reserved6

Reserved

3.3.2.6 FdxCmdTxQueueControl

Prototype:

```
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. 655µs (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. Note: Not yet supported on ASC-FDX-2

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.

3.3.2.7 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 Section 3.3.2.4 "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.

3.3.3 Individual (UDP Port oriented)Transmitter Functions

3.3.3.1 FdxCmdTxCreateVL

Prototype:

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

Purpose:

This function creates a virtual link in order to send frames in accordance to specific traffic shaping rules. It can be used only when the transmitter is not running.

Input:

TY_FDX_TRANSMIT_VL *px_TransmitVL

Pointer to a structure that contains the virtual link settings.

```
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. Range from 0 to 65535.

AiUInt32 ul_SubVls

Number of sub VL's associated to this VL. Range from 1 to 4.

AiUInt32 ul_Bag

Specifies the bandwidth allocation gap (BAG) for this virtual link in milliseconds. The bag limits the bandwidth of a VL. It defines the maximum rate at which data can be sent. Possible values are 1, 2, 4, 8, 16, 32, 64 and 128 ms.

AiUInt32 ul_NetSelect

This parameter is used to define the physical Interface of the MAC on which redundant frames will be sent. In case of delayed sending, the delay is defined with the

skew value (see parameter ul_Skew below).
Available options are:

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 parameter is only applicable in redundant port operation mode.

`AiUInt32 ul_MaxFrameLength`

Specifies the maximum length in bytes of frames that can be sent over this VL.

`AiUInt32 ul_FrameBufferSize`

Sets the size of the VL frame buffer in bytes. This translates to how many frames can be stored for this VL. If this value is set to zero a platform dependent default value is set.

`AiUInt32 ul_MACSourceMSLW`

Most significant 16 bit of the MAC source address in the format “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_MACSourceLSLW`

Least significant 32 bit of the MAC source address in the format “aa:bb:cc:dd:ee:ff”

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

`AiUInt32 ul_skew`

This value defines the skew in microseconds between the transmission of two redundant frames. The skew can be programmed in a range from 0 µs to 65,535 µs with a resolution of 1 µs. The ul_NetSelect parameter defines the port on which the frame is delayed. This value is only used if the card is configured for redundant operation and ul_NetSelect is defined as FDX_TX_FRAME_DLY_A or FDX_TX_FRAME_DLY_B.

Note:
This parameter is only applicable in redundant port operation mode.

Output:

None

Return Value:

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

3.3.3.2 FdxCmdTxCreateHiResVL

Prototype:

```
AiReturn FdxCmdTxCreateHiResVL(AiUInt32 ul_Handle,
                               const TY_FDX_TRANSMIT_VL* px_TransmitVL);
```

Purpose:

This function creates a virtual link with a high resolution bag in order to send frames in accordance to specific traffic shaping rules. It can be used only when the transmitter is not running. After starting transmission, frames on high resolution VLs are sent according to the BAG of the corresponding VL. Frames on high resolution Sub VLs are sent in round-robin manner.

Input:

`TY_FDX_TRANSMIT_VL *px_TransmitVL`

Pointer to a structure that contains the virtual link settings.

```
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.

`AiUInt32 ul_MaxFrameLength`

Specifies the maximum length in bytes of frames that can be sent over this VL.

`AiUInt32 ul_FrameBufferSize`

Sets the size of the VL frame buffer in bytes. This translates to how many frames can be stored for this VL. If this value is set to zero a platform dependent default value is set.

`AiUInt32 ul_MACSourceLSLW`

Least significant 32 bit of the MAC source address in the format 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 16 bit of the MAC source address in the format 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`

This parameter is used to define the network on which redundant frames will be sent. In case of delayed sending in conjunction with the defined skew value (see ul_Skew below). Available options are:

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 value defines the skew in microseconds between the transmission of two redundant frames. The skew can be programmed in a range from 0 µs to 65,535 µs with a resolution of 1 µs. The ul_NetSelect parameter defines the port on which the frame is delayed. This value is only used if the card is configured for redundant operation and ul_NetSelect is defined as 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.

3.3.3.3 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.

3.3.3.4 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 a Service Access Point transmitter port (SAP-Tx-Port), which is linked to a fixed specified UDP source port, and can be used to send messages to different IP/UDP-destinations with the function ***FdxCmdTxSAPWrite***. Like a queuing transmitter port a SAP-Tx-Port can store messages in a queue, send variable message sizes and split messages into single fragments/packets. Please note that several FxCmdTxUDP functions can also be used for SAP-Tx-Ports ***(FdxCmdTxUDPChgSrcPort, FdxCmdTxUDPGetStatus, FdxCmdTxUDPControl, FdxCmdTxUDPDestroyPort)***. In order to identify the SAP port in further functions the returned ul_UdpHandle must be used. Initial settings after creation of a 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_TxSapCreateIn`

Pointer to a structure that contains the SAP port settings.

```
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`

UDP source address of this SAP port. Range from 0 to 65535. Can be freely chosen but port number allocation schemes and ICANN administered numbers should possibly be considered. The UDP source address can be changed later while transmitter is running with FdxCmdTxUDPChgSrcPort.

`AiUInt32 ul_IpSrc`

The IPv4 source address used in the IP-Header in packets sent from this SAP port. The IP source address should be a Class A private IP unicast address and must respect specific addressing schemes.

AiUInt32 ul_VlId

The Virtual Link over which frames originating from this port shall be transmitted. The Virtual Link must have been created before with FdxCmdTxCreateVL or FdxCmdTxCreateHiResVL. Range from 0 to 65535

AiUInt32 ul_SubVlId;

Sub Virtual Link Identifier. Range from 1 to 4. This value must be consistent (<=) with parameter ul_SubVls of function FdxCmdTxCreateVL. If Sub VLs are not used, the Sub VL Id should be 1.

AiUInt32 ul_UdpNumBufMessages

Number of messages which can be stored by the SAP-Port in the associated queue. If this value is set to 0, the queue will be allocated with a default size.

AiUInt32 ul_UdpMaxMessageSize;

Maximum size of a message to send. Range from 0 to 8192 bytes.

Output:

TY_FDX_TX_SAP_CREATE_OUT* px_TxSapCreateOut

Pointer to a structure that contains the handle for the created SAP port.

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

AiUInt32 ul_UdpHandle

Handle to the SAP port. This handle must be stored by the application and is used to identify the SAP port in further functions.

Return Value:

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

3.3.3.5 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 message to a SAP-Tx-Port. The message is the UDP-payload with variable size and will be stored in the queue of the port. As soon as possible the message will be taken from the queue and used to build a single frame or multiple fragments, which will be buffered in the corresponding Sub-VL queue for later transmission with subject to traffic-shaping.

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

This handle identifies the SAP-Tx-Port to write to and is returned by **FdxCmdTxSAPCreatePort**.

AiUInt32 ul_UdpDst

The UDP destination port for the message. Valid range 0 to 65535.

AiUInt32 ul_IpDst

The IPv4 destination address used in the IP-Header in packets sent from this SAP port. The address should be a Class A private IP unicast address to identify the target subscriber or a Class D multicast address reflecting the VL. The destination address must respect specific addressing schemes.

AiUInt32 ul_ByteCount

Size of the message in bytes referenced by pv_Data.
Range from 0 to ul_UdpMaxMessageSize defined in **FdxCmdTxSAPCreatePort**.
Padding bytes will be added if ul_ByteCount is smaller than 17 bytes to have a valid Ethernet-Frame.

void* pv_Data

Pointer to a buffer that contains the message to send. Must be at least ul_ByteCount in size.

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. Will be zero if the queue of the SAP port is already full. The maximum queue size is defined by the parameter ul_NumBufMessages in **FdxCmdTxSAPCreatePort**.

Return Value:

When message has been successfully queued, function returns FDX_OK and sets ul_BytesWritten to the value of ul_ByteCount.
When message queue was full, function returns FDX_OK and sets ul_BytesWritten to zero. In this case the message will not be sent.

Returns a negative error code on other errors.

3.3.3.6 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.

3.3.3.7 FdxCmdTxUDPChgSrcPort

Prototype:

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

Purpose:

Changes the UDP source port of an existing SAP-Tx or UDP-Tx port. This function can be used while transmitter is running.

Input:

`AiUInt32 ul_UdpHandle`

This handle identifies the SAP-Tx- or UDP-Tx-Port to be changed and is returned by FdxCmdTxSAPCreatePort or FdxCmdTxUDPCreatePort.

`AiUInt32 ul_UdpSrc`

The new UDP source address of this port. Range from 0 to 65535. Can be freely chosen but port number allocation schemes and ICANN administered numbers should possibly be considered. There is no check if the new UDP source is already in use by another SAP-Tx or UDP-Tx port, so the user is responsible for a unique mapping between API ports and UDP source addresses.

Output:

None

Return Value:

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

3.3.3.8 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 settings and modify network and skew parameters.

Input:

AiUInt32 ul_UdpHandle

This handle identifies the UDP-Tx-Port to control and is returned by FdxCmdTxUDPCreatePort or 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.

Note:
Note: This parameter is not supported on ASC-FDX.

AiUInt32 ul_NetSelect

This parameter is used to define the network on which redundant frames will be sent. In case of delayed sending in conjunction with the defined skew value (see ul_Skew below).
Available options are:

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 ul_NetSelect setting of corresponding VL defined by function FdxCmdTxCreateVL or FdxCmdTxCreateHiResVL

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

AiUInt32 ul_Skew

This value defines the skew in microseconds between the transmission of two redundant frames. The skew can be programmed in a range from 1 µs to 65,535 µs with a resolution of 1 µs. The ul_NetSelect parameter defines the port on which the frame is delayed. This value is only used if the card is configured for redundant operation and ul_NetSelect is defined as FDX_TX_FRAME_DLY_A or FDX_TX_FRAME_DLY_B.
A ul_Skew value of 0 uses the skew setting of the corresponding VL defined by function FdxCmdTxCreateVL or FdxCmdTxCreateHiResVL.

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

Note:
Note: This parameter is not supported on ASC-FDX.

AiUInt32 ul_ErrorInjectionTyp

Refer to parameter ul_PhysErrorInjection of function FdxCmdTxQueueWrite.

Note:
Note: This parameter is not supported on ASC-FDX.

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.

Note:
Note: This parameter is not supported on ASC-FDX.

Output:

None

Return Value:

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

3.3.3.9 FdxCmdTxUDPCreatePort

Prototype:

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

Purpose:

Creates a UDP-Tx-Port which can be used to send messages to a specific IP/UDP-destination with the function **FdxCmdTxUDPWrite**. Please note that several FxCmdTxUDP functions can be used to modify settings or retrieve status information **(FdxCmdTxUDPChgSrcPort, FdxCmdTxUDPGetStatus, FdxCmdTxUDPControl, FdxCmdTxUDPDestroyPort)**. In order to identify the UDP port in further functions the returned ul_UdpHandle must be used. Initial settings after creation of a UDP port are:
UDP port enabled
Error injection: OFF
Skew (redundant mode only): 0 usec.
This function can only be used if the transmitter is not running.

Input:

`TY_FDX_UDP_DESCRIPTION* px_UdpDescription`

Pointer to a structure that contains the UDP port settings.

```
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 to create. It can be either sampling or queuing.

Value	Description
FDX_UDP_SAMPLING	Port is a sampling port. Each message is represented by one MAC frame. The size of the messages is constant.
FDX_UDP_QUEUING	Port is a queuing port. Each message can be represented by one or more MAC frames. Fragmentation will be handled at the IP layer. A message can have a size of up to 8kByte and can be different for each message.

struct TY_FDX_QUINTUPLET

This structure describes the full addressing of the communication port.

AiUInt32 x_Quint.ul_UdpSrc

UDP source address of this port. Range from 0 to 65535. Can be freely chosen but port number allocation schemes and ICANN administered numbers should possibly be considered. The UDP source address can be changed later with FdxCmdTxUD-PChgSrcPort while the transmitter is running.

AiUInt32 x_Quint.ul_IpSrc

The IPv4 source address used in the IP-Header in packets sent from this UDP port. The IP source address should be a Class A private IP unicast address and must respect specific addressing schemes.

AiUInt32 x_Quint.ul_VlId

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

AiUInt32 x_Quint.ul_IpDst

The IPv4 destination address used in the IP-Header in packets sent from this UDP port. The address should be a Class A private IP unicast address to identify the target subscriber or a Class D multicast address reflecting the VL. The destination address must respect specific addressing schemes.

AiUInt32 x_Quint.ul_UdpDst

The UDP destination port for the message. Valid range 0 to 65535.

AiUInt32 ul_SubVlId

Sub Virtual Link Identifier. Range from 1 to 4. This value must be consistent (<=) with parameter ul_SubVls of function FdxCmdTxCreateVL / FdxCmdTxCreateHiResVL. If Sub VLs are not used, the Sub VL Id should be 1.

AiUInt32 ul_UdpNumBufMessages

Number of messages which can be stored by the UDP-Port in the associated queue.
If this value is set to 0, the queue will be allocated with a default size.
For sampling ports the number of messages must be set to 1.

AiUInt32 ul_UdpMaxMessageSize

Maximum size of a message in bytes that can be sent. The size is without the header overhead (MAC, IP and UDP).

Port Type	Value
Sampling	This is the constant size of the sampling message. The message must fit into a single MAC frame that does not exceed ul_MaxFrameLength of the corresponding VL defined with the functions **FdxCmdTxCreateVL** or **FdxCmdTxCreateHiResVL**.
Queuing	Maximum size of a message to send. Range from 0 to 8192 bytes.

AiUInt32 ul_UdpSamplingRate

Specifies the message transmission rate for sampling ports in milliseconds and is therefore only applied for sampling ports. Valid range is starting from 1. Once transmission is started, the sampling port will automatically send the content of its message buffer with the configured sampling rate. The number of sampling ports with specific rates is restricted by the BAG of the corresponding VL. Refer to function FdxCmdTxCreateVL or FdxCmdTx-CreateHiResVL to see how to create and specify the BAG of a VL. The resulting load of a VL with a given set of sampling ports can be calculated by following formula:

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

If calculated VL load value is greater than 1, the VL is overloaded and configured sampling rates can not be met. Please note that queuing ports on the same VL may lead to additional load as soon as messages are queued for sending.

Note:
Note: With ASC-FDX devices an error will be returned if the VL is overloaded with sampling ports and the sampling port will not be created.

Output:

AiUInt32* pul_UdpHandle

Handle to the UDP port. This handle must be stored by the application and is used to identify the UDP port in further functions.

Return Value:

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

3.3.3.10 FdxCmdTxUDPDestroyPort

Prototype:

