Fax Software Reference

for Linux and Windows

05-2341-001

Publication Date: July, 2004Intel Converged Communications, Inc.
1515 Route 10
Parsippany NJ 07054

For **Technical Support**, visit the Intel Telecom Support Resources website:
http://developer.intel.com/design/telecom/support/

For **Products and Services Information**, visit the Intel Telecom Products website:
http://www.intel.com/design/network/products/telecom/

For **Sales Offices** and other contact information, visit the Where to Buy Intel Telecom Products page:
http://www.intel.com/buy/wtb/wtb1028.htm

THIRD-PARTY COPYRIGHT NOTICE

The Fax API Library for Linux makes use of the gd graphics library from Boutell.Com Inc. The following copyright statements apply to the gd library:

Portions copyright 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004 by Cold Spring Harbor Laboratory. Funded under Grant P41-RR02188 by the National Institutes of Health.

Portions copyright 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004 by Boutell.Com, Inc.

Portions relating to GD2 format copyright 1999, 2000, 2001, 2002, 2003, 2004 Philip Warner.

Portions relating to PNG copyright 1999, 2000, 2001, 2002, 2003, 2004 Greg Roelofs.

Portions relating to gdttf.c copyright 1999, 2000, 2001, 2002, 2003, 2004 John Ellson (ellson@graphviz.org).

Portions relating to gdft.c copyright 2001, 2002, 2003, 2004 John Ellson (ellson@graphviz.org).

Portions relating to JPEG and to color quantization copyright 2000, 2001, 2002, 2003, 2004, Doug Becker and copyright (C) 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004 Thomas G. Lane. This software is based in part on the work of the Independent JPEG Group. See the file README-JPEG.TXT for more information.

Portions relating to WBMP copyright 2000, 2001, 2002, 2003, 2004 Maurice Szmurlo and Johan Van den Brande.

Permission has been granted to copy, distribute and modify gd in any context without fee, including a commercial application, provided that this notice is present in user-accessible supporting documentation.

This does not affect your ownership of the derived work itself, and the intent is to assure proper credit for the authors of gd, not to interfere with your productive use of gd. If you have questions, ask. “Derived works” includes all programs that utilize the library. Credit must be given in user-accessible documentation.

This software is provided “AS IS”. The copyright holders disclaim all warranties, either express or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose, with respect to this code and accompanying documentation.

Although their code does not appear in gd 2.0.4, the authors wish to thank David Koblas, David Rowley, and Hutchison Avenue Software Corporation for their prior contributions.

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1. Purpose and Audience

This guide is written for the application developer who wants to create fax applications using the fax API library. The guide provides a complete reference to the fax API library functions, parameters, data structures, and error codes supported in the Linux* and Windows* environment, on DM3 boards and on Springware boards.

1.2. Using this Guide

This guide assumes that you are familiar with the Linux or Windows operating system and the C programming language.

1.3. Related Publications

1.3.1. Intel Publications

See the following documents for more information:

- *Voice API Programming Guide*
- *Voice API Library Reference*
- *Standard Runtime Library API Programming Guide*
- *Standard Runtime Library API Library Reference*
- *Global Call Voice API Programming Guide*
- *Global Call API Library Reference*
- *Digital Network Interface Software Reference*

In addition, see the *Release Guide* and *Release Update* that accompany a specific system release for system requirements, product support, feature support, known issues, and last-minute updates. The latest release-specific information is also available at the Technical Support website at *http://developer.intel.com/design/telecom/support/*.

1.3.2. Other Publications

The following documents are valuable references in understanding fax technology and ITU-T fax recommendations.

- Bodson, Dennis, McConnell, Kenneth R. and Schaphorst, Richard. *FAX: Digital Facsimile Technology and Applications, Second Edition*, Norwood, Massachusetts: Artech House, Inc., 1992.
- ITU-T (formerly CCITT). *Procedures for Document Facsimile Transmission in the General Switched Telephone Network, Recommendation T.30*, CCITT Blue Book, Volume VII - Fascicle VII.3, T.4 and T.6, Melbourne, Australia, November 1988.
  (Blue Book ordering information can be obtained by contacting the United Nations Bookshop, General Assembly Bldg., Room: G.A. 32 B, New York, NY 11017)

1.4. Documentation Conventions

The following documentation conventions are used in this manual:

Format	Examples	Description
boldface followed by parentheses	**dx_play( )**	API function call
boldface	**chdev**	parameter, field
italics	*\dlgc\samples* *asrdemo.exe* *Chapter 1*	• directory name • filename • chapter name, manual title
courier font	`#include <srllib.h>`	sample code

1.5. Contacting Intel

For technical support, visit the Intel website at *http://developer.intel.com/design/telecom/support/*.

1.6. What's in this Guide

This guide is organized into the following chapters and appendices:

Chapter 1 provides an introduction to this guide, its purpose and audience, documentation conventions, related publications and contact information.

Chapter 2 is an overview of the fax API. Included is information on product support, key product features, the fax library, sample demonstration programs, voice/fax integration, system configuration models and the Telephone Consumer Protection Act of 1991.

Chapter 3 describes fax API programming considerations for boards based on DM3 architecture (DM3 boards).

Chapter 4 presents the basics of fax communication. It covers fax terminology, the structure of a fax call (ITU-T T.30 protocol), fax data formats, data encoding schemes, types of fax transmission and more.

Chapter 5 discusses the implementation of fax send capability in your application. It presents guidelines and direction on using the fax API and other function calls to send a fax.

Chapter 6 covers the implementation of fax reception capability in your application. It presents guidelines and direction on using the fax API and other function calls to receive a fax.

Chapter 7 discusses the fonts supported in ASCII to fax conversion.

Chapter 8 discusses the fax demo programs for Linux.

Chapter 9 discusses the fax demo programs for Windows.

Chapter 10 describes the data structures used with fax library functions.

Chapter 11 is an overview of the fax API library. It describes the categories of fax functions, error handling and required include and library files.

Chapter 12 provides a complete alphabetical reference to the fax library functions and specifies the platform (DM3, Springware) supported by each function.

Appendices provide a reference for TIFF/F Tags and Values, Fax Phase D Status Values, Fax Phase E Status Values, Fax Error Codes, Fax Event Codes, ASCII to Fax Tables and acronyms used.

A *Glossary* and an *Index* are provided at the end of this guide for reference.

2. Basics of Fax Software

2.1. Introduction to Fax Software

This chapter introduces you to the basics of fax software. It lists the Intel® products that support the fax software, describes key product features and provides a general overview of the fax software and system configuration models.

The fax software provides a fax library fully integrated with the voice library. This integration enables you to build fax applications or add fax capability to your existing voice applications and create unified messaging systems.

The fax software is supported on specific boards and runs on Linux* and Windows* operating systems. It consists of fax library and header files, device drivers, sample demonstration programs, and a documentation set to help you develop voice/fax applications. The voice and fax libraries provide C-language interface.

The term R4 API ("System Software Release 4 Application Programming Interface") describes the direct interface used for creating computer telephony application programs. The R4 API is a rich set of proprietary APIs for building computer telephony applications tailored to hardware products from Intel. These APIs encompass technologies that include voice, network interface, fax, and speech. This document describes the fax API. The R4 API supports boards based on DM3 architecture and Springware (earlier-generation) architecture.

2.2. Product Terminology

The following terminology is used to describe Intel® products that support the fax API.

Product	Description
VFX	Integrated four-channel voice/fax boards based on Springware architecture. Each channel can process voice or fax. Through a time division multiplexing (TDM) bus interface, fax resources can be shared by other voice processing resources.
VFX/41JCT-LS	Four-port, voice/fax board with on-board analog telephone interfaces (loop start) and CT Bus connector. It is SCbus compatible with bus adapter. This board replaces older VFX boards such as the VFX/40ESCplus and VFX/PCI. The VFX/41JCT-LS board uses enhanced DSP Fax.

Product	Description
DSP-Based Group 3 Fax (DSP Fax)	Also known as **Softfax**. Multiple-port, software-based fax solution. This software allows you to use the fax API to develop fax applications on non-VFX Springware boards. Springware boards refer to boards based on earlier-generation architecture. Typically Springware fax boards have the prefix "VFX". Springware voice boards typically are prefaced with "D," such as the Intel® Dialogic® D/240JCT-T1. Some of these voice boards also support fax. Unlike the VFX boards, fax modem capability is implemented in software rather than hardware. In addition, ASCII to fax capability is provided by the fax library rather than by the DSP. However, fax imaging capability is still provided by the digital signal processor (DSP) on the board itself. There are two types of DSP Fax: basic and enhanced. See *Table 1. Key Fax Features and Specifications* for a list of features. Support for enhanced DSP Fax is available on the VFX/41JCT-LS board only. See the latest *Release Guide* for the most current list of products that support basic and enhanced DSP Fax. On some products, DSP Fax resources are in a 1:1 ratio with voice resources. However, an application may need only a limited number of DSP Fax resources in relation to voice channels. This configuration is provided by products, such as the D/82JCT-U, that offers DSP Fax resources which may be shared among the voice channels. The DSP Fax resource is automatically assigned to a channel through the **fx_open( )** function and de-assigned (or made available) with the **fx_close( )** function. See the *Voice API Library Reference* for more information on resource sharing, such as the **dx_GetRscStatus( )** function, which allows you to check whether a resource is assigned to a channel. Check the *Release Guide* and *Release Update* for any hardware-specific limitations on resource sharing and for information on the boards and channels that support DSP Fax.

Product	Description
DM3 Fax	Refers to boards based on DM3 architecture that support the Fax API. Examples are the DM/F, DM/VF (previously called VFN), and DM/V series boards. See the *Release Guide* for information on hardware support. DM3 boards is a collective name used in this document to refer to products that are based on the Intel® DM3 mediastream architecture. Typically DM3 board names have the prefix "DM," such as Intel® Dialogic® DMV160LP and Intel® NetStructure™ DM/V2400A-PCI.

For the most up-to-date list of products that support fax software, see the latest *Release Guide* and *Release Update* that accompany the software.

2.3. Key Product Features

This section describes key features of Intel® fax products in table format. *Table 1. Key Fax Features and Specifications* summarizes key fax features and specifications by product.

Intel® fax products are compatible with ITU-T Group 3 (T.4, T.30) and ETSI NET/30.

Complete technical specifications for fax products can be found on the Intel Telecom Products website at *http://www.intel.com/design/network/products/telecom/index.htm*. See the Glossary for definitions of unfamiliar terms. See the latest *Release Guide* for the system release you are using for information on hardware support.

The fax products described here do not support PCX and DCX file formats for sending and receiving faxes.

Table 1. Key Fax Features and Specifications

Fax Features and Specifications	DM3 Fax	Basic DSP Fax	Enhanced DSP Fax
Data rate			
Transmit 9,600 bits per second (bps)	•	•	•
Transmit 14,400 bps	•	•	•
Receive 9,600 bps	•	•	•
Receive 14,400 bps	•		•
Variable speed selection	•	•	•
Automatic step-down to lower speed if necessary	•	•	•
File storage format			
Raw MH encoded fax data from file or memory	•	•	•
Raw MR encoded fax data from file or memory	•		
Raw MMR encoded fax data from file or memory	•	•	•
TIFF/F (Tagged Image File Format, Class F) MH encoded	•	•	•
TIFF/F (Tagged Image File Format, Class F) MR encoded	•		
TIFF/F (Tagged Image File Format, Class F) MMR encoded	•	•	•
ASCII for transmit only	•	•	•
Japanese Katakana text for transmit only		•	•
Selectable storage of multi-page fax in a single TIFF/F, multiple TIFF/F or multiple raw files	•	•	•

Table 1. Key Fax Features and Specifications (cont.)

Fax Features and Specifications	DM3 Fax	Basic DSP Fax	Enhanced DSP Fax
Data transmission encoding scheme (over the phone line)			
MH (Modified Huffman)	•	•	•
MR (Modified Read)	•	•	•
MMR (Modified Modified Read)	•		•
Selectable data transmission encoding scheme	•	•	•
Selection of single, specified image resolution for all fax data during fax transmission	•	•	•
Data reception encoding scheme (over the phone line)			
MH (Modified Huffman)	•	•	•
MR (Modified Read)	•	•	•
MMR (Modified Modified Read)	•		•
Selectable data reception encoding scheme	•	•	•
ASCII to Fax conversion			
On the fly conversion—direct transmission of text files	•	•	•
Supports multiple fonts and language character sets		•	•
Supports entire selection of Windows fonts (Windows only)		•	•

Table 1. Key Fax Features and Specifications (cont.)

Fax Features and Specifications	DM3 Fax	Basic DSP Fax	Enhanced DSP Fax
ASCII to Fax conversion			
Supports use of tilde (~) and <ESC> in formatting ASCII documents; supports italicized text.		•	•
Supports embedded formatting commands	•	•	•
Creates page headers automatically	•	•	•
User-definable page header option	•	•	•
Merges ASCII and raw image on same page	•	• [1]	•
Error Correction			
Detects, reports and replaces faulty scan lines	•	•	•
Supports T.30 Error Correction Mode (ECM)	•		•
Image widths and scaling			
Supports multiple image widths: 215 mm (8.5 in), 255 mm (10 in) and 303 mm (11.9 in)	• [2]	•	•
Selectable maximum receive width and preferred receive length	•	•	•
Automatic horizontal and vertical scaling between page sizes	•	•	•

[1] For DSP Fax, no automatic page break occurs when page size exceeded.

[2] On DM/F and DM/VF boards, all image widths are supported. On DM/V boards, only 215 mm width is supported.

Table 1. Key Fax Features and Specifications (cont.)

Fax Features and Specifications	DM3 Fax	Basic DSP Fax	Enhanced DSP Fax
Fill bit processing			
Automatic fill bit insertion on transmit	•	•	•
Automatic fill bit removal on receive	•	•	•
Polling modes			
Normal and turnaround	•	•	•
Mixed coded images on one fax page			
Allows images from multiple sources (text and graphics) on one fax page (sub-pages)	•	•	•
Allows pages and sub-pages of different encoding schemes, resolutions and widths[3]	•	•	•
T.30 subaddress messaging			
Supports T.30 subaddress message protocol, which allows a fax to be routed to one or more telephone numbers once received		•	•

[3] If the widths of the consecutive sub-pages are different, the sub-pages are scaled to match the negotiated width. For **DSP Fax**, the resolution of each sub-page to be sent must be of the same resolution or you must explicitly set the resolution for the entire fax transfer.

Table 1. Key Fax Features and Specifications (cont.)

Fax Features and Specifications	DM3 Fax	Basic DSP Fax	Enhanced DSP Fax
System configuration model			
Supports stand-alone model	•	•	•
Supports TDM bus configuration	•	•	•
International fax support			
Supports German computer-based fax—automatically inserts two lines in the fax header	•	•	•
Supports Japanese Katakana character set for fax page and header		•	•
Fax Header			
Automatically creates one-line fax page header on every transmitted page	•	•	•
User-definable fax page header text option	•	•	•
Other			
Color fax (JPEG and JBIG format)	•		
Operator intervention (issue and accept voice request during data transmission and reception)		•	•
Reporting completion of T.30 Phase B and Phase D	•	•	•
Non-standard facilities (NSF), digital command signal (DCS), and digital information signal (DIS) information returned to the application, allowing proprietary communications above T.30	•	•	•

2.4. Fax API/Library Overview

The C-language application programming interface (API) included with the fax software provides a library of functions used to create fax applications. Fax data structures are also a part of the fax library. These fax functions interface with the voice driver and are tightly integrated with the voice library. This architecture enables you to add fax capability to an existing voice application. Your application can play or record voice files on one channel while another channel receives or sends fax calls; or the same channel can alternately process fax and voice calls.

The fax library, together with the voice libraries, provide the interface to the voice driver. The fax library contains all fax-specific functions. The voice libraries include the main voice library and the Standard Runtime Library. For more information on voice libraries, see the *Voice API Library Reference* and the *Standard Runtime Library API Library Reference*.

By convention, fax-specific functions begin with **fx_**, such as **fx_sendfax( )** and **fx_setparm( )**. Voice-specific functions typically begin with **dx_**, such as **dx_play( )** and **dx_getdigit( )**. Functions that are part of the Standard Runtime Library begin with **sr_**.

For more information on fax functions, see *Chapter 3. Fax API for DM3*, *Chapter 11. Using the Fax Library*, and *Chapter 12. Fax Library Function Reference*.

The fax library uses several fax data structures. These structures are described in detail in *Chapter 10. Fax Data Structures*.

2.5. Voice and Fax Integration

To develop fax and integrated voice/fax applications, you need to use fax library functions in conjunction with voice library functions and the Standard Runtime Library functions. For example, specific voice library functions control the hook state of your phone line, receiving and processing touch-tone digits, and so on, while the fax library controls such functions as sending and receiving faxes, and setting fax parameters.

On Springware boards and on DM/VF series boards (previously called VFN), a fax resource channel uses the same physical hardware channel as a voice resource channel, so a single channel can process either a voice call or a fax call; however, both voice and fax input/output cannot occur at the same time on the same device channel.

For DSP Fax, DM/V series boards, and DM/F series boards, a fax resource channel typically handles fax only and has no voice capability. On DM3 boards, fax and voice capability is determined by the media load in use. For information about media loads, see the Configuration Guide for DM3 products.

See the function reference examples in this guide to see how voice library functions are used with fax library functions. Also see Section *5.3. Opening and Closing a Fax Channel Device* for information on fax channel device operation. Information on discovering devices on DM3 boards is described in *Section 3.2. Device Discovery*.

2.6. Modes of Operation

The **fx_sendfax( )**, **fx_rcvfax( )**, **fx_rcvfax2( )**, and **fx_originate( )** functions have a mode argument that specifies the mode of operation for the function – synchronous or asynchronous. **All other fax functions operate synchronously**.

This section provides an overview of synchronous and asynchronous modes of operation.

NOTE: In this guide, the terms synchronous and asynchronous indicate only the function's mode of operation.

2.6.1. Synchronous Mode

Synchronous mode operation allows you to assign distinct applications to different channels in a system by simultaneously loading separate applications, each dedicated to a single channel.

In synchronous mode, a fax *send* or *receive* function returns control to the application only after the function has completed processing or an error has occurred. For example, a fax *send* function must transmit all the fax data (or return a processing error) before the application can issue another function call on the channel device.

For a full discussion of synchronous programming models, see the *Standard Runtime Library API Programming Guide* .

2.6.2. Asynchronous Mode

Asynchronous mode operation enables a single program to control multiple channels. Multiple tasks can be coordinated via the same process, controlling, for example, the timing and sequence of each task in a single call session.

In asynchronous mode, a *send* or *receive* fax function returns control to the application immediately after successful invocation. The application can then issue other functions while the first asynchronous function continues processing (for example, sending or receiving fax data). This allows the application to open several channels and issue functions on each channel from a single process. The application then monitors events from the channels it is controlling and maintains a state machine for every channel. Based on the event received from a channel, the application issues the next appropriate function for that channel.

The Standard Runtime Library generates events to indicate whether a function completed successfully or failed. See *Section 11.3. Error Handling* on *page 148* for more information on errors and *Appendix E* on *page 381* for a list of event codes.

NOTE: The application must keep track of all functions it is processing and monitor events generated by the Standard Runtime Library.

For a full discussion of asynchronous programming models, see the *Standard Runtime Library API Programming Guide.*

2.7. System Configuration Models

The fax software supports the following system configuration models on specific Intel® products:

- **Stand-alone** configuration. Used for system configurations that do not require TDM bus channel routing; for example, using on-board RJ-11 analog jacks to connect to the telephone network. Supported on VFX products.
- **TDM Bus** configuration. Used for system configurations that route channels to CT Bus time slots. Supported on the VFX/41JCT-LS, DSP-Based Group 3 Fax (also known as DSP Fax) and other fax products that have a TDM bus connection.

2.7.1. Stand-alone Model

In a stand-alone configuration model, the voice and fax resource channels are connected to the on-board network interface. There is no time slot routing.

This model is not supported on DM3 boards.

The program flow for the stand-alone model is as follows:

Step	Action
1	Open channels: • Open channel for voice using the voice API • Open channel for fax using the fax API
2	Call voice functions to set up the call.
3	Initiate fax send and receive on the channel: • Initialize the fax channel to send/receive faxes • Send/receive faxes
4	Continue with the application.
5	At the end of the application, close the open channels: • Close channel for voice using the voice API • Close channel for fax using the fax API

2.7.2. TDM Bus Model

In a TDM bus configuration model, you set up your application resource channels (network, voice, fax, and so on) to communicate with one another on the TDM bus by transmitting data on an assigned TDM bus transmit time slot (automatically assigned during download) and by listening to data transmitted on another resource's TDM bus transmit time slot.

Step	Action
1	Open channels: • Open channel for voice • Open channel for fax • Open network time slot (digital only)
2	Set TDM bus routing: fax/voice and network channel set for full duplex
3	Set up the call using call functions: • analog • digital
4	Before sending or receiving a fax on the channel: • Disconnect TDM bus routing: voice and network channel (full duplex) • Set TDM bus routing: fax resource and network channel (full duplex)
5	Initiate fax send and receive on the channel: • Initialize the fax channel to send/receive faxes • Send/receive faxes
6	After completing the fax send and receive on the channel, disconnect TDM bus routing: fax resource and network channel (full duplex)

Step	Action
7	To complete the call, additional voice processing may be required. Connect original voice channel to the TDM bus time slot: set TDM bus routing voice and network channel for full duplex.
8	Continue with the application currently set for voice API
9	At the end of the application, close the open channels: • Close the channel for voice • Close the channel for fax • Close the network time slot (digital only)

In this TDM bus model to include fax resources, note the following:

- TDM bus routing for the voice resource is set to communicate with the network resource in full duplex; that is, two-way communication, send and receive, between the two resources over the TDM bus.
- Before a fax send and receive is initiated (Step 5), the voice resource is disconnected from the TDM bus (Step 4). In this model, the same network channel (or time slot) is used for both voice and fax resources.
- Before a new resource is introduced, TDM bus time slot assignment is disconnected from the current resource and set for the new resource.

2.8. Complying with the Telephone Consumer Protection Act

The Telephone Consumer Protection Act of 1991 makes it unlawful for any person to use a computer or other electronic device, including fax machines, to send any message unless this message clearly contains, in a margin at the top or bottom of each transmitted page or on the first page of the transmission, the following information:

- date and time the message is sent

- an identification of the business, other entity, or individual sending the message
- the telephone number of the sending machine or such business, other entity, or individual. The telephone number provided may not be a 900 number or any other number for which charges exceed local or long-distance transmission charges.

To program this information into your fax application, complete the following steps:

- Use the **fx_setparm( )** function to set the company/sender in the FC_HDRUSER parameter and the telephone number in the FC_LOCALID parameter.
- By default, the fax sending functions will send a header at the top of every page containing the date and time as well as the parameters set through the **fx_setparm( )** function.

This function is discussed in detail in *Chapter 12. Fax Library Function Reference*.

3. Fax API for DM3

3.1. Overview of Fax API for DM3

The R4 Fax API supports a new generation of Intel® hardware products that are based on the DM3 mediastream architecture, in addition to original Springware products (also known as earlier-generation products).

Information on DM3 support is provided throughout this document as appropriate. This chapter provides programming considerations specific to DM3 boards and describes the color fax functionality.

The term “R4 for DM3” or “R4 on DM3” is used to refer to specific aspects of the R4 API interface that relate to support for DM3 boards.

Not all functions and parameters are supported on both DM3 boards and Springware boards. Information on these restrictions is noted in the function descriptions.

Similarly, not all data structures are supported and used on both DM3 boards and Springware boards. Information on these restrictions is noted in the data structure descriptions.

3.2. Device Discovery

To determine whether a DM3 board supports fax, follow this procedure:

1. Use Standard Runtime Library device mapper functions to return information about the structure of the system, such as a list of all the physical boards in a system, a list of all virtual boards on a physical board, and a list of all subdevices on a virtual board. For more information on these functions, see the *Standard Runtime Library API Library Reference*. The device mapper functions include **SRLGetAllPhysicalBoards( )**, **SRLGetVirtualBoardsOnPhysicalBoard( )**, and more.

2. Use **dx_open( )** to open the board device (virtual board device, not physical board device) and retrieve the device handle. The board device is in the format dxxxB*n*, such as dxxxB1.

   NOTE: Device enumeration on DM3 boards differs depending on the media load in use. For example, a DM/V600A-2E1 board that uses media load 5 (enhanced voice and fax) has more than 15 virtual boards. For more information on media loads, see the Configuration Guide for DM3 products.

3. To determine whether the board device supports fax before you attempt to open the fax device, use **dx_getfeaturelist( )** and specify the board device handle obtained in step 2. The FEATURE_TABLE structure contains the features of the device.

4. If this is a fax only channel, close this voice device handle using **dx_close( )**, as the handle will no longer be used.

5. Use **fx_open( )** on the voice channel device to open the associated fax device and retrieve the fax handle. The function will succeed if the channel device has fax capabilities; otherwise the function will fail.

6. If desired, call **fx_getctinfo( )** to find out more about the fax device, such as product ID, device family, and network interface. The CT_DEVINFO structure contains this fax device information.

3.3. Programming Considerations

The following programming considerations are provided to help you develop applications on DM3 boards:

- Use the TDM bus routing device information function, **fx_getctinfo( )**, to obtain information about DM3 devices, which is returned in a CT_DEVINFO data structure. This information can be used to identify whether a logical device belongs to DM3 hardware. For details on this structure, see the *Voice API Library Reference*.

- In flexible routing configurations, you must issue **fx_listen( )** prior to calling **fx_sendfax( )**, **fx_rcvfax( )** or **fx_originate( )**. Otherwise, these functions will return an error.

- R4 on DM3 does not support the use of a voice handle for fax commands; that is, you cannot use the device handle from **dx_open( )** to call fax API functions. You must use **fx_open( )** to open a channel device for fax processing and use that fax device handle.
- R4 on DM3 supports the use of **fx_open( )** on a board device as well as a channel device. To determine the number of subdevices (or channels) available, use **fx_open( )** on the board device followed by **ATDV_SUBDEVS( )**. If **fx_open( )** returns -1, then the subdevice does not support fax. For more information on **ATDV_SUBDEVS( )**, see the *Standard Runtime Library API Library Reference*. For more on device discovery, see *Section 3.2. Device Discovery*.
- Applications that create multiple handles for a single fax device should set the parameters consistently on all the handles that perform fax operations, for example, on **fx_sendfax( )** and **fx_rcvfax( )**.
- The only font that is supported on DM3 fax boards is the normal font. The compressed font is not supported on DM3 fax boards. Because of this, on DM3 boards the default font for the fax header is different from VFX boards. On DM3 boards, the normal font (ID 0) is used in the fax header, while on Springware boards, the compressed font (ID 3) is the default. Because of these font restrictions on DM3 fax boards (and because of the fixed left and right margins on DM3 fax boards as noted in Section *10.3.3. DF_ASCIIDATA Usage Rules*), DM3 fax boards provide fewer characters per line in the fax header. This means that the header may wrap to a second line.
- On DM3 boards, regardless of the page length you specify, the converted fax image has no maximum size (unlimited length). No pagination is performed by the firmware. Font is fixed at 10 lines per inch (each line is approximately 1/10 inch in height); prints approximately 12 characters per inch; 16 scan lines of MH data; 16 (horizontal) by 16 (vertical) pixels or 80 characters maximum per line. Top Margin is set to 3, Left Margin to 14 and Right Margin to 94.

3.4. Color Fax

Color fax functionality supports the sending and receiving of JPEG/JBIG files to and from Group 3 color fax devices.

3.4.1. Color Fax Features

Features of color fax include the following:

- Fax API Library support
- Transmission and reception of JPEG encoded color facsimile images to and from color fax devices
- Transmission and reception of JBIG grayscale facsimile images to and from fax devices
- Encoding of color fax images using the JPEG format as specified in ITU Rec. T.81 and T.42 standards and the ITU Rec. T.4 Annex E standard (ITU Rec. T.4 Annex E defines the specific JPEG profile for color fax)
- Encoding of fax images using the JBIG format as specified in ITU Rec. T.82 and T.43 standards and the ITU Rec. T.85 standard (ITU Rec. T.85 defines a specific profile for bilevel JBIG encoded fax images.)

The following baseline JPEG options are supported (as defined in ITU Rec. T.4 Annex E):

- Baseline DCT with Huffman entropy coding
- CIElab color space (L=Luminance [also used for grayscale], A=green/red hue, and B=blue/yellow hue)
- 8 bits/pel/component
- 4:1:1 sub-sampling
- One scan per image file
- Default CIE illuminant D.50
- Default gamut for LAB
- G3FAX APP1 marker: Version=1994 and resolution=200 dpi

3.4.2. Using the Fax API Library for Color Fax

This section includes information about the Fax API library used for color fax.

- Two new keywords have been added: DF_JPEG_GREY (for JBIG formatted files) and DF_JPEG_COLOR (for JPEG formatted files). They are intended to be used with a DF_IOTT structure (iott.io_coding field) that has io_datatype = DF_RAW.
- To enable the JPEG mode for sending or receiving, set FC_TXCODING in **dx_setparm( )** to DF_JPEG_COLOR (this implies automatic DF_MMR and DF_ECM).
- To enable the JBIG mode for sending or receiving, set FC_TXCODING in **dx_setparm( )** to DF_JPEG_GREY (this implies automatic DF_MMR and DF_ECM).
- To receive in JPEG, the application must receive the fax in raw format. Also, FC_TXCODING must be set to DF_JPEG_COLOR.
- To receive in JBIG, the application must receive the fax in raw format. Also, FC_TXCODING must be set to DF_JPEG_GREY and DF_JPEG_COLOR.

The R4 Fax API allows you to control many aspects of the T.30 protocol. The only commands you have to configure are the line settings:

- FC_TXCODING
- FC_TXBAUDRATE
- FC_RXBAUDRATE

The basic approach is to extend FC_TXCODING:

```
#define DF_MH           0     // 1-D Group 3 Modified Huffmann encoding
#define DF_MR           1     // 2-D Group 3, T.4 Modified Read encoding
#define DF_MMR          2     // T.6 Modified Modified Read encoding
#define DF_JPEG_GREY    3     // set ECM and T6 + JPEG    (<--- ignore DF_ECM value)
#define DF_JPEG_COLOR   4     // set ECM and T6 + JPEG Full Color
#define DF_ECM          0x8000     // OR with FC_TXCODING value to use ECM
```

This is valid for sending and receiving.

Sending a JPEG/JBIG Fax

To send a JPEG or JBIG fax, follow these instructions:

1. Set **fx_setparm( )** FC_TXCODING = DF_JPEG_GREY (for JBIG) or DF_JPEG_COLOR (for JPEG).
2. Fill the DF_IOTT structure:

```
Iott->io_type      = IO_DEV | IO_EOT;
Iott->io_fhandle   = dx_fileopen("d:\\F21_200.jpg", _O_RDONLY | _O_BINARY , 0);
Iott->io_bufferp   = NULL;
Iott->io_offset    = 0;
Iott->io_length    = -1;
Iott->io_nextp     = (DF_IOTT *) NULL;
Iott->io_prevp     = (DF_IOTT *) NULL;
Iott->io_width     = DF_WID1728;
Iott->io_resln     = DF_RESHI;
Iott->io_coding    = DF_JPEG_COLOR  (for JPEG: DF_JPEG_GREY or DF_JPEG_COLOR)
Iott->io_phdcont   = DFC_AUTO;
Iott->io_datatype  = DF_RAW #(mandatory for a JPEG)
```

NOTE: A file can also be sent from memory (IO_MEM instead of IO_DEV)

Sending a JPEG-only File

To send a JPEG-only file:

```
FC_TXCODING set to DF_JPEG_COLOR
     JPEG Color
               (MPS)
     JPEG Color
               (MPS)
     JPEG Color
               (EOP)
```

Additional information to note when sending a JPEG-only file:

- If the DF_IOTT contains a JPEG file, and the FC_TXCODING is not correct (for example, JPEG Color file and FC_TXCODING is JPEG_GREY), an error is reported.
- If DF_IOTT doesn't contain a JPEG entry, turn off JPEG, even if TX_CODING expects JPEG.
- When there is a JPEG file to send, headers are turned off for ALL the pages (also MH/MR/MMR/ASCII file).

- If you are sending a JPEG file, check that the DIS of the receiving side supports JPEG
- If the user forces an MPS as Phase D command, or if there is a different JPEG value (Disable/GREY/Color), then an EOM is forced.

Receiving a JPEG/JBIG File

To receive a JPEG or JBIG file:

NOTE: The only mode supported is RAW.

- FC_TXCODING = DF_JPEG_GREY (for JBIG) or DF_JPEG_COLOR (for JPEG)
- If the application sets FC_TXCODING to DF_JPEG_GREY or DF_JPEG_COLOR, and if the **fx_rcvfax( )** is not issued with the DF_RAW, the function will return an error.
- **ATFX_CODING( )** reports DF_JPEG_GREY or DF_JPEG_COLOR.
- As was done previously, FC_RXCODING specifies the receive file, except if **ATFX_CODING( )** reports DF_JPEG_GREY or DF_JPEG_COLOR.
- Backward compatibility is preserved with MH/MR/MMR reception.

Receiving a JPEG/JBIG Fax – Example

To receive a JPEG fax:

```
     FIS_PrmJPEG_JP_JPEG | FIS_PrmJPEG_JP_FULL_COLOR
DCS = QFC3_MsgReportCapsEvt_JPEG_JP_DISABLED
     ATFX_CODING = DF_MH, DF_MR or DF_MMR
     -> receive raw file
EOM
DCS = QFC3_MsgReportCapsEvt_JPEG_JP_JPEG
     ATFX_CODING = DF_JPEG_GREY
     -> receive raw file
EOM
DCS = QFC3_MsgReportCapsEvt_JPEG_JP_FULL_COLOR
     ATFX_CODING = DF_JPEG_GREY
     -> receive raw file
EOP
```

4. Background on Fax Communications

4.1. Overview

This chapter presents general background information on fax technology. It introduces you to the relevant fax terminology, describes the structure of a fax call, as encompassed in the ITU-T T.30 fax protocol recommendation, and discusses the types of fax transmission. This chapter also mentions fax library functions and structures as they relate to the topic discussed.

4.2. Fax Terminology

To understand how the fax API functions apply to sending and receiving fax documents, you should understand the distinction among the following terms.

Term	Definition
caller	An application or station that places a call. Also referred to as a calling application or station.
called	An application or station that receives a call.
fax transmitter	An application or station that sends or is capable of sending a fax document.
fax receiver	An application or station that receives or is capable of receiving a fax document.
fax session	The five phases of a fax call as defined by the ITU-T T.30 protocol recommendation. They are: • Phase A (set up fax call) • Phase B (pre-message procedure) • Phase C (transmit message) • Phase D (post-message procedure) • Phase E (release fax call).

Term	Definition
normal transmission	A type of fax transmission where the caller station transmits a fax to the called station.
polling transmission (fax on demand)	A type of fax transmission where the called station is asked (polled) to transmit a fax back to the caller station. Also referred to as polled transmission.
turnaround polling transmission	A type of fax transmission where the caller and called stations alternate between transmit and receive modes during the same call.