```
AiReturn FdxCmdTxUDPDestroyPort(AiUInt32 ul_Handle,
                                const AiUInt32 ul_UdpHandle);
```

Purpose:

This function is used to destroy an unneeded UDP transmit port. This function is therefore the opposite of functions **FdxCmdTxUDPCreatePort** and **FdxCmdTxSAPCreatePort**. This function can be used only when the transmitter is not running.

Input:

`AiUInt32 ul_UdpHandle`

Handle of the UDP Port to be destroyed. This is the handle returned when the UDP port is created via the functions
FdxCmdTxUDPCreatePort or **FdxCmdTxSAPCreatePort**.

Output:

None

Return Value:

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

3.3.3.11 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 message to a UDP-Tx-Port. The message is the UDP-payload of a before defined UDP sampling or a queuing port.
For queuing ports the message will be stored in the queue of the port. The transmission is initiated when data is written to a UDP port. As soon as possible the message will be taken from the queue and used to build a single frame or multiple fragments. For sampling ports there is only one queue entry which will be overwritten with each use of **FdxCmdTxUDPWrite**. At sampling time the message will be taken and used to build a single frame. For initialisation of the data contents this function should be called before the port is started.
The frames will be buffered in the corresponding Sub-VL queue for later transmission with subject to traffic-shaping.
This function can be used if the transmitter is running or not running.

Input:

`AiUInt32 ul_UdpHandle`

This handle identifies the UDP-Tx-Port to write to and is returned by **FdxCmdTxUDPCreatePort**.

`AiUInt32 ul_ByteCount`

Size of the message in bytes referenced by pv_Data. Range from 0 to ul_UdpMaxMessageSize defined in **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. Padding bytes will be added if ul_ByteCount is smaller than 17 bytes to have a valid Ethernet-Frame.

`void* pv_Data`

Pointer to a buffer that contains the message to send. Must be at least ul_ByteCount in size.

Output:

`AiUInt32* pul_BytesWritten`

Number of bytes actually written. Will be zero if the queue of the UDP port is already full. The maximum queue size is defined by the parameter ul_NumBufMessages in **FdxCmdTxUDPCreatePort**.

Return Value:

When message has been successfully written, function returns FDX_OK and sets *pul_BytesWritten to the value of ul_ByteCount.
When message queue was full for queuing ports, function returns FDX_OK and sets *pul_BytesWritten to zero. In this case the message will not be sent.

Returns a negative error code on other errors.

3.3.3.12 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 a UDP transmit port, specifically details about the messages sent from the port.

Input:

AiUInt32 ul_UdpHandle

This handle identifies the UDP-Tx-Port to read the status from and is returned by either **FdxCmdTxUDPCreatePort** or **FdxCmdTxSAPCreatePort**.

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 all messages successfully sent by this UDP port during its lifetime. This counter is not reset when restarting the transmitter port. The counter is maintained until the UDP port is destroyed via a call to either **FdxCmdTxUDPDestroyPort** or ***FdxCmdTxPortInit***.

AiUInt32 ul_CurrentIndex[1]

This parameter is only valid for UDP Sampling Ports which have been assigned an input queue.
The parameter shows the index of the message in the input queue which will be written to the Sub-VL next.

AiUInt32 ul_IndexCycleCount[1]

This parameter is only valid for UDP Sampling Ports which have been assigned an input queue.
The parameter counts the number of times the input queue has been cycled. A

cycle occurs when the index of the input queue reaches the end of the queue and is reset back to index 0.

[1] Parameter is not currently being supported on any ARINC664 board

Return Value:

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

3.3.3.13 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.

3.3.3.14 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 a frame 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 structure containing the VL settings and the frame to write.

```
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;

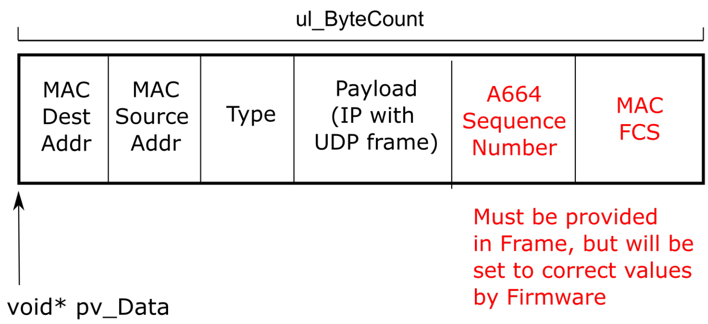
Sub Virtual Link Identifier. This value must be in a range from 1 to 4. If this VL has no Sub VLs, the Sub VL Id has to be set to 1.

AiUInt32 ul_ByteCount

Length of frame in bytes which shall be written to this VL.

void *pv_Data

Pointer to a buffer containing the frame to write. The size of this buffer has to correspond to ul_ByteCount. The frame contained in this buffer must be a complete MAC frame (IEEE 802.3 Ethernet Package), where sequence number and CRC will be overwritten and set to the correct value by the board firmware.



Output:

TY_FDX_TX_VL_WRITE_OUT *px_TxVLWriteOut

Pointer to a structure into which a value indicating whether the send process was successful will be written.

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

AiUInt32 ul_BytesWritten

Number of bytes actually written. Might be zero if the frame was not sent (see Return Value below).

Return Value:

Returns FDX_OK on success and sets ul_BytesWritten to ul_ByteCount.
Returns FDX_OK and sets ul_BytesWritten to zero if the VL buffer was full and not able to take this frame (and consequentially it was not sent).
Returns a negative error code on other errors.

3.3.3.15 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. 655µs (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 value defines the skew in microseconds between the transmission of two redundant frames. The skew can be programmed in a range from 0 µs to 65,535 µs with a resolution of 1 µs. The ul_NetSelect parameter defines the port on which the frame is delayed. This value is only used if the card is configured for redundant operation and ul_NetSelect is defined as FDX_TX_FRAME_DLY_A or FDX_TX_FRAME_DLY_B.

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 FdxCmdTx-TrgLineCtrl 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 is used to define the network on which redundant frames will be sent. In case of delayed sending in conjunction with the defined skew value (see ul_Skew above).
Available options are:

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:

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**.

3.3.4 Data Buffer Functions

3.3.4.1 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.

3.3.4.2 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.

3.3.4.3 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 FdxCmdTxBuffer-QueueAlloc).

`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.

3.3.4.4 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 FdxCmdTxBuffer-QueueAlloc).

`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.

3.3.4.5 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 FdxCmdTxBuffer-QueueAlloc).

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.

3.3.5 Generic Transmitter Sub-Queue-Functions

3.3.5.1 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.

3.3.5.2 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.

3.3.5.3 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:

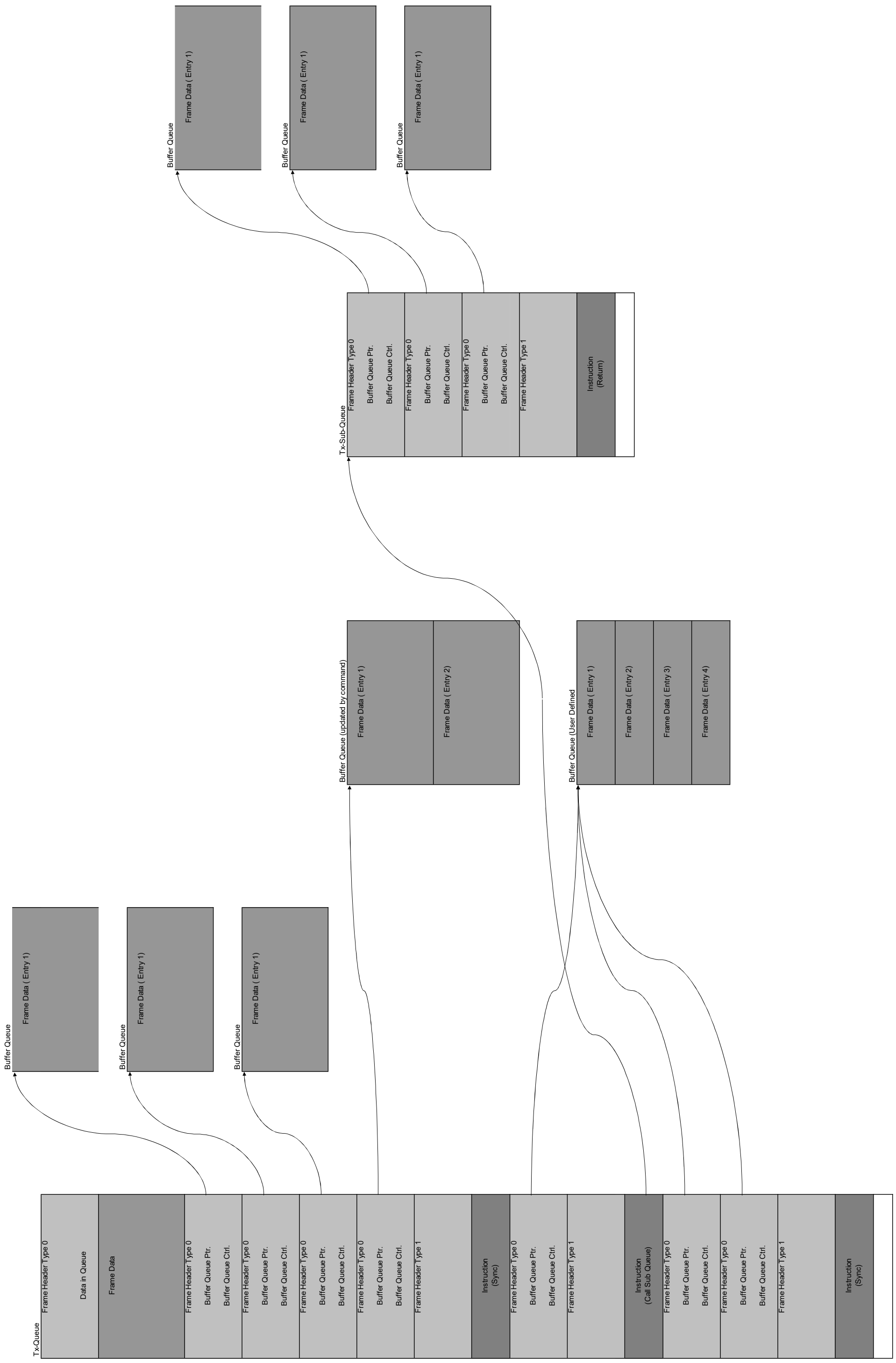
None

Return Value:

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

3.3.6 Theory 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:


Buffer Queue
Frame Data (Entry 1)
Buffer Queue
Frame Data (Entry 1)
Buffer Queue
Frame Data (Entry 1)
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 (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)
Buffer Queue
Frame Data (Entry 1)
Buffer Queue
Frame Data (Entry 1)
Buffer Queue
Frame Data (Entry 1)
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)

3.4 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 3.4: 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	nitializes 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 buffers for capturing of network traffic
FdxCmdMonContCapInvalidateMemory	Mark provided buffers as invalid for capturing network traffic
FdxCmdMonContCapForceDataTransfer	Force System to pass databuffer to Host and use next

3.4.1 Global Receiver Commands

3.4.1.1 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 used to control resetting of the statistics parameters in the target. (See Section 3.4.1.2 "FdxCmdRxGlobalStatistics") command for details.

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.

3.4.1.2 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:

Returns a set of statistics containing comprehensive information about the full bus traffic over all Virtual Links. The statistics are maintained by a set of counters which may be reset by the ul_Control parameter.

Input:

AiUint32 ul_Control

This parameter controls the resetting of the statistics counters on the board. All statistics counters are reset to zero when ul_Control is set to ***FDX_RX_GS_RES_ALL_CNT***. Several so-called "error counters" can be selectively reset by setting ul_Control to ***FDX_RX_GS_RES_ERR_CNT***. This will only affect a subset of the statistics. Refer to the Output section to see which statistics are reset by ***FDX_RX_GS_RES_ERR_CNT***.
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 counters
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 current 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 the total bytes received since the last counter reset.