The fax library allows your fax application to be a **caller** or **called** application, and a fax **transmitter** or fax **receiver**. This distinction is important in a **polling transmission** where the called application transmits documents back to the caller. Polling transmission is described in detail later in this chapter.

NOTE: The discussion of fax transmission and reception in this chapter is based on the use of the fax application as both a caller and called application. Your fax application will be either the caller or the called application. The other station (the remote station) may be a fax machine or another fax application.

As a fax **transmitter**, your application can perform the following fax procedures:

- Send a complete fax document or send data from various sources with independently defined page and document boundaries.
- Send an indication to the receiver requesting the receiver to send a fax document to the transmitter (polling).
- Both of the above during the same call.

As a fax **receiver**, your application can perform the following fax procedures:

- Receive incoming fax data.
- Indicate to the transmitter if polling is acceptable.
- Indicate to the caller that the called application only has fax transmission capabilities (caller can only receive a fax or disconnect).

4.3. Structure of a Fax Call

T.30 is an ITU-T recommendation that specifies a fax communications protocol for Group 3 fax. This recommendation describes how to establish and terminate communications between Group 3 fax machines. ITU-T is the International Telecommunication Union, a United Nations agency that develops and recommends international telecommunications standards.

The recommendation specifies five separate phases in a fax call or session. These phases are implemented using fax library API functions along with fax library data structures that accomplish the proper negotiation of each phase.

The five consecutive phases indicating the flow of a fax session are:

- **Phase A** - set up fax call (begin fax session)
- **Phase B** - pre-message procedure
- **Phase C** - transmit message
- **Phase D** - post-message procedure
- **Phase E** - release fax call and disconnect (end fax session)

4.3.1. Phase A - Set Up Fax Call

Phase A establishes communication between two stations: caller and called. This usually begins with a request for service and, in some cases, dialing the other station.

NOTE: The connection between the caller and called stations is implemented by functions other than fax library API functions. For analog TDM bus and stand-alone configurations, see the *Voice API Library Reference*; for digital TDM bus configurations, see the *Digital Network Interface Software Reference*. For DM3 products, also see the *Global Call API Programming Guide* and *Global Call API Library Reference*.

Once the line connection with the called party is established, the following takes place during Phase A:

- fax tone detection

- digital handshake detection

The caller station typically sends an 1,100 Hz tone known as the CNG tone. The called station responds with a 2,100 Hz tone, the CED.

In preparation for Phase B, the **caller** station is initially given fax **transmitter** status, and the **called** station is initially given fax **receiver** status. (The initial fax state for the fax application must be set by the application prior to issuing the first *send* or *receive* function of a fax session.)

4.3.2. Phase B - Pre-Message Procedure

Phase B is used by the transmitter and receiver to negotiate the parameters for sending and receiving the fax document/page, such as polling, type of data, transmission speed, resolution, width and more.

In this phase, the receiver identifies its capabilities to the transmitter, and the parameter values used depend on the capabilities of the **receiver**.

4.3.3. Phase C - Transmit Message

Phase C transmits the fax document page based on the parameters negotiated between the caller and called applications in Phase B.

4.3.4. Phase D - Post-Message Procedure

Phase D defines a continuation value to indicate to the receiver what to do after the transfer of the fax document/page is completed.

The fax *send* function transmits data from various sources with independently defined page and document boundaries. Phase D continuation values allow multiple send functions to be linked together to transmit data from many document file sources, building a fax transmission dynamically.

Phase D continuation values are used in a table of DF_IOTT structures to indicate Phase D commands to the receiver. The DF_IOTT table defines parameters for transmitting one or more files containing fax data. Each DF_IOTT table entry contains parameters describing the characteristics of the fax data to be sent. For

details on DF_IOTT, see *Section 10.6. DF_IOTT – Fax Transmit Data Description* on *page 128*.

4.3.5. Phase E - Release Fax Call

Phase E releases the fax call. The caller station sends a disconnect signal (DCN) and both fax stations disconnect from the phone line.

4.4. Types of Fax Transmission

Fax transmission is categorized as follows:

- normal
- polling
- turnaround polling

4.4.1. Normal Fax Transmission

A normal fax transmission occurs when a caller station sends a fax to the called station.

When the initial fax connection is made between the caller and the called stations, set the **caller** application to be the **transmitter** and the **called** application to be the **receiver**.

- The caller application issues a *send* function to transmit the fax to the called application.
- The called application issues a *receive* function to indicate readiness to receive a fax transmission.
- The caller application transmits the fax to the called application or disconnects for a reason indicated by an error code.

The following chart shows the sequence of a normal fax transmission in a caller and called fax application.

Table 2. Normal Fax Transmission Sequence

Caller Application	Called Application
Fax TRANSMITTER: • set initial fax state: CALLER • send function issued • (send function completes)	Fax RECEIVER: • set initial fax state: CALLED • receive function issued • (receive function completes)
Fax Transfer Status: • Fax transmitted to CALLED	Fax Transfer Status: • Fax received from CALLER

4.4.2. Polling Fax Transmission (Fax on Demand)

A polling fax transmission occurs when the called station is asked (polled) to send a fax to the calling station.

As in a normal fax transmission, when the initial fax connection is made between the caller and the called stations, set the **caller** to be the fax **transmitter** and the **called** to be the fax **receiver**.

To initiate a polling fax transmission, the caller application requests (polls) the called application to send a fax document to the caller. To make this request, the caller application issues a *receive* function rather than a *send* function as would be issued by the caller to indicate a normal fax transmission.

The called application accepts or rejects a poll request based on the value of the *poll* bit set in the initial called application's *receive* function.

- **Polling Valid.** If polling is valid and the caller issues a *receive* function, the called application's *receive* function returns a zero (in synchronous mode) or a completion event occurs (in asynchronous mode).

  After the called application examines the reason for termination and determines that a poll has occurred, the applications switch roles: the caller application becomes the fax receiver and the called application becomes the fax transmitter. The called application then transmits the fax document to the caller by issuing a *send* function.

  The called application must respond as quickly as possible with the *send* function.

- **Polling Invalid.** If polling is invalid and the caller issues a *receive* function, the called application indicates to the caller that it is not capable of a fax transmission and the fax session is terminated.

If the caller does not poll, the call progresses as in a normal fax transmission.

The caller application becomes the fax receiver under the following conditions:

- The caller polls by issuing an initial *receive* function.
- The *poll* bit is set to polling valid in the called application's *receive* function. Setting the *poll* bit causes the *receive* function to notify the application when polling has occurred.

The following chart shows the sequence of a polling fax transmission in a caller and called fax application.

Table 3. Polling Fax Transmission Sequence

Caller Application	Called Application
Fax TRANSMITTER:	Fax RECEIVER:
• set initial fax state: CALLER	• set initial fax state: CALLED
• receive function issued (poll request)	• receive function issued (poll bit = polling valid)
• (receive function still active)	• (receive function completes)
• (receive function still active)	• (examine termination reason: polling occurred)
CALLER is now a RECEIVER:	CALLED is now a TRANSMITTER:
• (receive function still active)	• send function issued
• (receive function completes)	• (send function completes)
Fax Transfer Status:	Fax Transfer Status:
• Fax received from CALLED	• Fax transmitted to CALLER

Blocking Incoming Faxes

A called application wishing to block incoming fax transfers and only transmit fax data indicates to the caller that a poll is required by issuing an initial *send* function. When a *send* function is initially issued by the called application, **the caller cannot transmit a fax**.

The following chart shows the sequence of a valid polling fax transmission where the called application issues an initial *send* function indicating **transmit only**:

Table 4. Polling Fax Transmission Sequence - Called Application Transmit Only

Caller Application	Called Application
Fax TRANSMITTER: • set initial fax state: CALLER • receive function issued CALLER is now a RECEIVER: • (receive function completes)	Fax RECEIVER: • set initial fax state: CALLED • send function issued CALLED is now a TRANSMITTER: • (send function completes)
Fax Transfer Status: • Fax received from CALLED	Fax Transfer Status: • Fax transmitted to CALLER

4.4.3. Turnaround Polling Fax Transmission

A turnaround polling fax transmission occurs when two stations alternate between send and receive modes during the same call. Each station becomes a fax transmitter and receiver at different times during the call.

The first fax sent in this fax session completes as a normal fax transmission.

To indicate that there is more fax data to follow, the caller application (the initial fax transmitter) specifies the proper transmit data continuation value at the end of the initial fax data transmission. (This value is set in the **io_phdcont** field of the DF_IOTT structure which is used by the *send* function.)

The caller application then follows the initial *send* function with a *receive* function to indicate that the current **transmitter** now wishes to become the fax **receiver**.

The *poll* bit set in the *receive* function of the initial receiver (called) application determines whether polling by the transmitter (caller) is valid.

- **Polling invalid.** If the *poll* bit is set to polling invalid, the called application will remain the fax receiver.
- **Polling valid.** If the *poll* bit is set to polling valid, the called application's *receive* function returns a zero (in synchronous mode) or a completion event occurs (in asynchronous mode). After the called application examines the reason for termination and determines that a poll has occurred, the applications switch roles. The caller application becomes the fax receiver and the called application becomes the fax transmitter. The channel remains open and the called application must respond as quickly as possible by issuing a *send* function. This *send* function is issued by the new transmitter application to send the fax to the new receiver application.

If the caller does not poll, the called application remains the fax receiver as in a normal fax transmission.

NOTE: In a valid turnaround polling fax transmission, the caller application is the transmitter, then the receiver.

The following chart shows the sequence of a turnaround polling fax transmission in a caller and called fax application.

Table 5. Turnaround Polling Fax Transmission Sequence

Caller Application	Called Application
Fax TRANSMITTER: • set initial fax state: CALLER • send function issued (continuation value: indicates additional fax data to follow) • (send function completes)	Fax RECEIVER: • set initial fax state: CALLED • receive function issued (poll bit = polling valid) • (receive function still active)
Fax Transfer Status: • Fax transmitted to CALLED	Fax Transfer Status: • Fax received from CALLER (waiting to receive more fax data) (receive function still active)
CALLER is still a TRANSMITTER: • receive function issued (poll request) • (receive function still active) CALLER is now a RECEIVER: • (receive function still active) • (receive function completes)	CALLED is still a RECEIVER: • (receive function completes) • (examine termination reason: polling occurred) CALLED is now a TRANSMITTER: • send function issued • (send function completes)
Fax Transfer Status: • Fax received from CALLED	Fax Transfer Status: • Fax transmitted to CALLER

4.5. File Storage Formats

Fax data can be stored in one of the following formats. Support for file storage formats varies by product; see *Section 2.3. Key Product Features* on *page 8.*

- raw or unstructured format
- TIFF/F (Tagged Image File Format meeting Class F specifications)
- ASCII for transmit only (includes the Intel® Dialogic® extended ASCII character set and the Katakana character set)

4.5.1. Raw Files

Fax data stored in raw, unformatted files contains only a single page of fax data per file. A description of the data, such as width, resolution and encoding scheme is specified in the DF_IOTT structure.

NOTE: The raw data must be in a fill order of Least Significant Bit (LSB) first.

Storage

Raw MH encoded data is recorded by the fax library with a fill order of Least Significant Bit (LSB) first and may not have EOL (End Of Line) sequences byte aligned.

Raw MR encoded data is recorded by the fax library with a fill order of Least Significant Bit (LSB) first.

Raw MMR encoded data is also recorded by the fax library with a fill order of Least Significant Bit (LSB) first. There is no zero fill, EOL (End Of Line) sequences or byte alignment for MMR stored files.

Transmission

Raw, unformatted files for transmission are treated as a byte stream of compressed fax data with the width, resolution and encoding scheme of the stored data specified in the fields of the DF_IOTT structure.

For transmission, **raw MH encoded data** must include EOL (End Of Line) flags, but may or may not contain RTC (Return To Control) sequences. RTC sequences are inserted by the firmware at the end of Phase C (message transmission) if the raw data does not contain them.

4.5.2. TIFF/F Files

TIFF/F refers to Tagged Image File Format meeting Class F specifications.

A TIFF/F file stores MH, MR, or MMR encoded data with additional header information and tags. Information such as the starting page number, page count and data type is specified in the DF_IOTT structure. Incoming fax data stored in TIFF/F format is written by the fax library with tags specified in *Appendix A*. TIFF/F files that include all mandatory tags (or subset) with valid values are accepted for transmission.

4.5.3. ASCII Files

During transmission, ASCII text files are converted to an encoded fax image. A description of the data, such as width, resolution and encoding scheme, is specified in the DF_IOTT structure. You can define additional attributes using the DF_ASCIIDATA structure. Note that the DF_ASCIIDATA structure is not used on DM3 boards. For more information, see *Section 10.3.3. DF_ASCIIDATA Usage* Rules in the DF_ASCIIDATA structure description.

To set fonts for use in ASCII to fax conversion, see *Chapter 7. Specifying Fonts in ASCII to Fax Conversion*.

4.6. Data Encoding Schemes

Several data encoding methods exist that compress fax data and reduce the size of the file to be transmitted, thereby increasing the speed of a fax transmission.

The following data encoding schemes are supported for transmitting fax data and storing incoming fax data. Support for data encoding schemes varies by product; for details, see *Section 2.3. Key Product Features* on *page 8*.

- **Modified Huffman** (MH) ITU-T T.4 Recommendation for Group 3 fax.
- **Modified Read** (MR) ITU-T T.4 Recommendation for Group 3 fax (transmit only). If negotiated during Phase B of the T.30 protocol, MH and MMR stored fax data is converted to MR line encoded data.
- **Modified Modified Read** (MMR) ITU-T T.6 Recommendation for Group 4 fax.

MH is a one-dimensional encoding scheme that compresses each horizontal scan line of the image.

Modified Read (MR) and Modified Modified Read (MMR) are two-dimensional encoding schemes that make use of the high degree of vertical correlation between each scan line in the fax image to achieve a higher compression than MH.

The highest data compression is achieved using the MMR encoding scheme.

Although not all fax machines can receive MR or MMR encoded data, some fax products are capable of converting MH or MMR stored fax data to MH, MR or MMR line encoding schemes during fax transmission, and converting incoming fax data to MH or MMR for data storage during fax reception.

NOTE: An error is returned to your application if unsupported encoding schemes are used.

4.7. Error Correction Mode (ECM)

Error Correction Mode (ECM) is a T.30 recommendation that provides more efficient error handling for noisy or distorted fax transmissions. It enables the receiver to check for and request retransmission of garbled data.

The Error Correction Mode (ECM) switch allows you to explicitly enable ECM T.30 protocol for a fax transmission. The use of ECM for a fax transmission is determined during Phase B negotiations and is based on the capabilities of the receiving station (remote station).

The encoding scheme in which the data is presented for transmission does not determine the phone line encoding scheme for data transmission. Rather, during Phase B negotiations, the FC_TXCODING fax parameter values (**fx_setparm( )** function) and the receiving station's capabilities determine the phone line encoding scheme, and whether to use ECM.

MMR line encoding always requires the use of ECM.

For more information on setting the ECM switch, see **fx_setparm( )** in *Chapter 12. Fax Library Function Reference*.

4.8. Image Scaling

Image scaling refers to the process by which the original image dimensions are reduced so that the full image (although in reduced form) is received at the remote station. The aspect ratio of the original image is maintained.

Image scaling is used when the remote station's recording width is smaller than the original image for transmission.

4.9. Image Resolution

Resolution refers to the level of picture detail of a fax image. The standard **horizontal** resolution is 203 lines per inch across the page. Two grades of **vertical** resolution are available:

- high or fine resolution at 196 lines per inch
- low or coarse resolution at 98 lines per inch

For more information on setting image resolution, see *Sections 5.7.5. Select Resolution for Fax Transmission* (*page 72*), *6.3.8. Resolution for Storing Incoming Fax Data* (*page 93*) and *10.6. DF_IOTT – Fax Transmit Data Description* (*page 128*).

4.10. Subaddress Fax Routing

A subaddress is a T.30 message protocol that allows a fax to be routed to one or more telephone numbers (or extensions) once it is received by the fax station.

This feature is not supported on DM3 boards.

Subaddress fax routing allows applications to do the following:

- **Transmit subaddress fax routing information:** Applications can send a 20-character (maximum) string that contains a combination of one or more phone numbers and/or extensions to allow a remote receiver with the capability of using the T.30 subaddress message to route the received fax data.
- **Receive subaddress fax routing information:** Based on the contents of the T.30 subaddress message received from the transmitter, applications can route incoming fax data to one or more phone numbers and/or extensions.

For more information on implementing this feature, see *Sections 5.7.6. Enable Subaddress Fax Routing* on *page 73* and *6.2.6. Routing Fax Data to Multiple Subaddresses* on *page 85*.

5. Implementing Send Fax Capability

5.1. Overview

This chapter and the next provide guidelines on how to use the fax library to implement fax capability in an application. Fax library functions and data structures used in completing a task are discussed in these chapters. For complete reference information on functions and data structures, see *Chapters 10. Fax Data Structures*, *11. Using the Fax Library* and *12. Fax Library Function Reference.*

This chapter focuses on the send fax capability and covers the following topics:

- *5.2. Guidelines for Implementing Fax*
- *5.3. Opening and Closing a Fax Channel Device*
- *5.4. Setting the Initial State of a Fax Channel*
- *5.5. Specifying Fax Data for Transmission in a DF_IOTT Table Entry*
- *5.6. Setting Parameters for Send Fax*
- *5.7. Setting the Bit Mask for a Send Fax Function*
- *5.8. Issuing a Send Fax Function*
- *5.9. Stopping a Fax Transmission or Reception*
- *5.10. Replacing Bad Scan Lines*
- *5.11. Creating User-Defined I/O Functions*

Support for the features described in this guide varies by product. For a listing of key features by product, see *Section 2.3. Key Product Features* on *page 8*.

5.2. Guidelines for Implementing Fax

Follow these guidelines to implement fax capability in your application.

Table 6. Guidelines for Creating Fax Applications

Step	Action
1.	Open a channel for voice and a channel for fax. Open a network channel if applicable.
2.	If your system configuration uses the TDM bus assign time slot routing for TDM bus. The program flow for these system configurations is given in *Section 2.7. System Configuration Models* on *page 17*.
3.	Set the initial state of the fax channel using **fx_initstat( )**. Following T.30 protocol, the caller is initially set to be the transmitter of a fax and the called station is initially set to be the receiver of a fax.
4.	Configure your fax device channel using **fx_setparm( )**. Several parameters are available to define values such as: • fax header • transmit and receive baud rate • retransmission count • preferred data transmission encoding scheme (over the phone line)
5.	When sending a fax, define a DF_IOTT table entry for each document (raw, TIFF/F and ASCII) to be transmitted. This table entry provides a description of the fax data to be transmitted and includes information such as the type of data, number of pages, width, resolution, and Phase D continuation values. You can use the **fx_setiott( )** function to initialize the DF_IOTT structure.
6.	When sending ASCII documents, you can further describe the ASCII data using the DF_ASCIIDATA structure. If you do not use this data structure, certain default values are assumed for your ASCII document.

Step	Action
7.	Set up the call by using voice API functions; that is, dial the number, wait for rings and so on.
8.	Initiate send fax or receive fax on the channel using **fx_sendfax( )**, **fx_rcv( )** or **fx_rcv2( )**. Send fax convenience functions can also be used instead of **fx_sendfax( )**.
9.	After the send fax and receive fax functions are completed, use the voice API functions to disconnect the call.
10.	At the end of the application, close the open channels.

Details on these guidelines are provided later in this chapter. For information on voice functionality, see the *Voice API Library Reference*.

5.3. Opening and Closing a Fax Channel Device

Before performing any operation on a fax channel device, open the device using **fx_open( )**. For additional information on device discovery on DM3 boards, see *Section 3.2. Device Discovery*.

NOTE: Compatibility is maintained with older VFX products for applications using the device handle from **dx_open( )** to call fax API functions. However, for DSP Fax and DM3 Fax products, you must use **fx_open( )** to open a device channel for fax processing. You cannot use **dx_open( )** for this operation. The same is true for **dx_close( )** and **fx_close( )**.

The **fx_open( )** function returns a unique device handle for that particular open process on that channel. The channel device handle is referred to as **dev**:

```
int dev;
dev = fx_open(channel_name,mode)
```

To use a fax library function on the channel, you must identify the channel with its channel device handle, **dev**. The channel name is used only when opening a channel, and all actions thereafter must use the handle **dev**.

You can open and use a fax channel without ever opening the board it is on. No board-channel hierarchy is imposed by the driver.

In applications which create child processes from a parent process, device handles are not inheritable from the parent process to the child process. Make sure that devices are opened in the child process.

Both the voice and fax channel need to be open for fax resource capability.

The voice driver supports specific fax library functions with synchronous/asynchronous modes of operation.

5.4. Setting the Initial State of a Fax Channel

Set the initial state of the fax channel using **fx_initstat( )**. Following T.30 protocol, the caller station is initially set to be the transmitter (DF_TX) of a fax and the called station is initially set to be the receiver (DF_RX) of a fax.

Use this function once before issuing the **first** *send* or *receive* function of a fax session. Fax session refers to the completion of a fax call from Phase A through Phase E, as defined by the T.30 protocol.

5.5. Specifying Fax Data for Transmission in a DF_IOTT Table Entry

The DF_IOTT structure contains fields describing the fax data for one fax document to be transmitted. Each structure describes one source for fax data: raw, TIFF/F or ASCII. A linked list or array of DF_IOTT structures (table) can be created to specify multiple fax documents for transmission using the fax *send* function, **fx_sendfax( )**. A pointer argument in the fax *send* function points to the DF_IOTT table.

The DF_IOTT table may contain entries specifying fax data of different widths, resolutions and encoding schemes. Before the fax data is transmitted, the validity of each DF_IOTT table entry is verified.

For complete reference information on all fields in the DF_IOTT structure, see *Section 10.6. DF_IOTT – Fax Transmit Data Description* on *page 128*.

The following topics on DF_IOTT are discussed:

5.5.1. Declaring a Table of DF_IOTT Entries

The following usage notes and cautions apply when you declare a table of DF_IOTT entries:

- Declare the DF_IOTT entries which are passed as an argument to **fx_sendfax( )** as global or static in your application.
- Do not modify the DF_IOTT entries until after **fx_sendfax( )** has completed. The DF_IOTT entries must exist for the duration of the fax transmission.
- In asynchronous mode, the fax library must repeatedly access the DF_IOTT entries during the fax transmission even after **fx_sendfax( )** has returned control to the application. Each channel controlled by the single process must have its own separate DF_IOTT table.
- The **io_type** field of the last DF_IOTT table entry must contain an IO_EOT to identify it as the last table entry.
- If the IO_EOT flag is set in the **io_type** field, then all the other flags are ignored.

The **fx_setiott( )** function can be used to initialize DF_IOTT structure values. For more information, see *Chapter 12. Fax Library Function Reference*.

5.5.2. Connecting DF_IOTT Table Entries

When sending more than one fax document in a single **fx_send( )** operation, you must build a linked list or array of DF_IOTT structures (table).

This DF_IOTT table may represent a combination of data in MH or MMR encoding schemes, or in ASCII format. The valid encoding scheme or format of the stored files specified for transmission depends on the capability of the fax product.

Specify the link between DF_IOTT table entries using the **io_type** logical OR field as follows:

- If the next entry is linked to the current one, specify IO_LINK and use **io_nextp** to point to the next DF_IOTT entry.
- If the entry is the last DF_IOTT entry in the chain, specify IO_EOT.
- If the next entry is contiguous, specify IO_CONT and set **io_nextp** and **io_prevp** fields to NULL. This is the default setting.

If neither IO_EOT nor IO_LINK is specified, the next entry is contiguous (next element in the array).

The fax library automatically builds the backward links for the DF_IOTT chain when **fx_sendfax( )** is issued. The **io_prevp** field of the first DF_IOTT entry is set to NULL.

5.5.3. Sending Data from Device or Memory

Use the **io_type** field of the DF_IOTT structure to specify whether you are sending data from a device or from memory.

- For data stored on a disk device, specify IO_DEV.
- For data stored in memory, specify IO_MEM.

IO_MEM is only valid when **io_datatype** is set for raw data (DF_RAW) or ASCII (DF_ASCII).

5.5.4. Specifying File Storage Format

Use the **io_datatype** field of the DF_IOTT structure to specify the file storage format for transmission as follows:

- For a **raw**, compressed unstructured file, specify DF_RAW.
- For a **TIFF/F** file, specify DF_TIFF.
- For an **ASCII** text file (converted to a fax image at the time of transmission), specify DF_ASCII.

5.5.5. Sending Raw Files

Fax data stored as a raw file contains no information on the format of the fax data. When the raw data is sent, the width, resolution and encoding scheme for the stored raw fax data must be specified in the DF_IOTT structure.

If the width and resolution of the data in the raw file do not match the capabilities of the receiving station, automatic image scaling is provided.

Raw files negotiated for transfer in MH or MR line encoding scheme are sent with a fill order of Least Significant Bit (LSB) first; End of Line (EOL) sequences are not byte aligned.

Table 7. DF_IOTT Fields for Raw Files lists DF_IOTT fields used to send raw files:

Table 7. DF_IOTT Fields for Raw Files

Field	Value or Description
io_datatype	DF_RAW
io_width	Width of the stored raw fax data.
io_resln	Vertical resolution for the stored raw fax data.
io_coding	Encoding scheme for raw files: DF_MH for Modified Huffman, DF_MR for Modified Read, and DF_MMR for Modified Modified Read. See *Section 5.5.8. Specifying Encoding Scheme* on *page 58* for more information.
io_offset	The starting byte location in the file/memory for the data transfer. Setting **io_offset** to zero starts the transfer from the beginning of the file/memory.
io_length	The number of bytes to transfer. This field is used with the **io_type** value.
io_type	The entry type: • If **io_type** is set to IO_DEV and **io_length** is set to -1, data is transferred until the end of the file is reached. • If **io_type** is set to IO_MEM, **io_length** indicates the exact number of bytes to transfer from the buffer. • If **io_type** is set to IO_MEM, **io_bufferp** points to the buffer in memory containing the raw image data.

5.5.6. Sending TIFF/F Files

The DF_IOTT structure may be set to send all or part of a single or multi-page TIFF/F file.

The fax library defaults to sending a single TIFF/F page beginning at page zero. To transmit TIFF/F files with a base 1 page numbering scheme, use the FC_TFPGBASE channel parameter in **fx_setparm( )**.

To send a subset of a TIFF/F file, use the **io_pgcount** and **io_firstpg** fields; see *Table 8. DF_IOTT Fields for TIFF/F Files.*

The **io_pgcount** value specifies the number of pages to send, and the **io_firstpg** value specifies the first page number to send. If the value of **io_firstpg** is zero (default), the number of pages specified by **io_pgcount** is sent. For example, to send document pages 0, 1 and 2 of a TIFF/F file, set **io_firstpg** to 0, and **io_pgcount** to 3.

If the width and resolution of the data in the TIFF/F file do not match the capabilities of the receiving station, automatic image scaling is provided.

Table 8. DF_IOTT Fields for TIFF/F Files lists DF_IOTT fields used to send TIFF/F files.

Table 8. DF_IOTT Fields for TIFF/F Files

Field	Value or Description
io_datatype	DF_TIFF
io_firstpg	The first page number to send. **The first page in the file is referenced as page zero**.
io_pgcount	The number of pages to send. If **io_pgcount** is -1 (default), all remaining pages in the file are sent.
io_coding	This field is ignored. The fax library reads the TIFF/F tags embedded in the file to determine the encoding scheme of the stored data.

Handling Multi-Page TIFF/F Files

In a multi-page TIFF/F file transmission, all pages preceding the final page are set to DFC_EOM (FC_SENDCONT parameter in **fx_setparm( )**) on Springware boards and DFC_AUTO on DM3 boards. The last selected page of the specified TIFF/F file uses the Phase D continuation value set in **io_phdcont** (DF_IOTT structure).

To change the default intermediate page continuation value, set the FC_SENDCONT parameter to a different value or to DFC_AUTO. For more information, see *Section 5.5.9. Setting Phase D Continuation Values* (*page 59*) and **fx_setparm( )** (*page 322*).

Troubleshooting

To ensure that TIFF/F files are sent successfully, the TIFF/F file must contain:

- all mandatory TIFF/F tags (or subset)
- valid TIFF/F tag values
- correct TIFF/F file header values
- valid PageNumber tag values

For a table of TIFF/F tags and values, see *Appendix A*. For information on error codes returned, see *Appendix D*.

5.5.7. Sending ASCII Files

ASCII files are converted to a fax image at the time of fax transmission and sent at the width, resolution and other values as specified in DF_IOTT.

The fax image is encoded over the phone line in MH or in the encoding scheme specified by the FC_TXCODING parameter of **fx_setparm( )**.

Converted ASCII files negotiated for transfer in the MH or MR line encoding scheme are sent with a fill order of Least Significant Bit (LSB) first; End of Line (EOL) sequences are byte aligned.

Table 9. DF_IOTT Fields for ASCII Files lists DF_IOTT fields used to send ASCII files.

Table 9. DF_IOTT Fields for ASCII Files

Field	Value or Description
io_datatype	DF_ASCII
io_width	Width of the fax image.
io_resln	Vertical resolution of the fax image.
io_offset	Byte offset in ASCII file/memory to start reading the ASCII data. Setting **io_offset** to zero starts reading the ASCII data from the beginning of the file/memory.
io_length	Number of bytes of ASCII data to read, convert and send. This field is used with the **io_type** field.
io_type	Entry type: • If **io_type** is set to IO_DEV and **io_length** is set to -1, data is transferred until the end of the file is reached. • If **io_type** is set to IO_MEM, **io_length** indicates the exact number of bytes of ASCII data to read from the buffer. This allows the application to select a portion of an ASCII file for transmission. • If **io_type** is set to IO_MEM, **io_bufferp** points to the buffer in memory containing the ASCII data.
io_datap	A pointer to an optional DF_ASCIIDATA structure that contains parameters and values for the ASCII data. If this pointer is NULL, the DF_ASCIIDATA structure defaults are used. For information on this structure, see *Chapter 10. Fax Data Structures.*
io_coding	This field is ignored for ASCII file transfer.

5.5.8. Specifying Encoding Scheme for Data Transmission

The encoding scheme used in transmitting data over the phone line varies by product. For product support, see *Section 2.3. Key Product Features* on *page 8*. **The negotiated encoding scheme is determined by the receiver's capability.**

The **io_coding** field in the DF_IOTT structure specifies the transmission encoding scheme and is used only in sending **raw** files. The available values for **io_coding** are:

- DF_MH – Modified Huffman
- DF_MR – Modified Read
- DF_MMR – Modified Modified Read

For TIFF/F and ASCII files, the **io_coding** field is ignored.

Some fax products provide the option to specify the preferred line encoding scheme for fax transmission. This option uses the FC_TXCODING parameter in **fx_setparm( )**. For more information on FC_TXCODING, see *Section 5.6.2. Specifying a Preferred Encoding Scheme for Transmission* on *page 65* and the **fx_setparm( )** function reference on *page 322*.

The transmitting channel uses the FC_TXCODING value during Phase B negotiation with the remote receiver. The fax image data provided via the DF_IOTT structures is automatically converted to the negotiated line encoding scheme at the time of transmission. The final negotiated line encoding scheme for transmission depends on the receiver's capability.

To determine the negotiated line encoding scheme, call **ATFX_CODING( )** after the negotiation of Phase B is completed.

5.5.9. Setting Phase D Continuation Values

Each DF_IOTT table entry specifies a continuation value for Phase D (post-message procedure) of the T.30 protocol in the **io_phdcont** field.

The **io_phdcont** field defines the way in which a following DF_IOTT entry is connected to the current DF_IOTT entry. Based on the **io_phdcont** field value, a message is sent from the transmitter to the receiver at the end of the current DF_IOTT entry's fax data. By selecting the appropriate value, you can transmit more data from the next DF_IOTT entry or terminate the fax session.

The **io_phdcont** field can have one of the values listed in *Table 10. Phase D Continuation Values*.

Table 10. Phase D Continuation Values

Value	Description
DFC_AUTO	**Automatic Phase D Messaging.** The fax driver automatically determines the T.30 protocol Phase D continuation value based on the width, resolution and position of the DF_IOTT entries. Possible values automatically assigned are DFC_EOM, DFC_EOP and DFC_MPS. This setting forces negotiation of Phase B when a page of a different width and/or resolution is found. If the following page has the same format as the current page, this setting bypasses Phase B negotiation for each page and saves transmit time. This is the default setting on DM3 boards.
DFC_MPG	**Merge-Page**. The data specified for the DF_IOTT entry directly following the current DF_IOTT entry is concatenated to the same page.
DFC_EOP	**End of Procedure (T.30)**. Terminate current fax session; progress to Phase E; and disconnect fax call.

Value	Description
DFC_EOM	**End of Message (T.30)**. End of current fax document page; more fax data to follow at different resolution or width; return to Phase B and negotiate parameters for next fax document page. This setting forces the negotiation of Phase B after each page. This is the default setting on Springware boards.
DFC_MPS	**Multi-Page Signal (T.30)**. End of current fax document page; next fax document page is in the same format as the current page; proceed directly to Phase C. This setting bypasses Phase B negotiation for each page and saves transmit time.

Hints

- DFC_AUTO and DFC_MPG are Intel® Dialogic® fax library terms, not T.30 protocol terminology.
- DFC_EOP, DFC_EOM and DFC_MPS are provided for backward compatibility and for applications where specific T.30 Phase D continuation values are required.
- If a DF_IOTT entry specifies DFC_EOP as a Phase D continuation value, but it is not the last entry in the table, **ATDV_LASTERR( )** returns an EFX_BADIOTT error.

More detail on each Phase D continuation value is provided next.

Automatic Phase D Messaging - DFC_AUTO

To enable automatic Phase D messaging, set the **io_phdcont** field in the DF_IOTT entry to DFC_AUTO.

By specifying DFC_AUTO for each DF_IOTT entry, Phase D messaging is simplified. The application does not have to determine the correct Phase D continuation value for each DF_IOTT entry; the fax library does this automatically. The fax library uses DFC_EOM, DFC_MPS or DFC_EOP as the continuation value based on the width, resolution, and position of the DF_IOTT entry in the chain as well as the remote receiver's capability.

For example, if you specify DFC_AUTO for the last DF_IOTT entry in the chain, the fax library automatically issues an EOP after transmitting all files specified in the last DF_IOTT entry.

For fax data containing more than one image file per page (also known as sub-page addressing), you must use DFC_MPG. For more information, see *Section 5.5.10. Merging Images from Different Sources or Sub-Page Addressing* on *page 63*.

Merge Page - DFC_MPG

To concatenate data for the DF_IOTT entry directly following the current DF_IOTT entry to the same page, specify DFC_MPG as the Phase D continuation value. This concatenation is also known as **sub-page addressing**.

When you use DFC_MPG, the DFC_MPG entries in a chain are followed by a DF_IOTT entry that specifies DFC_AUTO or a Phase D continuation value (DFC_EOP, DFC_EOM or DFC_MPS) for the last entry of the multi-source fax page.