AiUInt32 ul_FrameGoodCount;

Counter of the error-free frames received since the last counter reset.

AiUInt32 ul_FrameErrorCount;

Counter of all erroneous frames received since the last counter reset. Value will be reset when ul_Control is set to ***FDX_RX_GS_RES_ERR_CNT***

AiUInt32 ul_BytesPerSecond;

Instantaneous bytes received per second. This value is updated at a fixed interval.

AiUInt32 ul_FramesPerSecond;

Instantaneous frames received per second. This value is updated at a fixed interval.

TY_FDX_RX_GLOB_STAT_ERR x_StatErr

This structure gives more insight into the ul_FrameErrorCount statistic by present counts of specific frame errors which have been detect by the receiver. Note that an erroneous frame may contain more than one type of error.
All values in x_StatErr are reset when ul_Control is set to FDX_RX_GS_RES_ERR_CNT

```
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

> **Note:**
> For PCIe-based and new USB-based modules (e.g. APE and ASC) this value will always be 0. Possible errors of the category are reported with value ul_NoSfdErrorCount.

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[3]

Number of IP static header fields errors detected

AiUInt32 ul_MACErrorCount[3]

Number MAC static header fields errors detected

AiUInt32 ul_NoSfdErrorCount

Number Frames detected with no Start Frame Delimiter

AiUInt32 ul_VLLenErrorCount[23]

Number of VL specific frame length error detected

ul_SNIntegrityErrorCount[23]

Number of frames detected with sequence number integrity error.

AiUInt32 ul_TrafShapingViolationCount[13]

Traffic shaping violation detected

1 : only applicable if VL Descriptor setup and Traffic Shaping verification enabled
2 : only applicable if VL Descriptor setup
3 : not currently supported on the ASC-FDX

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

*: value will be reset when ul_Control is set to ***FDX_RX_GS_RES_ERR_CNT***

TY_FDX_RX_GLOB_STAT *px_GlobalStatPortB

This pointer is only necessary if the receive port is configured in Redundant Mode. In that case this structure reports the same information as *px_GlobalStat for the redundant port. If the port is configured in Single Mode then a NULL may be passed for this parameter.

Return Value:

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

3.4.1.3 FdxCmdRxModeControl

Prototype:

```
AiReturn FdxCmdRxModeControl(AiUInt32 ul_Handle,
                             const TY_FDX_RX_MODE_CTRL_IN* px_In,
                             TY_FDX_RX_MODE_CTRL_OUT* px_Out);
```

Purpose:

Configures the receive mode of a port. The following receive modes are supported dependent on the specific board-type.

- Chronological monitor mode All packets are stored in one chronological receive buffer. Various capture submodes can be configured by ***FdxCmdMonCaptureControl***
- single/redundancy mode
- Virtual Link oriented mode
  In this mode all received packets are scanned for the VL and can be stored in VL related queues. Only VLs enabled by ***FdxCmdRxVLControl*** for reception will be stored, all other VLs are just used for updating global statistics/VL activity and will be discarded afterwards. The stored packets are checked for their quintuplet and are forwarded to the Sampling & Queuing services or discarded, if no matching UDP or SAP receive port exists. This mode simulates how an end system typically receives packets and provides the received messages in UDP ports for applications running on the end system.
- Reros
  Packets are received and decoded using the address quintuplet, optionally modified, rerouted to one or more other ports and finally sent.

This function can only be called if the receiver is stopped.

Input:

TY_FDX_RX_MODE_CTRL_IN* px_In

Pointer to a structure that contains the receive port configuration options.

```
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

Dependent on the board type, up to three receive modes may be available:

Value	Description
FDX_RX_CHRONO	Chronological monitoring
FDX_RX_VL	VL oriented
FDX_RX_REROS	Rerouting

AiUInt32 ul_DefaultPayloadMode

> **Note:**
> This functionality is deprecated and only available for API/AMC/APU/GNET devices. With all other devices only FDX_PAYLOAD_FULL is applicable.

This mode can only be used for chronological monitoring. It defines the payload mode for data reduction. You can specify up to which level the data shall be stored in the chronological monitor buffer. In VL oriented simulation mode the full payload is stored.
Use this mode to monitor the traffic on the bus without saving full frames.

Value	Description
FDX_PAYLOAD_FULL (always used in VL-oriented mode)	Frames will be stored with full payload in the monitor. 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 bytes of frame data are stored to the data buffer. (MAC and IP header + 30 bytes of data
GNET_PORT_PAYLOAD_FRH32[1)]	Only frame header and 32 bytes of frame data are stored to the data buffer. (MAC header + 18 bytes MAC-payload).

> **Note:**
> The GNET_PORT_ parameters are only valid for APX-GNET boards.

[1)] These parameters for GNET_PORT_PAYLOAD_... must be selected either combined via bitwise "or" with the other standard payload modes or 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_DefaultCronoMode

This mode only applies to chronological monitoring. In VL oriented mode FDX_RX_DEFAULT_ENA_CNT will be applied regardless of the selected option so that frame receive statistics and VL related counters are available via FdxCmdRxVL-GetActivity. FdxCmdRxVLControl can be used to define a different ChronoMode for a VL. When monitoring chronologically, there are three possible default modes:

> **Note:**
> On ASC-FDX devices, only FDX_RX_DEFAULT_ MON_ ENA_ALL is supported at this time.

Value	Description
FDX_RX_DEFAULT_ENA_CNT (always used in VL-oriented mode)	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 data 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. In this option, only good (error free) frames are stored in the data buffer, defined with the parameters above. In parallel the VL oriented counters will be updated.

To change the enabled state from the default mode for a VL, use FdxCmdRxVL-Control 3.4.1.7.

AiUInt32 ul_GlbMonBufferSize

This parameter is only relevant if the chronological monitoring mode is selected. The parameter is not used in VL oriented mode.

It defines the requested size of the monitor buffer for this port resource. This value must be specified in bytes.

If this value is set to 0, the buffer size will be set to a default value. If a value smaller than the minimum value (which depends on the board type) is given, the minium size will be used.

If the requested buffer size exceeds the available memory, the FdxRxModeConrol command will fail and return a negative error code.

AiUInt32 ul_ RerosDefaultOutputPortmap

This parameter is not relevant for chronological monitoring and ignored in that case.

It is only used in reros mode, (See Section 3.6.1 "FdxCmdRerosVLReroute").

Output:

TY_FDX_RX_MODE_CTRL_OUT* px_Out

```
typedef struct {
    AiUInt32 ul_GlbMonBufferSize;
} TY_FDX_RX_MODE_CTRL_OUT;
```

AiUInt32 ul_GlbMonBufferSize

The allocated monitor buffer size is reported in this output parameter. This might be adjusted from the size requested via the input parameter, if that was less than the minimum size.

In VL oriented mode, the return value will always be 0.

Return Value:

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

3.4.1.4 FdxCmdRxPortInit

Prototype:

```
AiReturn FdxCmdRxPortInit(AiUInt32 ul_Handle,
                          const TY_FDX_PORT_INIT_IN* px_PortInitlIn,
                          TY_FDX_PORT_INIT_OUT* px_PortInitOut);
```

Purpose:

This function is used to reset the receive functionality of the port to an initial state.

The initial state is as follows:

- Receiver is stopped
- Global statistics available and set to 0
- 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 identification number. Only lowest 8 bits of the 32bit ul_PortMap value are used. This identification will be used in frame headers of received data to identify the port.

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

Number of logins that were performed on this port

AiUInt32 ul_GlobalMemFree

Size of Global Memory (in Bytes) which is not already allocated.
Only valid for AMC/API/APU-FDX cards.

AiUInt32 ul_SharedMemFree

Size of Shared Memory (in Bytes) which is not already allocated.
Only valid for AMC/API/APU-FDX cards.

Return Value:

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

3.4.1.5 FdxCmdRxStatus

Prototype:

```
AiReturn FdxCmdRxStatus(AiUInt32 ul_Handle,
                        TY_FDX_RX_STATUS* px_RxStatus);
```

Purpose:

This function is used to retrieve the receiver status of a certain port.

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 error information. Only valid if ul_Status set to FDX_STAT_ERROR.

Value	Description
FDX_RX_INFO_OVERLOAD	Receiver was not able to capture all frames. Frames may have been lost. It's necessary to restart receiver.

Return Value:

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

3.4.1.6 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.

3.4.1.7 FdxCmdRxVLControl

Prototype:

```
AiReturn FdxCmdRxVLControl(AiUInt32 ul_Handle,
                           const TY_FDX_RX_VL_CTRL* px_VLControl,
                           const TY_FDX_RX_VL_DESCRIPTION* px_VLDesc);
```

Purpose:

Controls the Virtual Link specific settings for a receive port. The standard behaviour for all Virtual Links is defined by the parameter ul_DefaultCronoMode in ***FdxCmdRxModeControl. FdxCmdRxVLControl*** can alter the storage behaviour for specific VLs. In extended operation mode enhanced checks and filter functions are available. As a result of these checks and filters packets might be discarded or the result will be reported in the receive header of each frame in chronologic receive mode (See 3.4.3.4 "TY_FDX_FRAME_BUFFER_HEADER" in ***FdxCmdMonQueueRead***). Furthermore, the selected enable mode and check related result counters and status-flags are available via ***FdxCmdRxVLGetActivity***.

In VL oriented receive mode all VLs are by default only enabled for Global Statistics and VL- Activity (See Section 3.4.1.2 "FdxCmdRxGlobalStatistics") and (See Section 3.4.1.9 "FdxCmdRxVLGetActivity"). If a VL shall be used for UDP- or SAP-receive ports, then it must be first enabled for extended operation mode by this function.

This function can only be called if the receiver is stopped.

Input:

TY_FDX_RX_VL_CTRL* px_VLControl

Pointer to a structure that contains the receive settings for the Virtual Link.

```
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. Range from 0 to 65535. This value is part of the MAC destination address.

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. Valid range is 1 to 65536–ul_VLId. If ul_EnableMode = FDX_RX_VL_ENA_EXT ul_VLRange must be 1

AiUInt32 ul_EnableMode

Mode which is used for the given VL.

Value	G	A	S	Comment
FDX_RX_VL_DIS				Virtual Link is disabled and all frames are silently discarded. No VL Activity counters and Global Statistics are updated.
FDX_RX_VL_ENA_STAT	X			Virtual Link is enabled for Global statistics only, frames are discarded
FDX_RX_VL_ENA_CNT	X	X		Virtual Link is enabled for Global Statistics and VL Activity, but frames are discarded. Helpful to see activity on the bus without storing any frames. This is the default for all Virtual Links in VL oriented receive mode.
FDX_RX_VL_ENA_MON_ALL	X	X	X	Virtual Link is enabled for Global Statistics and VL-Activity. Both good and erroneous frames are stored. Only applicable in chronologic receive mode
FDX_RX_VL_ENA_MON_GOOD	X	X	X	Virtual Link is enabled for Global Statistics and VL-Activity. Only good frames are stored, erroneus frames are discarded. Only applicable in chronologic receive mode.
FDX_RX_VL_ENA_EXT	X	X	X	Virtual Link is enabled for Global Statistics, VL-Activity and for extended operation mode. In this mode additional checks and verification modes can be configured for the VL with the structure TY_FDX_RX_VL_DESCRIPTION. Frames are stored dependent on the result of the checks and dependent on the FDX_RX_VL_CHECK_INVPAC flag.

G = enabled for Global Statistics
A = enabled for VL-Activity
S = frames are stored and forwarded to chronologic monitor or UDP-Ports

Note:
In VL oriented receive mode only ul_EnableMode = FDX_RX_VL_ENA_EXT is applicable.
On ASC-FDX devices only FDX_RX_VL_ENA_EXT and FDX_RX_VL_DIS is supported at this time.

AiUInt32 ul_PayloadMode

Defines the payload mode for data flow reduction. Dependent on the configured payload mode the entire frame or only parts of it will be stored.

Value	Description
FDX_PAYLOAD_FULL	Frames will be stored with full payload in the Monitor buffer. So the full Ethernet (MAC) frame is available for the application.
FDX_PAYLOAD_IP_EXT	Only the Frame header, the MAC- /IP- frame headers plus 20 bytes of the IP-Payload will be stored.
FDX_PAYLOAD_IP	Only the Frame header, the MAC- /IP- frame headers will be stored.
FDX_PAYLOAD_MAC	Only the Frame header and the MAC- frame header will be stored in the corresponding data buffer.
FDX_PAYLOAD_DEFAULT	use default Value

Note:
In VL oriented receive mode only ul_PayloadMode = FDX_PAYLOAD_FULL is applicable.
This functionality is deprecated and only available for API/AMC/APU/GNET devices. On all other devices only FDX_PAYLOAD_FULL is applicable.

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 Section 3.4.3.10 "FdxCmdMonTrgIndexWordIniVL") function.

Note:
In VL oriented receive mode this parameter is not applicable.
This functionality is deprecated and only available for API/AMC/APU/GNET devices.

TY_FDX_RX_VL_DESCRIPTION* px_VLDesc

Pointer to a structure that contains the verification mode and additional parameters of the VL. The structure is used only if ul_EnableMode is FDX_RX_VL_ENA_EXT, otherwise pointer can be set to NULL.