For more information on sub-page addressing, see *Section 5.5.10. Merging Images from Different Sources or Sub-Page Addressing* on *page 63*.

End of Procedure - DFC_EOP

To disconnect the fax call after Phase E is completed, specify DFC_EOP as the Phase D continuation value in the **io_phdcont** field for a DF_IOTT entry.

After Phase E, the line is still open and the application sets the channel on-hook, if necessary.

End of Message - DFC_EOM

To transmit more data in a different format, specify DFC_EOM as the Phase D continuation value in the **io_phdcont** field. This value allows you to:

- Change the type of data you are sending for the next DF_IOTT entry in a multiple page or multiple source fax transmission.
- Request turnaround polling fax transmission. After the current DF_IOTT entry's fax data is sent, you can send a message to the receiver requesting that the transmitter and receiver switch roles.

When initiating a **turnaround polling fax transmission**, set the **io_phdcont** field to DFC_EOM for the last DF_IOTT entry. This allows the fax session to return to Phase B after the initial fax data transmission is completed. The caller application (transmitter) can then continue with the turnaround polling fax transmission by issuing the **fx_rcvfax( )** or **fx_rcvfax2( )** function to indicate a poll request; see *Section 4.4.3. Turnaround Polling Fax Transmission* on *page 40.*

The current width and resolution values for the fax session remain in effect until a Phase D continuation value of DFC_EOM is reached in a DF_IOTT entry.

If the width and/or resolution of the data described in the next DF_IOTT entry is different from the current entry, specify DFC_EOM in the current entry to renegotiate Phase B.

Multi-Page Signal - DFC_MPS

To transmit more data in the same format as the current page, use DFC_MPS as the Phase D continuation value in the **io_phdcont** field.

All fields of the next DF_IOTT entry should be set to transmit data of the same format (image width, resolution, and so on) as the current DF_IOTT entry's data.

NOTE: When transmitting a multi-page TIFF/F file from a single DF_IOTT structure, the value specified in the **io_phdcont** field of the DF_IOTT structure is the Phase D continuation value after **all** the pages specified in that DF_IOTT structure are sent. The Phase D continuation value used between each page of the multi-page TIFF/F file is specified by the FC_SENDCONT parameter (see **fx_setparm( )**).

5.5.10. Merging Images from Different Sources or Sub-Page Addressing

A single page of fax data can be formed from images stored in different sources. **Each stored image is considered a sub-page**.

To concatenate fax data described in the next DF_IOTT entry to the current DF_IOTT entry on the same page, specify DFC_MPG in the **io_phdcont** field.

For example, to create a page of fax data from three different files (TIFF/F, raw and ASCII), three DF_IOTT entries are required. For each of the first two DF_IOTT entries, set the **io_phdcont** field to DFC_MPG to concatenate the data to the following DF_IOTT entry's data. For the third DF_IOTT entry, specify DFC_AUTO or a Phase D continuation value (DFC_EOP, DFC_EOM or DFC_MPS) in the **io_phdcont** field for the last entry of the page.

NOTE: DFC_MPG and DFC_AUTO are Intel® Dialogic® fax library terms, not T.30 protocol terminology.

The following rules and restrictions apply to sub-page addressing and the use of the DFC_MPG value.

DFC_MPG Usage

- DFC_MPG cannot be specified for the last DF_IOTT entry in a chain or array. The last entry in the chain or array should specify DFC_AUTO or a T.30 protocol Phase D continuation value (DFC_EOP, DFC_EOM or DFC_MPS).
- A DF_IOTT entry for a TIFF/F file specifying DFC_MPG is limited to sending a single page of data for the entry: a one-page TIFF/F file or one page of a multi-page TIFF/F file.

  When selecting a page from a multi-page TIFF/F file, set the **io_firstpg** field to the desired page number and the **io_pgcount** field to 1.

Resolution

- The resolution for the fax data page is determined by the resolution specified for the first sub-page entry or by the resolution specified in the **fx_sendfax( ) sndflag** argument.

Width

- If the width of consecutive sub-pages is different, the sub-pages are scaled to match the negotiated width.

Encoding schemes

- Stored encoded data to be specified as a sub-page may be in one of the supported encoding schemes: MMR, MH. Support for this feature varies by product. For a listing of key features by product, see *Section 2.3. Key Product Features* on *page 8.*

ASCII sub-pages

- If you concatenate multiple ASCII sub-pages on the same page, the top margin, bottom margin, page length and page padding values specified in the first DF_ASCIIDATA structure apply to the entire page.
- The left and right margins, font and line spacing can be set differently for each DF_ASCIIDATA structure sub-page. The margins specified for ASCII sub-pages only apply to the ASCII data and do not affect the image sub-pages.

- Multiple ASCII/image sub-pages concatenated to a single fax page may result in an image that exceeds the length of a single page. This may occur due to the choice of graphical attributes for the ASCII data, the size of the ASCII sub-pages or a large image sub-page. Your application must specify the correct choice of graphical attributes for the ASCII data and know how much space will be taken by an image sub-page.
  - If an image is present at the bottom of the page that exceeds the page length, the page is extended.
  - If the page length specified in the first ASCII sub-page is exceeded while an ASCII sub-page is being processed, the remaining ASCII text is placed on the next fax page.
- Formfeed characters in ASCII sub-pages are ignored.

If the DFC_MPG continuation value is not used properly, **ATDV_LASTERR( )** returns an EFX_BADIOTT error code.

5.6. Setting Parameters for Send Fax

The fax parameters described in this section are set using **fx_setparm( )**. For more information on **fx_setparm( )**, see *Chapter 12. Fax Library Function Reference*.

5.6.1. Selecting a Transmission Baud Rate

Using the FC_TXBAUDRATE parameter in **fx_setparm( )**, you can specify an initial transmission baud rate lower than the default which is the highest supported baud rate for a product. Issue **fx_setparm( )** prior to issuing **fx_sendfax( )**.

Support for transmission baud rate varies by product. For a listing of key features by product, see *Section 2.3. Key Product Features* on *page 8*.

5.6.2. Specifying a Preferred Encoding Scheme for Transmission

Using the FC_TXCODING parameter in **fx_setparm( )**, you can specify the preferred line encoding scheme in which to transmit fax data.

The available values for FC_TXCODING are:

- DF_MH - Modified Huffman
- DF_MR - Modified Read
- DF_MMR - Modified Modified Read
- DF_ECM - Error Correction Mode switch (logically “OR” this bit flag with an encoding scheme)
- DF_JPEG_COLOR (not supported on Springware boards)
- DF_JPEG_GREY (not supported on Springware boards)

The transmitting channel uses the FC_TXCODING value during Phase B negotiation with the remote receiver. The fax image data provided via the DF_IOTT structures is automatically converted to the negotiated line encoding scheme at the time of transmission. The final negotiated line encoding scheme for transmission depends on the receiver's capability.

To determine the negotiated line encoding scheme, call **ATFX_CODING( )** after the negotiation of Phase B is completed.

ECM can be explicitly specified for Phase B negotiation in fax transmission. Use of ECM is determined by the receiver’s capability.

The following guidelines are provided on the use of FC_TXCODING and the ECM switch:

- When you send fax data using MH line encoding, scan line correction can occur after each scan line.
- When you send fax data using MR line encoding, scan line correction can occur after every other scan line (coarse resolution) or every fourth scan line (fine resolution).
- For applications that require transmitted MH or MR encoded fax data to be received error-free, set ECM as an option. Fax machines and applications with ECM and MH or MR capability will receive the fax data exactly as it was sent.
- Sending fax data using MMR line encoding always requires ECM.

Using ECM adds time to the fax transfer based on the size of the fax and the quality of the transmission line.

The following chart shows the highest compression line encoding scheme for a fax transmission determined by the FC_TXCODING value in the transmitter application and the capabilities of the fax receiver.

	Fax Receiver Capabilities				
FC_TXCODING Parameter Setting	**MH**	**MH w/ECM**	**MR**	**MR w/ECM**	**MMR w/ECM**
DF_MH	MH	MH	MH	MH	MH
DF_MH | DF_ECM	MH	MH w/ECM	MH	MH w/ECM	MH w/ECM
DF_MR	MH	MH	MR	MR	MR
DF_MR | DF_ECM	MH	MH w/ECM	MR	MR w/ECM	MR w/ECM
DF_MMR	MH	MH	MR	MR	MMR w/ECM
DF_MMR | DF_ECM	MH	MH w/ECM	MR	MR w/ECM	MMR w/ECM

Setting the FC_TXCODING parameter to DF_MMR|DF_ECM specifies that ECM is used whenever the receiver is capable of ECM for receiving a fax at MMR, MR or MH line encoding.

Setting the FC_TXCODING parameter to DF_MMR specifies that ECM is not used even if the receiver is capable of ECM for receiving a fax at MR or MH line encoding. Note that MMR line encoding always requires ECM.

5.6.3. Defining a Fax Page Header

Fax page header parameters can be set to print a special line of text in a compressed font at the top of every transmitted fax page. There are two possible formats for the fax page header. For more information, see FC_HDRATTRIB and other FC_HDR*name* parameters in the **fx_setparm( )** function reference.

The Telephone Consumer Protection Act requires that a fax transmission include specific information identifying the sender. For more information on the requirements, see *Section 2.8. Complying with the Telephone Consumer Protection Act* on *page 20*.

5.6.4. Retransmitting a Fax

When a fax page is not successfully received, the fax receiver sends a Phase D status value of DFS_RTN (Retrain Negative) or DFS_PIN (Procedure Interrupt Negative) to the transmitter. The fax transmitter can automatically retransmit a fax page that is not successfully received.

If operator intervention (also called voice request) is disabled, only RTN (Retrain Negative) is sent to indicate unsuccessful reception of a fax page.

The number of attempts to retransmit pages from a file is set by the FC_RETRYCNT parameter in **fx_setparm( )**; the default is zero retries. The unsuccessfully received page can be retransmitted once (DF_RETRY1), twice (DF_RETRY2) or three (DF_RETRY3) times.

After the specified number of retry attempts, you can set the transmitter to disconnect the fax call. To do so, logically “OR” the DF_RETRYn value with the DF_RETRYDCN value.

Retry counter parameter values are set with **fx_setparm( )** and read with **fx_getparm( )**.

5.7. Setting the Bit Mask for a Send Fax Function

The **sndflag** parameter of the **fx_sendfax( )** function is a logical OR bit mask that can be set to indicate one or more conditions. For more information, see the following:

- *5.7.1. Mode of Operation* on *page 69*
- *5.7.2. Enable Phase B Event Generation* on *page 70*
- *5.7.3. Enable Phase D Event Generation* on *page 71*
- *5.7.4. Enable Operator Intervention (Voice Request)* on *page 72*
- *5.7.5. Select Resolution for Fax Transmission* on *page 72*
- *5.7.6. Enable Subaddress Fax Routing* on *page 73*

5.7.1. Mode of Operation

Two modes of operation are available for the send fax functions:

- synchronous mode – **sndflag** bit mask set to EV_SYNC
- asynchronous mode – **sndflag** bit mask set to EV_ASYNC

In synchronous mode, the function does not return control to the application until **fx_sendfax( )** completes (zero returned) or an error has occurred (-1 returned); see *Section 11.3. Error Handling* on *page 148*.

In asynchronous mode, the function returns control to the application immediately after invocation. **fx_sendfax( )** returns a zero to indicate successful invocation and a -1 to indicate an invocation error. Once control is returned to the application, the application may continue to send fax data on the specified device or issue voice/fax calls on other devices. The completion (or error termination) of **fx_sendfax( )** is indicated to the application via events generated by the Standard Runtime Library.

The following events are valid for **fx_sendfax( )**:

Event	Indicates…
TFX_FAXERROR	Error in processing
TFX_FAXSEND	Successful completion of **fx_sendfax**( )

See the *Standard Runtime Library API Library Reference* for event information.

5.7.2. Enable Phase B Event Generation

To enable Phase B event generation, specify DF_PHASEB in **sndflag** of the **fx_sendfax( )** function.

When this bit is set, a TFX_PHASEB event is generated each time Phase B of the T.30 protocol is completed while **fx_sendfax( )** is transmitting fax data.

If you issue **fx_sendfax( )** in synchronous mode (EV_SYNC), you must install an event handler to handle Phase B events using the **sr_enbhdlr( )** function of the Standard Runtime Library. For event handler details, see the *Standard Runtime Library API Library Reference.*

When a TFX_PHASEB event occurs, you can call these fax extended attributes for the following information:

Fax Extended Attribute	Returns…
ATFX_BSTAT( )	Phase B status information
ATFX_CODING( )	Negotiated line encoding scheme for the data transfer
ATFX_SPEED( )	Baud rate of the data transfer
ATFX_STATE( )	State of the fax channel device

5.7.3. Enable Phase D Event Generation

To enable Phase D event generation, specify DF_PHASED in **sndflag** of the **fx_sendfax( )** function.

When this bit is set, a TFX_PHASED event is generated each time Phase D of the T.30 protocol is completed during the send fax operation. A Phase D event is generated for every page except for the last page. After the last page, if your application is running in synchronous mode **fx_sendfax( )** completes or in asynchronous mode a TFX_FAXSEND event occurs.

Phase D events allow the application to monitor the progress of the fax transmission on a page-by-page basis.

If you issue **fx_sendfax( )** in synchronous mode (EV_SYNC), you must install an event handler to handle Phase D events using the **sr_enbhdlr( )** function of the Standard Runtime Library. See the *Standard Runtime Library API Library Reference* for event handler details.

When a TFX_PHASED or TFX_FAXSEND event occurs, you can call these fax extended attributes for the following information:

Fax Extended Attribute	Returns…
ATFX_PHDCMD( )	Phase D command
ATFX_PHDRPY( )	Phase D reply
ATFX_WIDTH( )	Width of the page
ATFX_RESLN( )	Resolution of the page
ATFX_SCANLINES( )	Total number of scan lines transferred
ATFX_BADSCANLINES( )	Number of bad scan lines transferred per page
ATFX_SPEED( )	Baud rate of the data transfer
ATFX_STATE( )	State of the fax channel device
ATFX_TRCOUNT( )	Number of bytes transferred

5.7.4. Enable Operator Intervention (Voice Request)

This feature is not supported on DM3 boards.

You can enable your application to send or receive an operator intervention (voice request) from a remote station.

DF_ACCEPT_VRQ in the **sndflag** argument enables the application to accept an operator intervention request from the remote station.

DF_ISSUE_VRQ in the **sndflag** argument enables the application to send an operator intervention request (DFS_PRI_EOP) to the remote station after the last fax page of the **fx_sendfax( )** fax session is transmitted.

If **fx_sendfax( )** completes successfully, the function returns a 0 in synchronous mode or a TFX_FAXSEND event occurs in asynchronous mode. The fax session is completed, but the connection between the two stations is still active for voice communication.

To determine the reason for termination of **fx_sendfax( ),** call **ATFX_TERMMSK( )**. This function returns a TM_FXTERM bitmap value indicating normal completion of the function or TM_VOICEREQ indicating completion due to a voice request issued or received.

5.7.5. Select Resolution for Fax Transmission

When the **sndflag** argument specifies a resolution (DF_TXRESLO or DF_TXRESHI), all fax data associated with the **fx_sendfax( )** call is transmitted at this resolution regardless of any resolutions previously specified.

The DF_IOTT entries for the fax may contain an arbitrary combination of raw, TIFF/F and ASCII files at different resolutions, specified in the **io_resln** field for raw and ASCII files. By setting the DF_TXRESHI or DF_TXRESLO bit in **sndflag**, the entire chain of DF_IOTT entries is sent at the specified resolution.

For example, if the DF_IOTT array specifies a TIFF/F file at high resolution followed by a raw file at low resolution, after the transmission of the TIFF/F file, Phase B would be entered to negotiate the change in resolution for the raw file. By setting **sndflag** to DF_TXRESLO, the entire fax session takes place at low

resolution. In this case, the high resolution data in the TIFF/F file is internally converted to low resolution at the time of transmission.

By default, the resolution in the TIFF/F file or in the DF_IOTT entry is used in the fax transmission. The **sndflag** argument overrides this resolution.

5.7.6. Enable Subaddress Fax Routing

This feature is not supported on DM3 boards.

As described in *Section 4.10. Subaddress Fax Routing* on *page 46*, a subaddress is a T.30 message protocol that allows a fax to be routed to one or more telephone numbers (or extensions) after it is received by the fax machine or server.

The T.30 subaddress message is a 20-character string containing a combination of one or more phone numbers and/or extensions sent during Phase B negotiation.

To enable subaddress fax routing during fax transmission:

- Set the FC_ENDDOC parameter in **fx_setparm( )** to DFS_REMOTESUBADDR.
- Set the FC_SENDCONT parameter in **fx_setparm( )** to DFC_AUTO.
- Set the DF_TXSUBADDR bit in the **sndflag** parameter of **fx_sendfax( )**.

When subaddress fax routing is enabled, the fax library issues an MPS (DFC_MPS) message between each page of a multiple page TIFF/F file. The width and resolution of the fax transmission is set by the first TIFF/F page and remains the same for the entire fax transmission regardless of changes in width or resolution.

To return Phase B status after a TFX_PHASEB event, use the **ATFX_BSTAT( )** extended attribute function. To return the setting of the fax channel for subaddress fax routing, use the FC_REMOTESUBADDR parameter (Windows only) in **fx_getparm( )**.

Sending Fax to a Single Subaddress

To send subaddress fax routing information with a fax transmission to a single subaddress, use the following procedure:

1. Initialize a table of DF_IOTT entries for fax data to be sent to the subaddress.

   For the last entry in a table of DF_IOTT entries, set **io_phdcont** to DFC_EOP (End of Procedure) and **io_type** to IO_EOT.

2. Set the following fax parameters using **fx_setparm( )**:

   - Set the FC_TXSUBADDR parameter to the desired subaddress: phone number and/or extension.
   - If a multi-page fax is to be sent to the subaddress, set the FC_SENDCONT parameter value to DFC_AUTO.

     DFC_AUTO automatically sets the FC_SENDCONT parameter value to DFC_MPS, allowing all pages of the multi-page fax to be transmitted to the specified subaddress in the least amount of time.

3. Set the initial state of the fax channel to transmitter using **fx_initstat( )**.
4. Dial the number of the receiving fax machine/server. See the *Voice API Library Reference* for functions to use for dialing.
5. Call **fx_sendfax( )** with the DF_TXSUBADDR bit set in the **sndflag** parameter.

Sending Fax to Multiple Subaddresses

To send subaddress fax routing information with a fax transmission to multiple subaddresses, use the following procedure:

1. Initialize a table of DF_IOTT entries for fax data to be sent to the subaddress.

   The specified fax data is sent to the first subaddress. For the last entry in a table of DF_IOTT entries, set **io_phdcont** to DFC_EOM (End of Message) and **io_type** to IO_EOT (last DF_IOTT entry). The DFC_EOM causes renegotiation of Phase B and indicates to the receiver that more fax data will be transferred.

2. Set the following fax parameters using **fx_setparm( )**:

   - Set the FC_TXSUBADDR parameter to the desired subaddress: phone number(s) and/or extension(s).
   - If a multi-page fax is to be sent to the subaddress, set the FC_SENDCONT parameter value to DFC_AUTO.

     DFC_AUTO automatically sets the FC_SENDCONT parameter value to DFC_MPS, allowing all pages of the multi-page fax to be transmitted to the specified subaddress in the least amount of time.

3. Set the initial state of the fax channel to transmitter using **fx_initstat( )**.
4. Dial the number of the receiving fax machine/server. See the *Voice API Library Reference* for information on dialing.
5. Call **fx_sendfax( )** with the DF_TXSUBADDR bit set in the **sndflag** parameter. After **fx_sendfax( )** completes, the fax session is still active.
6. Call **fx_setparm( )** to update the FC_TXSUBADDR parameter with the new subaddress.
7. Initialize a DF_IOTT table for fax data for the new subaddress.

   NOTE: To route more than two DF_IOTT tables of fax data to different subaddresses, set **io_phdcont** for each entry in the table to DFC_EOM (End of Message) as in Step 1 above except for the final entry in the last DF_IOTT table. Set **io_phdcont** for the final entry to DFC_EOP and **io_type** to IO_EOT to end the fax session.

8. Call **fx_sendfax( )** with the DF_TXSUBADDR bit set in the **sndflag** parameter.
9. Repeat steps 2 through 8 until all fax data has been sent to all specified subaddresses during the same fax session.

5.8. Issuing a Send Fax Function

After defining fax data in one or more DF_IOTT structures and following other recommended steps as outlined in *Section 5.2. Guidelines for Implementing Fax* (*page 48*), you can issue the **fx_sendfax( )** function in your application.

The **fx_sendfax( )** function transmits fax data as specified by a table of DF_IOTT entries. This function can be issued by the caller application or the called application. The called application issues **fx_sendfax( )** in a polling fax transmission or to block incoming faxes. See *Section 4.4. Types of Fax Transmission* on *page 35* for more information on the types of fax transmission and application flow.

When the initial fax connection is made, the caller station is initially set as the fax transmitter and the called station as the fax receiver.

5.8.1. Send Fax Issued by the Transmitter

The transmitter (which can be the caller or the called application depending on the type of fax transmission) issues **fx_sendfax( )** to send fax data to the receiver as specified by DF_IOTT entries. Any receiver incompatibility disconnects the call and **ATDV_LASTERR( )** returns an error code of EFX_DISCONNECT.

In synchronous mode, the function returns a -1 to indicate that an error has occurred. In asynchronous mode, a TFX_FAXERROR event is generated.

In a **turnaround polling fax transmission**, set the **io_phdcont** field to DFC_EOM for the last DF_IOTT entry before issuing the initial **fx_sendfax( )** function. For more information, see *Sections 4.4.3. Turnaround Polling Fax Transmission* (*page 40*) and *5.5.9. Setting Phase D Continuation Values* (*page 59*).

5.8.2. Send Fax Issued by the Called Application

A called application issues an initial **fx_sendfax( )** function to the caller application to indicate that the called application is only capable of transmitting a fax.

- If the caller is capable of receiving or wishes to receive a fax transmission, the caller issues an **fx_rcvfax( )** or **fx_rcvfax2( )** function to receive the fax data.
- If the caller is not capable of receiving or does not wish to receive a fax transmission, the called application disconnects the fax call.

The called application can also issue **fx_sendfax( )** in a polling fax transmission. For more information, see *Sections 4.4.2. Polling Fax Transmission* (*page 36*) and *4.4.3. Turnaround Polling Fax Transmission* (*page 40*).

5.8.3. Status of Fax Transmission

Status information on the fax transmission is available using the fax extended attribute functions.

- During fax transmission, the state of the channel device is set to CS_SENDFAX. To obtain the current state of the channel device, issue **ATFX_STATE( )**.
- If the function successfully completes, the final Phase D status is available using **ATFX_PHDCMD( )** and **ATFX_PHDRPY( ).** For more information on Phase D status values, see *Appendix B.*
- To obtain a pointer to the last DF_IOTT entry that was processed, issue **ATFX_LASTIOTT( )**.

5.9. Stopping a Fax Transmission or Reception

At any time, you can stop a fax transmission or reception in progress by issuing the **fx_stopch( )** function.

5.10. Replacing Bad Scan Lines

Before stored MH or MR encoded fax data is transmitted, the fax library checks the integrity of every scan line in the data stream. Scan lines that do not have the correct pixel count are replaced. (If the image was stored without scan line errors, the data being transmitted should have no bad scan lines.)

When a scan line error is detected, Bad Line Replacement (BLR) automatically replaces the bad scan line(s) with the last scan line that had the correct pixel count.

To determine the number of bad scan lines detected and replaced on the transmitted, call **ATFX_BADSCANLINES( )**.

MMR encoded data always requires the use of T.30 Error Correction Mode (ECM). The capabilities of the receiving station determine if ECM is used.

5.11. Creating User-Defined I/O Functions

In your fax application, you may want to replace standard I/O functions **lseek( )**, **read( )** and **write( )** with your own I/O functions.

To register user-defined I/O functions, set up a DF_UIO structure with pointers to the application's own seek, read and write functions. Call **fx_setuio( )** to register the functions with the fax library. The fax library stores the pointers to the user-defined seek, read and write functions and calls them with the same arguments as it would call the standard I/O seek and read functions.

For user-defined I/O functions to access fax data, set the IO_UIO bit in the **io_type** field of the DF_IOTT structure. The **io_fhandle** field of the DF_IOTT structure specifies the file descriptor passed to the functions.

NOTE: User-defined I/O functions are called only for the DF_IOTT table entries with IO_UIO bit set in **io_type**. For all other DF_IOTT table entries, the standard I/O functions are used.

For more information on the DF_UIO structure, see *Section* 10.8. DF_UIO – User-Defined I/O on *page 134*.

6. Implementing Receive Fax Capability

6.1. Overview

This chapter provides guidelines on how to use the fax library to implement **receive** fax capability in an application. Fax library functions and data structures used in completing a task are included in this chapter. For complete reference information on functions and data structures, see *Chapters 10. Fax Data Structures*, *11. Using the Fax Library* and *12. Fax Library Function Reference.*

The following topics are covered in this chapter:

- *6.2. Setting Parameters for Receive Fax*
- *6.3. Setting the Bit Mask for a Receive Fax Function*
- *6.4. Issuing a Receive Fax Function*
- *6.5. Creating User-Defined I/O Functions*

Support for the features described varies by product. For a listing of key features by product, see *Section 2.3. Key Product Features* on *page 8*.

6.2. Setting Parameters for Receive Fax

The fax parameters described in this section are specified using **fx_setparm( )** (see *page 322*).

6.2.1. Specifying Encoding Scheme to Store Incoming Fax Data

The incoming fax data may be stored in one of the following encoding schemes based on the capability of the receiving station: MH or MMR.

On most Intel® products, the encoding scheme for storing incoming fax data is determined by the FC_RXCODING parameter in the **fx_setparm( )** function. For details on product support, see *Section 2.3. Key Product Features* on *page 8*.

- When incoming fax data is stored in TIFF/F files, the encoding scheme specified in the FC_RXCODING parameter is included in the TIFF/F tags embedded in the stored file. For TIFF/F tag details, see *Appendix A*.
- When incoming fax data is stored in raw image files, the application must keep track of the encoding scheme by referring to the value specified in the FC_RXCODING parameter.

The fax software automatically converts the incoming fax data to the encoding scheme specified in the FC_RXCODING parameter regardless of the encoding scheme negotiated during Phase B of the fax transfer.

To determine the negotiated line encoding scheme, call **ATFX_CODING( )** after the negotiation of Phase B has completed.

6.2.2. Storing Incoming Fax Data

Incoming fax data is delimited by Phase D command values sent from the transmitter to the receiver. The transmitter application sets these values using the FC_ENDDOC parameter in **fx_setparm( )**.

Storing in a Single TIFF/F File

For most applications, a multi-page fax document is stored in a single TIFF/F file.

The DFS_EOP value indicates to the receiver that all incoming pages will be stored in a single, multi-page TIFF/F file. This is the default setting.

For information on specifying the file storage format (TIFF/F or raw), see *Section 6.3.1. File Format for Incoming Fax Data* on *page 87*.

Storing in Multiple TIFF/F Files

An application can store each page (or group of pages) of a multi-page fax in a separate TIFF/F file.

The DFS_MPS and/or DFS_EOM value tells the receiver to store individual pages of a multi-page fax in separate TIFF/F files. When you execute **fx_rcvfax( )** or **fx_rcvfax2( )** in a loop delimited by a DFS_MPS or DFS_EOM (and the default, DFS_EOP), the application will specify a different file each time the receive fax function is issued.

Incoming fax data stored as TIFF/F files can be delimited by the following Phase D command values, sent from the transmitter to the receiver:

- DFS_EOP (End of Procedure) - default. Indicates to the receiver that the fax procedure has completed. This setting stores all incoming pages into a single multi-page TIFF/F file. The fax phone line is still active after a DFS_EOP. To terminate the call, set the voice channel on-hook.
- DFS_MPS (Multi-Page Signal). Indicates to the receiver that there is more fax data to follow, and the next page is in the same format as the page just received. The application proceeds to T.30 Phase C.
- DFS_EOM (End of Message). Indicates to the receiver that there is more fax data to follow. The application returns to T.30 Phase B and negotiates parameters for the next page.

By default, incoming fax data is delimited by the reception of DFS_EOP from the transmitter. For example, if the FC_ENDDOC parameter is set to DFS_EOM, the incoming fax is delimited on DFS_EOM as well as the default value of DFS_EOP.

When the receiver station receives DFS_EOM or DFS_EOP from the transmitter, **fx_rcvfax( )** completes, returns a 0 (in synchronous mode) or a TFX_FAXRECV event is generated (in asynchronous mode), and control is returned to the application.

The application must then check the Phase D command using **ATFX_PHDCMD( )** to determine if more fax pages will follow or if the last page of the fax has been stored (DFS_EOP). If there are more fax pages to follow, the application must issue another **fx_rcvfax( )** function specifying a different file for storage. When the last page of the fax has been stored, a DFS_EOP value is present indicating the end of fax reception.

Storing in a Raw File

Incoming fax data stored in a raw file contains unstructured fax data that does not conform to TIFF/F or other formats.

Incoming fax data stored in a raw file in MH or MMR encoding is written with a fill order of LSB (Least Significant Bit) first.

- For MH: EOL (end of line) sequences are not byte aligned.
- For MMR: No EOL (end of line) sequences, byte alignment or zero fill.

Storage in raw files is automatically delimited by all of the following Phase D status command values sent from transmitter to receiver. This means that the receive function returns control to the application on all FC_ENDDOC values. The transmitter application sets these values using the FC_ENDDOC parameter in **fx_setparm( )**.

- DFS_EOM (End of Message)
- DFS_EOP (End of Procedure)
- DFS_MPS (Multi-Page Signal)

At the end of every fax page received for storage in a raw file, the **fx_rcvfax( )** function completes, returns a 0 (in synchronous mode) or a TFX_FAXRECV event occurs (in asynchronous mode). The application must then check the Phase D command using **ATFX_PHDCMD( )** to determine if more fax pages will follow (DFS_EOM or DFS_MPS) or if the last page of the fax has been stored (DFS_EOP). If there is another page of fax data to follow, the application must issue another **fx_rcvfax( )** function for that page specifying a different raw file for storage. When the last page of the fax has been stored, a DFS_EOP Phase D value is present indicating the end of fax reception.

6.2.3. Setting Acceptable Percentage of Bad Scan Lines

You can specify the percentage of bad scan lines acceptable during a fax page reception before an RTP (Retrain Positive) and an RTN (Retrain Negative) message are sent to the transmitter at the completion of the fax page. To set this percentage, use the fax channel parameters FC_RTP and FC_RTN in **fx_setparm( )**.

To determine the number of pages received that required an RTN (Retrain Negative) to be returned to the remote station, call the fax extended attribute **ATFX_RTNPAGES( )** (this function not supported on DM3 boards).

6.2.4. Selecting Preferred Maximum Receive Baud Rate

You can specify the maximum preferred baud rate for fax data reception.

To receive fax transmissions at a lower baud rate than the default, set the FC_RXBAUDRATE parameter in **fx_setparm( )** to one of the supported baud rates. The default baud rates are the highest supported rates for a product (see *Section 2.3. Key Product Features* on *page 8*).

This parameter is useful when receiving fax transmissions over known noisy lines. By setting a lower baud rate than the default, no time is wasted in negotiating baud rates.

6.2.5. Replacing Bad Scan Lines

Noise on the telephone line can cause scan line errors. During the reception of MH encoded fax data, the integrity of every scan line in the data stream is checked. When a scan line error is detected, Bad Line Replacement (BLR) automatically replaces the bad scan line(s) with the last correctly received scan line.

MMR encoded data uses T.30 Error Correction Mode (ECM) to ensure error-free transfer. The fax library verifies the integrity of the MMR encoded data before storage. The capabilities of the remote station determine if ECM is used.

To return the number of bad scan lines detected and replaced on a received page, call the fax extended attribute **ATFX_BADSCANLINES( )**.

If the number of bad scan lines received per page is higher than the percentage of bad scan lines you will accept, the fax channel may request retraining before receiving the next page or retransmission of the current page (see the FC_RTN and FC_RTP parameters in **fx_setparm( )**).

NOTE: When fax data is stored in a TIFF/F file, the bad scan line count is written to the BadFaxLines TIFF/F tag. To verify that the bad scan lines were replaced before storage, the CleanFaxData TIFF/F tag is set to zero.

6.2.6. Routing Fax Data to Multiple Subaddresses

This feature is not supported on DM3 boards.

When the T.30 subaddress message is received from the transmitter during Phase B negotiation, the fax data following the negotiation can be routed to the subaddress specified.

One model for implementing the subaddress feature is to store the subaddresses in a file and the incoming fax data in separate files, one file containing the fax data for each subaddress. After receiving fax data for all subaddresses and the fax session is complete, the application can distribute the fax data as specified in each subaddress.

Set the application to receive fax data containing subaddress information as follows:

1. Set the FC_ENDDOC fax parameter to DFS_REMOTESUBADDR in **fx_setparm( )** to keep fax pages destined for different subaddresses in separate files. When this bit is set and a T.30 subaddress message is received from the transmitter, control is returned to the application after **fx_rcvfax( )** or **fx_rcvfax2( )** receives the fax data specified for the subaddress sent during the last Phase B negotiation.

   NOTE: When the FC_ENDDOC parameter is set with the DFS_REMOTESUBADDR flag and an EOM is received by the remote fax machine, the application may receive both a TFX_PHASED and a TFX_FAXRECV event for the same page. Under normal circumstances the last TFX_PHASED is replaced by the TFX_FAXRECV, but in this case the **fx_rcvfax( )** or **fx_rcvfax2( )** function will not know that it needs to return to the application until after it has already completed Phase D.

2. Set the DF_PHASEB **rcvflag** bit in **fx_rcvfax( )** or **fx_rcvfax2( ).** This bit enables the generation of Phase B events.

3. After a Phase B event is generated, call the **ATFX_BSTAT( )** function to see if the DFS_REMOTESUBADDR bit flag is set. If this bit is set, the incoming fax data contains valid subaddress information.

4. Store the subaddress information contained in the **fx_getparm( )** FC_REMOTESUBADDR fax parameter (Windows only) for later use when directing the fax data to the phone number(s) and/or extension(s) specified in the subaddress message.

5. When **fx_rcvfax( )** or **fx_rcvfax2( )** completes, check the Phase D command from the sender using **ATFX_PHDCMD( )**. If the Phase D command is not DFS_EOP (End Of Procedure), call **fx_rcvfax( )** or **fx_rcvfax2( )** again to receive more fax data from the transmitter.

The application must call **fx_rcvfax( )** immediately after subaddress information is received or an EFX_NXTCMDRX error may be generated indicating a time out while waiting for the next **fx_rcvfax( )** call.

6. Continue to check for T.30 SUB messages during Phase B negotiation and collect the subaddress information for each subaddress during the fax session for later routing. When a DFS_EOP is returned by **ATFX_PHDCMD( )** in Step 5, no additional fax data is sent for the fax session.

6.3. Setting the Bit Mask for a Receive Fax Function

The **rcvflag** parameter of the **fx_rcvfax( )** and **fx_rcvfax2( )** functions is a logical OR bit mask that can be set to indicate one or more conditions. For more information, see the following:

- *6.3.1. File Format for Incoming Fax Data*
- *6.3.2. Mode of Operation*
- *6.3.3. Enable Phase B Event Generation*
- *6.3.4. Enable Phase D Event Generation*
- *6.3.5. Enable Operator Intervention (Voice Request)*
- *6.3.6. Selectable Receive Width*
- *6.3.7. Selectable Receive Length*
- *6.3.8. Resolution for Storing Incoming Fax Data*

6.3.1. File Format for Incoming Fax Data

Once received, fax data can be stored in one of the following formats:

- raw, unstructured file – **rcvflag** bit mask set to DF_RAW
- TIFF/F structured, formatted file – **rcvflag** bit mask set to DF_TIFF

For information on delimiters for multi-page fax documents, see *Section 6.2.2. Storing Incoming Fax Data* on *page 81*.

6.3.2. Mode of Operation

Two modes of operation are available for the receive fax functions:

- synchronous mode – **rcvflag** bit mask set to EV_SYNC
- asynchronous mode – **rcvflag** bit mask set to EV_ASYNC

In synchronous mode (EV_SYNC), the function does not return control to the application until **fx_rcvfax( )** completes or an error has occurred.

In asynchronous mode (EV_ASYNC), the function returns control to the application immediately after it is invoked. The **fx_rcvfax( )** function returns a zero to indicate successful invocation and a -1 to indicate an invocation error. If successfully invoked, the function returns control to the application and the specified device continues to receive fax data. Once control is returned to the application, the application may issue voice/fax calls on other devices. The completion (or error termination) of **fx_rcvfax( )** is indicated through events generated by the Standard Runtime Library.

The following events are valid for **fx_rcvfax( )**:

Event	Description
TFX_FAXERROR	Error in processing
TFX_FAXRECV	Successful completion of **fx_rcvfax**( )

See the *Standard Runtime Library API Library Reference* for event handling information.

6.3.3. Enable Phase B Event Generation

To enable Phase B event generation, specify DF_PHASEB in **rcvflag** of the receive fax function.

When this bit is set, a TFX_PHASEB event is returned each time Phase B is completed during the receive fax operation.

When a TFX_PHASEB event occurs, the application can call these fax extended attributes for the following information:

Fax Extended Attribute	Indicates
ATFX_BSTAT( )	Phase B information available
ATFX_CODING( )	Negotiated line encoding scheme for the data transfer
ATFX_SPEED( )	Baud rate of the data transfer
ATFX_STATE( )	State of the fax channel device

If you issue **fx_rcvfax( )** or **fx_rcvfax2( )** in synchronous mode (EV_SYNC), you must install an event handler to handle Phase B events using **sr_enbhdlr( )** of the Standard Runtime Library. See the *Standard Runtime Library API Library Reference* for event handler details.

6.3.4. Enable Phase D Event Generation

To enable Phase D event generation, specify DF_PHASED in **rcvflag** of the receive function.

When this bit is set, a TFX_PHASED event is returned each time Phase D is completed during the receive fax operation, except for the last page. After the last page, **fx_rcvfax( )** completes (synchronous mode) or a TFX_FAXRECV event occurs (asynchronous mode).

Phase D events allow the application to monitor the progress of the fax session on a page-by-page basis.

When a TFX_PHASED or TFX_FAXRECV event occurs, the application can call these fax extended attributes for the following information:

Fax Extended Attribute	Returns
ATFX_BADSCANLINES( )	Number of bad scan lines transferred (per page)
ATFX_PHDCMD( )	Phase D command
ATFX_PHDRPY( )	Phase D reply
ATFX_RESLN( )	Resolution of the page
ATFX_SCANLINES( )	Total number of scan lines transferred
ATFX_SPEED( )	Baud rate of the data transfer
ATFX_STATE( )	State of the fax channel device
ATFX_TRCOUNT( )	Number of bytes transferred
ATFX_WIDTH( )	Width of the page

NOTES:

1. When you enable Phase D events, the generation of the Phase D event is skipped for the **last** fax page received of the **fx_rcvfax( )** call since a TFX_FAXRECV event is generated indicating the successful completion of the **fx_rcvfax( )** function.
2. If you issue the **fx_rcvfax( )** function in synchronous mode (EV_SYNC), an event handler must be installed to handle Phase D

events using the **sr_enbhdlr( )** function of the Standard Runtime Library.

3. If the RTN message is returned to the TRANSMITTER, the generation of the Phase D event would occur on that page each time Phase D event generation is enabled. In this case, Phase D event would not be skipped for the **last** page of the **fx_sendfax( )** call if the RTN message is returned from the RECEIVER on the **last** page.
4. If the RTN message is returned to the TRANSMITTER, either TFX_FAXSEND or TFX_FAXERROR would be generated. If none of pages are sent successfully, TFX_FAXERROR event is generated indicating fax failure. If one of pages is sent successfully, TFX_FAXSEND event is generated indicating the successful completion of the **fx_sendfax( )** function.

6.3.5. Enable Operator Intervention (Voice Request)

DF_ACCEPT_VRQ in the **rcvflag** argument enables the application to accept an operator intervention request from the remote station. This feature is not supported on DM3 boards.

DF_ISSUE_VRQ in the **rcvflag** argument enables the application to send an operator intervention request (PIN/PIP) to the remote station after the **last** fax page of the receive fax operation is received. This feature is not supported on DM3 boards.

To indicate successful completion, **fx_rcvfax( )** returns a 0 in synchronous mode or a TFX_FAXRECV event occurs in asynchronous mode. The fax session is completed, but the connection between the two stations is still active for voice communication.

To determine the reason for termination of **fx_rcvfax( )**, call the **ATFX_TERMMSK( )**. This function returns a TM_FXTERM bit value to indicate normal completion or TM_VOICEREQ to indicate termination due to a voice request issued or received.

6.3.6. Selectable Receive Width

Set the **rcvflag** bit mask to restrict the maximum width (in number of pixels) at which the application receives fax data. Possible values are:

DF_1728MAX
DF_2048MAX
DF_2432MAX (default)

The DF_2432MAX value (default) allows reception of fax data at a width of 1728, 2048 or 2432 pixels. The transmitter is notified of the maximum width of the transmitted page during negotiation of Phase B. It is up to the transmitting fax machine to scale large pages to the maximum receive width specified.

T.30 protocol specifies that a receiver in a fax session be able to receive fax data at the following width combinations:

1728 pixels only
1728 and 2048 pixels
1728, 2048 and 2432 pixels (default)

6.3.7. Selectable Receive Length

Set the **rcvflag** bit mask to indicate the preferred page length for receiving fax data.

DF_A4MAXLEN (approximately 11 inches)
DF_B4MAXLEN (approximately 14 inches)
DF_NOMAXLEN (unlimited) (default)

You can override the default setting via parameter initialization during installation.

The transmitter is notified of the receiver's preferred page length during negotiation of Phase B. The receiving fax channel does not actually paginate the incoming image to the specified page length; the transmitter must send the image so that the maximum specified length is not exceeded.

6.3.8. Resolution for Storing Incoming Fax Data

Two grades of vertical resolution can be specified using the **rcvflag** argument:

- high or fine resolution – DF_RXRESHI
- low or coarse resolution – DF_RXRESLO

When the **rcvflag** argument specifies a resolution (high or low), all fax data associated with **fx_rcvfax( )** or **fx_rcvfax2( )** is stored at this resolution regardless of the resolution specified by the transmitter. These bit flags can be used to reduce storage requirements or to support third-party utility programs that can only handle certain resolutions.

6.4. Issuing a Receive Fax Function

When the initial fax connection is made between the caller and the called applications, the caller application is initially set to be the fax transmitter and the called application is initially set to be the fax receiver, as in a normal fax transmission.

6.4.1. Receive Fax Issued by the Receiver

When the receiver issues **fx_rcvfax( )** and the transmitter does not poll, the call progresses as in a normal fax transmission; that is, the transmitter sends fax data to the receiver.

If the transmitter polls, the poll bit in the **fx_rcvfax( )** function issued by the receiver indicates whether a poll by the transmitter is valid.

- **Polling Invalid.** If the poll bit is set to DF_NOPOLL, the receiver application remains the receiver of the fax data. A poll by the transmitter is invalid; the **fx_rcvfax( )** function fails and the fax session is terminated. In this case, **ATDV_LASTERR( )** returns an EFX_DISCONNECT error.

  If the receiver's poll bit is set to DF_NOPOLL, the transmitter can only send fax data to the receiver, as in a normal fax transmission.

- **Polling Valid.** If the poll bit is set to DF_POLL and the transmitter application polls, the receiver's **fx_rcvfax( )** function returns a 0 in

synchronous mode or a TFX_FAXRECV event occurs in asynchronous mode. The receiver application calls the fax extended attribute **ATFX_TERMMSK( )** and determines that a poll has occurred (TM_POLLED). The receiver application can now become the fax transmitter.

The fax session remains active and the original receiver must immediately issue a *send* function to complete the poll and become the new fax transmitter.

If the transmitter does not poll, the receiver's **fx_rcvfax( )** function completes as it would for a normal fax transmission.

6.4.2. Receive Fax Issued by the Transmitter

When the transmitter issues the **fx_rcvfax( )** function, this indicates a poll request; that is, the transmitter application wishes to be a fax receiver.

- **Polling Valid.** If the poll bit is set to DF_POLL on the receiver side, polling is valid. The initial receiver and transmitter switch roles. The fax is sent to the new fax receiver.
- **Polling Invalid.** If the poll bit is set to DF_NOPOLL on the receiver side, polling is invalid. The **fx_rcvfax( )** function in the transmitter application fails and the fax session is terminated. In this case, **ATDV_LASTERR( )** returns a EFX_NOPOLL error.

6.4.3. Status of Fax Reception

Status information on the fax reception is available using the fax extended attribute functions.

- During fax reception, the state of the channel device is set to CS_RECVFAX. To obtain the current state of the channel device, call **ATFX_STATE( )**.
- To obtain the final transfer Phase D status, call **ATFX_PHDCMD( )** and **ATFX_PHDRPY( )**. For more information on Phase D status values, see *Appendix B*.

6.5. Creating User-Defined I/O Functions

In your fax application, you may want to replace the use of the standard I/O functions **lseek( )**, **read( )** and **write( )** with your own I/O functions.

To receive fax data using user-defined I/O functions, you must "OR" the **rcvflag** argument of the **fx_rcvfax2( )** function with the IO_UIO bit. The fax library calls the user-defined seek and write functions every time fax data is written to the I/O device. The **fd** argument in **fx_rcvfax2( )** specifies the file descriptor passed to the I/O functions.

For information on the DF_UIO structure, see *Section 10.8. DF_UIO – User-Defined I/O* on *page 134*.

7. Specifying Fonts in ASCII to Fax Conversion

7.1. Overview

This chapter discusses the use of fonts on Windows operating systems when an ASCII text file is converted to fax format and transmitted. The following topics are covered:

- *7.2. Fonts Supported in ASCII to Fax Conversion*
- *7.3. Using fx_setparm( ) and fx_getparm( ) to Select Fonts*
- *7.4. Overriding Fonts Set with fx_setparm( )*
- *7.5. Preserving Proprietary Fonts as Default Fonts*

The information in this chapter does not apply to DM3 boards. For ASCII to fax information applicable to DM3 boards, see *Section 10.3.3. DF_ASCIIDATA Usage Rules*.

7.2. Fonts Supported in ASCII to Fax Conversion

DSP Fax uses Windows font handles in ASCII to fax conversion rather than proprietary fonts from Intel. (These proprietary fonts -- see ASCII to Fax tables in *Appendix F* -- were used on older generation fax boards.) By using Windows font handles, you can choose from the entire selection of fonts available in Windows or you can supply your own font resources.

For details on how to create font handles and font resources, see your Software Development Kit documentation. For information on how to enable proprietary fonts as default fonts in ASCII to fax conversion in DSP Fax, see *Section 7.5. Preserving Proprietary Fonts as Default Fonts* on *page 100*.

On DSP Fax, the conversion of ASCII text to fax format is performed on the host CPU rather than on the fax board. The following font features are available on boards that support DSP Fax:

- two fonts available per channel, reset to the default for each **fx_open( )**
- two fonts active for each **fx_sendfax( )**
- no limitation on language or character set
- no limitation on point sizes

For ASCII to fax information applicable to DM3 boards, see *10.3.3. DF_ASCIIDATA Usage Rules*.

7.3. Using fx_setparm( ) and fx_getparm( ) to Select Fonts

Use the following parameter IDs with **fx_setparm( )** and **fx_getparm( )** to select or return fonts: FC_FONT0 and FC_FONT3.

The parameter IDs are defined as follows:

- FC_FONT0 – defaults to OEM_FIXED_FONT 12-point. The font specified by FC_FONT0 is applied by default to the fax document.
- FC_FONT3 – defaults to OEM_FIXED_FONT 9-point. FC_FONT3 defines a second font to be available for use. It also sets the font for the header.

If you use DSP Fax out of the box, your ASCII to fax document (without any special control characters within the document itself) is rendered in a default 12-point font similar to the Windows standard OEM_FIXED_FONT. Headers are rendered in a default 9-point font.

You can select two fonts for each fax channel device by specifying font handles using **fx_setparm( )** and FC_FONT0 and FC_FONT3 as parameter IDs.

Each parameter ID initializes a font handle to make the specified font available for use in rendering an ASCII document. Two font handles can be active at one time on a fax channel device; they are stored in the fax library. The font handle must remain open for the duration of the fax transmission; that is, the font handle must not be deleted until the fax transmission has completed. You must delete the handle when it is no longer needed.

You can replace the default values with any other Windows font or your own font resource. For details on how to create font handles and font resources, see your Software Development Kit documentation.

To override the default font or specify the use of a different font, see *Section 7.4. Overriding Fonts Set with fx_setparm( )* on *page 99* for more information.

If you don't specify FC_FONT0 or FC_FONT3, your ASCII document will use the default font previously available.

For more information on **fx_getparm( )** and **fx_setparm( )**, see *Chapter 12. Fax Library Function Reference*.

7.4. Overriding Fonts Set with fx_setparm( )

As described in *Section 7.3. Using fx_setparm( ) and fx_getparm( ) to Select Fonts*, fonts used in ASCII to fax conversion are selected for each fax device channel using **fx_setparm( )**. The following methods override the default font specified in **fx_setparm( )**. These methods are optional and are described in order of precedence.

- using the **font** field of the DF_ASCIIDATA data structure
- using control characters in the ASCII document prior to sending

7.4.1. Specify a Font in DF_ASCIIDATA

The **font** field in the DF_ASCIIDATA data structure specifies the font in use for a specific ASCII document associated with a specific DF_IOTT structure. Valid values are DF_FONT_0 and DF_FONT_3.

This font overrides FC_FONT0 and becomes the default font for the current fax transmission. The subsequent fax transmission reverts to using the font specified in FC_FONT0 as the default font.

For example, if you want a specific ASCII document (associated with a specific DF_IOTT structure) to use FC_FONT3 as the default font, then specify DF_FONT_3 as the value in the **font** field.

For further information on DF_ASCIIDATA, see *Section 10.3. DF_ASCIIDATA – ASCII Data Description* on *page 119*.

7.4.2. Use Control Characters in ASCII Document Prior to Sending

To apply a second font in your fax document in addition to the default font, you must edit your ASCII document and insert the proper control characters before the line on which the change will take place. The new format takes effect on the next full line of text. These control characters override the font set in the DF_ASCIIDATA data structure. This method may be useful when applying a font to specific parts of your document (such as headings) rather than to the entire document.

For example, if you want to use the font specified by FC_FONT3, insert <ESC>F3 or ~F3 in your ASCII document before the line on which the change will take place. The new format takes effect on the next full line of text. To return to the default font, insert <ESC>F0 or ~F0.

For further information on control characters, see the ASCII to Fax tables in *Appendix F*.

7.5. Preserving Proprietary Fonts as Default Fonts

In VFX products, ASCII to fax conversion is performed in the firmware and ASCII documents are rendered using proprietary fonts from Intel (see ASCII to Fax tables in *Appendix F*).

The use of Windows font handles in DSP Fax enables you to specify any Windows font or to supply your own font resources. The default fonts provided by DSP Fax are Windows fonts.

You can enable proprietary fonts as default fonts in ASCII to fax conversion in DSP Fax by following the directions in this section.

7.5.1. Location of Proprietary Fonts

The proprietary font files are provided with the system software and installed by default in the …*\dialogic\fonts* subdirectory. To enable applications to use these fonts, you must package and re-distribute these font files with their applications.

7.5.2. Steps to Enable Proprietary Fonts

To use the proprietary fonts for rendering ASCII to fax documents, modify your application as follows:

1. Call the Win32 API **AddFontResource**(**font_filename**) to notify the operating system of the presence of new fonts, namely, the proprietary fonts. You will also need to notify other applications of the new fonts. For more information, see your Software Development Kit documentation.
2. Establish an LFONT structure with the appropriate parameter to prepare for the creation of a new font.
3. Call the Win32 API **CreateFontIndirect( )** to obtain a font handle.
4. Use the **fx_setparm( )** function call to store this font handle in the fax library.
5. Call the Win32 API **RemoveFontResource( )** to remove the font. You will also need to notify other applications of the removal.

For more information on the Win32 API, see the *Microsoft Win32 API Programmer's Reference*.

Example

The following example shows one way to specify the use of proprietary fonts (Japanese Katakana character set) in rendering ASCII to fax documents.

To see an example of how to specify the use of a Windows font, see the example for **fx_setparm( )** and FC_FONT0 on *page 349*.

```
// …
// open device using fx_open()
// …
// The next line of code is required when your application needs to use old
// Dialogic fonts.
```

```
// It specifies the font-resource filename. Assume in same directory as
// application.

LOGFONT lFont;
HFONT hFont;

ret = AddFontResource("katakna0.fon");

memset(&lFont, 0, sizeof(lFont));
lFont.lfCharSet = OEM_CHARSET;
strcpy(lFont.lfFaceName, "Katakna0");

hFont = CreateFontIndirect(&lFont);

if ((fx_setparm ( dev, FC_FONT_0, (void *)&hFont)) == -1)
{
printf("LastError: %d, ErrorMsg = %s\n", ATDV_LASTERR ( dev ),
         ATDV_ERRMSGP( dev ));
         fx_close ( dev );
         exit ( 0 );
}
```

8. Fax Demo Programs for Linux

8.1. Overview

This chapter provides the following information on fax demonstration programs supported on Linux.

- An overview of fax synchronous and asynchronous demos
- The physical connections necessary to run the fax demos
- The software required to run the fax demos
- Running the demos
- The fax demo programs flow

8.2. Fax Demo Programs Overview

The following fax demo programs for stand-alone applications are included with the fax software:

- *faxdemo*
- *faxasync*
- *faxsr*

These fax demo programs are not supported on DM3 boards.

The *faxdemo* demonstrates the use of the fax library functions in synchronous mode in a normal fax reception and transmission application. When executed, the program initializes the channel to wait for rings and receive a fax when a call is placed to this channel.

By default the received file is stored as a TIFF/F file. The filename takes the form <channel_name>.tif (for example, *dxxxB1C1.tif*). If the file type specified on the command line is raw, the received file is stored as a raw unformatted image file. The filename takes the form <channel_name>.raw (for example, *dxxxB1C1.raw*).

After the fax is received, the channel waits a few seconds and dials the number specified on the command line and transmits the received fax file. Set this number to the phone number of a fax machine.

The fax demo *faxasync* demonstrates the use of the fax library functions in asynchronous mode with multi-channel control within a single process in a normal fax reception and transmission application. When faxasync is executed, the program reads *fax.cfg* and intializes the channels specified in *fax.cfg* to wait for rings. When a call is placed from a fax machine to any one of these channels, the channel picks up the call and receives the fax into a file.

If the file type specified in *fax.cfg* is tiff, the received file is stored as a TIFF/F file. The filename takes the form <channel_name>.tif (for example, *dxxxB1C1.tif*). If the file type specified in *fax.cfg* is raw, the received file is stored as a raw unformatted image file. The filename takes the form <channel_name>.raw (for example, *dxxxB1C1.raw*).

After the fax is received, the channel waits a few seconds and dials the number specified in *fax.cfg* for that channel and transmits the received fax file. Set this number to the phone number of a fax machine.

The fax demo *faxsr* sends or receives faxes using the Fax API on a single Voice/Fax channel. Send up to four files specified on the command line. File extensions are used to determine data type, *.raw* for raw files and *.tif* for TIFF/F files. Files with any other extension are assumed to be ASCII format. Use the DF_ASCIIDATA structure to change the default settings for faxing ASCII information.

faxsr can also receive faxes and store them to disk in either raw or TIFF/F format. The extension of the specified receive file is used to determine how the fax is stored, *.raw* for raw format or *.tif* for TIFF/F format. If raw is specified and the fax contains more than one page, additional pages are stored in files with incrementing extensions, for example, *filename02.raw*, *filename03.raw*, etc.

8.3. Fax Demo Programs Physical Connections

The following physical connections are required to run the fax demos:

- The installed system with fax resources connected to the telephone network
- A fax machine connected to the telephone network via a separate phone line

8.4. Fax Demo Programs Software

The fax demo program software is contained on the System Release software distribution media and is installed during fax software installation. Refer to the *Installation Guide* for software installation procedures.

The files include the fax demo program source code, an executable version of the demo), and a *makefile* used to compile the fax demo program.

The source code for the fax demo programs is written in C and is instructional to those with a C and Linux programming background.

The *faxdemo* demo program implements the single channel per process model, which follows the suggested synchronous mode model for building applications using the Voice Driver.

The *faxasync* demo program implements a multi-channel per process model for building applications using the Voice Driver.

The *faxsr* demo program sends or receives faxes using the Fax API.

A toolkit of fax C routines is also included on the System Release software distribution media. This toolkit aids in the development of fax synchronous mode applications. The toolkit source code is contained in the file */user/dialogic/fx_demos/faxconv/faxconv.c.*

NOTE: The fax convenience functions (**fx_sendascii( )**, **fx_sendraw( )** and **fx_sendtiff( )**) are located in the *faxconv.c* file.

8.5. Before Running the Fax Demo Programs

After installing the System Release software and before running fax demo programs, you may recompile the fax demo programs .

While logged in the system with root privileges, at the */usr/dialogic/fx_demos* subdirectory on your system, enter the following command:

```
make clean
```

The **make clean** command deletes the fax demo programs executable files and the fax demo program object files from your system.

To recompile the fax demo programs executable files and to create the object files, enter the following command at the */usr/dialogic/fx_demos* subdirectory prompt on your system while logged in the system with root privileges:

```
make
```

After recompilation, the **make** command links the object files with the *libdxxx.a*, *libsrl.a* and *libfax.a* library files previously installed on your system during System Release software installation (refer to the *Installation Guide* for information regarding software installation).

8.5.1. Modify *fax.cfg* Configuration File

The *fax.cfg* configuration file is only used when running the *faxasync* fax asynchronous demo program. The *fax.cfg* file must contain the channel devices and fax phone numbers that are used during the asynchronous demo.

The syntax for each entry in the *fax.cfg* file is as follows:

```
devname   faxnumber   <filetype>
```

devname	Specifies the fax application's channel device that the faxasync demo program uses to receive and transmit a fax document (Example: dxxxB1C1). The channel device specified must have fax capability.
faxnumber	Specifies the fax machine telephone number (where the fax demo application sends the fax document it receives).
filetype	Specifies the type of file (TIFF/F or raw) in which to store the fax data received (optional argument). If the filetype argument is not included, TIFF/F is assumed, otherwise specify raw (arguments must be in lower case letters).

Sample configuration file entries:

```
devname        faxnumber    filetype
dxxxB1C1       5551234      tiff
dxxxB1C2       5555678      raw
```

Modify the *fax.cfg* configuration file to include all devices, fax phone numbers and file types to be used for the asynchronous *(faxasync)* demo, then save the file.

8.5.2. Fax Demo Program Execution Considerations

During the execution of the demo programs *faxdemo* and *faxasync*, a file(s) is created to store the fax data in the directory from which the demo is executed. The user must have write permission to that directory.

The user has the following options:

- Execute the demo from the */usr/dialogic/fx_demos/sync_demos/<fax demo>* ,the */usr/dialogic/fx_demos/async_demos/faxasync* (for *faxasync*), or the */usr/dialogic/fx_demos/scc_demos/<fax demo>* directory.

NOTE: Running the demos from the directories stated above requires that you be logged into the Linux system with root privileges.

- Copy the fax demo file (and *fax.cfg* for asynchronous demo) to the user's local directory (after recompilation), then run the demo from the user's local directory.

8.6. Running the Fax Demo Programs

The following sections explain how to start demo programs.

8.6.1. Starting *faxdemo*

The command-line syntax to execute the Fax synchronous demo program *faxdemo* is as follows:

```
faxdemo devname faxnumber <filetype>
```

devname	Specifies the fax application's channel device that the fax demo program uses to receive and transmit a fax document (Example: dxxxB1C1). The channel device specified must have fax capability.
faxnumber	Specifies the fax machine telephone number where the fax application sends the fax document it receives.
filetype	Specifies the type of file (TIFF/F or raw) in which to store the fax data received (optional argument). If the filetype argument is not included, TIFF/F is assumed, otherwise specify raw (arguments must be in lower case letters) (see the command-line examples below). If TIFF/F is specified, the fax demo stores the data into a single TIFF/F file. The document name is devname and appended with .tif (Example: dxxxB1C1.tif). If raw is specified, the fax demo stores the data as a single page. If more than one page of fax data is sent to the fax demo application, an error occurs and the demo stops. The document name is devname and appended with .raw (Example: dxxxB1C1.raw). The one page limitation for raw files only applies to the fax demo program. This is not a limitation of the fax resource.

8.6.2. Starting *faxasync*

After modifying the *fax.cfg* configuration file (*Section 8.5.1. Modify fax.cfg Configuration File)*, use the following command-line syntax to execute the *faxasync* fax asynchronous demo program:

```
faxasync
```

A screen is displayed on the monitor that shows the device name(s) and status during the running of the demo.

If an error occurs on any channel during the execution of the *faxasync* demo program, a message is displayed on the screen to indicate that a problem has occurred on a specific channel. The execution of the other active channels

(specified in the *fax.cfg* file) continues unaffected. The messages include the following (as applicable):

- Fax demo program error message
- Fax or system error code
- Fax Phase E status message

NOTE: See the Appendices for information on fax error codes and Phase E status values.

Refer to the *faxasync.c* source code file for fax demo error processing information.

8.6.3. Starting *faxsr*

The command-line syntax to execute the Fax synchronous demo program *faxsr* is as follows:

```
faxsr devname faxnumber <filetype>
```

For example, to send *fonttest.txt* from fax channel 1 on board 1, to a fax machine with the phone number 555-1234, type:

```
faxsr -d"5551234" -s"fonttest.txt"
```

In addition, the User Header field, Local ID, and channel can be specified on the command line.

faxsr parses the command line parameters to determine what to do. If the dial string and send file list are present, *faxsr* builds an IOTT chain, dials the number and sends the fax. If a receive file is specified, *faxsr* waits for rings. If rings are received, *faxsr* goes offhook and receives the fax.

Defaults:

Channel	First channel in system
Local ID	"FAXSR CH: nn" where nn is the channel number
User Header	"" a null string

Command-line Examples

In the following command-line example, *faxdemo* uses the application's channel device one, on board one, to do the following:

- Receive a fax document and store it in TIFF/F format
- Send the document just received (stored in TIFF/F format) to the fax machine associated with the telephone line number 555-1234

```
faxdemo dxxxB1C1 5551234
```

In the following command-line example, *faxdemo* uses the application's channel device one, on board one, to do the following:

- Receive a fax document (single page) and store it as a raw file
- Send the document just received (stored as a raw file) to the fax machine associated with the telephone line number 555-1234

```
faxdemo dxxxB1C1 5551234 raw
```

If an error occurs during the execution of the fax demo program, the demo stops and messages are displayed on the monitor screen to indicate the problem. They include the following (as applicable):

- Fax demo program error message,
- Fax or system error code,
- Fax Phase E status message.

NOTE: See the Appendices for information on fax error codes and Phase E status values.

An error message is displayed on the system monitor if a fax demo program function returns a -1.

Refer to the *faxdemo.c* source code file for fax demo error processing information.

8.7. Fax Demo Program Flow

Both the synchronous and asynchronous demo programs follow the same basic flow of execution as outlined in this section. The use of the fax main library functions and the fax convenience functions (for synchronous use) are demonstrated in the demo programs.

Fax demo program flow:

1. The fax application's channel device specified in **devname** is opened.
2. The local ID of the fax application's channel device is set to the name specified in **devname** (Example: dxxxB1C1).
3. The fax application is ready to receive a fax document and waits for a call (incoming rings) from a remote fax machine.
4. The user prepares to send a document (single or multi-page to send to TIFF/F file; single page to send to raw file) on a remote fax machine and places a call from the remote fax machine to the telephone number associated with the fax application's channel device specified in **devname**.
5. The fax application detects the incoming rings and takes the fax application's channel device specified in **devname** off-hook.

NOTE: The local ID of the fax application's channel device is displayed on the remote fax machine if the remote fax machine has this display capability.

6. The fax application receives and stores the document in either a TIFF/F file (default) or **raw** file (if **raw** is specified in **filetype**).
    - If TIFF/F file, the fax demo accepts a single or multi-page fax and store it in a single TIFF/F document file. The document name is **devname** and appended with *.tif* (Example: *dxxxB1C1.tif*).
    - If **raw** file, the fax demo accepts **only one page**. If more than one page is sent to the fax application, an error occurs. The document name is **devname** and appended with *.raw* (Example: *dxxxB1C1.raw*).

NOTE: The one page limitation for raw files only applies to the fax demo program. This is not a limitation of the fax resource.

7. After the document page(s) has been successfully received by the fax application, information is displayed on the monitor screen to indicate successful reception.
8. The fax application's channel device is placed on-hook.
9. A ten second pause.
10. The fax application dials the remote fax machine at the telephone number specified in **faxnumber**.
11. The fax application sends the stored document to the remote fax machine via the fax application's channel device specified in **devname**.
    - If the document was stored as a TIFF/F file, the entire TIFF/F document is transmitted
    - If **raw** was specified (in **filetype**), the synchronous demo transmits the single raw page twice as if it were a two page document. The asynchronous demo transmits the raw page only once
12. After the document page(s) has been successfully transferred to the remote fax machine from the fax application, information is displayed on the fax application's monitor screen to indicate successful transmission.
    - The number of pages transmitted
    - The remote ID (ID of the receiving fax machine)
13. The fax application's channel device is placed on-hook and waits for the next incoming fax call.

NOTES:
1. Transmitting the raw file twice during the synchronous demo allows the use of the **fx_sendfax( )** and **fx_sendraw( )** functions for the fax demo.
2. Exit the fax demo program by issuing an interrupt, quit, terminate or hang up to the system (Example: Press the <Del> (delete) keyboard key).

Refer to the *faxdemo.c, faxasync.c, and faxsr.c* source code files for fax demo source code details.

9. Fax Demo Program for Windows

This chapter briefly describes the fax demonstration program supported on Windows operating systems. This demonstration program is not supported on DM3 boards.

The stand-alone fax demo, *dspfaxsr.exe*, is designed to send or receive faxes without channel routing. It can send up to four files of the following types: raw, TIFF/F or ASCII. Use this demo on boards that support DSP-based Group 3 fax (Softfax).

The demonstration program is installed by default in the *demos* subdirectories. Source code written in C as well as instructions are supplied with each demonstration program. For further instruction on running the demo, see the online help distributed with the demo. You can access this help from the **Help** menu.

10. Fax Data Structures

10.1. Overview

This chapter describes fax data structures used by the fax library.

Fax library data structures are defined in *faxlib.h,* which resides by default in the …*dialogic**inc* directory.

The data structures are shown in *Table 11. Fax Data Structures.*

Table 11. Fax Data Structures

Structure Name	Description
DF_ASCIIDATA	ASCII Data Description Structure. Specifies the formatting of the ASCII data for a fax transmission. Parameters to be set include margins, page length, font selection, spacing between ASCII lines and number of tab stops on a line. A pointer in the DF_IOTT structure specifies the location of the DF_ASCIIDATA structure.
DF_DCS	Digital Command Signal Information Structure. Provides the T.30 Digital Command Signal information in LSB format for the **fx_getDCS( )** function call.
DF_DIS	Digital Identification Signal Information Structure. Provides the T.30 Digital Identification Signal information in LSB format for the **fx_getDIS( )** function call.
DF_IOTT	Fax Transfer Table Structure. Describes the characteristics of the fax data to be sent. You can build an array or linked list of DF_IOTT structures to specify fax transmission from multiple sources within a single **fx_sendfax( )** call.

Structure Name	Description
DF_TXNSF	Transmit NSF Message Structure. Describes the characteristics of the customized T.30 non-standard facilities (NSF) message to be sent. Used with FC_TXNSF parameter in **fx_setparm**( ).
DF_UIO	User-definable I/O Structure. Contains user-defined I/O functions that replace standard **read**( ), **write**( ) and **lseek**( ) functions. The application installs the user-defined functions using the **fx_setuio**( ) function.

10.2. Declaring Fax Data Structures

You must declare the DF_IOTT, DF_ASCIIDATA and DF_UIO structures to be global or static in your application. Do not modify the contents of the structures until after the completion of the fax session.

In asynchronous mode, the fax library needs to repeatedly access the DF_IOTT, DF_ASCIIDATA and DF_UIO table entries during the fax transmission even though **fx_sendfax( )** returns control to the application after initiating the fax transmission.

10.3. DF_ASCIIDATA – ASCII Data Description

The DF_ASCIIDATA data structure describes graphical attributes such as margins, font and line spacing to use in converting an ASCII file to a fax image for transmission.

Use of DF_ASCIIDATA is optional. If used, the DF_IOTT data structure must contain a pointer (**io_datap**) cast as a void * to the location of the specific DF_ASCIIDATA structure. If the pointer is NULL, default ASCII values are used.

On DM3 boards, the DF_ASCIIDATA structure is not used. See *Section 10.3.3. DF_ASCIIDATA Usage* Rules for more information.

10.3.1. DF_ASCIIDATA Definition

The DF_ASCIIDATA structure consists of the following fields.

```
struct df_asciidata {

    ushort pagelength;     /* Page Length */
    ushort pagepad;        /* Pad blank scan lines to end of page */
    ushort topmargin;      /* Top Margin */
    ushort botmargin;      /* Bottom Margin */
    ushort leftmargin;     /* Left Margin */
    ushort rightmargin;    /* Right Margin */
    ushort linespace;      /* Spacing between ASCII lines */
    ushort font;           /* Font selection */
    ushort tabstops;       /* Number of tabstops on a line */
    uchar  units;          /* Units for specifying margins/lengths */
    uchar  flags;          /* Reserved for future use */
};
```

10.3.2. DF_ASCIIDATA Field Descriptions

For easier reference, the fields in the DF_ASCIIDATA structure are described in alphabetical order in *Table 12. DF_ASCIIDATA Fields*.