```
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;
```

AiUInt32 ul_VerificationMode

The Verification mode for the given VL is a bitfield of the flags below. It includes additional checks and functions. It is only applicable if the VL is set to ***FDX_RX_VL_ENA_EXT*** mode. The result of the checks will be reported in the receive header of each frame (See 3.4.3.4 TY_FDX_FRAME_BUFFER_HEADER in ***FdxCmdMonQueueRead***). Furthermore, the verification mode, result counters and status-flags of the checks are available via ***FdxCmdRxVLGetActivity***.

If the FDX_RX_VL_CHECK_INVPAC flag is set, all frames will be stored in the chronological monitor or corresponding UDP ports. Otherwise, frames which failed the checks will be discarded.

Details of the verification modes and checks are described in ARINC664 Part 7 Specification [7]

Value	Description	Comment
FDX_RX_VL_CHECK_DISA	Verification disabled	
FDX_RX_VL_CHECK_ENA_DEFAULT	Default Setting	This setting depends on the port configuration (see FdxCmdBoardControl). If the port is configured as single: FDX_RX_VL_CHECK_TRAFIC and FDX_RX_VL_CHECK_FRAMESIZE are enabled. If the port is configured as redundant: FDX_RX_VL_CHECK_REDMAM, FDX_RX_VL_CHECK_FRAMESIZE and FDX_RX_VL_CHECK_SNINTEG are enabled.
FDX_RX_VL_CHECK_REDMAM	Redundancy Management	Enable redundancy Management
FDX_RX_VL_CHECK_TRAFIC	Traffic Policing	Enable traffic policing verification. A byte based policing algorithm is used.
FDX_RX_VL_CHECK_FRAMESIZE	VL specific frame size 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 is checked
FDX_RX_VL_CHECK_INVPAC	Invalid Packet processing	All Packets, including the erroneous or those which would be discarded by the checks, will be stored and forwarded

AiUInt32 ul_Bag

Bandwidth Allocation Gap (BAG) for the defined Virtual Link. The BAG is necessary for the Redundancy Management and Traffic Policing function.
As shown below, this parameter is encoded 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, a range from 1 to 1000 is allowed. However the ARINC664 Part 7 Specification [7] defines only 8 valid values for the BAG (1, 2, 4, 8,..128ms).
If specifying the range in microseconds, the 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 also set 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***).

`AiUInt32 ul_Jitter`

Maximum allowed jitter value in µs, for the given Virtual Link. Necessary for the traffic policing function.
Possible range for the jitter 1 to 65535 µs. In any case the jitter should be less or equal than the BAG.

`AiUInt32 ul_MaxFrameLength`

Maximum length of a MAC frame on this Virtual Link in bytes. Necessary for the VL specific frame size check and traffic policing function. Valid range ul_MinFrameLength ... 1518

`AiUInt32 ul_MinFrameLength`

Minimum length of a MAC frame on this Virtual Link in bytes. Necessary for the traffic policing function. Valid range 64 ... ul_MaxFrameLength

`AiUInt32 ul_MaxSkew`

The maximum time difference in µs between the arrival time of two redundant frames with the same sequence number. Possible range for the MaxSkew is 0 to 65535 µs. This setting is necessary for the redundancy management function.

`AiUInt32 ul_VLBufSize`

Size of the local buffer in bytes which should be used to store data of the selected VL. This parameter is only applicable if the receive port works in VL oriented mode. If this value is 0 a default value will be used.

> **Note:**
> On ASC-FDX devices only 0 is applicable

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

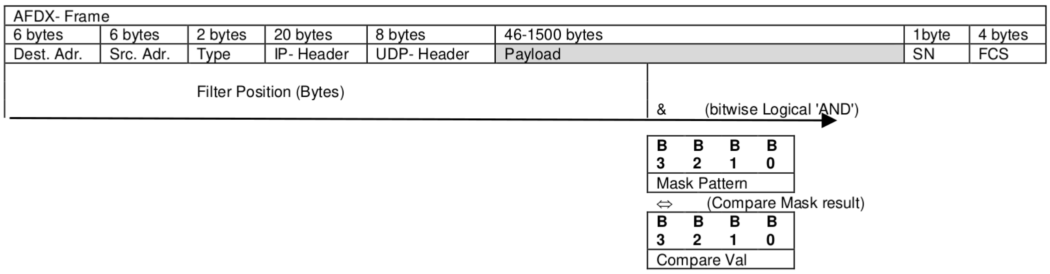


Figure 3.1: Mechanism of second level Filter

```
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_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

> **Note:**
> On ASC-FDX devices only FDX_DIS is applicable

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.

3.4.1.8 FdxCmdRxVLControlEx

Prototype:

```
AiReturn FdxCmdRxVLControlEx(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_STR	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.

3.4.1.9 FdxCmdRxVLGetActivity

Prototype:

```
AiReturn FdxCmdRxVLGetActivity(AiUInt32 ul_Handle,
                               const TY_FDX_RX_VL_ACTIVITY_IN* px_VLActivityIn,
                               Y_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). A VL is considered to be active when it has received data. 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	Get 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 if ul_Mode is set to FDX_RX_VL_ACT_VL (see above).

AiUInt32 ul_MaxReadBytes

Maximum number of bytes which can be written to the provided output array *pax_VLActivity.

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 array *pax_VLActivity. The array won't have any entries if ul_Mode is set to FDX_RX_VL_ACT_CNT.

If ul_Mode is FDX_RX_VL_ACT_VL, ul_NumOfActivVL is always 1.

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 device limitations, API/AMC/APU/APE/ACE/AXC/AMCX boards can collect statistics for a maximum number of 511 VLs per port. ASC-FDX boards can collect statistics for at most 291 VLs per port.

TY_FDX_RX_VL_ACTIVITY *pax_VLActivity

Pointer to an array of structured elements. 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

Operational mode of the given VL. (See Section 3.4.1.7 "FdxCmdRxVLControl") for details.

AiUInt32 ul_VerificationMode

Verification mode which is selected for the specified Virtual Link. (See Section 3.4.1.7 "FdxCmdRxVLControl") for details.

AiUInt32 ul_PayloadMode

Defines what part of the payload will be stored in the data buffer for a given VL. (See Section 3.4.1.7 "FdxCmdRxVLControl") for details.

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 with forward slash ('/') separated error abbreviations ((See Section 4.1 "List of Abbreviations")). Note: Error types may occur as combinations of the following constants.

[1] The "B" variables are only applicable if the handle used for this function references a port set to redundant. Then the VL-specific counter and error information for both ports are provided.

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_FRAM_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]

[1] The “B” variables are only applicable if the handle used for this function references a port set to redundant. Then the VL-specific counter and error information for both ports are provided.
[2] This counter is only applicable if the handle used for this function references a port set to redundant.

Each redundant frame received by this port must be discarded by the receiver. This counter counts all these frames.

AiUInt32 ul_EntriesRead

Number of VL entries of type TY_FDX_RX_VL_ACTIVITY written to the provided buffer. If the size of 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 the number of active VLs (ul_NumOfActivVL).

Return Value:

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

3.4.1.10 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. ul_VlRangeMin and ul_VlRangeMax.
FDX_ON	All erroneous frames are passed through to the upper following instances. ul_VlRangeMin and ul_VlRangeMax.

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. ul_VlRangeMin and ul_VlRangeMax.

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.

3.4.2 VL oriented Receiver Functions

These functions are only applicable, if the receive port is switched to Virtual Link oriented mode.

3.4.2.1 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 number of bytes that can be read per call to FdxCmdRxSAPBlockRead 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 shall be initialized with the size of pv_ReadBuffer in bytes. Initialization to 0 is also possible and assumes pv_ReadBuffer to have the required size (see description of pv_ReadBuffer below).

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 preceding 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.

3.4.2.2 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 Service Access Point receive port (SAP-Rx-Port), which is linked to a fixed specified UDP destination port, and can be used to receive messages from different IP/UDP-sources. Like a queuing receive port a SAP-Rx-Port can store received messages in a queue, handle variable message sizes and reassemble messages from multiple received fragments/packets. Messages can be retrieved from the queue with the function FdxCmdRxSAPRead.

Please note that several FxCmdRxUDP functions can also be used for SAP-Rx-Ports ***(FdxCmdRxUDPChgDestPort, FdxCmdRxUDPGetStatus, FdxCmdRxUDPControl, FdxCmdRxUDPDestroyPort)***. In order to identify the SAP port in further functions the returned ul_UdpHandle must be used.

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.

Input:

TY_FDX_RX_SAP_CREATE_IN* px_SapCreateIn

Pointer to a structure that contains the SAP port settings.

```
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

UDP destination address of this SAP port. Range from 0 to 65535. Can be freely chosen but port number allocation schemes and ICANN administered numbers should possibly be considered. The UDP destination address can be changed later while receiver is running with ***FdxCmdRxUDPChgDestPort***.

AiUInt32 ul_IpDst

The IPv4 destination address of this SAP port. The address should be a Class A private IP unicast address to identify the target subscriber or a Class D multicast address reflecting the VL. The destination address must respect specific addressing schemes.

`AiUInt32 ul_VlId`

The Virtual Link used to receive packets for this SAP port. The Virtual Link must have been enabled before for extended operation with ***FdxCmdRxVLControl***. Range from 0 to 65535

`AiUInt32 ul_UdpNumBufMessages`

Number of messages which can be stored by the SAP-Port in the associated queue. If this value is set to 0 a default value will be used.

`AiUInt32 ul_UdpMaxMessageSize;`

Maximum size of a message which can be received by this SAP port. Range from 0 to 8192 bytes.

> **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 that contains the handle for the created SAP port.

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

`AiUInt32 ul_UdpHandle`

Handle to the SAP port. This handle must be stored by the application and is used to identify the SAP port in further functions.

Return Value:

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

3.4.2.3 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 can be used for reading messages from a Service Access Point receive port (SAP- Rx-Port). SAP-Rx-Ports store received messages in a queue. ***FdxCmdRxSAPRead*** takes messages from this queue in FIFO order.
Please note that messages may be discarded depending on thesettings of ***FdxCmdRxVLControl*** (parameter ul_VerificationMode).

Input:

TY_FDX_RX_SAP_READ_IN* px_SapReadIn

```
typedef struct {
    AiUInt32 ul_UdpHandle;
    AiUInt32 ul_MsgCount;
} TY_FDX_RX_SAP_READ_IN;
```

AiUInt32 ul_UdpHandle

Handle of the SAP port to read messages from. This is the handle returned when the SAP port is created via ***FdxCmdRxSAPCreatePort***.

AiUInt32 ul_MsgCount

Maximum number of messages to read. The queue size has been defined by parameter ul_UdpNumBufMessages in ***FdxCmdRxSAPCreatePort***.

Output:

TY_FDX_RX_SAP_READ_OUT* px_SapReadOut

```
typedef struct {
    AiUInt32 ul_MsgRead;
    void*    pv_ReadBuffer;
} TY_FDX_RX_SAP_READ_OUT;
```

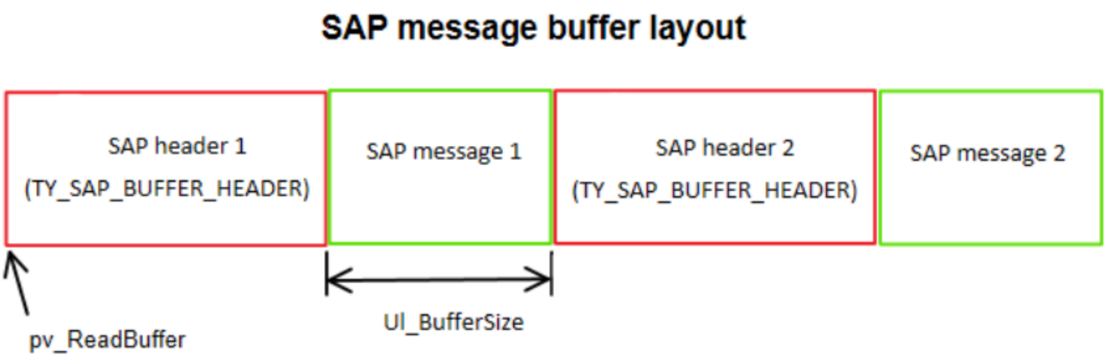
AiUInt32 ul_MsgRead

Number of messages actually read.

void* pv_ReadBuffer

Pointer to the data buffer where the messages will be stored. Required size of buffer can be calculated by (sizeof(TY_SAP_BUFFER_HEADER) + ul_UdpMaxMessageSize) * ul_MsgCount. The ul_UdpMaxMessageSize is defined with function ***FdxCmdRxSAPCreatePort***. User is responsible for the memory management of the buffer.

The messages are stored consecutively in this buffer. Each message is prepended by a buffer header that contains detailed information about the message. The buffer header of the first message starts at address pv_ReadBuffer .



TY_SAP_BUFFER_HEADER

```
typedef struct sap_buffer_Header {
    TY_FDX_FW_IRIG_TIME x_FwTimeTag;
    AiUInt32            ul_BufferSize;
    AiUInt32            ul_IPSrc;
    AiUInt32            ul_UDPSrcPort;
    AiUInt32            ul_ErrorInfo;
} TY_SAP_BUFFER_HEADER;
```

TY_FDX_FW_IRIG_TIME x_FwIrigTime

The firmware IRIG time tag information is from last received fragment of the message.

```
typedef struct {
    AiUInt32 ul_TtHigh;
    AiUInt32 ul_TtLow;
} TY_FDX_FW_IRIG_TIME;
```

AiUInt32 ul_TtHigh;

Time tag word in firmware format. The higher part of the time tag, containing the minutes of hour, hours of day and day of year.

AiUInt32 ul_TtLow;

Time tag word in firmware format. The lower part of the time tag, containing the microseconds of second, seconds of minutes and minutes of hour. (See Section 3.5.6 "FdxStructIrig2FwIrig") to get a 'C' structured information of the time tag.

`AiUInt32 ul_BufferSize;`

Payload size in bytes of received message.

`AiUInt32 ul_IPSrc`

The IPv4 source address of the transmitting UDP/SAP port.

`AiUInt32 ul_UDPSrcPort`

UDP source port address of the transmitting UDP/SAP port.

`AiUInt32 ul_ErrorInfo`

This parameter holds all errors, which occurred in the message reception, in a bitfield. Each error type is represented by a one bit error flag.

Bit	Error Type	Abbreviation
0-3	not used	
4	Reassembly error: First fragment missing	IP_REASS_ERROR_SYNC
5	Reassembly error: Fragments out of order	IP_REASS_ERROR_ORDER
6	Reassembly error: Fragmented message although fragmenting not allowed	IP_REASS_ERROR_FRAG
7	Reassembly error: Fragment size too big/small	IP_REASS_ERROR_SIZE
8	Reassembly error: Message exceeded maximum message size	IP_REASS_ERROR_BUF
9-15	not used	
16	Wrong physical symbol during frame reception.	PHY
17	Wrong Preamble/Start Frame Delimiter received.	PRE
18	Unaligned frame length received (Triple Nibble Error).	TRI
19	MAC CRC Error.	CRC
20	Short Interframe Gap Error (<960ns)	IFG
21	AFDX IP Framing Error (AFDX-IP frame specific settings violated).	IPE
22	AFDX MAC Framing Error (AFDX-MAC frame specific settings violated).	MAE
23	Frame without valid Start Frame Delimiter received	SFD
24	Long Frame Received (> 1518 Bytes)	LNG
25	Short Frame Received (< 64 Bytes)	SHR
26	VL specific Frame size Violation	VLS
27	Sequence No. mismatch	SNE
28	not used	RS2
29	Traffic Shaping Violation	TRS
30-31	not used	

Return Value:

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

3.4.2.4 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_BLOCK_READ_IN_PORT;
```

AiUInt32 ul_UdpHandle

The handle of the UDP port from which the message(s) will be read. 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 number of bytes that can be read per call to FdxCmdRxUDPBlockRead 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 shall be initialized with the size of pv_ReadBuffer in bytes. Initialization to 0 is also possible and assumes pv_ReadBuffer to have the required size (see description of pv_ReadBuffer below).

`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.

3.4.2.5 FdxCmdRxUDPControl

Prototype:

```
AiReturn FdxCmdRxUDPControl(AiUInt32 ul_Handle,
                            const AiUInt32 ul_UdpHandle,
                            const TY_FDX_RX_UDP_CONTROL* px_UdpControl);
```

Purpose:

This function is used to configure several settings of a previously created UDP receive port.

Input:

`AiUInt32 ul_UdpHandle`

Handle of the UDP receive port to configure. This is the handle returned by FdxCmdRx-UDPCreatePort (3.4.2.7).

`TY_FDX_RX_UDP_CONTROL* px_UdpControl`

Pointer to a structure containing the parameters to configure.

```
typedef struct {
    AiUInt32 ul_NetSelect;
    AiUInt32 ul_InterruptControl;
} TY_FDX_RX_UDP_CONTROL;
```

`AiUInt32 ul_NetSelect`

Specifies the network the UDP receive port is bound to.

Value	Description
FDX_RX_UDP_BOTH	Receive messages from both networks
FDX_RX_UDP_ONLY_A	Receive messages from network A only
FDX_RX_UDP_ONLY_B	Receive messages from network B only

`AiUInt32 ul_InterruptControl`

Specifies if user shall be notified about reception of messages on the port.

Value	Description
FDX_RX_UDP_NOINT	Notification disabled
FDX_RX_UDP_INT	Notification enabled
FDX_RX_UDP_UDF	Not implemented

Note:
Not supported on ASC-FDX-2

Output:

None

Return Value:

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

3.4.2.6 FdxCmdRxUDPChgDestPort

Prototype:

```
AiReturn FdxCmdRxUDPChgDestPort(AiUInt32 ul_Handle,
                                AiUInt32 ul_UdpHandle,
                                AiUInt32 ul_UdpDst);
```

Purpose:

Changes the UDP destination port of an existing SAP-Rx or UDP-Rx port. This function can be used while receiver is running.

Input:

`AiUInt32 ul_UdpHandle`

This handle identifies the SAP-Rx- or UDP-Rx-Port to be changed and is returned by ***FdxCmdRxSAPCreatePort*** or ***FdxCmdRxUDPCreatePort***.

`AiUInt32 ul_UdpDst`

Destination UDP port. Part of the UDP quintuplet.
The new UDP destination address of this port. Range from 0 to 65535. Can be freely chosen but port number allocation schemes and ICANN administered numbers should possibly be considered. There is no check if the new UDP destination is already in use by another SAP-Rx or UDP-Rx port, so the user is responsible for a unique mapping between API ports and UDP destination addresses.

Output:

None

Return Value:

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

3.4.2.7 FdxCmdRxUDPCreatePort

Prototype:

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

Purpose:

This function creates a UDP-Rx-Port to receive message data from a specified UDP connection oriented port over a specified VL.

Messages received by a UDP port can be retrieved via a call to ***FdxCmdRxUDPRead***.

UDP ports come in two types: sampling or queuing. A UDP sampling port can store only a single received message and any new message sent to the port will overwrite the message currently being stored. Therefore a UDP sampling port must be read promptly to avoid data loss caused by the reception of a new message.

If it is desired that the port capture a stream of messages, then a UDP queuing port should be used. As the name suggests a UDP queuing port stores received messages in a queue by order of reception. Newly received messages are placed in the back of the queue rather than overwriting older messages. The queue has a fixed size which can be set in the ***px_UdpDescription*** input parameter when calling ***FdxCmdRxUDPCreatePort***. UDP queuing ports may also receive larger messages compared to UDP sampling ports (see the ***ul_UdpMaxMessageSize*** below).

A connection between a transmitting and receiving UDP port is defined by the quintuplet ***X_Quint*** in the input parameter ***px_UdpDescription***. The UDP port will only receive messages whose source and destination addresses match the connection defined by the port's quintuplet. Once created a UDP-Rx-Port's destination address can be changed with the function ***FdxCmdRxUDPChgDestPort***.

This function assumes VL reception mode was previously enabled via a call to ***FdxCmdRxVLControl***. This function cannot be used while the receiver is running.

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 Frames. Reassembling of the MAC frames into the original message will be done in the IP layer. A Message can have a size of up to 8kByte.

struct _quintuplet

This structure defines a connection between a transmitting and receiving UDP port.

AiUInt32 x_Quint.ul_UdpSrc

UDP source address of the transmitting UDP port from which this UDP port will receive messages. Range from 0 to 65535. Can be freely chosen but port number allocation schemes and ICANN administered numbers should possibly be considered.

AiUInt32 x_Quint.ul_IpSrc

The IPv4 source address of the transmitting UDP port from which this UDP port will receive messages. The address should be a Class A private IP unicast address to identify the target subscriber or a Class D multicast address reflecting the VL.

AiUInt32 x_Quint.ul_VlId

Virtual Link Identifier for the virtual link on which the incoming frames will be received. Range from 0 to 65535. The virtual link must have been enabled previous to calling FdxCmdRxUDPCreatePort via a call to FdxCmdRxVLControl.

AiUInt32 x_Quint.ul_IpDst

The IPv4 destination address of this UDP port. The address should be a Class A private IP unicast address to identify the target subscriber or a Class D multicast address reflecting the VL.

AiUInt32 x_Quint.ul_UdpDst

UDP destination address of this UDP port. Range from 0 to 65535. Can be freely chosen but port number allocation schemes and ICANN administered numbers should possibly be considered. The UDP destination address can be changed later while receiver is running.

AiUInt32 uw_SubVlId;

Not relevant in Rx Mode.

AiUInt32 ul_UdpNumBufMessages

Maximum number of messages which can be stored by the UDP port. For sampling ports this value must be set to 1. For queuing ports an adequate size must be provided. If this value is set to zero a default value will be used.

AiUInt32 ul_UdpMaxMessageSize;

Maximum size of a message which this port is able to receive. For a UDP sampling port this is the message size without the header overhead (MAC, IP and UDP).

Port Type	Value
Sampling	0..1471 bytes
Queuing	0..8 kBytes

Note:
If a received message exceeds the maximum size this message will be truncated 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. This handle must be stored by the application and is used to identify the UDP port in further functions.

Return Value:

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

3.4.2.8 FdxCmdRxUDPDestroyPort

Prototype:

```
AiReturn FdxCmdRxUDPDestroyPort(AiUInt32 ul_Handle,
                                const AiUInt32 ul_UdpHandle);
```

Purpose:

This function is used to destroy an unneeded UDP receive port. It is therefore the opposite of functions ***FdxCmdRxUDPCreatePort*** and ***FdxCmdRxSAPCreatePort***. It can be used only when the receiver is not running.

Input:

`AiUInt32 ul_UdpHandle`

Handle of the UDP port to be destroyed. This is the handle returned when the UDP port is created via one of the functions ***FdxCmdRxUDPCreatePort*** or ***idxCmdRxSAPCreatePort***.

Output:

None

Return Value:

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

3.4.2.9 FdxCmdRxUDPGetStatus

Prototype:

```
AiReturn FdxCmdRxUDPGetStatus(AiUInt32 ul_Handle,
                              const AiUInt32 ul_UdpHandle,
                              TY_FDX_RX_UDP_STATUS* px_UdpRxStatus);
```

Purpose:

This function is used to retrieve the status of the specified UDP/SAP receive port.

Input:

`AiUInt32 ul_UdpHandle`

This handle identifies the SAP-Rx- or UDP-Rx-Port to get status of and is returned by ***FdxCmdRxSAPCreatePort*** or ***FdxCmdRxUDPCreatePort***.

Output:

`TY_FDX_RX_UDP_STATUS* px_UdpRxStatus`

```
typedef struct {
        AiUInt32 ul_MsgCount;
        AiUInt32 ul_MsgErrorCount;
} TY_FDX_RX_UDP_STATUS;
```

`AiUInt32 ul_MsgCount`

Total number of messages received on this specific port since it had been created. Counts only error-free messages.

`AiUInt32 ul_MsgErrorCount`

Total number of erroneous messages received on this specific port since it had been created. Counts only erroneous messages.

Erroneous messages are also stored in message buffer and can be retrieved by calling FdxCmdRxUDPRead, FdxCmdRxSAPRead respectively. Detailed information about the errors occurred is provided in the header of these messages after reading them from message buffer.

Note:
ul_MsgCount and ul_MsgErrorCount reflect all messages received for this particular UDP/SAP port. Though user may not be able to retrieve all of them from port's message buffer in case the message buffer is full and messages get discarded.

Return Value:

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

3.4.2.10 FdxCmdRxUDPRead

Prototype:

```
AiReturn FdxCmdRxUDPRead(AiUInt32 ul_Handle,
                         const AiUInt32 ul_UdpHandle,
                         AiUInt32 ul_MsgCount,
                         AiUInt32* pul_MsgRead,
                         void* pv_ReadBuffer);
```

Purpose:

This function can be used for reading messages from a specified UDP port.

UDP Queuing RxPorts store received messages in a queue. ***FdxCmdRxUDPRead*** takes messages from this queue in FIFO order.

UDP Sampling RxPorts store at most a single message. ***FdxCmdRxUDPRead*** returns the message stored in the sampling port. Further calls to ***FdxCmdRxUDPRead*** will return the same message until a new sampling message is received.

Please note that messages may be discarded depending on the settings of ***FdxCmdRxVLControl*** (parameter ul_VerificationMode).

Input:

`AiUInt32 ul_UdpHandle`

Handle of the UDP port to read messages from. This is the handle returned when the UDP port is created via ***FdxCmdRxUDPCreatePort***.

`AiUInt32 ul_MsgCount`

Maximum number of messages to read.

Output:

`AiUInt32* pul_MsgRead`

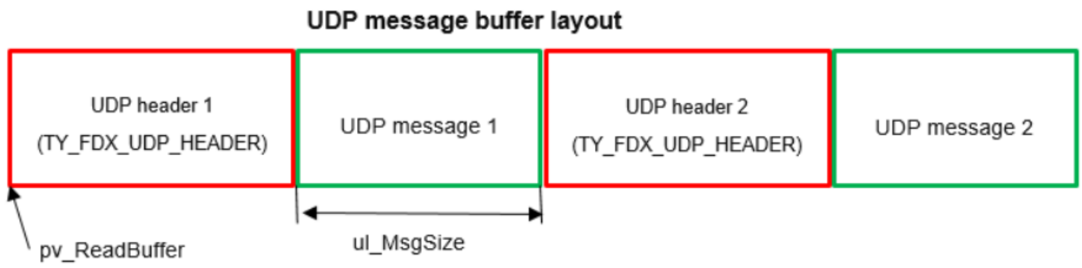
Number of messages actually read. Might be smaller than the number of messages requested with ul_MsgCount. E.g. in the case of sampling ports, which can only store a single message. Or i n the case of queuing ports when fewer messages than requested are available in the queue.