For additional usage information, see *Section 10.3.3. DF_ASCIIDATA Usage* Rules on *page 124*.

Table 12. DF_ASCIIDATA Fields

Field	Description
botmargin	**Default:** 2 (corresponds to 0.2 inches) Bottom margin for ASCII text in units. See **units** field (default for **units** is 1/10 inch units). Valid values are: • between 0 and 52 in 0.1 inch units • between 0 and 132 in mm units • between 0 and 512 in pels (coarse scan lines)
flags	Reserved for future use.
font	**Default:** DF_FONT_0 The font used in rendering the ASCII document. See *Appendix F* for ASCII character set supported. Valid values are: • DF_FONT_0 (default) – normal 6 lines/inch; prints approximately 10 characters per inch at an approximate 12 point character; 112 characters maximum per line; 16 scan lines of MH data; 20 (horizontal) by 16 (vertical) pixel font in sans serif Helvetica style. • DF_FONT_3 – compressed 8 lines/inch; prints approximately 17 characters per inch at an approximate 9 point character; 192 characters maximum per line; 12 scan lines of MH data; 12 (horizontal) by 12 (vertical) pixel font in sans serif Helvetica style. When using Windows font handles, the font specified in the **font** field overrides FC_FONT0 set in **fx_setparm( )**. For more information on Windows font handles, see *Section 7.3. Using fx_setparm( ) and fx_getparm( ) to Select Fonts*.

Table 12. DF_ASCIIDATA Fields (continued)

Field	Description
leftmargin	**Default:** 3 (corresponds to 0.3 inches) Left margin for ASCII text in units. See **units** field (default for **units** is 1/10 inch units). Valid values are: • between 0 and 25 in 0.1 inch units • between 0 and 64 in mm units • between 0 and 512 in pels (pixels)
linespace	**Default:** DF_SINGLESPACE Linespace can be used to set one of the following: • The line space between ASCII text. The font height determines the line spacing for each row. **OR** • The number of lines per inch. Valid values for the **line space between ASCII text**: • DF_SINGLESPACE – single space between lines (default) • DF_DOUBLESPACE – 2 spaces between lines • DF_TRIPLESPACE – 3 spaces between lines • DF_HALFSPACE – add a half space to single or double space.

Table 12. DF_ASCIIDATA Fields (continued)

Field	Description
linespace (cont.)	Valid values for the **number of lines** per inch: • DF_8LPI – 8 lines per inch • DF_6LPI – 6 lines per inch • DF_3LPI – 3 lines per inch • DF_4LPI – 4 lines per inch • DF_2_4LPI – 2.4 lines per inch If DF_8LPI is specified for font DF_FONT_0, the font is too large to fit within the 8 lpi requirement.
pagelength	**Default:** 110 (corresponds to 11 inches) Page length for ASCII text in units. See **units** field (default for **units** is 1/10 inch units). To allow embedded formfeed characters in the ASCII data to control the paging, set the value to exceed the longest page of ASCII data. Valid values are: • ≥52 – in 0.1 inch units • ≥133 – in mm • ≥513 – in scan lines
pagepad	**Default:** DF_PAD Pad last page with blank lines (default) or do not pad with blank lines. All pages received are the same length as specified in **pagelength**. Valid values are: • DF_PAD – Pad last page with blank lines (default). • DF_NOPAD – Do not pad with blank lines.

Table 12. DF_ASCIIDATA Fields (continued)

Field	Description
rightmargin	**Default:** 3 (corresponds to 0.3 inches) Right margin for ASCII text in units. See **units** field (default for **units** is 1/10 inch units). Valid values are: • between 0 and 25 in 0.1 inch units • between 0 and 64 in mm units • between 0 and 512 in pels (pixels)
tabstops	**Default:** 8 The number of tab stops per line equally spaced based on the total page width. Any tab stops set outside right and left margins are not usable. Valid values: ≥0
topmargin	**Default:** 2 (corresponds to 0.2 inches) Top margin for ASCII text in units. See **units** field (default for **units** is 1/10 inch units). Valid values are: • between 0 and 52 in 0.1 inch units • between 0 and 132 in mm units • between 0 and 512 in pels (coarse scan lines)

Table 12. DF_ASCIIDATA Fields (continued)

Field	Description
units	**Default:** DF_UNITS_IN10 The decimal values specified in the margin fields (top, bottom, left and right) and the pagelength field of the DF_ASCIIDATA structure are based on the value in this field. For further detail, see *Springware Boards – Maximum* Values for Margins in *Section 10.3.3. DF_ASCIIDATA Usage* Rules on *page 124*. Valid values are: • DF_UNITS_IN10 – 1/10" units (default) for specifying margins and page length. • DF_UNITS_MM – Millimeter units for specifying margins and page length. • DF_PELS – Number of pixels (horizontal) and number of coarse scan lines (vertical) for specifying margins and page length.

10.3.3. DF_ASCIIDATA Usage Rules

The following rules apply when you use the DF_ASCIIDATA structure.

DM3 Boards

On DM3 boards, the DF_ASCIIDATA structure is not used. When an ASCII file is converted to a fax image, the following rules apply.

Regardless of the page length you specify, the converted fax image has no maximum size (unlimited length). No pagination is performed by the firmware. Font is fixed at 10 lines per inch (each line is approximately 1/10 inch in height); prints approximately 12 characters per inch; 16 scan lines of MH data; 16 (horizontal) by 16 (vertical) pixels or 80 characters maximum per line. Top Margin is set to 3, Left Margin to 14 and Right Margin to 94.

Springware Boards – Applicability

The values specified in the DF_ASCIIDATA structure apply to the DF_IOTT entry it is associated with, unless the values are overridden by escape sequences from an Intel Dialogic ASCII to Fax command set (see *Appendix F*).

Springware Boards – Using Escape Sequences from ASCII to Fax Command Set

The ASCII file may contain embedded escape sequences from an Intel Dialogic ASCII to Fax Command Set (see *Appendix F*). These escape sequences specify graphical attributes within the ASCII file and override the values specified in the DF_ASCIIDATA structure.

Springware Boards – Maximum Values for Margins

The **units** field determines the unit of measurement for the top, bottom, left and right margin fields in the DF_ASCIIDATA structure.

The maximum decimal value that can be specified for the margin fields is listed in *Table 13. Maximum Values for Margins* for each of the supported units:

Table 13. Maximum Values for Margins

Units	Top	Bottom	Left	Right
DF_UNITS_IN10 (0.1 inch units)	52	52	25	25
DF_UNITS_MM (mm)	132	132	64	64
DF_PELS (coarse scan lines)	512	512	-	-
DF_PELS (pixels)	-	-	512	512

Springware Boards – Page Size of Converted ASCII Document

The conversion of the ASCII text to a fax image takes place at the time of transmission. Line wrapping occurs if the line of ASCII text is longer than the negotiated width of the fax data.

The ASCII data is separated into pages based on the following:

- The maximum number of ASCII lines that can fit on a page determined by the page length, margins and font selection.
- The presence of formfeed characters in the ASCII data.

  The line wrapping and paging is transparent to the application. The page padding option automatically fills in blank lines to the end of the last page (default). See *Appendix F* for embedded escape sequences that override the defaults.

For more information on specifying fonts for ASCII to fax, see *Chapter 7. Specifying Fonts in ASCII to Fax Conversion*.

10.4. DF_DCS – Digital Command Signal

The DF_DCS data structure provides T.30 Digital Command Signal information (in LSB format) for the **fx_getDCS( )** function call.

The T.30 Digital Command Signal specifies caller transmit parameters and provides information on Phase B negotiated settings between the transmitter and receiver. For a complete description of the information in the DCS signal, see the ITU-T publication *Procedures for Document Facsimile Transmission in the General Switched Telephone Network, Recommendation T.30* (see *Section 1.3.2. Other Publications* on *page 2*).

```
typedef struct {
     char dcs_data[10];     /* DCS information in LSB format */
} DF_DCS;
```

10.5. DF_DIS – Digital Identification Signal

The DF_DIS data structure provides T.30 Digital Identification Signal (DIS) information (in LSB format) for the **fx_getDIS( )** function call.

The T.30 Digital Identification Signal specifies called unit capabilities. The DIS message provides information on the receiver's capabilities. For a complete description of the information in the DIS signal, see the ITU-T publication *Procedures for Document Facsimile Transmission in the General Switched Telephone Network, Recommendation T.30* (see *Section 1.3.2. Other Publications* on *page 2*).

```
typedef struct {
     char dis_data[10];     /* DIS information in LSB format */
} DF_DIS;
```

10.6. DF_IOTT – Fax Transmit Data Description

The DF_IOTT structure describes the characteristics of the fax data for one fax document to be transmitted.

Your application can build an array, linked list or any combination of linked list and array of DF_IOTT structures to specify multiple fax documents for transmission using the *send* fax function. When the *send* function is issued, each DF_IOTT structure is checked for valid parameters. A pointer argument in the *send* fax function points to the DF_IOTT table.

The structure can define raw, TIFF/F or ASCII data.

For usage information, see *Section 5.5. Specifying Fax Data for Transmission in a DF_IOTT Table Entry* on *page 50* and the code examples in the **fx_sendfax( )** function reference on *page 286*.

10.6.1. DF_IOTT Definition

The DF_IOTT structure consists of the following fields.

```
typedef struct df_iott DF_IOTT;
struct df_iott {

     unsigned long io_offset;     /* Starting page number or offset */
     unsigned long io_length;     /* Number of pages or length of data */
     char *io_bufferp;            /* Memory transfer start buffer location */
     DF_IOTT *io_prevp;           /* (Optional) Pointer to previous DF_IOTT */
     DF_IOTT *io_nextp;           /* Pointer to next DF_IOTT entry (for linked list) */
     void *io_datap;              /* Pointer to additional data associated */
                                  /* with io_datatype */
     int io_fhandle;              /* File descriptor */
     unsigned short io_type;      /* Entry type (file, memory; linked, contiguous, */
                                  /* last structure; select user-defined I/O */
                                  /* functions for transmit) */
     unsigned short io_datatype;  /* Type of data to transmit */
     unsigned short io_phdcont;   /* Phase D continuation value to send */
     unsigned short io_width;     /* Width of image  (raw and ASCII) */
     unsigned char io_resln;      /* Vertical resolution of image (raw and ASCII) */
     unsigned char io_coding;     /* Encoding of stored data (raw) */
     unsigned char rfu[2];        /* Reserved for future use */
};
```

The following defines are used with the DF_IOTT structure for clarity:

```
#define io_firstpg     io_offset
#define io_pgcount     io_length
```

10.6.2. DF_IOTT Field Descriptions

For reference, the fields in the DF_IOTT structure are described in alphabetical order in *Table 14. DF_IOTT Fields*.

Table 14. DF_IOTT Fields

Field	Description
io_bufferp	Memory transfer start buffer location.
io_coding	Used for **raw** files only. Indicates the encoding scheme of the stored raw data. Valid values: • DF_MH – Modified Huffman. One-dimensional encoding. • DF_MR – Modified Read. Two-dimensional coding. • DF_MMR – Modified Modified Read. Two-dimensional encoding.
io_datap	Pointer to additional data associated with **io_datatype** (cast as void *).
io_datatype	The source of the data to be transmitted. Valid values: • DF_RAW – source of data is raw file • DF_TIFF – source of data is TIFF/F file • DF_ASCII – source of data is ASCII file For more usage information, see *Sections 5.5.5. Sending Raw Files* (*page 53*), *5.5.6. Sending TIFF/F Files* (*page 55*) and *5.5.7. Sending ASCII Files* (*page 56*). For more information on ASCII file source used on DM3 boards, see *Section 10.3.3. DF_ASCIIDATA Usage* Rules for DF_ASCIIDATA structure.
io_fhandle	File descriptor. For more information, see *Section 10.8. DF_UIO – User-Defined I/O* on *page 134*.
io_firstpg	Used for **TIFF/F** files only. Indicates the starting page number (decimal value) or the first page to send. Note that **page numbering begins at zero**. Valid values: ≥0

Table 14. DF_IOTT Fields (continued)

Field	Description
io_length	For **raw** files, indicates the number of bytes to transfer. For **ASCII** files, indicates the number of bytes of ASCII data to read. >0 = number of bytes to transfer or to read -1 = transfer entire raw file or read entire ASCII file
io_nextp	Pointer to next DF_IOTT entry (for linked list).
io_offset	For **raw** files, indicates the starting byte location (offset) to start data transfer. For **ASCII** files, indicates the byte offset in the ASCII file (or memory) to start reading ASCII data. A value of 0 means no offset. Valid values: ≥0
io_pgcount	Used for **TIFF/F** files only. Indicates the number of consecutive pages to send. Valid values: >0 = Number of consecutive pages to send -1 = Send all remaining pages from specified page number

Table 14. DF_IOTT Fields (continued)

Field	Description
io_phdcont	The continuation value for Phase D (post-message procedure) of the T.30 protocol. Valid values: • DFC_AUTO – Automatic Phase D Messaging. The fax driver automatically determines the T.30 Phase D continuation value based on the width, resolution and position of the DF_IOTT entries. • DFC_MPG – Merge-Page. The data specified for the DF_IOTT entry directly following the current DF_IOTT entry is concatenated to the same page. • DFC_EOP – End of Procedure (T.30). Terminate current fax session, progress to Phase E and end fax call. • DFC_MPS – Multi-Page Signal (T.30). End of current fax document page; next fax document page is in the same format as the current page; proceed directly to Phase C. • DFC_EOM – End of Message (T.30). End of current fax document page; more fax data to follow at different resolution or width; return to Phase B and negotiate parameters for next fax document page. For more usage information, see *Section 5.5.9. Setting Phase D Continuation Values* on *page 59*.
io_prevp	Pointer to previous DF_IOTT (optional).
io_resln	Used for **raw** and **ASCII** files only. Indicates the vertical resolution of the image in lines per inch: • DF_RESHI – high or fine vertical resolution (196 lpi) • DF_RESLO – low or coarse vertical resolution (98 lpi) The horizontal resolution of a fax image is fixed at 203 lines per inch across the page.

Table 14. DF_IOTT Fields (continued)

Field	Description
io_type	**Default:** IO_CONT This is a bit masked logical OR field that specifies (a) whether the fax data is in a file or in memory, (b) whether the structure entry is an array, a linked list or the last DF_IOTT entry, and (c) whether the structure entry selects user-defined I/O functions. • IO_DEV – Transfers fax from a file • IO_MEM – Transfers fax from memory (raw or ASCII data only) • IO_CONT – Next DF_IOTT entry is contiguous in memory (default) • IO_LINK – Next DF_IOTT entry is linked to the current entry • IO_EOT – Indicates the last DF_IOTT entry. If the IO_EOT flag is set in the **io_type** field, then all the other flags are ignored. • IO_UIO – Selects user-defined I/O functions for fax transmission For more information on using the **io_type** field, see *Section*s *5.5.2. Connecting DF_IOTT Table Entries* (*page 52*), *5.5.3. Sending Data from Device or Memory* (*page 52*), *5.11. Creating User-Defined I/O Functions* (*page 78*) and *10.8. DF_UIO – User-Defined I/O* (*page 134*).
io_width	Used for **raw** and **ASCII** files only. Indicates the width of the image: • DF_WID1728 – 1728 pixels per line • DF_WID2048 – 2048 pixels per line • DF_WID2432 – 2432 pixels per line
rfu[2]	Reserved for future use.

10.7. DF_TXNSF – Transmit NSF Message

The DF_TXNSF data structure describes the characteristics of the T.30 non-standard facilities (NSF) message sent by the transmitter. Use of this data structure and the FC_TXNSF define in **dx_setparm( )** enables the transmitter to send a customized NSF message during Phase B negotiation.

This data structure is only supported on DM3 boards; it is not supported on Springware boards.

10.7.1. DF_TXNSF Definition

The DF_TXNSF structure consists of the following fields:

```
typedef struct df_txnsf {

    unsigned char length;
    unsigned char nsf[255];

} DF_TXNSF;
```

10.7.2. DF_TXNSF Field Descriptions

The DF_TXNSF fields are described in *Table 15. DF_TXNSF Fields*.

Table 15. DF_TXNSF Fields

Field	Description
length	Specifies the number of bytes of data contained in the nsf field.
nsf	Contains the T.30 non-standard facilities (NSF) customized message. **NOTE:** Although the nsf field specifies 255, the maximum length valid for this field is 127 bytes.

10.8. DF_UIO – User-Defined I/O

The DF_UIO structure contains pointers to user-defined functions that replace the standard I/O functions **read( )**, **write( )** and **lseek( )**. This structure is passed to **fx_setuio( )**.

The user-defined I/O functions are passed the same arguments and must specify the same return type as the standard I/O functions. The fax library stores the pointers to the user-defined read, write and seek functions.

10.8.1. DF_UIO Definition

The DF_UIO structure consists of the following fields.

```
typedef struct df_uio {
     int (*u_read)( );     /* User defined replacement for read( ) */
     int (*u_write)( );    /* User defined replacement for write( ) */
     long (*u_seek)( );    /* User defined replacement for lseek( ) */
};
```

10.8.2. DF_UIO Field Descriptions

The DF_UIO fields are described in *Table 16. DF_UIO Fields*.

Table 16. DF_UIO Fields

Field	Description
u_read	A pointer to the user-defined **read( )** function.
u_write	A pointer to the user-defined **write( )** function.
u_seek	A pointer to the user-defined **lseek( )** function.

10.8.3. DF_UIO Usage Rules

The following rules apply to the use of the DF_UIO structure:

- specifying user I/O for **fx_sendfax( )**
- specifying user I/O for **fx_rcvfax2( )**

Specifying User I/O for fx_sendfax( )

To specify user-defined I/O functions for use in sending a fax, you must "OR" the **io_type** field of the appropriate DF_IOTT structure with IO_UIO.

The fax library then calls the user-defined seek and read functions when processing the DF_IOTT structure. The file descriptor argument passed to the user-defined functions is the value specified in the **io_fhandle** field of the DF_IOTT structure.

NOTE: In an array of DF_IOTT structures passed to the **fx_sendfax( )** function, the user-defined I/O functions are only called for those structures whose IO_UIO bit is set in the **io_type** field. The standard I/O functions are used for all other DF_IOTT structures.

Specifying User I/O for fx_rcvfax2( )

To specify user-defined I/O functions for use in receiving a fax, you must "OR" the **rcvflag** argument of the **fx_rcvfax2( )** function with the IO_UIO bit.

The fax library then calls the user-defined seek and write functions every time fax data is to be written to the I/O device. The file descriptor passed to the user-defined I/O functions is the **fd** argument to the **fx_rcvfax2( )** function.

11. Using the Fax Library

11.1. Overview

This chapter presents an overview of the fax main library functions and convenience functions.

11.2. Function Categories

The fax main library (*libfaxmt.lib*) provides functions used to create fax applications. These functions interface with the voice driver.

Intel also supplies fax convenience functions, which are built on fax main library functions. Convenience functions enable you to easily implement basic functionality of the fax main library functions. The source code for these functions is provided in *faxconv.c* and in the function reference.

See *Chapter 12. Fax Library Function Reference* for details on all functions. Not all functions are supported on all platforms (DM3, Springware). Platform support is indicated in the Platform line in each function reference.

The fax library functions can be grouped as shown in *Table 17. Categories of Fax Functions*:

Table 17. Categories of Fax Functions

Category	Description
Send fax	Send single or multi-page fax data as defined by the DF_IOTT structure.
Receive fax	Receive fax data and write to a file; or send a request to receive fax data (polling).
Set initial fax state	Set the initial fax state of the application to caller (transmit) or called (receive).
Initialize DF_IOTT structure	Set default values for a DF_IOTT structure.
Configuration	Set and retrieve fax parameters such as fax header attributes.
Extended attribute	Return information specific to a fax device.
Resource management	Open, close and stop a fax channel device.
TDM bus routing	Connect and disconnect fax channel to a TDM bus time slot.
Convenience	A set of functions built on fax main library functions; these functions simplify application development.
Miscellaneous	Fax functions that don't fall into any other category.

11.2.1. Send Fax

The Send Fax function transmits fax data as defined by the DF_IOTT structure.

NOTE: Convenience functions can also be used to send fax data; see *Section 11.2.10. Convenience Functions* on *page 147*.

Function	Description
fx_sendfax( )	Sends fax data as defined by the DF_IOTT structure: • single or multi-page TIFF/F fax file • single page raw image file • ASCII data Indicates to the caller application that the called application only has transmit capability.

11.2.2. Receive Fax

The Receive Fax functions receive fax data and write it to a specified file, or send a request to the caller application to receive fax data (poll request).

Functions	Description
fx_rcvfax( )	Receives fax data and writes it to a file: • single or multi-page TIFF/F fax file • single page raw image file Requests the caller application to receive a fax document (poll request).
fx_rcvfax2( )	Same description as **fx_rcvfax( )**, except that this function takes a file descriptor argument instead of a file name. Use this function to enable user-defined I/O functions.

11.2.3. Set Initial Fax State

The caller and called fax applications issue the Set Initial Fax State fax function to establish the initial fax state as caller (transmit state) or called (receive state) before issuing the initial *send* or *receive* fax function of the fax call.

NOTE: Once the fax application sets the initial fax state, the correct fax state is maintained by the fax library throughout the fax session even when polling occurs.

Function	Description
fx_initstat( )	Sets the initial fax state for a specified fax channel device: • caller = transmit state • called = receive state

11.2.4. Initialize DF_IOTT

This function initializes a DF_IOTT structure which specifies the fax data to send.

Function	Description
fx_setiott( )	Initializes a DF_IOTT structure and sets default values.

11.2.5. Configuration

Configuration functions set and read various fax parameters.

Functions	Description
fx_getparm( )	Reads fax parameter.
fx_setparm( )	Sets fax parameter such as fax page header attributes.

11.2.6. Extended Attribute

Fax extended attribute functions have the prefix **ATFX_**. These functions take one parameter, the device handle for the fax channel, and return status information about the fax session.

Fax extended attributes are included in the fax library file (*libfaxmt.lib).*

Fax extended attribute function names are case-sensitive and must be written in uppercase letters.

Fax Extended Attribute	Function Description
ATFX_BADIOTT( )	Returns a pointer to an invalid DF_IOTT structure.
ATFX_BADPAGE( )	Returns a bad TIFF/F page number within the DF_IOTT structure.
ATFX_BADSCANLINES( )	Returns the number of bad scan lines in the last page transmitted or received.
ATFX_BSTAT( )	Returns the Phase B status bitmap after a TFX_PHASEB event. Returns bits indicating that the following messages are available: Non-Standard Facilities (NSF) information, Digital Information Signal (DIS) message and Digital Command Signal (DCS) message.
ATFX_ECM( )	Returns information on use of Error Correction Mode (ECM) for fax data transfer.
ATFX_ESTAT( )	Returns Phase E information describing errors during the fax session.
ATFX_CHTYPE( )	Returns the fax channel hardware type.
ATFX_CODING( )	Returns negotiated line encoding scheme.
ATFX_FXVERSION( )	Returns fax library version number string.
ATFX_LASTIOTT( )	Returns a pointer to the last DF_IOTT structure that was processed.
ATFX_PGXFER( )	Returns the number of pages transferred.
ATFX_PHDCMD( )	Returns the Phase D command after a TFX_PHASED event.
ATFX_PHDRPY( )	Returns the Phase D reply after a TFX_PHASED event.

Fax Extended Attribute	Function Description
ATFX_RESLN( )	Returns a decimal value indicating the vertical resolution of the transferred page after Phase D is completed.
ATFX_RTNPAGES( )	Returns the number of received pages that returned RTN to the transmitter during reception.
ATFX_SCANLINES( )	Returns the total number of scan lines in the last page transmitted or received.
ATFX_SPEED( )	Returns the final transfer speed (in baud) after Phase B completed; returns the equate DF_14400BAUD to indicate 14.4 Kbps transfer speed.
ATFX_STATE( )	Returns the current fax device channel state.
ATFX_TERMMSK( )	Returns a bitmap of termination reasons.
ATFX_TFBADTAG( )	Returns a bad TIFF/F tag number if EFX_BADTAG error is returned to **ATDV_LASTERR( )**.
ATFX_TFNOTAG( )	Returns the tag number of missing TIFF/F mandatory tag if EFX_BADTIF error is returned to **ATDV_LASTERR( )**.
ATFX_TFPGBASE( )	Returns a value indicating the base page numbering scheme for the last transmitted TIFF/F file.
ATFX_TRCOUNT( )	Returns the total number of bytes transferred during the current fax session.
ATFX_WIDTH( )	Returns a decimal value (in pixels per line) of negotiated width after Phase D completed. Returns the width of the fax page as it was transferred.

11.2.7. Resource Management

Resource Management functions start and stop fax resources, and stop a fax transfer.

Functions	Description
fx_open( )	Opens a fax channel or board device.
fx_close( )	Closes a fax channel device.
fx_stopch( )	Stops a fax channel device I/O.

Before a fax transfer can occur, the fax channel device must be opened. The **fx_open( )** function specifies a unique Intel Dialogic device handle. This handle is the only way a device can be identified once it is open. The **fx_close( )** function closes a device via its handle.

The **fx_open( )** and **fx_close( )** functions do not cause a device to be busy. The functions work on a device whether the device is busy or idle.

See *Section 5.3. Opening and Closing a Fax Channel* (*page 49*) for more information on opening and using devices.

Hints

- Issuing an **fx_open( )** or **fx_close( )** while the fax device is in use by another process does not affect the current operation of the fax device.
- The device handle which is returned is **Dialogic defined**. The device handle is not a standard operating system file descriptor. Any attempts to use operating system commands such as **read( )**, **write( )**, or **ioctl( )** will produce unexpected results.
- In an application that creates a child process from a parent process, a device handle is not inheritable by the child process. Devices must be opened in the child process.

The **fx_stopch( )** function stops a fax *send* or *receive* in progress on a channel device.

11.2.8. TDM bus Routing

Use the fax-specific TDM bus routing functions in combination with TDM bus routing functions of other resources to set up TDM bus routing to send or receive a fax.

Functions	Description
fx_listen( )	Connects fax listen channel to TDM bus time slot.
fx_unlisten( )	Disconnects fax listen channel from TDM bus.
fx_getxmitslot( )	Returns fax device channel's TDM bus transmit time slot.
fx_getctinfo( )	Returns information about a fax device handle

The fax TDM bus routing functions are included as part of the fax library.

11.2.9. Miscellaneous

The following functions don't fall into any other fax category. They are used to get T.30 messaging data, load fonts for ASCII data, set up user-defined I/O functions and for other miscellaneous purposes.

Functions	Description
fx_getDCS( )	Gets T.30 Digital Command Signal data.
fx_getDIS( )	Gets T.30 Digital Information Signal data.
fx_getNSF( )	Gets T.30 Non-Standard Facilities data.
fx_GetDllVersion( )	(Windows only) Returns the fax DLL version number.
fx_libinit( )	(Windows only) Initializes the fax library DLL.
fx_setuio( )	Installs user-defined I/O functions.
fx_originate( )	Allows the DCS on hold feature.

The **fx_getDCS( )** function allows an application to retrieve the most recent T.30 Digital Command Signal message, if available, for a specified channel. The DCS message contains information about the Phase B negotiated settings between the transmitter and receiver.

The **fx_getDIS( )** function allows an application to retrieve the most recent T.30 Digital Information Signal message, if available, for a specified channel. The DIS message contains information about the receiver's capabilities. The DIS message is sent by the receiver to the transmitter as part of the Phase B negotiation.

The **fx_getNSF( )** function allows an application to retrieve the T.30 Non-Standard Facilities message, if available, for a specified channel. The NSF message is a variable length message that can contain manufacturer-specific information. Manufacturers can use this message to support proprietary features for their products. The NSF message is sent as part of the Phase B negotiation.

The **fx_GetDllVersion( )** function returns the fax DLL version number, while the **fx_libinit( )** function initializes the fax library DLL.

The **fx_setuio( )** function allows an application to install user-defined **read( )**, **write( )** and **lseek( )** I/O functions. The DF_UIO data structure provides pointers to user-defined I/O functions.

11.2.10. Convenience Functions

Fax convenience functions are built on fax main library functions. They allow you to easily implement some of the basic functionality of the fax main library functions.

Functions	Description
fx_sendascii( )	Sends a single ASCII file.
fx_sendraw( )	Sends a single page of raw, unformatted, compressed fax data.
fx_sendtiff( )	Sends a single TIFF/F file.

The fax convenience functions are based on the **fx_sendfax( )** function and use the DF_IOTT data structure. The source code for these functions is provided in the function reference as well as in the file *faxconv.c*.

You have the option of building your own convenience functions or using the functions provided in *faxconv.c*. If you use the convenience functions in *faxconv.c*, you must compile *faxconv.c* and link it with the object code when building your application.

These convenience functions are written to operate in synchronous mode (EV_SYNC). To use the asynchronous mode of operation, you can modify the source code for the call to **fx_sendfax( )** in the *faxconv.c* file**.** See the **fx_sendfax( ), fx_sendascii**, **fx_sendraw( )** and **fx_sendtiff( )** function references and sample code in *Chapter 12. Fax Library Function Reference*.

11.3. Error Handling

This *Section* describes error handling in general and for specific modes of operation, namely synchronous and asynchronous. For a list of fax error codes, see *Appendix D*.

All fax library functions return a value to indicate success or failure of the function.

- To indicate success, the library returns a value of zero or a non-negative number.
- To indicate failure, the library returns a value of -1.

Extended attribute functions that do not return a pointer indicate failure by returning AT_FAILURE. Extended attribute functions that return a pointer indicate failure with AT_FAILUREP.

If a fax library function fails, call the standard attribute function **ATDV_LASTERR( )** to return the error code and **ATDV_ERRMSGP( )** to return a string describing the error. These functions are described in the *Standard Runtime Library API Library Reference*.

If **ATDV_LASTERR( )** returns the error EDX_SYSTEM, a system error has occurred. On Linux, check the global variable errno for more information. On Windows, use **dx_fileerrno( )** to obtain the system error value. Refer to the **dx_fileerrno( )** function in the *Voice API Library Reference* for a list of possible system error values.

NOTE: **fx_open( )** and **fx_close( )** are exceptions to the above error handling rules. If these functions fail, the return code is -1 and an error from the operating system has occurred.

If a fax send or receive function successfully completes, you can learn the final Phase D status of the fax transfer using fax extended attributes. If the function fails, Phase E status (using the fax extended attribute **ATFX_ESTAT( )**) provides additional error information for the T.30 fax protocol. Values for Phase D and Phase E status are described in *Appendix B* and *Appendix C*, respectively.

On DM3 boards, if you execute a standard fax function that is not supported by DM3 boards, it produces an EFX_NOTIMP ("not implemented") error. If you execute a supported fax function with a parameter that is not supported on DM3 boards, it produces an EFX_UNSUPPORTED error.

11.3.1. Synchronous Mode

Fax library functions that operate in synchronous mode return a value to indicate successful completion or failure of the function.

- To indicate successful completion, the function returns a value of zero.
- To indicate failure, the function returns a value of -1.

If a fax library function operating in synchronous mode fails, an error code is generated. To learn more about this failure, call the standard attribute function **ATDV_LASTERR( )** to return the error code and **ATDV_ERRMSGP( )** to return a string describing the error.

11.3.2. Asynchronous Mode

The **fx_rcvfax( )**, **fx_rcvfax2( )**, **fx_sendfax( ),** and **fx_originate(**) functions can be specified to operate in synchronous or asynchronous mode. All other fax library functions operate in synchronous mode.

Fax library functions that operate in asynchronous mode return a value to indicate invocation success or failure, immediately after the function has been initiated:

- Invocation success is indicated by a return value of zero.
- Invocation failure is indicated by a return value of -1.

If the function is successfully invoked in asynchronous mode, it completes processing or terminates due to a processing error and an event is generated by the Standard Runtime Library.

The Standard Runtime Library may return the following events on completion of **fx_rcvfax( )**, **fx_rcvfax2( )** or **fx_sendfax( )** operating in asynchronous mode:

Event	Description
TFX_FAXERROR	Error in processing
TFX_FAXRECV	Successful completion of **fx_rcvfax( )** or **fx_rcvfax2( )**
TFX_FAXSEND	Successful completion of **fx_sendfax( )**

If a function fails, a TFX_FAXERROR event is generated. Call the standard attribute function **ATDV_LASTERR( )** to return the error code and **ATDV_ERRMSGP( )** to return a string describing the error.

11.4. Include (Header) Files

Applications that use the fax library functions must contain the following statements for header files **in this order** before other statements:

```
#include <srllib.h>  /* For Voice and Fax development purposes. */
#include <dxxxlib.h> /* For Voice development purposes. */
#include <faxlib.h>  /* For Fax development purposes. */
```

NOTE: List *srllib.h* in the code before *dxxxlib.h* and *faxlib.h*.

For default directory path information, see the software installation information included with the system software.

11.5. Compiling Applications

In Linux, you must link the following libraries **in this order** when compiling your single or multi-threaded application:

```
libdxxx.so
libsrl.so
libfax.so
```

In Windows, you must link the following libraries **in this order** when compiling your single or multi-threaded application:

```
libdxxmt.lib
libsrlmt.lib
libfaxmt.lib
```

For default path information, see the software installation information included with the system software.

The fax header and library files are part of the fax software. The function prototypes and equates are defined in the header file. The voice and Standard Runtime Library header files and library files are part of the voice software.

These files are installed on your hard disk during the software installation.

NOTE: If you use the fax convenience functions contained in the *faxconv.c* file, you must also compile and link the *faxconv.c* file when compiling your application. The *faxconv.c* file is included with the fax software.

12. Fax Library Function Reference

Fax Library Overview

This chapter provides an alphabetical reference to the functions in the fax main library (*libfaxmt.lib*) as well as the fax convenience functions (*faxconv.c*).

The following information is included to describe the fax function:

- Reference header information
- Description
- Details (when applicable)
- Cautions
- Example
- Source Code (when applicable)
- See Also (list of related functions, when applicable)

Reference Header Information

The function reference header contains the following information at the beginning of each fax function description:

Name:	The function name (with parameters).
Inputs:	The function parameter input types with a brief description of each parameter.
Returns:	The returns for the function.
Includes:	The include files required as displayed in the function example.
Category:	The group in which the function belongs.
Mode:	The mode of operation for the function (synchronous or asynchronous), if applicable.
Platform:	The board type (DM3, Springware)

Description

The function reference description provides the following information:

- A brief description of the purpose and operation of the function
- The function parameters and values
- A detailed description of the function to include, where applicable, how associated fax features apply to the function

Details

Where applicable, the chapter provides a detailed description of the fax function.

Cautions

The function reference cautions provide important information regarding restrictions on the use of the fax function.

■ Example

An example is provided to show how the function is used in a fax application. The specific fax function and parameters are printed in **bold** type. When applicable, the examples are commented to explain how each function is used in the example.

■ Source Code

Where applicable, the source code for the function is provided as part of the function reference.

■ See Also

Where applicable, a list of related functions is provided at the end of each function reference.