`void* pv_ReadBuffer`

Pointer to the data buffer where the messages will be stored. Required size of buffer can be calculated by (sizeof(TY_FDX_UDP_HEADER) + ul_UdpMaxMessageSize) * ul_MsgCount.

The ul_UdpMaxMessageSize is defined with function FdxCmdRxUDPCreatePort. User is responsible for the memory management of the buffer.

The messages are stored consecutively in this buffer. Each message is prepended by a buffer header that contains detailed information about the message. The buffer header of the first message starts at address pv_ReadBuffer.



TY_FDX_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 fragment of the message.

```
typedef struct {
    AiUInt32 ul_TtHigh;
    AiUInt32 ul_TtLow;
} TY_FDX_FW_IRIG_TIME;
```

AiUInt32 ul_TtHigh;

Time tag word in firmware format. The higher part of the time tag, containing the minutes of hour, hours of day and day of year.

AiUInt32 ul_TtLow;

Time tag word in firmware format. The lower part of the time tag, containing the microseconds of second, seconds of minutes and minutes of hour.

(See Section 3.5.3 "FdxFwIrig2StructIrig") to get a ‘C’ structured information of the Time Tag.

AiUInt32 ul_MsgSize;

UDP payload size in bytes of received Frame.

AiUInt32 ul_ErrorInfo

This parameter holds all errors, which occurred in the message reception, in a bitfield. Each error type is represented by a one bit error flag.

Bit	Error Type	Abbreviation
0-3	not used	
4	Reassembly error: First fragment missing	IP_REASS_ERROR_SYNC
5	Reassembly error: Fragments out of order	IP_REASS_ERROR_ORDER
6	Reassembly error: Fragmented message although fragmenting not allowed	IP_REASS_ERROR_FRAG
7	Reassembly error: Fragment size too big/small	IP_REASS_ERROR_SIZE
8	Reassembly error: Message exceeded maximum message size	IP_REASS_ERROR_BUF
9-15	not used	
16	Wrong physical symbol during frame reception.	PHY
17	Wrong Preamble/Start Frame Delimiter received.	PRE
18	Unaligned frame length received (Triple Nibble Error).	TRI
19	MAC CRC Error.	CRC
20	Short Interframe Gap Error (<960ns)	IFG
21	AFDX IP Framing Error (AFDX-IP frame specific settings violated).	IPE
22	AFDX MAC Framing Error (AFDX-MAC frame specific settings violated).	MAE
23	Frame without valid Start Frame Delimiter received	SFD
24	Long Frame Received (> 1518 Bytes)	LNG
25	Short Frame Received (< 64 Bytes)	SHR
26	VL specific Frame size Violation	VLG
27	Sequence No. mismatch	SNE
28	not used	RS2
29	Traffic Shaping Violation	TRS
30-31	not used	

Return Value:

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

3.4.3 Chronological Monitor

These functions are only applicable, if the receive port is switched to chronological mode.

3.4.3.1 FdxCmdMonCaptureControl

Prototype:

```
AiReturn FdxCmdMonCaptureControl(AiUInt32 ul_Handle,
                                 const TY_FDX_MON_CAP_MODE *px_MonCapMode);
```

Purpose:

This function is used to select a specific capture mode when monitoring network traffic chronologically

Input:

TY_FDX_MON_CAP_MODE *px_MonCapMode

Pointer to a structure that holds the capture mode options.

```
typedef struct {
    AiUInt32 ul_CaptureMode;
    AiUInt32 ul_TriggerPosition;
    AiUInt32 ul_Strobe;
} TY_FDX_MON_CAP_MODE;
```

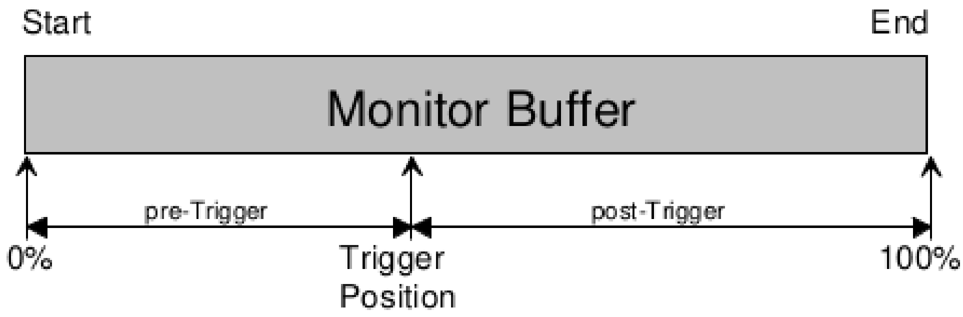
AiUInt32 ul_CaptureMode

Selects the capture mode for chronological monitoring.

Mode	Value	Description	API/AMC-FDX	APE/ACE/AXC/AMCX-FDX	ASC-FDX
Single	FDX_MON_SINGLE	The monitor buffer will be filled one time. Then the capturing stops. The data will be available after first frame captured and can be retrieved by using FdxCmdMonQueueRead.	✓	✓	✓
Continuous	FDX_MON_CONTINUOUS	The monitor buffer will be written continuously until capturing is stopped. The data will be available after first frame captured and can be retrieved by using FdxCmdMonQueueRead. The function FdxCmdMonQueueRead must be called subsequently fast enough to prevent overflow condition.	✓	✓	✓
Continuous_2	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. See chapter 'Continuous Capture Second Edition Functions' for detailed information.	✓	✓	X
Selective	FDX_MON_SELECTIVE	The monitor buffer will be filled one time. Then the capturing stops. Only frames are stored to the buffer which match certain trigger conditions. Trigger conditions can be defined by using Trigger Control Blocks ((See Section 3.4.3.8 "FdxCmdMonTCBSetup")) to filter network traffic. (e.g. capture only frames that: - are erroneous or - have a specific bit pattern set or - are received while an external event is active ...)..	✓	X	X
Selective Continuous	FDX_MON_CONT_SELECTIVE	Same mode as described for Selective but monitor buffer will be filled as in Continuous mode.	✓	X	X
Selective Continuous_2	FDX_MON_CONT_SE_SELECTIVE	Same mode as described for Selective but monitor buffer will be filled as in Continuous_2 mode.	✓	X	X
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.	✓	X	X
Selective Recording	FDX_MON_RECORDING_SELECTIVE	Same mode as described for Selective but with capabilities of Recording Mode.	✓	X	X

AiUInt32 ul_TriggerPosition

This is a value between 0 and 100 %.
The trigger position is only relevant in Single mode and Selective 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`

> **Note:**
> The strobe generation is only available for API-FDX, AMC-FDX, APU-FDX and ASC-FDX.

Value	Description
FDX_MON_STROBE_DIS	Discrete output strobe disabled
FDX_MON_STROBE_STOP	Discrete output strobe is generated if capturing is stopped (monitor buffer is full) in Single or Selective Capture mode. In Continuous or Recording mode, the Discrete output strobe is generated, each time half of the monitor buffer has been filled. **Note:** ASC-FDX does not support output strobe when half of the buffer has been filled. Instead, the strobe is generated when the buffer is full.
FDX_MON_STROBE_START	Discrete output strobe is generated once, when capturing is triggered (In Single, Continuous or Recording mode). In Selective capture mode, the strobe is generated each time the capturing of a frame is triggered.

Output:

None

Return Value:

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

3.4.3.2 FdxCmdMonGetStatus

Prototype:

```
AiReturn FdxCmdMonGetStatus(AiUInt32 ul_Handle
                            TY_FDX_E_MON_STATUS *pe_MonStatus,
                            TY_FDX_MON_REC_STATUS *px_MonRecStatus);
```

Purpose:

This function is used to get the monitor status of a certain port.

Input:

None

Output:

TY_FDX_E_MON_STATUS *pe_MonStatus

Pointer to a monitor port status information structure

```
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;
```

Status information of the monitor

Status:	Description
FDX_MON_OFF	Monitor is not running. Captured frames still in buffer can be read using FdxCmdMonQueueRead.
FDX_MON_WAIT_FOR_TRIGGER	Monitor is waiting for trigger.
FDX_MON_TRIGGERED	Start trigger occurred. Frames are being captured.
FDX_MON_STOPPED	Capturing has been suspended due to an incoming frame that triggered stop condition. Only valid in selective capture modes.
FDX_MON_FULL	Only valid in single shot capture mode. Monitor buffer is full and no more frames are captured.
FDX_MON_OVERLOAD	Capturing was stopped due to overload. Not all frames could be stored to the associated buffers. Monitor has to be restarted.

Figure 3.2 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.

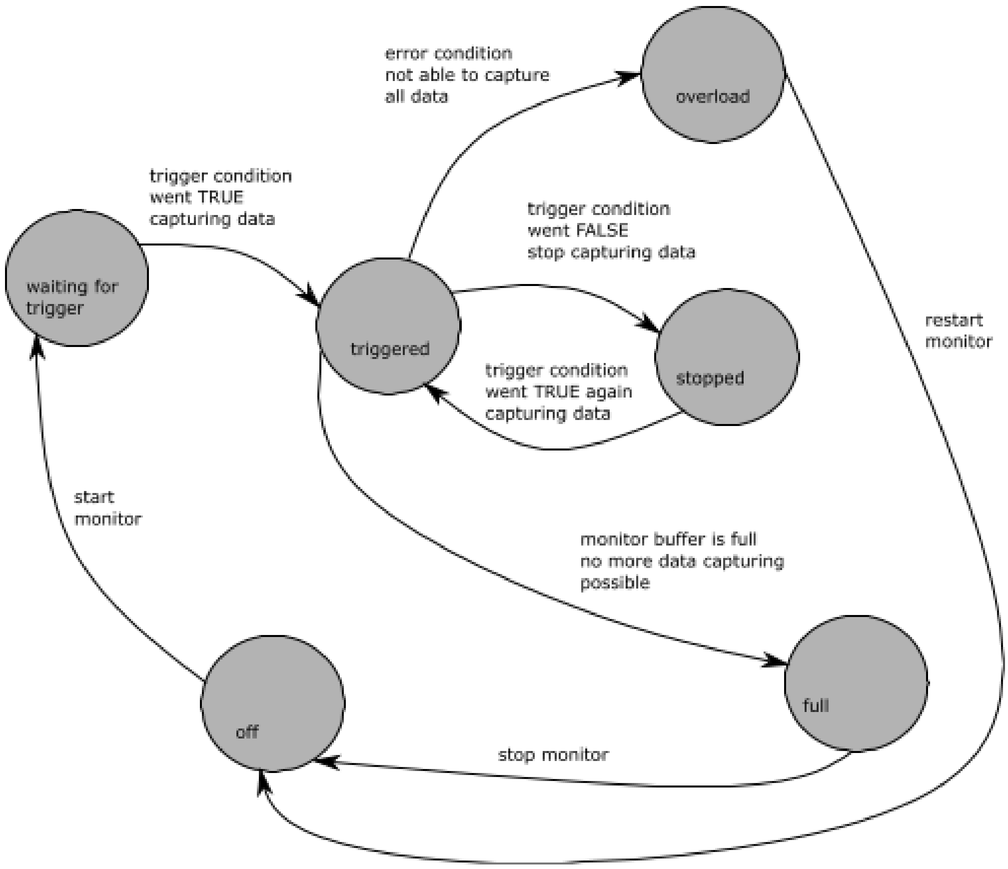


Figure 3.2: States of the Chronological Monitor

`TY_FDX_MON_REC_STATUS *px_MonRecStatus`

Additional information about captured frames

```
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`

Only valid in mode FDX_MON_RECORDING and FDX_MON_RECORDING_SELECTIVE. Indicates how many bytes have been recorded yet.

Return Value:

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

3.4.3.3 FdxCmdMonQueueControl

Prototype:

```
AiReturn FdxCmdMonQueueControl(AiUInt32 ul_Handle,
                               const TY_FDX_MON_QUEUE_CTRL_IN *px_In,
                               TY_FDX_MON_QUEUE_CTRL_OUT *px_Out);
```

Purpose:

This function is used to create or delete a queue, used to read from the chronological monitor. If the queue is created a queue ID is generated, which must be used in all following queue related functions like ***FdxCmdMonQueueRead*** or ***FdxCmdMonQueueStatus***. The queue can only be created once per port. Before this function can be used, chronological monitoring ***FDX_RX_CHRONO*** must be configured with ***FdxCmdRxModeControl*** and a specific capture mode should be selected by ***FdxCmdMonCaptureControl***.

Input:

`TY_FDX_MON_QUEUE_CTRL_IN * px_In`

Input structure to control the queue

```
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`

Identifies the queue to delete (only needed if the Queue Control Code is set to ***FDX_MON_QUEUE_DELETE***)

`AiChar ac_RecordingFileName`

Specifies the filepath of the recording file to be created. Only relevant for recording mode ***FDX_MON_RECORDING*** 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 A664-board is present). So drive and path information is related to the server.

`AiUInt32 ul_MaxFileSize`

Specifies the maximum file size in Mbytes for the recording file to be created. Onlyrelevant for recording mode ***FDX_MON_RECORDING*** and if the Queue Control Code is set to ***FDX_MON_QUEUE_CREATE***

Output:

`TY_FDX_MON_QUEUE_CTRL_OUT *px_Out`

Holds the ID of the created queue

```
typedef struct {
    AiUInt32 ul_QueueId;
} TY_FDX_MON_QUEUE_CTRL_OUT;
```

`AiUInt32 ul_QueueId`

Queue Identifier.

Return Value:

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

3.4.3.4 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 frames of a chronological capturing queue into a user provided buffer. If you need to manually set (or get) the reading start position, (See Section 3.4.3.5 "FdxCmdMonQueueSeek") (or (See Section 3.4.3.6 "FdxCmdMonQueueTell")).

> **Note:**
> On boards that support replay, 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 input parameter structure for this function.

```
typedef struct {
    AiUInt32 ul_EntryCount;
    AiUInt32 ul_ReadQualifier;
    AiUInt32 ul_MaxReadBytes;
} TY_FDX_MON_QUEUE_READ_IN;
```

AiUInt32 ul_EntryCount,

Number of frames to read from queue.

AiUInt32 ul_ReadQualifier

This is a qualifier, indicating which part of the frame should be read.

Value	Description:
FDX_MON_READ_FULL	Read the full frame
FDX_MON_READ_HEADER	Read only the fixed header
FDX_MON_READ_DATA	Read only the data, starting with the MAC Frame without fixed header

Note:
On ASC-FDX only FDX_MON_READ_FULL is supported

AiUInt32 ul_MaxReadBytes,

Size of the provided data buffer (vpv_ReadBuffer) in bytes. The size of data which can be read in a single call to FdxCmdMonQueueRead is limited. The limit is dependent on the board you are working on, and the operating system used by the host.
The following table outlines the limits:

Board	Windows / Linux		LabView RT
Mode	Continuous Capture	Single Capture	
API-FDX, AMC-FDX, APU-FDX	12 Mbyte	48 kbyte	48 kbyte
APE-FDX, AMCX-FDX, AXC-FDX, ACE-FDX	12 Mbyte	12 Mbyte	48 kbyte
ASC-FDX	16 kbyte	16 kbyte	

Output:

TY_FDX_MON_QUEUE_READ_OUT* px_QueueReadOut

Pointer to structure that holds output parameters.

```
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 frames actually read.

AiUInt32 ul_BytesRead

Number of bytes actually read.

AiUInt32 ul_Synchronized

In mode FDX_MON_SELECTIVE or FDX_MON_CONTINUOUS, a value other than 0 indicates an error. Some frames may be lost. Capture data is invalid and capturing must be restarted.

AiUInt64 vpv_ReadBuffer

Address of the data buffer where the frames are to be stored. The size of this buffer must correspond to ul_MaxReadBytes. The user of the API needs to allocate an appropriate buffer and provide the address here.

Data Buffer:

The frames are stored consecutively in this buffer. Each frame is prepended by a receive header that contains detailed information about the frame. The receive header of first frame starts at address vpv_ReadBuffer.
Different device platforms use different types of receive headers. These are
TY_FDX_FRAME_BUFFER_HEADER
for PCI/USB based devices (API/AMC/APU/ASC-FDX) and
TY_GNET_FRAME_BUFFER_HEADER
for PCIe based devices (APE/ACE/AXC/AMCX-FDX).
In order to iterate over all the frames in the buffer, add the buffer size as encoded in the current receive header to the address of the current receive header to get the address of the next receive header. I.e. in case of TY_FDX_FRAME_BUFFER_HEADER:
next_address = current_address + receive_header.x_FrameHeaderInfo.uw_BufferSize
In case of TY_GNET_FRAME_BUFFER_HEADER:
next_address = current_address + receive_header.x_EntrySizeWord.TotalDataBufferSize
Following figure visualizes the general layout of the frame buffer after calling FdxCmdMonQueueRead:

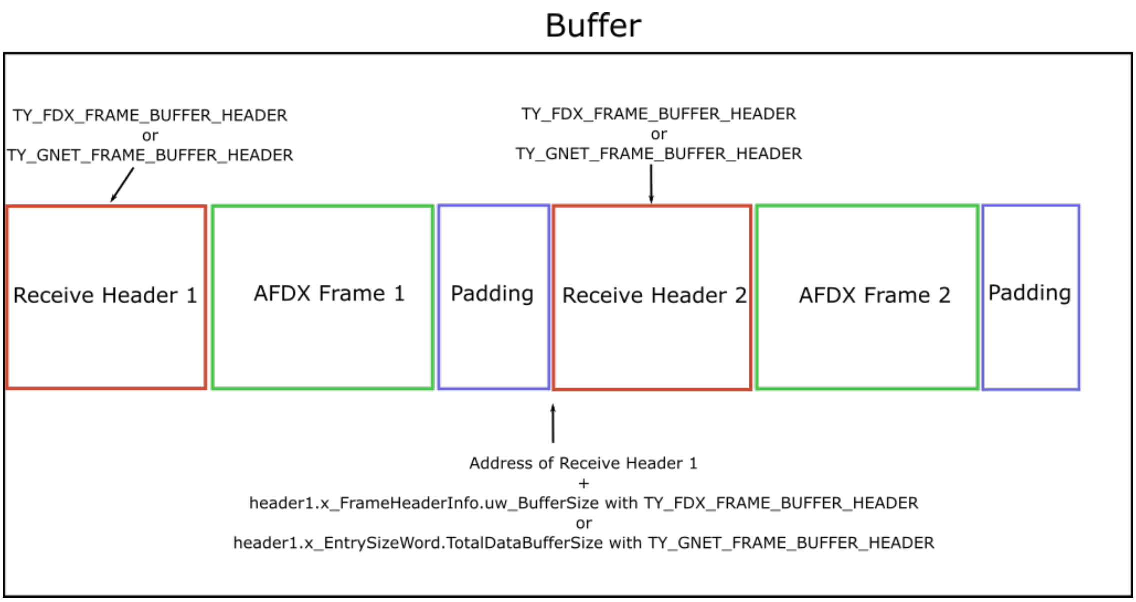


Figure 3.3: Frame Buffer Layout

Receive Headers:

The API offers C structures to decode the individual fields of the receive headers. Now follows a detailed description for both header variants. If your application is running on a Big Endian System, you need to call FdxProcessMonQueue() before accessing any of the fields in the receive headers.

TY_FDX_FRAME_BUFFER_HEADER

This is a structural description of the receive header for frames of PCI and USB based devices.

```
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;

Reserved.

AiUInt32 ul_Next;

Reserved.

AiUInt32 ul_EntryControl

Additional Information about the captured frame

Value	Bit	Description
Frame receive header type	31..30	0: Default receive frame header 3: APE-FDX /APX-GNET frame header 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 of IP payload stored 2: Frame header, MAC and IP header stored 3: Frame header and MAC header stored other values: Reserved
Start trigger flag	26	This bit indicates this frame has triggered the start of capturing
	25..24	Reserved
AFDX Port Map ID	23..16	User defined port number which is associated with the physical port (PortMapID (See Section 3.4.1.4 "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.

```
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 the following information:

AiUInt16 uw_VlId

Virtual Link Identifier associated with the frame

AiUInt16 uw_ErrorField

This bit-oriented information is a cumulative list of the error types which have occurred. The library function ***FdxTranslateErrorWord*** translates the given Error Word into a zero terminated string containing '/' separated error abbreviations ((See Section 4.1 "List of Abbreviations"))

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
14-15	not used	

`AiUInt32 ul_FrameHeaderWord_1`

Frame Header Word 1 contains the following information:

Value	Bit	Description
Network ID	31..29	The network on which the frame was received 001: Network A 010: Network B others: reserved
Interframe Gap High	12	Indicates 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 Error.	
Byte Count	10..0	Byte count of the received AFDX / MAC – frame (incl. CRC bytes).

Frame Header Word 2 contains the following information:

AiUInt8 uc_SequenceNr

Sequence number (copied from AFDX frame)

AiUInt8 uc_Reserved1

Reserved

AiUInt16 uw_BufferSize

Size of the frame in bytes including alignment padding.

Note:
On API/AMC/APU boards, the CRC bytes do not contain valid data on this level.

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 {
typedef struct {
    AiUInt32 ul_TtHigh;
    AiUInt32 ul_TtLow;
} TY_FDX_FW_IRIG_TIME;
```

AiUInt32 ul_TtHigh;

Time tag word in firmware format. The higher part of the time tag contains the minutes of hour, hours of day and day of year.

AiUInt32 ul_TtLow;

Time tag word in firmware format. The lower part of the time tag contains the microseconds of second, seconds of minutes and minutes of hour.

(See Section 3.5.3 "FdxFwIrig2StructIrig") to get a 'C' structured information of the time tag.

TY_GNET_FRAME_BUFFER_HEADER

This is a structural description of the frame receive header for frames of PCIe-based devices.

```
typedef struct _gnet_frame_buffer_header {
    TY_GNET_FRAME_BUFFER_CONTROL_WORD x_EntryControlWord;
    TY_GNET_FRAME_BUFFER_SIZE_WORD x_EntrySizeWord;
    TY_GNET_FRAME_HEADER_TYPE_WORD x_FrameHeaderTypeWord;
    AiUInt32 ul_NextPointer;
    TY_GNET_FRAME_HEADER_INFO_WORD_0 x_FrameHeaderInfoWord0;
    TY_GNET_FRAME_HEADER_INFO_WORD_1 x_FrameHeaderInfoWord1;
    TY_FDX_FW_IRIG_TIME x_FwIrigTime;
} TY_GNET_FRAME_BUFFER_HEADER;
```

For each structure member variable there is an additional structure defined in the appropriate header file.

TY_GNET_FRAME_BUFFER_CONTROL_WORD x_EntryControlWord

Additional information about the captured frame

Value	Bit	Description
Reserved	31..30	Reserved
Frame discarded	27	Frame was marked as discarded during RX verification tests.
Statistics	26..24	0 = short frame error 0-63Bytes, 1 = 64-127 Bytes, 2 = 128-255 Bytes, 3 = 256-511 Bytes, 4 = 512-1023 Bytes, 5 = 1024-1518 Bytes, 6 = >1518 Bytes received
AfdxPortNumber / AFDX Port Map ID	23..16	User defined port number which is associated with the physical port (PortMapID see FdxCmdRxPortInit).
Receiver Speed mode	15..14	0: 100 Mbit/sec 1: 10 Mbit/sec 2: 1GBit/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.
SequenceNumber	7..0	Sequence number of received frame

TY_GNET_FRAME_BUFFER_SIZE_WORD x_EntrySizeWord

Additional information about the buffer sizes, start and store mode

Value	Bit	Description
MonitorStartTriggerFlag	16	This bit indicates this frame has triggered the start of capturing.
ADSM	15..14	Applied Data Store Mode bits
TotalDataBufferSize	11..0	Size of the frame entry (Header + Data) in bytes including alignment padding.

TY_GNET_FRAME_HEADER_TYPE_WORD x_FrameHeaderTypeWord

Additional information about the captured frame

Value	Bit	Description
Frame receive header type	31..30	0: Default receive header type 3: APE-FDX /APX-GNET receive header type other values: Reserved
ADSM	15..14	Applied Data Store Mode bits
TotalDataBufferSize	11..0	Size of the frame entry (Header + Data) in bytes including alignment padding.

TY_GNET_FRAME_HEADER_TYPE_WORD x_FrameHeaderTypeWord

Additional information about the captured frame

Value	Bit	Description
Frame receive header type	31..30	0: Default receive header type 3: APE-FDX /APX-GNET receive header type other values: Reserved

AiUInt32 ul_NextPointer

Reserved

TY_GNET_FRAME_HEADER_INFO_WORD_0 x_FrameHeaderInfoWord0

MAC ID, Errors, VL ID

Value	Bit	Description
VirtualLinkID	15..0	Virtual Link
PhysicalSymbolError	16	PHY: Wrong physical symbol during frame reception.
StartFrameDelimiterError	17	PRE/SFD: Wrong Preamble/Start Frame Delimiter
Reserved	18	Unused
CrcError	19	CRC: MAC CRC error.
InterFrameGapError	20	IFG: short interframe gap error (<960ns)
IpFrameError	21	IPE: AFDX IP framing error (AFDX-IP frame specific settings violated).
MacFrameError	22	MAE: AFDX MAC framing error (AFDX-MAC frame specific settings violated).
ShortFrameError	23	SHR: short frame received (< 64 Bytes)
LongFrameError	24	LNG: long frame received (> 1518 Bytes)
FrameSizeError	25	VLS: VL specific frame size violation
SequenceNumberError	26	SNE: Sequence No. mismatch
Reserved	27	Unused
TrafficShapingError	28	TRS: Traffic Shaping Violation
MacID	31..29	MAC identifier field. 001=A, 010=B

TY_GNET_FRAME_HEADER_INFO_WORD_1 x_FrameHeaderInfoWord1

IFG information, frame bytes received

Value	Bit	Description
ByteCount	11..0	Byte count of the received AFDX / MAC – frame (incl. CRC Bytes).
InterFrameGapCounter	27..14	Counter of the interframe gap with a resolution of 8 ns, if the gap is lower than 131.072 µs. (Example: A 'Inter-frame Gap Count' value of 5 means a gap time of 5 * 8 ns = 40 ns)
InterFrameGapHigh	30	Indicates that the gap between two frames was greater than 131.072 µs

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;

Time tag word in firmware format. The higher part of the time tag contains the minutes of hour, hours of day and day of year.