Name:	DF_IOTT * ATFX_BADIOTT(dev)
Inputs:	int dev • fax channel device handle
Returns:	pointer to invalid DF_IOTT if successful AT_FAILUREP if error
Includes:	srllib.h dxxxlib.h faxlib.h
Category:	extended attribute
Mode:	synchronous
Platform:	DM3, Springware

Description

The **ATFX_BADIOTT( )** function returns a pointer to an invalid DF_IOTT structure if one is detected after transmission begins. If a bad DF_IOTT structure is detected, a TFX_FAXERROR event occurs and **ATDV_LASTERR( )** returns the error EFX_BADIOTT.

See the *Voice API Library Reference* for information on **ATDV_LASTERR( )**.

Parameter	Description
Dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

The value returned by **ATFX_BADIOTT( )** at the end of a fax session remains available to the application until a new *send* is initiated on that channel.

Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

DF_IOTT iott[10];
DF_IOTT * badiotp;
int dev;

/*
 * Open the channel using fx_open( ) and obtain the
```

```
 * FAX device handle in dev.
 */
.
.
/* Call fx_sendfax( ) after setting up the DF_IOTT array. */
if (fx_sendfax(dev, iott, EV_SYNC) == -1) {

   /* Check if error was due to an invalid DF_IOTT. */
   if (ATDV_LASTERR(dev) == EFX_BADIOTT) {
      /* Get pointer to bad DF_IOTT element. */
      badiotp = ATFX_BADIOTT(dev);
   }
   .
   .
}
```

■ Errors

If one of the following conditions is present, this function fails and returns AT_FAILUREP:

- An invalid fax channel device handle is specified in **dev.**
- An EFX_BADIOTT error did not occur during the last call to **fx_sendfax( )** on the specified channel.

Name:	long ATFX_BADPAGE(dev)
Inputs:	int dev • fax channel device handle
Returns:	bad page number within DF_IOTT structure if successful AT_FAILURE if error
Includes:	srllib.h dxxxlib.h faxlib.h
Category:	extended attribute
Mode:	synchronous
Platform:	DM3, Springware

■ Description

The **ATFX_BADPAGE( )** function returns the fax page number (if error during processing) within the DF_IOTT structure that is being processed when an error occurs.

To determine the last DF_IOTT processed, call the fax extended attribute **ATFX_LASTIOTT( )**.

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

The value returned by **ATFX_BADPAGE( )** at the end of a fax session remains available to the application until a new *send* is initiated on that channel.

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

DF_IOTT iott[10];
DF_IOTT * lastiotp;
long pagenum;
int dev;

/*
```

```
 * Open the channel using fx_open( ) and obtain the
 * FAX device handle in dev.
 */
.
.
/* Call fx_sendfax( ) after setting up the DF_IOTT array. */
if (fx_sendfax(dev, iott, EV_SYNC) == -1) {

   /*
    * Get pointer to DF_IOTT being processed when error
    * occurred.
    */
   lastiotp = ATFX_LASTIOTT(dev);
   /*
    * Page being processed within this DF_IOTT when error
    * occurred.
    */
   pagenum = ATFX_BADPAGE(dev);
   .
   .
}
```

■ Errors

If one of the following conditions is present, this function fails and returns AT_FAILURE:

- An invalid fax channel device handle is specified in **dev.**
- An EFX_BADPAGE error has not occurred during the last call to **fx_sendfax( )** on the specified channel.

Name:	long ATFX_BADSCANLINES(dev)
Inputs:	int dev • fax channel device handle
Returns:	number of bad scan lines in last page transferred if successful AT_FAILURE if error
Includes:	srllib.h dxxxlib.h faxlib.h
Category:	extended attribute
Mode:	synchronous
Platform:	DM3, Springware

Description

The **ATFX_BADSCANLINES( )** function returns the number of bad scan lines detected and replaced in the last page transmitted or received. This information is available at the end of Phase D for every page sent or received.

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

The data provided by this function is updated each time the fax transfer completes Phase D of the T.30 protocol. For an application to monitor the number of bad scan lines, you must enable Phase D events and issue **ATFX_BADSCANLINES( )** when the TFX_PHASED event occurs. Note that since Phase D also occurs at the end of a send or receive when a Phase D event is not generated, you can also call this function after a TFX_FAXSEND or TFX_FAXRECV event.

The final bad scan line value returned by **ATFX_BADSCANLINES( )** at the end of a fax session remains available to the application until a new send or receive is initiated on that channel.

NOTE: Between multiple Phase D completions during the same fax session, **ATFX_BADSCANLINES( )** returns the bad scan line information from the previously completed page.

Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int dev;

/* Handler for Phase D events. */
int phd_hdlr( );

main( )
{
   /*
    * Open the channel using fx_open( ) and obtain the
    * FAX device handle in dev.
    */
   .
   .
   /*
    * Install handler to service TFX_PHASED events.
    */
    if (sr_enbhdlr(dev, TFX_PHASED, phd_hdlr) == -1) {
        printf("Failed to install Phase D handler \n");
        return;
    }

   /*
    * Call fx_rcvfax( ) in asynchronous mode to receive
    * TIFF/F file. Set DF_PHASED bit in mode field
    * to enable generation of Phase D events.
    */
   if (fx_rcvfax(dev,"fax.tif", EV_ASYNC|DF_PHASED) == -1) {
      printf("Error - %s (error code %d)\n",
         ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
      if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
         /* Perform system error processing */
      }
   }
   .
   .
}

/*
 * Handler registered with SRL to handle TFX_PHASED events.
 */

int phd_hdlr( )
{

   int dev = sr_getevtdev( );

   /*
    * Number of bad scan lines of the page just
    * received is available at this point.
   */
   printf("Bad scan lines in page received: %ld\n",
```

```
            ATFX_BADSCANLINES(dev));

    .
    .
    return(0);
}
```

■ Errors

If one of the following conditions is present, this function fails and returns AT_FAILURE:

- An invalid fax channel device handle is specified in **dev.**
- The function is called prior to the completion of the first page transfer of the fax session.

Name:	long ATFX_BSTAT(dev)
Inputs:	int dev • fax channel device handle
Returns:	Phase B status bitmap after TFX_PHASEB event if successful AT_FAILURE if error
Includes:	srllib.h dxxxlib.h faxlib.h
Category:	extended attribute
Mode:	synchronous
Platform:	DM3, Springware

■ Description

The **ATFX_BSTAT( )** function returns a bitmap to indicate Phase B status. Valid values are:

Phase B Status	Description
DFS_REMOTEID	Remote station ID available
DFS_NSF	Remote station Non-Standard Facilities (NSF) message available
DFS_DIS	Digital Information Signal (DIS) message available, sent from receiver to transmitter
DFS_DCS	Digital Command Signal (DCS) message available, sent from transmitter to receiver
DFS_REMOTESUBADDR	Subaddress routing message available, sent from transmitter to receiver (not supported on DM3 boards)

To obtain the remote station ID, use the FC_REMOTEID parameter value in the **fx_getparm( )** function.

For details on receiving the DCS, DIS and NSF messages, see the **fx_getDCS( )**, **fx_getDIS( )** and **fx_getNSF( )** function references.

Parameter	Description
dev	Specifies the channel device handle for the **fax** channel obtained when the channel was opened.

The data provided by this function is updated each time the fax transfer completes Phase B of the T.30 protocol. After a fax session terminates, the value from the last fax transfer is available until the start of a new fax session.

To be notified when Phase B information is available, you must enable Phase B events (DF_PHASEB in the **fx_rcvfax( ), fx_rcvfax2( )** or **fx_sendfax( )** function) and issue **ATFX_BSTAT( )** when the Phase B event (TFX_PHASEB) occurs.

NOTES:
1. Between multiple Phase B negotiations during the same fax session, **ATFX_BSTAT( )** returns the Phase B availability information from the previously completed Phase B negotiation.
2. If **fx_sendfax( )**, **fx_rcvfax( )** and **fx_rcvfax2( )** are issued in synchronous mode (EV_SYNC), you must install an event handler to handle Phase B events by using the **sr_enbhdlr( )** function of the Standard Runtime Library.

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

DF_IOTT iott[10];
int dev;

/* Handler for Phase B events. */
int phb_hdlr( );

main( )
{
   /*
    * Open the channel using fx_open( ) and obtain the
    * FAX device handle in dev.
    */
   .
   .
   /*
    * Install handler to service TFX_PHASEB events.
    */
    if (sr_enbhdlr(dev, TFX_PHASEB, phb_hdlr) == -1) {
```

```
        printf("Failed to install Phase B handler \n");
        return;
}

   /*
    * Call fx_sendfax( ) in asynchronous mode after setting
    * up the DF_IOTT array.  Set DF_PHASEB bit in mode field
    * to enable generation of Phase B events.
    */
   if (fx_sendfax(dev, iott, EV_ASYNC|DF_PHASEB) == -1) {
      printf("Error - %s (error code %d)\n",
         ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
      if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
         /* Perform system error processing */
      }
   }
   .
   .
}

/*
 * Handler registered with SRL to handle TFX_PHASEB events.
 */

int phb_hdlr( )
{

   int dev = sr_getevtdev( );

   if (ATFX_BSTAT(dev) & DFS_REMOTEID) {
      /*
       * Remote ID available - get remote id using
       * fx_getparm( ).
       */
       .
       .
      }
      /* Remote data rate capability. */
      printf("Data rate for fax transmission: %ld\n",
         ATFX_SPEED(dev));
   .
   .
   return(0);
}
```

■ Errors

If one of the following conditions is present, this function fails and returns AT_FAILURE:

- An invalid **fax** channel device handle is specified in **dev.**
- The function is called prior to the completion of the first Phase B event.

See Also

- **fx_getDCS( )**
- **fx_getDIS( )**
- **fx_getNSF( )**
- **fx_getparm( )**

Name: long ATFX_CHTYPE(dev)
Inputs: int dev • fax channel device handle
Returns: fax channel base hardware type if successful
AT_FAILURE if error
Includes: srllib.h
dxxxlib.h
faxlib.h
Category: extended attribute
Mode: synchronous
Platform: DM3, Springware

■ Description

The **ATFX_CHTYPE( )** function returns the fax channel's base hardware type. Valid values are:

Hardware Type Value	Description
DFS_FAX40	VFX/40SC board (no longer supported)
DFS_FAX40E	VFX/40ESC board (no longer supported)
DFS_FAX40EPLUS	VFX/40ESCplus board (no longer supported)
DFS_FAX40EPLUSREX	board that supports DSP Fax
DFS_FAXDM3	DM3 board that supports fax

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

The base hardware type can be used to determine the capabilities of the fax channel. For more information, see *Section 2.3. Key Product Features* on *page 8.* On DM3 boards, you cannot use **ATFX_CHTYPE( )** to determine if a channel device has fax capabilities, because its **dev** parameter requires a fax device handle previously obtained from **fx_open( )**. If **fx_open( )** succeeds, this means that the channel device is already fax-capable. The **ATFX_CHTYPE( )** function only

identifies the hardware on which the fax-capable channel device sits. If the hardware is DM3, then this function returns DFS_FAXDM3.

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int dev;

main( )
{
   /*
    * Open the channel using fx_open( ) and obtain the
    * FAX device handle in dev.
    */
   .
   .
   /* Determine channel hardware type. */
   switch(ATFX_CHTYPE(dev))  {
   case DFS_FAX40E:
      /*
       * Enable VFX/40ESC (return type DFS_FAX40E) supported features:
       * For example, set FC_RXCODING parameter to DF_MMR to receive
       * all files in MMR encoding scheme.
       */
      .
      .
      break;
   case DFS_FAX40:
      /* VFX/40SC (return type DFS_FAX40) device */
      .
      .
      break;
   }
   .
}
```

■ Errors

If one of the following conditions is present, this function fails and returns AT_FAILURE:

- An invalid fax channel device handle is specified in **dev.**
- The channel device handle specified does not have fax support.

■ See Also

- **fx_sendfax( )**

Name: long ATFX_CODING(dev)
Inputs: int dev • fax channel device handle
Returns: negotiated line encoding scheme if successful
AT_FAILURE if error (or if initial Phase B not completed)
Includes: srllib.h
dxxxlib.h
faxlib.h
Category: extended attribute
Mode: Synchronous
Platform: DM3, Springware

■ Description

The **ATFX_CODING( )** function returns most recently negotiated fax encoding scheme between the transmitter and receiver for the specified fax channel.

Valid values are:

DFS_MH	Modified Huffman line encoding scheme
DFS_MR	Modified Read line encoding scheme
DFS_MMR	Modified Modified Read line encoding scheme

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

The data provided by this function is updated each time the fax transfer completes Phase B of the T.30 protocol. After a fax session terminates, the value from the last fax transfer is available until the start of a new fax session.

To be notified when Phase B has completed, you must enable Phase B events (DF_PHASEB in the **fx_rcvfax( )**, **fx_rcvfax2( )** or **fx_sendfax( )** function) and issue **ATFX_CODING( )** when the Phase B event (TFX_PHASEB) occurs.

NOTES: **1.** Between multiple Phase B completions during the same fax session, **ATFX_CODING( )** returns the Phase B encoding information from the previously completed Phase B negotiation.

2. For a receiver application, the line encoding scheme negotiated between the transmitter and receiver for the fax transfer may be different than the encoding scheme of the stored fax data.
3. For a transmitter application, the line encoding scheme negotiated between the transmitter and receiver for the fax transfer may be different than the encoding scheme of the stored fax data or the encoding scheme specified in the FC_TXCODING parameter. The final negotiated data transmission line encoding scheme is based on the receiver's capabilities.

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>


DF_IOTT iott[10];
int dev;

/* Handler for Phase B events. */
int phb_hdlr( );

main( )
{
   /*
    * Open the channel using fx_open( ) and obtain the
    * FAX device handle in dev.
    */
   .
   .
   /*
    * Install handler to service TFX_PHASEB events.
    */
    if (sr_enbhdlr(dev, TFX_PHASEB, phb_hdlr) == -1) {
        printf("Failed to install Phase B handler \n");
        return;
}

   /*
    * Call fx_sendfax( ) in asynchronous mode after setting
    * up the DF_IOTT array.  Set DF_PHASEB bit in mode field
    * to enable generation of Phase B events.
    */
   if (fx_sendfax(dev, iott, EV_ASYNC|DF_PHASEB) == -1) {
      printf("Error - %s (error code %d)\n",
         ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
      if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
         /* Perform system error processing */
      }
   }
   .
```

```
   .
}

/*
 * Handler registered with SRL to handle TFX_PHASEB events.
 */

int phb_hdlr( )
{

   int dev = sr_getevtdev( );

   /* Negotiated line encoding scheme. */
   printf("Negotiated data encoding scheme: %ld\n",
      ATFX_CODING(dev));
   .
   .
   return(0);
}
```

■ Errors

If one of the following conditions is present, this function fails and returns AT_FAILURE:

- An invalid fax channel device handle is specified in **dev.**
- The function is called prior to the completion of the first Phase B event.

■ See Also

- **fx_rcvfax( )**
- **fx_sendfax( )**
- **fx_setparm( )**

Name:	int ATFX_ECM (dev)
Inputs:	int dev • fax channel device handle
Returns:	whether fax data was transferred using ECM if successful -1 if function called before completion of Phase B
Includes:	srllib.h dxxxlib.h faxlib.h
Category:	extended attribute
Mode:	synchronous
Platform:	DM3, Springware

■ Description

The **ATFX_ECM( )** function returns information on use of ECM for fax data transfer. Valid values are:

DFS_NOECM	Error Correction Mode (ECM) not used to transfer fax data
DFS_ECM	Error Correction Mode (ECM) used to transfer fax data

This function has the following parameter:

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

After completion of Phase B negotiation, you can call **ATFX_ECM( )** to determine whether ECM was used in the fax data transfer. The value in **ATFX_ECM( )** remains valid until a new fax session is initiated.

Example

```
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

DF_IOTT iott[10];
int dev;

/* Handler for Phase B events. */
int phb_hdlr( );

main( )

{
   /*
    * Open the channel and obtain the device handle
    * in dev.
    */
   .
   .
   /*
    * Install handler using sr_enbhdlr( ) to service
    * TFX_PHASEB events.
    */
     if (sr_enbhdlr(dev, TFX_PHASEB, phb_hdlr) == -1) {
         printf("Failed to install Phase B handler \n");
         return;
     }

   /*
    * Call fx_sendfax( ) in asynchronous mode after setting
    * up the DF_IOTT array.  Set DF_PHASEB bit in mode field
    * to enable generation of Phase B events.
    */
   if (fx_sendfax(dev, iott, EV_ASYNC|DF_PHASEB) == -1) {
      printf("Error - %s (error code %d)\n",
         ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
      if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
         /* Perform system error processing */
      }
   }
   .
   .
}

/*
 * Handler registered with SRL to handle TFX_PHASEB events.
 */

int phb_hdlr( )
{

   int dev = sr_getevtdev( );

   if (ATFX_ECM(dev) == DFS_ECM) {
      printf("ECM was used during transfer\n");
      .
      .
   }
```

```
   return(0);
}
```

■ Errors

If an invalid fax channel device handle is specified in **dev**, this function fails and returns AT_FAILURE.

■ See Also

- **fx_setparm( )** (FC_TXCODING parameter)

Name: long ATFX_ESTAT(dev)
Inputs: int dev • fax channel device handle
Returns: Phase E information if successful
AT_FAILURE if error
Includes: srllib.h
dxxxlib.h
faxlib.h
Category: extended attribute
Mode: synchronous
Platform: DM3, Springware

Description

The **ATFX_ESTAT( )** function returns Phase E information describing errors that occurred during the T.30 fax protocol.

When the **fx_rcvfax( )**, **fx_rcvfax2( )** or **fx_sendfax( )** function returns a -1, or **ATDV_LASTERR( )** returns a EFX_DISCONNECT error, use this function to determine the reason for disconnection. See *Appendix C* for Phase E values returned.

If a T.30 protocol error does not occur, **ATFX_ESTAT( )** returns a zero.

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

DF_IOTT iott[10];
int dev;

/*
 * Open the channel using fx_open( ) and obtain the
 * FAX device handle in dev.
 */
.
.
/* Call fx_sendfax( ) after setting up the DF_IOTT array. */
if (fx_sendfax(dev, iott, EV_SYNC) == -1) {
   printf("Error - %s (error code %d)\n",
      ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
      /* Perform system error processing */
   }   else if (ATDV_LASTERR(dev) == EFX_DISCONNECT) {
      /*
       * Additional error processing - check Phase E status to
       * determine cause of error during fax protocol.
       */
      printf("Phase E status: %ld\n", ATFX_ESTAT(dev));
      }
}
.
.
```

■ Errors

This function fails and returns AT_FAILURE if an invalid fax channel device handle is specified in **dev**.

Name: char * ATFX_FXVERSION(dev)
Inputs: int dev • fax channel device handle
Returns: fax library version number string if successful
AT_FAILUREP if error
Includes: srllib.h
dxxxlib.h
faxlib.h
Category: extended attribute
Mode: synchronous
Platform: DM3, Springware

■ Description

The **ATFX_FXVERSION( )** returns the fax library version number string (format: *x.xx*).

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int dev;

/*
 * Open the channel using fx_open( ) and obtain the
 * FAX device handle in dev.
 */
.
.
/*
 * Optional display of version number of Fax
 * library.
 */
printf("%s\n", ATFX_FXVERSION(dev));
```

■ Errors

This function fails and returns AT_FAILUREP if an invalid fax channel device handle is specified in **dev**.

Name:	DF_IOTT * ATFX_LASTIOTT(dev)
Inputs:	int dev • fax channel device handle
Returns:	pointer to last DF_IOTT if successful AT_FAILUREP if error
Includes:	srllib.h dxxxlib.h faxlib.h
Category:	extended attribute
Mode:	synchronous
Platform:	DM3, Springware

■ Description

The **ATFX_LASTIOTT( )** function returns a pointer to the last processed DF_IOTT structure.

Use this function to determine which DF_IOTT was processed when an error occurred.

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

DF_IOTT iott[10];
DF_IOTT * lastiotp;
long pagenum;
int dev;

/*
 * Open the channel using fx_open( ) and obtain the
 * FAX device handle in dev.
 */
.
.
/* Call fx_sendfax( ) after setting up the DF_IOTT array. */
if (fx_sendfax(dev, iott, EV_SYNC) == -1) {
```

```
   /*
    * Get pointer to DF_IOTT being processed when error
    * occurred.
    */
   lastiotp = ATFX_LASTIOTT(dev);
   /*
    * Page being processed within this DF_IOTT when error
    * occurred.
    */
   pagenum = ATFX_BADPAGE(dev);
}
.
.
```

■ Errors

This function fails and returns AT_FAILUREP if an invalid fax channel device handle is specified in **dev**.

Name:	long ATFX_PGXFER(dev)
Inputs:	int dev • fax channel device handle
Returns:	number of pages transferred if successful AT_FAILURE if error
Includes:	srllib.h dxxxlib.h faxlib.h
Category:	extended attribute
Mode:	synchronous
Platform:	DM3, Springware

Description

The **ATFX_PGXFER( )** function returns the number of transferred fax pages during the current fax call.

This function typically indicates the number of pages transferred by the **fx_rcvfax( )**, **fx_rcvfax2( )** or **fx_sendfax( )** function. You can issue **ATFX_PGXFER( )** any time during a fax transfer to return the cumulative page count for the fax session.

The final page count value returned by **ATFX_PGXFER( )** at the end of a fax session remains available to the application until a new *send* or *receive* is initiated on that channel.

NOTE: In turnaround polling, this function provides a cumulative page count to include both sending and receiving on the specified channel.

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int dev;

/*
 * Open the channel using fx_open( ) and obtain the
 * FAX device handle in dev.
 */
.
.
/*
 * Call fx_rcvfax( ) to receive a fax into the file
 * "myfax.tif".
 */
if (fx_rcvfax(dev, "myfax.tif", DF_TIFF|DF_NOPOLL|EV_SYNC)
   == -1) {
   printf("Error - %s (error code %d)\n",
      ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
      /* Perform system error processing */
   }
   /*
    * Additional error processing - check Phase E status to
    * determine cause of error during fax protocol.
    */
   printf("Phase E status: %ld\n", ATFX_ESTAT(dev));
   .
   .
}
printf("Number of pages received: %ld\n", ATFX_PGXFER(dev));
```

Errors

This function fails and returns AT_FAILURE if an invalid fax channel device handle is specified in **dev**.

Name:	long ATFX_PHDCMD(dev)
Inputs:	int dev • fax channel device handle
Returns:	Phase D command after TFX_PHASED event if successful AT_FAILURE if error
Includes:	srllib.h dxxxlib.h faxlib.h
Category:	extended attribute
Mode:	synchronous
Platform:	DM3, Springware

Description

The **ATFX_PHDCMD( )** function returns the Phase D command. The following are valid Phase D command values:

DFS_EOP	End of Procedure – Terminate fax session. Progress to Phase E and disconnect fax call.
DFS_MPS	Multi-Page Signal – End of current fax document page, more fax data to follow. Next fax document page is in the same format as the current page, so proceed directly to Phase C.
DFS_EOM	End of Message – End of current fax document page, more fax data to follow. Return to Phase B and negotiate parameters for next fax document page.
DFS_POLL	A poll request was sent
DFS_PRI_EOP	Request for operator intervention sent (PRI_EOP) (not supported on DM3 boards)
DFS_PRI_MPS	Request for operator intervention sent (PRI_MPS) (not supported on DM3 boards)
DFS_PRI_EOM	Request for operator intervention sent (PRI_EOM) (not supported on DM3 boards)

See *Appendix B* for Phase D command details.

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

The data provided by this function is updated each time the fax transfer completes Phase D of the T.30 protocol.

To monitor the Phase D commands, you must enable Phase D events and issue **ATFX_PHDCMD( )** when the TFX_PHASED event occurs. Note that since Phase D also occurs at the end of a send or receive when a Phase D event is not generated, you can also issue this function after a TFX_FAXSEND or TFX_FAXRECV event.

The final Phase D command value returned by **ATFX_PHDCMD( )** at the end of a fax session remains available to the application until a new send or receive is initiated on that channel.

NOTE: Between multiple Phase D completions during the same fax session, **ATFX_PHDCMD( )** returns the previously completed Phase D command information.

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int dev;

/* Handler for Phase D events. */
int phd_hdlr( );

main( )
{
   /*
    * Open the channel using fx_open( ) and obtain the
    * FAX device handle in dev.
    */
   .
   .
   /*
    * Install handler (phd_hdlr( )) using sr_enbhdlr( ) to service
    * TFX_PHASED events.
    */
    if (sr_enbhdlr(dev, TFX_PHASED, phd_hdlr) == -1) {
```

```
        printf("Failed to install Phase D handler \n");
        return;
    }

   /*
    * Call fx_rcvfax( ) in synchronous mode to receive
    * TIFF/F file. Set DF_PHASED bit in mode field
    * to enable generation of Phase D events.
    */
   if (fx_rcvfax(dev,"fax.tif", EV_SYNC|DF_PHASED) == -1) {
      printf("Error - %s (error code %d)\n",
         ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
      if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
         /* Perform system error processing */
      }
   }
   /*
    * Examine Phase D command for last page.
    */
   printf("Phase D command: %ld\n", ATFX_PHDCMD(dev));
   .
   .
}

/*
 * Handler registered with SRL to handle TFX_PHASED events.
 */

int phd_hdlr( )
{

   int dev = sr_getevtdev( );

      /*
       * Examine Phase D command - e.g., DFS_MPS, DFS_EOM,
       * DFS_EOP.
       */
      phdcmd = ATFX_PHDCMD(dev);
      printf("Phase D command: %ld\n", phdcmd);
   .
   .
   return(0);
}
```

■ Errors

If one of the following conditions is present, this function fails and returns AT_FAILURE:

- An invalid fax channel device handle is specified in **dev.**
- The function is called prior to the completion of the first Phase D event.

Name:	long ATFX_PHDRPY(dev)
Inputs:	int dev • fax channel device handle
Returns:	Phase D reply after TFX_PHASED event if successful AT_FAILURE if error
Includes:	srllib.h dxxxlib.h faxlib.h
Category:	extended attribute
Mode:	synchronous
Platform:	DM3, Springware

■ Description

The **ATFX_PHDRPY( )** function returns the Phase D reply. The following are valid Phase D reply values:

DFS_MCF	Message confirmation - valid fax image received, ready for more pages
DFS_RTN	Retrain negative - bad fax image received, retrain and resend image
DFS_RTP	Retrain positive - valid fax image received but retraining required
DFS_PIP	Procedure interrupt positive - operator intervention request
DFS_PIN	Procedure interrupt negative - operator intervention request

See *Appendix B* for Phase D reply details.

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

The data provided by this function is updated each time the fax transfer completes Phase D of the T.30 protocol.

To monitor Phase D replies, you must enable Phase D events and issue **ATFX_PHDRPY( )** when the TFX_PHASED event occurs. Note that since Phase D also occurs at the end of a send or receive when a Phase D event is not generated, you can also issue this function after a TFX_FAXSEND or TFX_FAXRECV event.

The final Phase D reply value returned by **ATFX_PHDRPY( )** at the end of a fax session remains available to the application until a new send or receive is initiated on that channel.

NOTE: Between multiple Phase D completions during the same fax session, **ATFX_PHDRPY( )** returns the previously completed Phase D reply information.

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

DF_IOTT iott[10];
int dev;

/* Handler for Phase D events. */
int phd_hdlr( );

main( )
{
   /*
    * Open the channel using fx_open( ) and obtain the
    * FAX device handle in dev.
    */
   .
   .
   /*
    * Install handler (phd_hdlr( )) using sr_enbhdlr( ) to service
    * TFX_PHASED events.
    */
    if (sr_enbhdlr(dev, TFX_PHASED, phd_hdlr) == -1) {
        printf("Failed to install Phase D handler \n");
        return;
    }

   /*
    * Call fx_sendfax( ) in synchronous mode after setting
    * up the DF_IOTT array. Set DF_PHASED bit in mode field
    * to enable generation of Phase D events.
    */
   if (fx_sendfax(dev, iott, EV_SYNC|DF_PHASED) == -1) {
      printf("Error - %s (error code %d)\n",
         ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
```

```
      if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
         /* Perform system error processing */
      }
   }
   /*
    * Examine Phase D reply for last page.
    */
   printf("Phase D reply: %ld\n", ATFX_PHDRPY(dev));
   .
   .
}

/*
 * Handler registered with SRL to handle TFX_PHASED events.
 */

int phd_hdlr( )
{

   long phdrpy;
   int dev = sr_getevtdev( );

      /*
       * Examine Phase D reply - e.g., DFS_MCF, DFS_RTN,
       * DFS_RTP.
       */
      phdrpy = ATFX_PHDRPY(dev);
      printf("Phase D reply: %ld\n", phdrpy);
   .
   .
   return(0);
}
```

■ Errors

If one of the following conditions is present, this function fails and returns AT_FAILURE:

- An invalid fax channel device handle is specified in **dev.**
- The function is called prior to the completion of the first Phase D event.

Name:	long ATFX_RESLN(dev)
Inputs:	int dev • fax channel device handle
Returns:	Vertical resolution of the transferred page if successful AT_FAILURE if error
Includes:	srllib.h dxxxlib.h faxlib.h
Category:	extended attribute
Mode:	synchronous
Platform:	DM3, Springware

■ Description

The **ATFX_RESLN( )** function returns the vertical resolution of the page that is sent or received. Valid values are:

DF_RESHI	High vertical resolution (fine) - 196 lines or pels per inch
DF_RESLO	Low vertical resolution (coarse) - 98 lines or pels per inch

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

On DM3 boards, the data provided by the **ATFX_RESLN( )** function is updated each time the fax transfer completes Phase B (rather than Phase D) of the T.30 protocol. By enabling the Phase B event, you can issue **ATFX_RESLN( )** when the TFX_PHASEB event occurs.

On Springware boards, the data provided by this function is updated each time the fax transfer completes Phase D of the T.30 protocol.

To monitor the vertical resolution, you must enable Phase D events and issue **ATFX_RESLN( )** when the TFX_PHASED event occurs. Note that since Phase D also occurs at the end of a send or receive when a Phase D event is not

generated, you can also issue this function after a TFX_FAXSEND or TFX_FAXRECV event.

The final, vertical resolution value returned by **ATFX_RESLN( )** at the end of a fax session remains available to the application until a new send or receive is initiated on that channel.

NOTE: Between multiple Phase D completions during the same fax session, **ATFX_RESLN( )** returns the vertical resolution information from the previously completed page.

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

DF_IOTT iott[10];
int dev;

/* Handler for Phase D events. */
int phd_hdlr( );

main( )
{
   /*
    * Open the channel using fx_open( ) and obtain the
    * FAX device handle in dev.
    */
   .
   .
   /*
    * Install handler (phd_hdlr( )) using sr_enbhdlr( ) to service
    * TFX_PHASED events.
    */
    if (sr_enbhdlr(dev, TFX_PHASED, phd_hdlr) == -1) {
        printf("Failed to install Phase D handler \n");
        return;
    }
   /*
    * Call fx_sendfax( ) in synchronous mode after setting
    * up the DF_IOTT array. Set DF_PHASED bit in mode field
    * to enable generation of Phase D events.
    */
   if (fx_sendfax(dev, iott, EV_SYNC|DF_PHASED) == -1) {
      printf("Error - %s (error code %d)\n",
         ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
      if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
         /* Perform system error processing */
      }
   }
   /*
    * Vertical resolution of the page just transferred is
```

```
    * available at this point.
    */
   printf("Page was transferred at vertical resolution: %ld\n",
      ATFX_RESLN(dev));
   .
   .
}

/*
 * Handler registered with SRL to handle TFX_PHASED events.
 */

int phd_hdlr( )
{

   long phdrpy;
   int dev = sr_getevtdev( );

      /*
       * Vertical resolution of the page just transferred is
       * available at this point.
       */
      printf("Page was transferred at vertical resolution: %ld\n",
         ATFX_RESLN(dev));
   .
   .
   return(0);
}
```

■ Errors

If one of the following conditions is present, this function fails and returns AT_FAILURE:

- An invalid fax channel device handle is specified in **dev.**
- The function is called prior to the completion of the first Phase D event.

Name: long ATFX_RTNPAGES(dev)
Inputs: int dev • fax channel device handle
Returns: number of pages RTN was returned to the remote station
AT_FAILURE if error
Includes: srllib.h
dxxxlib.h
faxlib.h
Category: extended attribute
Mode: synchronous
Platform: Springware

■ Description

The **ATFX_RTNPAGES( )** function returns the number of RTN pages; that is, the number of received pages for which the receiver returned an RTN (Retrain Negative) message to the remote transmitter.

If an unacceptable percentage of bad scan lines is received for a fax page (controlled by the FC_RTN parameter in **fx_setparm( )**), an RTN is returned to the remote station. The received page is still written to the specified receive file. After receiving the RTN, the transmitter may or may not retransmit the same page.

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

At the completion of the **fx_rcvfax( )** or **fx_rcvfax2( )** function, the receiver application may call **ATFX_RTNPAGES( )** to determine the *total* number of RTN pages received.

NOTE: To monitor the RTN status for each page of a multi-page fax, you must enable Phase D events (DF_PHASED) in each **fx_rcvfax( )** or **fx_rcvfax2( )** function called and issue **ATFX_PHDRPY( )** to monitor for an RTN message.

The data provided by this function is updated each time the fax transfer completes Phase D of the T.30 protocol.

To monitor the number of RTN pages, you must enable Phase D events and issue **ATFX_RTNPAGES( )** when the TFX_PHASED event occurs. Note that since Phase D also occurs at the end of a send or receive when a Phase D event is not generated, you can issue this function after a TFX_FAXSEND or TFX_FAXRECV event.

The final RTN page count value returned by **ATFX_RTNPAGES( )** at the end of a fax session remains available to the application until a new send or receive is initiated on that channel.

If the receive file is a TIFF/F file, you can examine the tag 'BadFaxLines' for each page to determine the page's image quality (see TIFF/F tags in *Appendix A*).

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int dev;
long badpages;

/*
 * Open the channel using fx_open( ) and obtain the
 * FAX device handle in dev.
 */
.
.
/*
 * Call fx_rcvfax( ) to receive a fax into the file
 * "myfax.tif".
 */
if (fx_rcvfax(dev, "myfax.tif", DF_TIFF|EV_SYNC) == -1) {
   printf("Error - %s (error code %ld)\n",
      ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
      /* Perform system error processing */
   }
   /*
    * Additional error processing - check Phase E status to
    * determine cause of error during fax protocol.
    */
   printf("Phase E status: %ld\n", ATFX_ESTAT(dev));
   .
   .
}
/*
 * Check if the received file has any pages for
 * which a RTN was returned.
 */
badpages = ATFX_RTNPAGES(dev);
```

■ Errors

This function fails and returns AT_FAILURE if an invalid fax channel device handle is specified in **dev.**

Name:	long ATFX_SCANLINES(dev)
Inputs:	int dev • fax channel device handle
Returns:	total number of scan lines in last page transferred if successful AT_FAILURE if error
Includes:	srllib.h dxxxlib.h faxlib.h
Category:	extended attribute
Mode:	synchronous
Platform:	DM3, Springware

■ Description

The **ATFX_SCANLINES( )** function returns the number of scan lines in the last page transmitted or received. This information is available at the end of Phase D for every page sent or received.

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

The data provided by this function is updated each time the fax transfer completes Phase D of the T.30 protocol.

To monitor the number of scan lines, you must enable Phase D events and issue **ATFX_SCANLINES( )** when the TFX_PHASED event occurs. Note that since Phase D also occurs at the end of a send or receive when a Phase D event is not generated, you can issue this function after a TFX_FAXSEND or TFX_FAXRECV event.

The final scan line value returned by **ATFX_SCANLINES( )** at the end of a fax session remains available to the application until a new send or receive is initiated on that channel.

NOTE: Between multiple Phase D completions during the same fax session, **ATFX_SCANLINES( )** returns the total number of scan lines from the previously completed page.

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int dev;

/* Handler for Phase D events. */
int phd_hdlr( );

main( )
{
   /*
    * Open the channel using fx_open( ) and obtain the
    * FAX device handle in dev.
    */
   .
   .
   /*
    * Install handler to service TFX_PHASED events.
    */
    if (sr_enbhdlr(dev, TFX_PHASED, phd_hdlr) == -1) {
        printf("Failed to install Phase D handler \n");
        return;
}

   /*
    * Call fx_rcvfax( ) in asynchronous mode to receive
    * TIFF/F file. Set DF_PHASED bit in mode field
    * to enable generation of Phase D events.
    */
   if (fx_rcvfax(dev,"fax.tif", EV_ASYNC|DF_PHASED) == -1) {
      printf("Error - %s (error code %d)\n",
         ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
      if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
         /* Perform system error processing */
      }
   }
   .
   .
/*
 * Handler registered with SRL to handle TFX_PHASED events.
 */

int phd_hdlr( )
{

   int dev = sr_getevtdev( );

   /*
    * Total number of scan lines on the page just
    * received is available at this point.
    */
      printf("Total number of scan lines in page received: %ld\n",
         ATFX_SCANLINES(dev));
   .
   .
   return(0);
```

```
}
```

■ Errors

If one of the following conditions is present, this function fails and returns AT_FAILURE:

- An invalid fax channel device handle is specified in **dev.**
- The function is called prior to the completion of the first Phase D event.

Name:	long ATFX_SPEED(dev)
Inputs:	int dev • fax channel device handle
Returns:	final transfer speed if successful AT_FAILURE if error
Includes:	srllib.h dxxxlib.h faxlib.h
Category:	extended attribute
Mode:	synchronous
Platform:	DM3, Springware

■ Description

The **ATFX_SPEED( )** function returns the fax transfer speed (in baud) of the last transmitted page. This information is available after Phase B is completed. For transfers that do not renegotiate Phase B, issuing **ATFX_SPEED( )** at the completion of a fax session returns the transfer baud rate for the entire session.

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

The data provided by this function is updated each time the fax transfer completes Phase B of the T.30 protocol. After a fax session terminates, the value from the last fax transfer is available until the start of a new fax session.

For example, **ATFX_SPEED( )** returns the equate DF_14400BAUD to indicate 14.4 Kbps transfer speed.

NOTE: Between multiple Phase B negotiations during the same fax session, **ATFX_SPEED( )** returns the Phase B transfer rate information from the previously completed Phase B negotiation.

To monitor the transfer speed for each completed Phase B negotiation, you must enable Phase B events (DF_PHASEB in the **fx_rcvfax( )**, **fx_rcvfax2( )** or **fx_sendfax( )** function) and issue **ATFX_SPEED( )** when the Phase B event (TFX_PHASEB) occurs.

If the application has enabled the generation of Phase B events, you can determine the baud rate set for the transmission by calling the **ATFX_SPEED( )** function in the handler routine for the Phase B event.

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

DF_IOTT iott[10];
int dev;

/* Handler for Phase B events. */
int phb_hdlr( );

main( )
{
   /*
    * Open the channel using fx_open( ) and obtain the
    * FAX device handle in dev.
    */
   .
   .
   /*
    * Install handler to service TFX_PHASEB events.
    */
    if (sr_enbhdlr(dev, TFX_PHASEB, phb_hdlr) == -1) {
        printf("Failed to install Phase B handler \n");
        return;
}

   /*
    * Call fx_sendfax( ) in asynchronous mode after setting
    * up the DF_IOTT array. Set DF_PHASEB bit in mode field
    * to enable generation of Phase B events.
    */
   if (fx_sendfax(dev, iott, EV_ASYNC|DF_PHASEB) == -1) {
      printf("Error - %s (error code %d)\n",
         ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
      if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
         /* Perform system error processing */
      }
   }
   .
   .
}

/*
 * Handler registered with SRL to handle TFX_PHASEB events.
 */

int phb_hdlr( )
{

   int dev = sr_getevtdev( );
```

```
   /* Remote data rate capability. */
   printf("Data rate for fax transmission: %ld\n",
      ATFX_SPEED(dev));
   .
   .
   return(0);
}
```

■ Errors

If one of the following conditions is present, this function fails and returns AT_FAILURE:

- An invalid fax channel device handle is specified in **dev.**
- The function is called prior to the completion of the first Phase B event.

Name: long ATFX_STATE(dev)
Inputs: int dev • fax channel device handle
Returns: current state of fax channel device if successful
AT_FAILURE if invalid fax channel device handle specified
Includes: srllib.h
dxxxlib.h
faxlib.h
Category: extended attribute
Mode: synchronous
Platform: DM3, Springware

■ Description

The **ATFX_STATE( )** function returns the current state of the fax channel device specified in **dev**.

The function parameter is defined as follows:

Parameter	Description
dev	Specifies the valid fax channel device handle obtained when the fax channel was opened.

This function returns one of the following values indicating the current state of the channel:

CS_IDLE	fax channel is idle
CS_SENDFAX	fax channel is transmitting (**fx_sendfax( )** active)
CS_RECVFAX	fax channel is receiving (**fx_rcvfax( )** or **fx_rcvfax2( )** active)
CS_FAXIO	fax channel is between pages OR when *send* or *receive* functions have returned but the fax session is still active

NOTE: A fax device channel is idle when no I/O is active on the channel.

Example

```
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

DF_IOTT iott[10];
int dev;


main( )
{
   /*
    * Open the channel using fx_open( ) and obtain the
    * FAX device handle in dev.
    */
   .
   .
   .
   .
   /*
    * Check state of the FAX channel.
    * If idle, call fx_sendfax( ) in asynchronous mode after setting
    * up the DF_IOTT array.
    */
   if (ATFX_STATE(dev) == CS_IDLE) {
      if (fx_sendfax(dev, iott, EV_ASYNC) == -1) {
         printf("Error - %s (error code %d)\n",
            ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
         if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
            /* Perform system error processing */
         }
      }
   }
   .
   .
}
```

Errors

This function fails and returns AT_FAILURE if an invalid fax channel device handle is specified in **dev**.

Name: long ATFX_TERMMSK(dev)
Inputs: int dev • fax channel device handle
Returns: bitmap of termination reasons if successful
AT_FAILURE if error
Includes: srllib.h
dxxxlib.h
faxlib.h
Category: extended attribute
Mode: Synchronous
Platform: DM3, Springware

■ Description

The **ATFX_TERMMSK( )** function returns a bitmap of termination reasons.

Call this function after the successful completion of **fx_rcvfax( )**, **fx_rcvfax2( )**, or **fx_sendfax( )** to determine the termination reason. Valid values are:

TM_FXTERM	Normal completion of fax send/receive
TM_POLLED	Poll request received from transmitter
TM_VOICEREQ	Voice request (operator intervention) issued/received (not supported on DM3 boards)

The termination reason is available until the next send or receive is issued on the channel.

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int dev;
```

```
long lTermMask;

/*
 * Open the channel using fx_open( ) and obtain the
 * FAX device handle in dev.
 */
.
.
/*
 * Call fx_rcvfax( ) to receive a fax into the file
 * "myfax.tif".
 */
if (fx_rcvfax(dev, "myfax.tif", DF_TIFF|EV_SYNC) == -1) {
   printf("Error - %s (error code %ld)\n",
      ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
      /* Perform system error processing */
   }
   /*
    * Additional error processing - check Phase E status to
    * determine cause of error during fax protocol.
    */
   printf("Phase E status: %ld\n", ATFX_ESTAT(dev));
   .
   .
}
/* Check termination reasons. */
lTermMask = ATFX_TERMMSK (dev);
/* Evaluate success/failure separately because all bits are turned on for -1 */
if (lTermMask == -1) {
   printf("Failed to retrieve the termination mask!\n");
   /* Process error */

}
if (lTermMask  & TM_POLLED) {
   printf("Poll received\n");
   /* Respond to poll by issuing a fx_sendfax( ). */
   .
   .
}
if (lTermMask  & TM_VOICEREQ) {
   printf("Voice request received\n");
   /* Respond to voice request (PRI_EOP). */
   .
   .
}
```

■ Errors

This function fails and returns AT_FAILURE if an invalid fax channel device handle is specified in **dev**.

Name:	long ATFX_TFBADTAG(dev)
Inputs:	int dev • fax channel device handle
Returns:	bad TIFF/F tag number if successful AT_FAILURE if error
Includes:	srllib.h dxxxlib.h faxlib.h
Category:	extended attribute
Mode:	synchronous
Platform:	Springware

■ Description

The **ATFX_TFBADTAG( )** function returns the invalid TIFF/F tag number when **ATDV_LASTERR( )** returns an EFX_BADTAG error. This error is returned during the transmission of a TIFF/F file if an invalid TIFF/F tag value is found.

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

The invalid TIFF/F tag value returned by **ATFX_TFBADTAG( )** at the end of a fax session remains available to the application until a new send is initiated on that channel.

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

DF_IOTT iott;
int dev;

/*
 * Open the channel using fx_open( ) and obtain the
 * FAX device handle in dev.
 */
.
.
/* Call fx_sendfax( ) after setting up the DF_IOTT. */
if (fx_sendfax(dev, &iott, EV_SYNC) == -1) {
   if (ATDV_LASTERR(dev) == EFX_BADTAG) {
      printf("Bad Tag in TIFF/F file. Tag number %ld\n",
         ATFX_TFBADTAG(dev));
      .
      .
   }
}
.
.
```

■ Errors

This function fails and returns AT_FAILURE if an invalid fax channel device handle is specified in **dev**.

Name:	long ATFX_TFNOTAG(dev)
Inputs:	int dev • fax channel device handle
Returns:	missing TIFF/F mandatory tag number if successful AT_FAILURE if error
Includes:	srllib.h dxxxlib.h faxlib.h
Category:	extended attribute
Mode:	synchronous
Platform:	Springware

■ Description

The **ATFX_TFNOTAG( )** function returns missing TIFF/F mandatory tag number if **ATDV_LASTERR( )** returns an EFX_BADTIF error.

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

The missing TIFF/F tag value returned by **ATFX_TFNOTAG( )** at the end of a fax session remains available to the application until a new send is initiated on that channel.

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

DF_IOTT iott;
int dev;

/*
 * Open the channel using fx_open( ) and obtain the
 * FAX device handle in dev.
 */
.
.
/* Call fx_sendfax( ) after setting up the DF_IOTT. */
if (fx_sendfax(dev, &iott, EV_SYNC) == -1) {
   if (ATDV_LASTERR(dev) == EFX_BADTIF) {
      printf("Missing Tag in TIFF/F file. Tag number %ld\n",
         ATFX_TFNOTAG(dev));
      .
      .
   }
}
.
.
```

■ Errors

This function fails and returns AT_FAILURE if an invalid fax channel device handle is specified in **dev**.

Name: long ATFX_TFPGBASE(dev)
Inputs: int dev • fax channel device handle
Returns: base page number scheme for TIFF/F file if successful
AT_FAILURE if error
Includes: srllib.h
dxxxlib.h
faxlib.h
Category: extended attribute
Mode: synchronous
Platform: Springware

■ Description

The **ATFX_TFPGBASE( )** returns the base page numbering scheme for the most recently transmitted TIFF/F file. Valid values are:

TF_BASE0	First page number is zero (TIFF/F standard)
TF_BASE1	First page number is one

NOTE: According to TIFF/F requirements, the pages of a multi-page TIFF/F file are numbered internally starting at zero, but some utilities may not adhere strictly to these requirements. See the FC_TFPGBASE parameter in the **fx_setparm( )** function reference for more information.

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

DF_IOTT iott;
int dev;
long tfpgbase;

/*
```

```
 * Open the channel using fx_open( ) and obtain the
 * FAX device handle in dev.
 */
.
.
/*
 * If you are unsure of the page numbering scheme used in
 * the TIFF/F file to be transmitted, call fx_setparm( ) to
 * set the FC_TFPGBASE to TF_AUTOPG to enable the auto-
 * paging mode (determines the page numbering scheme
 * automatically) (Note: A multi-page TIFF/F file should
 * have its pages internally numbered starting at zero, but
 * some utilities may not adhere strictly to TIFF/F
 * requirements).
 */
.
.
/*
 * Call fx_sendfax( ) after setting up the DF_IOTT to
 * send the TIFF/F file.
 */
if (fx_sendfax(dev, &iott, EV_SYNC) == -1) {
   printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
         ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
      /* Perform system error processing */
   }
   .
   .
}

/* Determine page numbering scheme. */
tfpgbase = ATFX_TFPGBASE(dev);

/*
 * Once the page numbering scheme of the TIFF/F file has
 * been determined, the FC_TFPGBASE parameter may be set
 * correctly for future transmission of this file (or files)
 * created by the TIFF utility that was used.
 */
```

■ Errors

This function fails and returns AT_FAILURE if an invalid fax channel device handle is specified in **dev**.

Name:	long ATFX_TRCOUNT(dev)
Inputs:	int dev • fax channel device handle
Returns:	number of bytes transferred for fax transfer if successful AT_FAILURE if invalid fax device handle specified
Includes:	srllib.h dxxxlib.h faxlib.h
Category:	extended attribute
Mode:	synchronous
Platform:	DM3, Springware

■ Description

The **ATFX_TRCOUNT( )** function returns the number of bytes transferred so far during the current *send* or *receive* on the fax channel device specified in **dev**.

Parameter	Description
dev	Specifies the valid fax channel device handle obtained when the fax channel was opened.

The transfer byte count value returned by **ATFX_TRCOUNT( )** at the end of a fax session remains available to the application until a new fax session is initiated on that fax device channel.

On DM3 boards, when an error occurs during a send, this value is reset to zero (0).

■ Example

```
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

DF_IOTT iott[10];
int dev;


main( )
{
   /*
    * Open the channel using fx_open( ) and obtain the
    * FAX device handle in dev.
    */
   .
   .
   /*
    * Call fx_sendfax( ) in synchronous mode after setting
    * up the DF_IOTT array.
    */
   if (fx_sendfax(dev, iott, EV_SYNC) == -1) {
      printf("Error - %s (error code %d)\n",
         ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
      if (ATDV_LASTERR(dev)==EDX_SYSTEM) {

      }
   }
   .
   .
   /* Check the transfer count */
   printf("Transfer count is %d\n", ATFX_TRCOUNT(dev));
   .
   .
}
```

■ Errors

This function fails and returns AT_FAILURE if an invalid fax channel device handle is specified in **dev**.

Name: long ATFX_WIDTH(dev)
Inputs: int dev • fax channel device handle
Returns: Width of transferred page if successful
AT_FAILURE if error
Includes: srllib.h
dxxxlib.h
faxlib.h
Category: extended attribute
Mode: synchronous
Platform: DM3, Springware

Description

The **ATFX_WIDTH( )** function returns the decimal value of the negotiated width (in pixels per line) of the fax page transmitted. Valid values are:

1728	(pixels per line)
2048	(pixels per line)
2432	(pixels per line)

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.

On DM3 boards, the data provided by the **ATFX_WIDTH( )** function is updated each time the fax transfer completes Phase B (rather than Phase D) of the T.30 protocol. By enabling the Phase B event, you can issue **ATFX_WIDTH( )** when the TFX_PHASEB event occurs.

On Springware boards, the data provided by this function is updated each time the fax transfer completes Phase D of the T.30 protocol.

To monitor width, you must enable Phase D events and issue **ATFX_WIDTH( )** when the TFX_PHASED event occurs. Note that since Phase D also occurs at the

end of a send or receive when a Phase D event is not generated, you can also issue this function after a TFX_FAXSEND or TFX_FAXRECV event.

The width value returned by **ATFX_WIDTH( )** at the end of a fax session remains available to the application until a new send or receive is initiated on that channel.

NOTE: Between multiple Phase D completions during the same fax session, **ATFX_WIDTH( )** returns the width information from the previously completed page.

If you have enabled generation of Phase D events, you can call **ATFX_WIDTH( )** in the handler routine for the Phase D event to determine the width of the transferred page (see programming example).

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

DF_IOTT iott[10];
int dev;

/* Handler for Phase D events. */
int phd_hdlr( );

main( )
{
   /*
    * Open the channel using fx_open( ) and obtain the
    * FAX device handle in dev.
    */
   .
   .
   /*
    * Install handler to service TFX_PHASED events.
    */
    if (sr_enbhdlr(dev, TFX_PHASED, phd_hdlr) == -1) {
        printf("Failed to install Phase D handler \n");
        return;
}

   /*
    * Call fx_sendfax( ) in synchronous mode after setting
    * up the DF_IOTT array. Set DF_PHASED bit in mode field
    * to enable generation of Phase D events.
    */
   if (fx_sendfax(dev, iott, EV_SYNC|DF_PHASED) == -1) {
      printf("Error - %s (error code %d)\n",
         ATDV_ERRMSGP(dev), ATDV_LASTERR(dev));
```

```
      if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
         /* Perform system error processing */
      }
   }
   /*
    * Width of the page just transferred is available at
    * this point.
    */
   printf("Page width: %ld\n", ATFX_WIDTH(dev));
   .
   .
}

/*
 * Handler registered with SRL to handle TFX_PHASED events.
 */

int phd_hdlr( )
{

   int dev = sr_getevtdev( );

   /*
    * Width of the page just transferred is available at
    * this point.
    */
   printf("Page width: %ld\n", ATFX_WIDTH(dev));
   .
   .
   return(0);
}
```

■ Errors

If one of the following conditions is present, this function fails and returns AT_FAILURE:

- An invalid fax channel device handle is specified in **dev.**
- The function is called prior to the completion of the first Phase D event.

Name: int fx_close(dev)
Inputs: int dev • fax channel device handle
Returns: 0 if successful
-1 if error
Includes: srllib.h
dxxxlib.h
faxlib.h
Category: resource management
Mode: synchronous
Platform: DM3, Springware

■ Description

The **fx_close( )** function closes a fax channel device opened previously using **fx_open( )**. It releases the handle and breaks any link the calling process has with the fax device channel through this handle, regardless of whether the device is busy or idle.

NOTE: **fx_close( )** disables the generation of all fax events; it does not affect the hookstate or parameter settings for the voice channel device.

Parameter	Description
dev	Specifies the valid fax device handle obtained when the channel device was opened.

■ Cautions

- Once a fax channel device is closed, a process can no longer perform an action on this fax channel device using this device handle. Other handles for this channel device that exist in the same process or in other processes are still valid. The only process affected by **fx_close( )** is the process that called the function.
- **fx_close( )** does not affect any action occurring on a fax channel device; it only breaks the link between the calling process and the fax channel device by freeing the specified fax channel device handle. Other links through different device handles are still valid.

- Do not use the Windows **close( )** function to close a fax channel device. Unpredictable results will occur.
- **fx_close( )** discards any outstanding fax events on the fax handle.
- On DM3 boards, if an **fx_close( )** is sent in the middle of a fax transmission, the fax is aborted (which is equivalent to issuing **fx_stopch( )** and **fx_close( )**). Do not issue an **fx_close( )** during a fax send.

Example

```
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

main()
{
   int dev;    /* Fax channel device handle. */

/* Open the Voice channel device using dx_open( ). */
.
.
/* Open the FAX channel device. */
   if ((dev = fx_open("dxxxB1C1", NULL)) == -1) {
      /* Error opening device */
      /* Perform system error processing */
      exit(1);
   }
   /* FAX transfers (send/receive) calling FAX API functions using dev. */
   .
   .
/* Close the FAX channel device. */
   if (fx_close(dev) == -1) {
      /* Error closing device. */
      printf("Error closing channel\n");
       printf("Error - %s (error code %d)\n",
         ATDV_ERRMSGP(dev), ATDV_LASTERR(dev));
      if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
         /* Perform system error processing */
      }
      exit(1);
   }
   .
   .
}
```

Errors

If this function returns -1 to indicate failure, a system error has occurred. On Linux, check the global variable errno for more information. On Windows, use

dx_fileerrno( ) to obtain the system error value. Refer to the **dx_fileerrno( )** function in the *Voice API Library Reference* for a list of the possible system error values.

■ See Also

- **fx_open( )**

Name:	int fx_getctinfo(chdev, ct_devinfop)	
Inputs:	int chdev	• fax channel device handle
	CT_DEVINFO *ct_devinfop	• pointer to device information structure
Returns:	0 on success -1 on failure	
Includes:	srllib.h dxxxlib.h faxlib.h	
Category:	TDM bus routing	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **fx_getctinfo( )** function returns information about a fax channel device handle. Use this function to identify whether a device belongs to DM3 hardware or Springware hardware.

Parameter	Description
chdev	Specifies the valid fax channel device handle obtained when the channel was opened using **fx_open**( ).
ct_devinfop	Specifies a pointer to the data structure CT_DEVINFO.

On return from the function, the CT_DEVINFO structure contains the relevant information. The valid values for each member of the CT_DEVINFO structure are defined in *ctinfo.h*, which is referenced by *dxxxlib.h*. For details on this data structure, see the *Voice API Library Reference*.

■ Cautions

This function will fail if an invalid fax channel device handle is specified.

■ Example

```
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

main()
{
   int  chdev;                 /* Channel device handle */
   CT_DEVINFO ct_devinfo;      /* Device information structure */

   /* Open board 1 channel 1 devices */
   if ((chdev = fx_open("dxxxB1C1", 0)) == -1) {
     printf("Cannot open channel\n");
     /* Perform system error processing */
     exit(1);
   }

   /* Get Device Information */
   if (fx_getctinfo(chdev, &ct_devinfo) == -1) {
     printf("Error message = %s", ATDV_ERRMSGP(chdev));
     exit(1);
   }

   printf("%s Product Id = 0x%x, Family = %d, Mode = %d, Network = %d, "
        " Bus mode = %d, Encoding = %d", ATDV_NAMEP(chdev), ct_devinfo.ct_prodid,
        ct_devinfo.ct_devfamily, ct_devinfo.ct_devmode, ct_devinfo.ct_nettype,
        ct_devinfo.ct_busmode, ct_devinfo.ct_busencoding);
}
```

■ Errors

If the function returns -1, use the SRL Standard Attribute function **ATDV_LASTERR( )** to obtain the error code or use **ATDV_ERRMSGP( )** to obtain a descriptive error message. One of the following error codes may be returned:

Equate	Returned When
EFX_BADPARM	Invalid value for fax parameter (on DM3 boards only)
EDX_BADPARM	Invalid value for parameter (on Springware boards)
EDX_SH_BADEXTTS	TDM bus time slot is not supported at current clock rate
EDX_SH_BADINDX	Invalid Switch Handler library index number
EDX_SH_BADTYPE	Invalid channel type (voice, analog, etc.)
EDX_SH_CMDBLOCK	Blocking command is in progress
EDX_SH_LIBBSY	Switch Handler library is busy

Equate	Returned When
EDX_SH_LIBNOTINIT	Switch Handler library is uninitialized
EDX_SH_MISSING	Switch Handler is not present
EDX_SH_NOCLK	Switch Handler clock fallback failed
EDX_SYSTEM	Operating system error

See Also

- **dt_getctinfo( )**
- **dx_getctinfo( )**

Name:	int fx_getDCS(dev,dcs_buf)	
Inputs:	int dev	• fax channel device handle
	DF_DCS * dcs_buf	• pointer to DF_DCS structure
Returns:	0 if success	
	-1 if failure	
Includes:	srllib.h	
	dxxxlib.h	
	faxlib.h	
Category:	miscellaneous	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **fx_getDCS( )** function returns the most recent DCS message (T.30 Digital Command Signal), if available, for the specified channel.

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.
dcs_buf	A pointer to the DF_DCS structure where the DCS message information is stored.

The DCS message contains information about negotiated settings between the transmitter and receiver. The DCS message is sent by the transmitter to the receiver as part of Phase B negotiation of a fax transfer.

NOTE: Use this function only when your application requires the specific Phase B negotiation information provided in the DCS message. For most applications, using **fx_getDCS( )** to retrieve the DCS message information is not required because the fax extended attribute functions provide access to most of the information contained in the DCS message; see **ATFX_RESLN( )**, **ATFX_SPEED( )**, **ATFX_WIDTH( )**.

The most recent DCS message sent from the transmitter is available to the application after the completion of the first Phase B negotiation. If available, the DCS message can be retrieved after each Phase B negotiation during the **fx_sendfax( )**, **fx_rcvfax( )** or **fx_rcvfax2( )** function call. The DCS message

information remains valid until the next Phase B negotiation is completed for the current function call or until a new *send* or *receive* is initiated.

To determine when the DCS message is available, call **ATFX_BSTAT( )**. This function returns a bitmap with the DFS_DCS bit set indicating that the transmitter's DCS message is available.

NOTE: Phase B negotiation takes place at the beginning of a fax *send* or *receive* function call and after a T.30 End of Message (EOM) is sent by the transmitter station during a fax *send* or *receive* function call.

For DCS message details, see the ITU-T publication *Procedures for Document Facsimile Transmission in the General Switched Telephone Network, Recommendation T.30* (see *Section 1.3.2. Other Publications).*

■ Example

```
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

DF_IOTT iott[10];

/* Handler for Phase B events. */
int phb_hdlr( );

main( )
{
   int voxdev;   /* Voice channel device handle. */
   int dev;   /* Fax channel device handle. */

   /*
    * Open the channel using dx_open( ) to obtain the
    * VOICE device handle in voxdev.
    * Open the channel using fx_open( ) to obtain the FAX channel
    * device handle in dev.
    */
   .
   .
   /*
    * Install handler using sr_enbhdlr( ) to service
    * TFX_PHASEB events.
    */
    if (sr_enbhdlr(dev, TFX_PHASEB, phb_hdlr) == -1) {
        printf("Failed to install Phase B handler \n");
        return;
    }

   /*
    * Call fx_sendfax( ) in asynchronous mode after setting
    * up the DF_IOTT array.  Set DF_PHASEB bit in mode field
```

```
    * to enable generation of Phase B events.
    */
   if (fx_sendfax(dev, iott, EV_ASYNC|DF_PHASEB) == -1) {
      printf("Error - %s (error code %d)\n",
         ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
      if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
         /* Perform system error processing */
      }
   }
   .
   .
}

/*
 * Handler registered with SRL to handle TFX_PHASEB events.
 */

int phb_hdlr( )
{
   int dev = sr_getevtdev( );
   DF_DCS dcs_buf;

   if (ATFX_BSTAT(dev) & DFS_DCS) {
      /* T.30 DCS available. */
      if (fx_getDCS(dev, &dcs_buf) == -1) {
         /* Error processing */
         .
         .
      } else {
         /* Application specific analysis of the DCS */
         .
         .
      }
   }
   .
   .
   return(0);
}
```

■ Errors

ATDV_LASTERR( ) returns these fax error codes for the following reasons:

EFX_NODATA	The function is called before completion of the initial Phase B negotiation.
EFX_UNSUPPORTED	The function is called for an unsupported board.

See *Appendix D* for a list of error codes that may be returned for this function.

Name:	int fx_getDIS(dev,dis_buf)	
Inputs:	int dev	• fax channel device handle
	DF_DIS * dis_buf	• pointer to DF_DIS structure
Returns:	0 if success	
	-1 if failure	
Includes:	srllib.h	
	dxxxlib.h	
	faxlib.h	
Category:	miscellaneous	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **fx_getDIS( )** function returns the most recent DIS message (T.30 Digital Information Signal), if available, for the specified channel.

The DIS message contains information about the receiver's capabilities. The DIS message is sent by the receiver to the transmitter as part of Phase B negotiation of a fax transfer.

NOTE: Only use this function when your application requires specific receiver capability information provided in the DIS message. For most applications, retrieving the DIS message information is not required.

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.
dis_buf	A pointer to the DF_DIS structure where the DIS information is stored.

The most recent DIS message from the receiver is available to the application after the completion of the first Phase B negotiation. If available, the DIS message can be retrieved after each Phase B negotiation during the **fx_sendfax( )**, **fx_rcvfax( )** or **fx_rcvfax2( )** function call. The DIS message information remains valid until the next Phase B negotiation is completed for the current function call or until a new *send* or *receive* is initiated.

NOTE: Phase B negotiations take place at the beginning of a fax *send* or *receive* function call and after a T.30 End of Message (EOM) message is sent by the transmitter during a fax *send* or *receive* function call.

To determine when the DIS message is available, call the **ATFX_BSTAT( )** function. This function returns a bitmap with the DFS_DIS bit set indicating that the receiver's DIS message is available.

For DIS message details, see the ITU-T publication *Procedures for Document Facsimile Transmission in the General Switched Telephone Network, Recommendation T.30* (see *Section 1.3.2. Other Publications*).

■ Example

```
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

DF_IOTT iott[10];

/* Handler for Phase B events. */
int phb_hdlr( );

main( )
{
   int voxdev;   /* Voice channel device handle. */
   int dev;   /* Fax channel device handle. */

   /*
    * Open the channel using dx_open( ) to obtain the
    * VOICE device handle in voxdev.
    * Open the channel using fx_open( ) to obtain the FAX channel
    * device handle in dev.
    */
   .
   .
   /*
    * Install handler using sr_enbhdlr( ) to service
    * TFX_PHASEB events.
    */
    if (sr_enbhdlr(dev, TFX_PHASEB, phb_hdlr) == -1) {
        printf("Failed to install Phase B handler \n");
        return;
    }

   /*
    * Call fx_sendfax( ) in asynchronous mode after setting
    * up the DF_IOTT array.  Set DF_PHASEB bit in mode field
    * to enable generation of Phase B events.
    */
   if (fx_sendfax(dev, iott, EV_ASYNC|DF_PHASEB) == -1) {
      printf("Error - %s (error code %d)\n",
```

```
            ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
         if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
            /* Perform system error processing */
         }
      }
      .
      .
}

/*
 * Handler registered with SRL to handle TFX_PHASEB events.
 */

int phb_hdlr( )
{
   int dev = sr_getevtdev( );
   DF_DIS dis_buf;

   if (ATFX_BSTAT(dev) & DFS_DIS) {
      /* T.30 DIS available. */
      if (fx_getDIS(dev, &dis_buf) == -1) {
         /* Error processing. */
         .
         .
      } else {
         /* Application specific analysis of the DIS. */
         .
         .
      }
   }
   .
   .
   return(0);
}
```

■ Errors

ATDV_LASTERR( ) returns the following fax error codes for the following reasons:

EFX_NODATA	The function is called before completion of the initial Phase B negotiation.
EFX_UNSUPPORTED	The function is called for an unsupported board.

See *Appendix D* for a list of error codes that may be returned for this function.

Name:	fx_GetDllVersion (dwfileverp, dwprodverp)	
Inputs:	LPDWORD dwfileverp	• fax DLL version number
	LPDWORD dwprodverp	• product version of this release
Returns:	0 if success	
	-1 if failure	
Includes:	srllib.h	
	dxxxlib.h	
	faxlib.h	
Category:	miscellaneous	
Mode:	synchronous	
Platform:	DM3, Springware (Windows only)	

■ Description

Windows only. The **fx_GetDllVersion( )** function returns the fax DLL version number for the file and product.

Parameter	Description
dwfileverp	pointer to where to return file version information
dwprodverp	pointer to where to return product version information

■ Example

```
/*$ fx_GetDllVersion( ) example $*/

#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int InitDevices( )
{
   DWORD dwfilever, dwprodver;

   /*******************************************************************
   * Initialize all the DLLs required. This will cause the DLLs to be
   * loaded and entry points to be resolved. Entry points not resolved
   * are set up to point to a default not implemented function in the
   * 'C' library. If the DLL is not found all functions are resolved
   * to not implemented.
   *******************************************************************/

   if (sr_libinit(DLGC_MT) == -1) {
      /* Must be already loaded, only reason if sr_libinit( ) was already called */
```

```
   }

   /* Call technology specific dx_libinit( ) functions to load Voice DLL */
   if (dx_libinit(DLGC_MT) == -1) {
      /* Must be already loaded, only reason if dx_libinit( ) was already called */
   }
   /* Call technology specific fx_libinit( ) functions to load VFX Fax DLL */
   if (fx_libinit(DLGC_MT) == -1) {
      /* Must be already loaded, only reason if dx_libinit( ) was already called */
   }
   /*****************************************************************************
   * Fax library initialized so all other VFX functions may be called as normal.
   * Display the version number of the DLL
   *****************************************************************************/
   fx_GetDllVersion(&dwfilever, &dwprodver);
   printf("File Version for FAX DLL is %d.%02d\n",
                              HIWORD(dwfilever), LOWORD(dwfilever));
   printf("Product Version for FAX DLL is %d.%02d\n",
                              HIWORD(dwprodver), LOWORD(dwprodver));

   /* Now open all the Voice devices */
}
```

See Also

- **dx_GetDllVersion( )**
- **sr_GetDllVersion( )**
- **dt_GetDllVersion( )**
-

Name:	int fx_getNSF(dev,nsf_length,nsf_data)	
Inputs:	int dev	• fax channel device handle
	unsigned short nsf_length	• number of bytes from NSF message to return
	char * nsf_data	• pointer to buffer for NSF data
Returns:	0 if success	
	-1 if failure	
Includes:	srllib.h	
	dxxxlib.h	
	faxlib.h	
Category:	miscellaneous	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **fx_getNSF( )** function returns the remote station's NSF message (T.30 Non-Standard Facilities), if available, for the specified channel containing the specified number of bytes.

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.
nsf_length	Total number of bytes in the NSF message to be stored in the buffer and number of bytes of the NSF message to retrieve.
nsf_data	A pointer to the buffer location where the NSF message specified in **nsf_length** is stored. The first word of **nsf_data** contains the actual length of the entire NSF message. The remaining bytes of **nsf_data** contain the bytes of the NSF message information.

To understand how the parameters are used, consider this example. If **nsf_length** is 10 bytes, then the **nsf_data** format is as follows:

- the first 2 bytes contain the number of bytes for the entire NSF message

- the remaining 8 bytes contain the first eight bytes of NSF message

NOTE: If the actual NSF message requires fewer bytes than was specified in **nsf_length,** the number of bytes remaining is blank. If the actual NSF message contains more bytes than was specified in **nsf_length**, the NSF message is truncated.

The NSF message information is an optional, variable-length message that can contain fax hardware manufacturer-specific information. Manufacturers can use this information to support proprietary features for their products. The NSF message is sent by the remote station's fax machine and is available to the application after the completion of the first Phase B negotiation for a **fx_sendfax( )**, **fx_rcvfax( )** or **fx_rcvfax2( )** function call.

The NSF message information remains valid until the next Phase B negotiation is completed for the current function or until a new *send* or *receive* is initiated.

NOTE: Phase B negotiations take place at the beginning of a fax *send* or *receive* function call and after a T.30 EOM (End of Message) message is sent by the transmitter.

To determine if the remote station sent an NSF message, call the **ATFX_BSTAT( )** function. If the NSF message is available, this function returns a bitmap with the DFS_NSF bit set.

■ Example

```
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

#define NSFMAX 128

DF_IOTT iott[10];


/* Handler for Phase B events. */
int phb_hdlr( );

main( )
{
   int voxdev;   /* Voice channel device handle. */
   int dev;   /* Fax channel device handle. */

   /*
    * Open the channel using dx_open( ) to obtain the
```

```
    * VOICE device handle in voxdev.
    * Open the channel using fx_open( ) to obtain the FAX channel
    * device handle in dev.
    */
   .
   .
   /*
    * Install handler using sr_enbhdlr( ) to service
    * TFX_PHASEB events.
    */
    if (sr_enbhdlr(dev, TFX_PHASEB, phb_hdlr) == -1) {
        printf("Failed to install Phase B handler \n");
        return;
    }
   /*
    * Call fx_sendfax( ) in asynchronous mode after setting
    * up the DF_IOTT array.  Set DF_PHASEB bit in mode field
    * to enable generation of Phase B events.
    */
   if (fx_sendfax(dev, iott, EV_ASYNC|DF_PHASEB) == -1) {
      printf("Error - %s (error code %d)\n",
         ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
      if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
         /* Perform system error processing */
      }
   }
   .
   .
}

/*
 * Handler registered with SRL to handle TFX_PHASEB events.
 */

int phb_hdlr( )
{
   char nsf_data[NSFMAX];
   char * nsfp:
   unsigned short nsflen;
   int dev = sr_getevtdev( );

   if (ATFX_BSTAT(dev) & DFS_NSF) {
      /* Remote NSF available. */
      if (fx_getNSF(dev, NSFMAX, nsf_data) == -1) {
         /* Error processing */
         .
         .
      } else {
         /* Obtain number of bytes of NSF returned */
         nsflen = * ((unsigned short *)&nsf_data[0]);
         if (nsflen > NSFMAX) {
            /*
             * More NSF data available -- call fx_getNSF( )
             * with larger data buffer if needed.
             */
             .
             .
         }
         /* Set pointer to NSF data. */
         nsfp = &nsf_data[2];
         /* Display NSF (application specific handling). */
         .
```

```
            .
        }
    }
    .
    .
    return(0);
}
```

Errors

ATDV_LASTERR( ) returns the following fax error codes for the following reasons:

EFX_NODATA	The function is called before completion of the initial Phase B or the NSF message was not sent by the remote station.
EFX_NSFBUFF	**nsf_length** value is less than 2 (bytes).
EFX_UNSUPPORTED	The function is called for an unsupported board.

See *Appendix D* for a list of error codes that may be returned for this function.

See Also

- FC_TXNSF define in **dx_setparm( )**
- DF_TXNSF data structure

Name:	int fx_getparm(dev,parm,valuep)	
Inputs:	int dev	• fax channel device handle
	unsigned long parm	• parameter
	void *valuep	• pointer location to store parameter value
Returns:	0 if success -1 if failure	
Includes:	srllib.h dxxxlib.h faxlib.h	
Category:	configuration	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **fx_getparm( )** function returns the current parameter setting for an open fax channel device.

Parameter	Description
dev	Specifies the device handle returned for the fax channel when the channel was opened.
parm	Specifies the define for the parameter ID whose value is returned in the location pointed to by **valuep**.
valuep	Points to the location where the **parm** value is stored.

Many of the same parameter IDs are available for **fx_setparm( )** and **fx_getparm( )**; any differences are noted. The **fx_getparm( )** function allows you to retrieve parameters set for an open fax channel device, and the **fx_getparm( )** function allows you to configure a channel device. For details on parameter IDs, see the **fx_setparm( )** function reference.

■ Cautions

The address of the variable passed to receive the value of the requested parameter must be cast as (void *) as shown in the example. Intel also recommends that you clear this variable prior to calling **fx_getparm( )**.

Allocate sufficient memory to receive the value of the parameter specified. Note that some parameters require only 2 bytes while other parameters may be ASCII strings.

NOTE: Do not use the voice driver library function **dx_getparm( )** to retrieve fax parameter values.

■ Example

Example 1: fx_getparm( ) and FC_RETRYCNT

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int dev;
unsigned short value;

/* Clear value. */
value = 0;

/*
 * Open device using fx_open( ). Obtain FAX device
 * handle in dev.
 */
.
.
/*
 * FC_RETRYCNT parameter uses 2 bytes. Pass the address of
 * the variable value (unsigned short) to fx_getparm( ).
 */
if (fx_getparm(dev,FC_RETRYCNT,(void *)&value) == -1) {
   /* Error processing. */
   printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
      ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
}
printf("Number of retries was %ld\n",value);
```

Example 2: fx_getparm( ) and FC_FONT0 (Windows only)

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int dev;
HFONT hMyFont;
/*
 * Open device using fx_open( ). Obtain fax device handle in dev.
 */
.
.
.