AiUInt32 ul_TtLow;

Time tag word in firmware format. The lower part of the time tag contains the Microseconds of second, seconds of minutes and minutes of hour.

(See Section 3.5.3 "FdxFwIrig2StructIrig") to get a 'C' structured information of the Time Tag.

AFDX Frame

The frame receive header is directly followed by the actual AFDX frame:

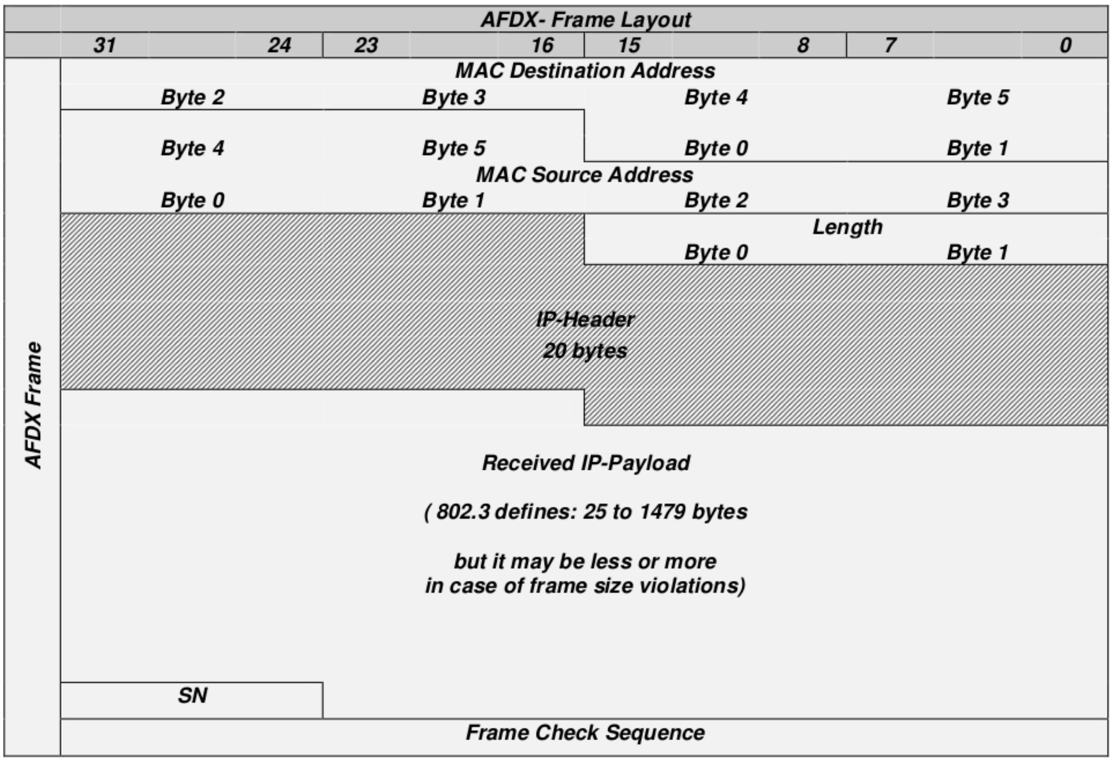


Figure 3.4: AFDX Frame Layout

Return Value:

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

3.4.3.5 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.

3.4.3.6 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.

3.4.3.7 FdxCmdMonQueueStatus

Prototype:

```
AiReturn FdxCmdMonQueueStatus(AiUInt32 ul_Handle,
                              AiUInt32 ul_QueueId,
                              TY_FDX_MON_QUEUE_ STATUS_OUT
                              *px_QueueStatusOut );
```

Purpose:

This function is used to get the status of a certain capture queue. It is useful for getting information about number of frames that have been received and are currently ready for processing.

Input:

AiUInt32 ul_QueueId

ID of capture queue to get status of. Queue identifiers are returned when creating a capture queue via either FdxCmdMonQueueControl or FdxCmdMonQueueContCapControl (when in FDX_MON_CONTINUOUS_SE capture mode) 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 capture queue.

Define	Description
FDX_MON_STAT_EMPTY	The queue is empty
FDX_MON_STAT_FILLED	There are frames to read
FDX_MON_STAT_FULL	Only valid in single shot capture mode. Capture queue is full and no more frames are captured.
FDX_MON_STAT_OVERFLOW	Capturing was stopped due to overload. Not all frames could be stored to the associated buffers. Capturing has to be restarted.
FDX_MON_STAT_ERROR	Internal capture queue error. Capturing has to be restarted

`AiUInt32 ul_ FramesToRead`

Returns number of frames actually in queue to read out.

`AiUInt32 ul_ BytesToRead`

Returns number of bytes needed to read all frames that are currently in queue. This value can be higher than the real number of frame data, because each frame buffer is internally aligned.

Return Value:

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

3.4.3.8 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 3.5 , 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.	

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 (Triple Nibble Error).	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
16-21	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.

Figure 3.5 shows the dependencies of the trigger engine and the related commands for setting up the corresponding items.

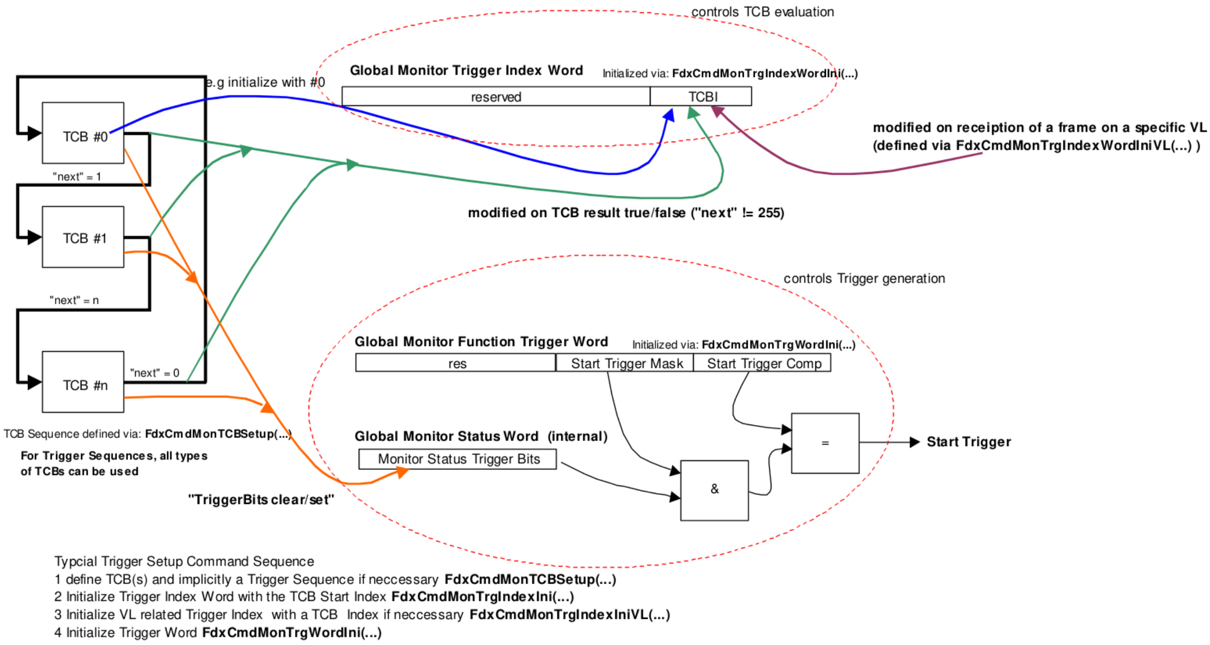


Figure 3.5: Trigger Engine Dependencies

3.4.3.9 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 Section 3.4.3.8 "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.

3.4.3.10 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 3.5 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.

3.4.3.11 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 3.5 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 Section 3.4.3.8 "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.

3.4.4 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	fdXTap	APU-FDX	ASC-FDX	APE/ACE/AXC/AMCX-FDX
Windows						•
Linux						•

3.4.4.1 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 G lobal 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.

3.4.4.2 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	fter 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.

3.4.4.3 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.

3.4.4.4 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.

3.5 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 3.5: 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.

3.5.1 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 Section 3.5.3 "FdxFwIrig2StructIrig") function above.

TY_FDX_IRIG_TIME *px_IrigTimeB

Format (See Section 3.5.3 "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

3.5.2 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.

3.5.3 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 year1 to 365	Hours of day 0 to 23	Minutes of hour 0 to 59

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.

3.5.4 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 Section 3.3.2.4 "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.

3.5.5 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.

3.5.6 FdxStructIrig2FwIrig

Prototype:

```
oid 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 Section 3.5.3 "FdxFwIrig2StructIrig") function above.

Return Value:

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

3.5.7 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 (Triple Nibble Error).	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.

3.5.8 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 Section 3.5.7 "FdxTranslateErrorWord")

3.5.9 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:**
> 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

he 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_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.

3.5.10 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.

3.6 Reros Functions

3.6.1 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 Section 3.3.1.2 "FdxCmdTxModeControl") and (See Section 3.4.1.3 "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.

3.6.2 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 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!

Note:
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 */
    rpfBCD,                 /* 1...64 */
    rpfScaledSigned,        /* 1...64 */
    rpfScaledUnsigned,      /* 1...64 */
    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 Section 3.6.5 "FdxCmdRerosParamControlInteractive") or automatically by the board in accordance with the corresponding setup ((see (See Section 3.6.4 "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.

3.6.3 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 Section 3.6.2 "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
                                  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 Section 3.6.2 "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 Section 3.6.4 "FdxCmdRerosParamControlAutomatic") and (See Section 3.6.2 "FdxCmdRerosParamCreate")).
rtsIsWaitingForStartTrigger	Waiting for Start Trigger for an Automatic modification, (See Section 3.6.4 "FdxCmdRerosParamControlAutomatic")
rtsIsWaitingForStopTrigger	Waiting for Stop Trigger for an Automatic modification, (See Section 3.6.4 "FdxCmdRerosParamControlAutomatic")

TY_FDX_E_REROS_MODIFICATION_STATE e_ModificationState

(See Section 3.6.4 "FdxCmdRerosParamControlAutomatic") and functions for the setup of the modification functions.

```
typedef enum rerosParamModificationState
{
    rpmsIsNotModified,
    rpmsIsLimitingToMin,
    rpmsIsLimitingToMax,
    rpmsIsModifiedByDynamicFunction,
    rpmsIsModifiedInteractive
} TY_FDX_E_REROS_MODIFICATION_STATE;
```

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 Section 3.3.2.4 "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 Section 3.3.2.4 "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.

3.6.4 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 Section 3.6.2 "FdxCmdRerosParamCreate").

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 Section 3.6.2 "FdxCmdRerosParamCreate") 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 $\neq$ ‚d_Value’
rtstartmInternalGreaterThan	associated parameter $>$ ‚d_Value“
rtstartmInternalLessThan	associated parameter $<$ ‚d_Value“
rtstartmInternalInRange	‚d_MinVal’ $\leq$ associated parameter $\leq$ ‚d_MaxVal’
rtstartmInternalOutOfRange	‚d_MinVal’ $>$ associated parameter $>$ ‚d_MaxVal’
rtstartmExternal	Start Trigger is issued by external Trigger (on HWTrigger 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

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 Section 3.3.2.4 "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
Section 3.3.2.4 "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.

3.6.5 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 Section 3.6.2 "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`

```
/* 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.

4 Notes

4.1 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 WaitTime	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.

List of Abbreviations

usec	microseconds
AFDX	Avionic Full Duplex Switched Ethernet
API	Application Programming Interface
ARINC	Aeronautical Radio, Incorporated
ASCII	American Standard Code for Information Exchange
ASP	Application Support Processor
BAG	Bandwidth Allocation Gap
BITE	Built IN Test
BIU	Bus Interface Unit
BSP	Board Support Package
COM Port	Communication Port, an UDP Sampling- or Queuing Port ()COMM Port
CRC	Cyclic Redundancy Check
DST	Destination
Frame	A single Ethernet-packet, has normally an Ethernet- and IP-Header
FIFO	First in - First out
GTM	Generic Transmit Mode
GTU	Gap Time Unit
IC	Integrity Checking
ICANN	Internet Corporation for Assigned Names and Numbers
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
Message	UDP-Payload without any protocol headers, valid range 1 to 8192 bytes
MSB	Most Significant Byte
ns	nanoseconds
OSI	Open System Interconnect
Packet	A single Ethernet-packet, valid range 64 to 1518 bytes
PCI	Peripheral Component Interconnect
PGWT	Packet group wait time
PMC	PCI Mezzanine Card
RAM	Random Access Memory
RM	Redundancy Management
RMA	Redundancy Management Algorithm
RP(M)	Replay Mode
Rx	Receiver
SAP	Service Access Point
SFD	Start Frame Delimiter
SN	Sequence Number
SRC	Source
STM	Simulator Transmit Mode
TAP	Test Access Point

TBD	To be defined
TCB	Monitor Trigger Control Block
TFTP	Trivial File Transfer Protocol
TM	Time Manager
Tx	Transmission
UDP	User Datagram Protocol
VL	Virtual Link

LIST OF FIGURES

LIST OF TABLES