/* pass the handle to the fax library as one of the 2 internal fonts.*/
if (fx_getparm(dev,FC_FONT0,(void *)&hMyFont) == -1) {
   printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
      ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
}
```

■ Errors

See *Appendix D* for a list of error codes that may be returned for this function.

If you issue this function for a parameter that is not supported by your fax hardware channel, **ATDV_LASTERR( )** returns an EFX_UNSUPPORTED error code.

Name:	int fx_getxmitslot(dev,sc_tsinfop)	
Inputs:	int dev	• fax channel device handle
	SC_TSINFO *sc_tsinfop	• pointer to TDM bus time slot information structure
Returns:	0 on success -1 on failure	
Includes:	srllib.h dxxxlib.h faxlib.h	
Category:	TDM bus routing	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **fx_getxmitslot( )** function provides TDM bus time slot number of the fax transmit channel. It returns the TDM bus time slot information contained in a SC_TSINFO structure that includes the number of the TDM bus time slot connected to the fax transmit channel.

NOTE: TDM bus convenience function **nr_scroute( )** includes **fx_getxmitslot( )** functionality. See the *Voice API Library Reference* for more information on nr_ convenience functions.

Parameter	Description
dev	Specifies the channel device handle obtained when the fax device was opened using **fx_open**( ).
sc_tsinfop	Specifies a pointer to the data structure SC_TSINFO.

The SC_TSINFO structure is declared as follows:

```
typedef struct {
     unsigned long     sc_numts;
     long          *sc_tsarrayp;
} SC_TSINFO;
```

The sc_numts member of the SC_TSINFO structure must be initialized with the number of TDM bus time slots requested (1 for a fax channel). The sc_tsarrayp member of the SC_TSINFO structure must be initialized with a pointer to a valid array. Upon return from the function, the first element of the array will contain the number (between 0 and 1023) of the TDM bus time slot on which the fax channel transmits.

■ Cautions

This function will fail when an invalid fax channel device handle is specified.

■ Example

```
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

main( )
{
   int dev;                /* Fax channel device handle. */
   SC_TSINFO sc_tsinfo;    /* Timeslot information structure. */
   long scts;              /* TDM bus time slots. */
   .
   .
   /* Open the FAX channel resource device. */
   if ((dev = fx_open("dxxxB7C1", NULL)) == -1) {
      /* Error opening device. Process error. */
      exit(1);
   }
   /* Fill in the SC_TSINFO structure time slot information. */
   sc_tsinfo.sc_numts = 1;
   sc_tsinfo.sc_tsarray = &scts;
   /* Get FAX device channel TDM bus transmit time slot. */
   if (fx_getxmitslot(dev, &sc_tsinfo) == -1) {
      printf("Error message = %s", ATDV_ERRMSGP(dev));
      exit(1);
   }

   printf("Fax channel is transmitting on TDM bus time slot %d\n", scts);
   .
   .
```

■ Errors

If this function returns -1, use **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to retrieve one of the following error reasons:

Equate	Returned When
EDX_BADPARM	Parameter error
EDX_SH_BADCMD	Command is not supported in current bus configuration
EDX_SH_BADINDX	Invalid Switch Handler index number
EDX_SH_BADLCLTS	Invalid channel number
EDX_SH_BADMODE	Function not supported in current bus configuration
EDX_SH_BADTYPE	Invalid channel type (voice, analog, etc.)
EDX_SH_CMDBLOCK	Blocking command is in progress
EDX_SH_LCLDSCNCT	Channel is already disconnected from TDM bus
EDX_SH_LIBBSY	Switch Handler library busy
EDX_SH_LIBNOTINIT	Switch Handler library uninitialized
EDX_SH_MISSING	Switch Handler is not present
EDX_SH_NOCLK	Switch Handler clock fallback failed
EDX_SYSTEM	System Error

■ See also

- **ag_listen( )**
- **dt_listen( )**
- **dx_listen( )**
- **fx_listen( )**

Name:	int fx_initstat(dev,state)	
Inputs:	int dev	• fax channel device handle
	int state	• initial fax state
Returns:	0 if success	
	-1 if failure	
Includes:	srllib.h	
	dxxxlib.h	
	faxlib.h	
Category:	set initial fax state	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **fx_initstat( )** function sets the initial fax state. You must issue this function in your application to establish the initial fax state of the specified fax channel.

Following T.30 protocol, you must always initially set a caller party to be the transmitter of a fax and a called party to be the receiver of a fax.

NOTE: Only use the **fx_initstat( )** function prior to issuing the **first** *send* or *receive* function of a fax session. Once you issue the **fx_initstat( )** function for a fax session, the correct fax state of the application is maintained automatically by the fax library throughout the fax session, even if turnaround polling is specified.

Parameter	Description	
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.	
state	Specifies the initial fax state. Valid values:	
	DF_RX	called application (receive state)
	DF_TX	caller application (transmit state)

■ Cautions

You must issue the **fx_initstat( )** function before issuing the first *send* or *receive* function for a fax call to select the appropriate protocol for the fax session.

NOTE: Existing applications that use the voice library functions **dx_dial( )** and **dx_wtring( )** to set the initial state will run unmodified. When developing new applications, you must use **fx_initstat( )**.

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int voxdev;   /* Voice channel device handle. */
int dev;      /* Fax channel device handle. */


/*
 * Open the channel using dx_open( ) and obtain the
 * VOICE channel device handle in voxdev. Use voxdev for all
 * Voice API calls.
 */
.
.

/*
 * Open the channel using fx_open( ) and obtain the
 * FAX channel device handle in dev. Use dev for all
 * Fax API calls.
 */
.
.
/*
 * Set channel on-hook using dx_sethook( ) in synchronous
 * mode.
 */
.
.

/*
 * Wait for 1 ring and go off-hook. */
.
.

/*
 * Set the initial FAX state to be RECEIVER. */
if (fx_initstat(dev,DF_RX) == -1) {
   printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
      ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
```

```
}

/* Issue a fx_rcvfax( ). */
.
.
```

Errors

See *Appendix D* for a list of error codes that may be returned for this function.

Name:	fx_libinit (flags)	
Inputs:	unsigned short flags	• programming model
Returns:	0 if success	
	-1 if failure	
Includes:	srllib.h	
	dxxxlib.h	
	faxlib.h	
Category:	miscellaneous	
Mode	synchronous	
Platform:	DM3, Springware (Windows only)	

Description

Windows only. The **fx_libinit( )** function initializes the fax library DLL by loading and resolving all entry points in *libfaxmt.dll*.

Parameter	Description	
flags	Specifies the programming model. Valid values:	
	DLGC_MT	Specifies a multi-threaded or window callback model.
	DLGC_ST	Specifies a single-threaded model.

Cautions

You must call **sr_libinit( )** prior to using **fx_libinit( )**.

Example

```
/*$ fx_libinit( ) example $*/

#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>
int InitDevices( )
{
   DWORD dwfilever, dwprodver;

/*************************************************************************
```

```
   * Initialize all the DLLs required. This will cause the DLLs to be
   * loaded and entry points to be resolved. Entry points not resolved
   * are set up to point to a default not implemented function in the
   * 'C' library. If the DLL is not found all functions are resolved
   * to not implemented.
   ************************************************************************/
   if (sr_libinit(DLGC_MT) == -1) {
      /* Must be already loaded, only reason if sr_libinit( ) was already called */
   }
   /* Call technology specific dx_libinit( ) functions to load Voice DLL */
   if (dx_libinit(DLGC_MT) == -1) {
      /* Must be already loaded, only reason if dx_libinit( ) was already called */
   }
   /* Call technology specific fx_libinit( ) functions to load VFX Fax DLL */
   if (fx_libinit(DLGC_MT) == -1) {
      /* Must be already loaded, only reason if dx_libinit( ) was already called */
   }
   /*****************************************************************************
   * Fax library initialized so all other VFX functions may be called as normal.
   * Display the version number of the DLL
   *****************************************************************************/
   fx_GetDllVersion(&dwfilever, &dwprodver);
   printf("File Version for FAX DLL is %d.%02d\n",
                              HIWORD(dwfilever), LOWORD(dwfilever));
   printf("Product Version for FAX DLL is %d.%02d\n",
                              HIWORD(dwprodver), LOWORD(dwprodver));

/* Now open all the Voice devices */
}
```

■ Errors

The **fx_libinit( )** function fails if the library has already been initialized; for example, if you try to make a second call to **sr_libinit( )**.

■ See Also

- **dx_libinit( )** (in the *Voice API Library Reference*)
- **sr_libinit( )** (in the *Standard Runtime Library API Library Reference*)

Name:	int fx_listen(dev,sc_tsinfop)	
Inputs:	int dev	• fax channel device handle
	SC_TSINFO *sc_tsinfop	• pointer to TDM bus time slot information structure
Returns:	0 on success -1 on failure	
Includes:	srllib.h dxxxlib.h faxlib.h	
Category:	TDM bus routing	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **fx_listen( )** function connects fax listen channel to TDM bus time slot. This function uses the information stored in the SC_TSINFO structure to connect the fax receive (listen) channel to a TDM bus time slot. This function sets up a half-duplex connection. For a full-duplex connection, the receive (listen) channel of the other device must be connected to the fax transmit channel.

NOTE: TDM bus convenience function **nr_scroute( )** includes **fx_listen( )** functionality. See the *Voice API Library Reference* for more information on nr_ convenience functions.

Parameter	Description
dev	Specifies the valid fax channel device handle obtained when the channel was opened using **fx_open**( ).
sc_tsinfop	Specifies a pointer to the data structure SC_TSINFO.

The SC_TSINFO structure is declared as follows:

```
typedef struct {
        unsigned long    sc_numts;
        long             *sc_tsarrayp;
} SC_TSINFO;
```

The sc_numts member of the SC_TSINFO structure must be set to 1. The sc_tsarrayp field of the SC_TSINFO structure must be initialized with a pointer to

a valid array. The first element of this array must contain a valid TDM bus time slot number (between 0 and 1023) which was obtained by issuing a **xx_getxmitslot**( ) function (xx = ag, dl, dt or fx). Upon return from the **fx_listen**( ) function, the fax receive channel will be connected to this time slot.

Although multiple TDM bus device channels may listen (be connected) to the same TDM bus time slot, the fax receive (listen) channel can connect to only one TDM bus time slot.

■ Cautions

This function will fail when:

- An invalid fax channel device handle is specified.
- An invalid TDM bus time slot is specified.

■ Example

```
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

main( )
{
   int voxdev;          /* Voice channel device handle. */
   int dev;             /* Fax channel device handle. */
   SC_TSINFO sc_tsinfo;     /* TDM bus time slot information structure. */
   long scts;           /* TDM bus time slot. */
   .
   .
   /* Open the FAX channel device. */
   if ((dev = fx_open("dxxxB7C1", NULL)) == -1) {
      /* Error opening device. Process error. */
      exit(1);
   }
   /* Open the VOICE channel device on the D/160SC-LS. */
   if ((voxdev = dx_open("dxxxB1C1", NULL)) == -1) {
      /* Error opening device. Process error. */
      exit(1);
   }
     .
     .
   /*
    * Break the full-duplex connection between the Voice
    * channel device and the Network analog device.
    * Use the TDM bus routing cconvenience function nr_scunroute( ).
    */
   if (nr_scunroute(voxdev, SC_VOX, voxdev, SC_LSI, SC_FULLDUP)== -1) {
       /* Error during TDM bus unroute. */
```

```
    printf("Error unrouting channel\n");
    printf("Error - %s (error code %d)\n",
       ATDV_ERRMSGP(voxdev), ATDV_LASTERR(voxdev));
    if (ATDV_LASTERR(voxdev) == EDX_SYSTEM) {
       /* Perform system error processing */
    }
}
  /*
   * Set full-duplex connection between the FAX
   * channel device and the Network analog device.
   */

/* Fill in the SC_TSINFO structure time slot information. */
sc_tsinfo.sc_numts = 1;
sc_tsinfo.sc_tsarray = &scts;


/* Get Network analog device's TDM bus transmit time slot. */
if (ag_getxmitslot(voxdev, &sc_tsinfo) == -1) {
    printf("Error message = %s", ATDV_ERRMSGP(voxdev));
    exit(1);
}
/*
 * Connect the FAX channel to "listen" to the Network
 * channel's TDM bus transmit time slot. Pass the time slot
 * information in the SC_TSINFO structure to fx_listen( ).
 */
if (fx_listen(dev, &sc_tsinfo) == -1) {
   printf("Error message = %s", ATDV_ERRMSGP(dev));
   exit(1);
}
  .
  .
/* Complete full-duplex connection between the FAX channel device
 * and the Network channel device using fx_getxmitslot( )
 * and ag_listen( ).
 */
  .
  .
/* Call FAX API functions for FAX transfers. */
  .
  .
```

■ Errors

If this function returns -1, use **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to retrieve one of the following error reasons:

Equate	Returned When
EDX_BADPARM	Parameter error
EDX_SH_BADCMD	Command is not supported in current bus configuration
EDX_SH_BADEXTTS	TDM bus time slot is not supported at current

Equate	Returned When
	clock rate
EDX_SH_BADINDX	Invalid Switch Handler index number
EDX_SH_BADLCLTS	Invalid channel number
EDX_SH_BADMODE	Function not supported in current bus configuration
EDX_SH_CMDBLOCK	Blocking command is in progress
EDX_SH_LCLTSCNCT	Channel is already connected to TDM bus
EDX_SH_LIBBSY	Switch Handler library busy
EDX_SH_LIBNOTINIT	Switch Handler library uninitialized
EDX_SH_MISSING	Switch Handler is not present
EDX_SH_NOCLK	Switch Handler clock fallback failed
EDX_SYSTEM	System Error

■ See also

- **ag_getxmitslot( )**
- **dt_getxmitslot( )**
- **dx_getxmitslot( )**
- **fx_unlisten( )**

Name:	int fx_open(namep,mode)
Inputs:	char *namep • pointer to device name to open
	int mode • reserved for future use
Returns:	>0 to indicate valid device handle if successful
	-1 if failure
Includes:	srllib.h
	dxxxlib.h
	faxlib.h
Category:	resource management
Mode:	synchronous
Platform:	DM3, Springware

Description

The **fx_open( )** function opens a fax channel or board device and returns a unique device handle to identify the fax channel or board device.

All subsequent fax API calls to the opened fax channel/board device must be made using the fax channel/board device handle until the fax channel/board device is closed.

A fax device can be opened more than once by any number of processes.

Issuing an **fx_open( )** while the fax device is in use by another process does not affect the current operation of the fax device.

Parameter	Description
namep	Pointer to a NULL-terminated ASCII string (ASCIIZ string) that contains the name of the valid fax channel or board device. The valid device names are automatically generated during installation according to the following naming conventions. The value in the name field takes one of the following forms (by default): Board device: dxxxB*n* Channel device: dxxxB*n*C*m* where: *n* is the decimal number of the board in the system *m* is the decimal number of the channel on the board Boards and channels are numbered starting from one.
mode	Reserved for future use. This parameter should be set to NULL.

■ Cautions

- Use **fx_open( )** to open a fax device or DSP fax resource only. On DM3 boards, see *Section 3.2. Device Discovery* for more information on opening DM3 devices.
- The fax device handle returned by this function is defined by Intel. It is not a standard Windows file descriptor. Any attempts to use Windows operating system commands will produce unexpected results.
- In applications that create child processes from a parent process, the fax device handle is not inheritable by the child process. Make sure fax channel/board devices are opened by the child process.
- By default, the maximum number of times you can simultaneously open the same channel in your application is set to 30 in the Windows Registry.

■ Example

```
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

main()
{
   int dev;   /* Fax channel device handle. */

/* Open the Voice channel resource (device) using dx_open( ). */
.
.
/* Open the FAX channel resource (device). */
   if ((dev = fx_open("dxxxB1C1", NULL)) == -1) {
      /* Error opening device. */
      /* Perform system error processing */
      exit(1);
   }
   .
   .
   /* FAX transfers (send/receive) calling FAX API functions using dev. */
   .
   .
}
```

Errors

If this function returns -1 to indicate failure, a system error has occurred. On Linux, check the global variable errno for more information. On Windows, use **dx_fileerrno( )** to obtain the system error value. Refer to the **dx_fileerrno( )** function in the *Voice API Library Reference* for a list of the possible system error values.

See Also

- **fx_close( )**

Name:	int fx_originate(dev, mode)	
Inputs:	int dev	• fax channel device handle
	int mode	• asynchronous/synchronous
Returns:	0 on success	
	-1 on failure	
Includes:	srllib.h	
	dxxxlib.h	
	faxlib.h	
Category:	miscellaneous	
Mode:	asynchronous/synchronous	
Platform:	DM3	

■ Description

The **fx_originate( )** function allows the DCS on hold feature to be used. The function results in a TFX_ORIGINATE or TFX_FAXERROR event, and the command can be stopped by **fx_stopch( )**.

Upon receipt of a TFX_ORIGINATE event, an **fx_sendfax( )** should follow, and the application can call **ATFX_BSTAT( )** and access FC_REMOTEID, **fx_getDIS( )**, **fx_getNSF( )**, **ATFX_SPEED( )**, **ATFX_CODING( )**, **ATFX_ECM( )**. All functions are filled with DF_DIS (Digital Identification Signal) values to avoid having to decode the DIS frame. Values are updated later, during the **fx_sendfax( )**, with DF_DCS (Digital Command Signal) values. When the **fx_sendfax( )** is issued, the file/image format can be specified, including JPEG. See *Section 3.4. Color Fax* for additional information on sending

Parameter	Description	
dev	Specifies the device handle returned for the fax channel when the channel was opened.	
mode	The operation mode specifies whether the function should run asynchronously or synchronously.	
	EV_ASYNC	• Run asynchronously. Returns -1 to indicate failure to initiate. Returns 0 to indicate successful initiation and generates a TFX_ORIGINATE message once the

Parameter	Description	
		DIS is detected, or TFX_FAXERROR if it is not detected.
	EV_SYNC	• Run synchronously. Returns 0 on success and -1 on failure.

■ Cautions

- This function is supported on DM3 boards only.
- You must call the **fx_initstat(DF_TX)** function at least once prior to calling **fx_originate( )**.
- In Fax Resource Only Cluster (FROC) configurations (flexible routing), you must issue **fx_listen( )** prior to calling **fx_sendfax( )**, **fx_rcvfax( )** or **fx_originate( )**. Otherwise, these functions will return an error.
- If the **fx_sendfax( )** is not received within the 20s (limited by the T1 timer + security margin), the fax session will be aborted and result in TFX_FAXERROR.

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
#include <conio.h>

int        dev; /* Fax channel device handle. */
DF_IOTT   iott;

int catchall( );

int main(int argc, char* argv[])
{
    int CanRun = 1;
    int mode = SR_STASYNC;

    /* Set SRL to turn off creation of internal thread */
    if( sr_setparm( SRL_DEVICE, SR_MODELTYPE, &mode ) == -1 ){
        printf( "Error: cannot set srl mode\n" );
        exit( 1 );
```

```
}
/*
* Open the channel using fx_open( ) to obtain the FAX
* channel device handle in dev.
*/
if ((dev = fx_open("dxxxB1C1", NULL)) == -1) {
    /* Error opening device. */
    /* Perform system error processing */
    exit(1);
}

/*
* If this is not a Dm3 Fax channel (return type DFS_FAXDM3)
* warn the user.
*/
if (ATFX_CHTYPE(dev) != DFS_FAXDM3) {
    printf("Function fx_originate is not supported\n");
} else {

    /* Open the file as read-only. */
    iott.io_fhandle = dx_fileopen("sample.tif" ,O_RDONLY|O_BINARY,0);
    /*
    * Set up the DF_IOTT structure as the default and then
    * change the necessary fields.
    */
    fx_setiott(&iott, iott.io_fhandle, DF_TIFF, DFC_AUTO);
    iott.io_type |= IO_EOT;
    printf("Press SPACE to show fx_originate usage, or ESC to exit\n");
    while (CanRun) {
        if (sr_waitevt(100) != -1) {
            catchall();
        }
        if (kbhit()) {
            switch(getch()) {
            case 27: /* Esc */
                CanRun = 0;
                break;
            case ' ': /* Space */
                /* Set initial state of FAX channel to TRANSMITTER. */
                if (fx_initstat(dev, DF_TX) == -1) {
                    printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
                        ATDV_LASTERR(dev));
                    if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
                        /* Perform system error processing */
                    }
                } else {
                    if (iott.io_fhandle != -1) {
                        printf("Issue our fx_originate\n");
                        fx_originate(dev, EV_ASYNC);
                    }
                }
                break;
            }
        }
    }
    dx_fileclose(iott.io_fhandle);
}
/*
* close the channel using fx_close( )
*/
fx_close(dev);
return 0;
```

```
}


/* Event handler. */
/*
* This routine gets called when sr_waitevt( ) receives any event.
* Maintain a state machine for every channel and issue the
* appropriate function depending on the next action to be
* performed on the channel.
*/
int catchall( )
{
    int dev;
    dev = sr_getevtdev( );

    /* Determine the event. */
    switch(sr_getevttype( )) {
    case TFX_ORIGINATE:
        {
            long BStat = ATFX_BSTAT(dev);
            char DisBuf[100]={"N/A"};
            char RemoteId[100]={"N/A"};
            if (BStat & DFS_REMOTEID) {
                if (fx_getparm(dev, FC_REMOTEID, RemoteId)!=0) {
                }
            }
            if (BStat & DFS_DIS) {
                DF_DIS DfDis;
                if (fx_getDIS(dev, &DfDis)==-1) {
                }
                else {
                    /* should translate DIS to a string */
                    strcpy(DisBuf, "Present");
                }
            }
            if (BStat & DFS_NSF) {
            /*...    */
            }
            printf("Receiver is capable of: speed %ld, resln %ld,"
                "width %ld, Ecm %ld\n", ATFX_SPEED(dev),
                ATFX_RESLN(dev), ATFX_WIDTH(dev),
                ATFX_ECM(dev));
            printf("Information: Csid='%s' and Dis is %s\n",
                RemoteId, DisBuf);
            printf("Issue our fx_sendfax\n");
            /*
            * Depending the capabilities, Parameter can be adjusted
            * and the user is capable of pointing to the appropriate
            * iott structure (e.g., Raw Color Fax or simple tiff image)
            */
            fx_sendfax(dev, &iott,EV_ASYNC|DF_PHASEB|DF_PHASED);
        }
        break;
    case TFX_FAXSEND:
        /* The document has been successfully sent. */
        printf("Sent %ld pages at speed %ld, resln %ld,"
            "width %ld\n", ATFX_PGXFER(dev), ATFX_SPEED(dev),
            ATFX_RESLN(dev), ATFX_WIDTH(dev));
        /* Fax session completed. */
        printf("Press ESC to exit\n");
        break;
    case TFX_PHASEB:
```

```
        printf("Phase B event\n");
        /* extract usual information from here */
        break;
    case TFX_PHASED:
        printf("Phase D event\n");
        /* extract usual information from here */
        break;
    case TFX_FAXERROR:
        /* Error during the fax session. */
/*        print_err(dev); */
        printf("Phase E status %ld\n", ATFX_ESTAT(dev));
        /* Application specific error handling. */
            break;
    default:
            break;
    } /* End of switch. */
    return(0);
}
```

■ Errors

TFX_ORIGINATE	Successful fax origination
TFX_FAXERROR	Error in processing
EFX_UNSUPPORTED	Unsupported function. This error returned if this function is attempted on a non-DM3 board.

Error defines can be found in *faxlib.h*.

■ See Also

- **fx_getDIS( )**
- **fx_getNSF( )**
- **fx_stopch( )**
- **ATFX_SPEED( )**
- **ATFX_CODING( )**
- **ATFX_ECM( )**

Name:	int fx_rcvfax(dev, faxname, rcvflag)	
Inputs:	int dev	• fax channel device handle (to receive fax data)
	char * faxname	• name to assign received document
	unsigned long rcvflag	• mode flag
Returns:	0 if success (on invocation in asynchronous mode)	
	-1 if failure (on invocation in asynchronous mode)	
Includes:	srllib.h	
	dxxxlib.h	
	faxlib.h	
Category:	receive fax	
Mode:	synchronous/asynchronous	
Platform:	DM3, Springware	

■ Description

The **fx_rcvfax( )** function receives fax data from an open channel device and stores it as a TIFF/F file or a raw file.

NOTE: A raw file stores fax data as a single page of unstructured, unformatted data.

The **fx_rcvfax( )** function can be issued by the fax receiver or the fax transmitter. To stop a fax reception in progress, use **fx_stopch( )**.

The encoding scheme in which the incoming fax data may be stored (MH or MMR) is based on the capability of the Intel® fax product. For product capabilities, see *Section 2.3. Key Product Features* on *page 8*.

For more information on setting up the channel device to receive fax data, see *Chapter 6. Implementing Receive Fax Capability*.

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.
faxname	Specifies the file name to assign to the incoming fax data. The TIFF/F or raw file, named in **faxname**, is created or overwritten as needed. When storing multi-page fax data in raw files, you must specify a different file for each incoming fax page.
rcvflag	A logical OR bit mask that indicates the following: • The file format in which to save the incoming fax data • Polling request from the transmitter is valid or not • The mode of operation, synchronous or asynchronous • Enable generation of Phase B events (T.30 pre-message procedure) • Enable generation of Phase D events (T.30 post-message procedure) • Enable accepting and issuing operator intervention (voice request) from remote station • Set maximum receive width • Set preferred receive length • Store all incoming fax data at low (coarse) or high (fine) vertical resolution • Enable user-defined I/O functions (**fx_rcvfax2( )** only) The **rcvflag** bit mask can have the following values: File format bit:

Value	Description
DF_TIFF	TIFF/F structured formatted fax data
DF_RAW	Raw, unformatted fax data

Poll bit:

rcvflag (cont.)

Value	Description
DF_NOPOLL	Polling invalid (default)
DF_POLL	Polling valid

Mode bit (for more information, see *Section 6.3.2. Mode of Operation* on *page 88*):

Value	Description
EV_SYNC	Synchronous mode operation.
EV_ASYNC	Asynchronous mode operation.

Phase B, Phase D and Voice Request enable bits. Set one or more of the following (the default is disabled):

Value	Description
DF_PHASEB	Enable Phase B event generation. When set, a TFX_PHASEB event is returned each time Phase B is completed during the receive fax operation. For more information, see *Section 6.3.3. Enable Phase B Event Generation* on *page 89*.
DF_PHASED	Enable Phase D event generation. When set, a TFX_PHASED event is returned each time Phase D is completed during the receive fax operation, except for the last page. After the last page, **fx_rcvfax( )** completes (synchronous mode) or a TFX_FAXRECV event occurs (asynchronous mode). For more information, see *Section 6.3.4. Enable Phase D Event Generation* on *page 90.*
DF_ACCEPT_VRQ	Enable accepting operator intervention (voice request) from remote station. This value is not supported on DM3 boards.
DF_ISSUE_VRQ	Enable issuing operator intervention (voice request) to remote station. This value is not supported on DM3 boards.

rcvflag (cont.)

Maximum receive width bits:

Value	Description
DF_1728MAX	Maximum receive width: 1728 pixels
DF_2048MAX	Maximum receive width: 2048 pixels
DF_2432MAX	Maximum receive width: 2432 pixels (default)

Preferred receive length bits:

Value	Description
DF_A4MAXLEN	Maximum receive length: A4 size (approximately 11 inches)
DF_B4MAXLEN	Maximum receive length: B4 size (approximately 14 inches)
DF_NOMAXLEN	Maximum receive length: unlimited (default)

Vertical resolution of fax data storage. The default is the incoming fax data's specified resolution.

Value	Description
DF_RXRESLO	Store all incoming fax data at low vertical resolution.
DF_RXRESHI	Store all incoming fax data at high vertical resolution.

Enable user-defined I/O bit, available for **fx_rcvfax2( )** only:

Value	Description
IO_UIO	User-defined I/O functions for **fx_rcvfax2( )**

■ Examples

Examples 1 and **2** use **fx_rcvfax( )** for receiving fax data into TIFF/F and raw format files in synchronous mode. The synchronous programming code fragments

shown can be used in a multi-threaded application where the program creates a separate thread for every channel. Each thread would control a single channel using a synchronous mode of operation.

Example 3 uses **fx_rcvfax( )** in asynchronous mode. The asynchronous programming code fragments shown can be used in a multi-threaded application where the program creates multiple threads. Each thread could control a single channel or multiple channels using an asynchronous mode of operation. See the *Standard Runtime Library* documentation for information on programming modes and the Standard Runtime Library (SRL) functions.

Example 1: Receive Fax Data into TIFF/F File Format - Synchronous

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int voxdev;   /* Voice channel device handle. */
int dev;      /* Fax channel device handle. */


unsigned long rcvflag = DF_NOPOLL|DF_TIFF|EV_SYNC;
unsigned short value;


/*
 * Open the channel using dx_open( ) to obtain the
 * VOICE channel device handle in voxdev. Use voxdev for
 * all Voice API calls.
 */
if ((voxdev = dx_open("dxxxB1C1", NULL)) == -1) {
   /* Error opening device. */
      /* Perform system error processing */
      exit(1);
}

/*
 * Open the channel using fx_open( ) to obtain the FAX
 * channel device handle in dev. Use dev for all Fax API
 * calls.
 */
if ((dev = fx_open("dxxxB1C1", NULL)) == -1) {
   /* Error opening device. */
   /* Perform system error processing */
   exit(1);
}
.
.
/*
 * Set channel on-hook using dx_sethook( ) in synchronous
 * mode.
 */
.
```

```
.
/*
 * Wait for 1 ring and go off-hook using dx_wtring( ).
 */
.
.
/* If this is a channel on a VFX/40SC (return type DFS_FAX40)
 * or VFX/40ESC (return type DFS_FAX40E),
 * a vertical resolution for the receive file can
 * be specified in rcvflag. For the VFX/40ESC, the
 * received data can be stored as MMR encoded data.
 */
switch (ATFX_CHTYPE(dev))  {

case DFS_FAX40:
   /* Store the received file in low vertical resolution. */
   rcvflag |= DF_RXRESLO;
   break;

case DFS_FAX40E:
   /*
    * Store the received file in low vertical resolution
    * and MMR encoding.
    */
   rcvflag |= DF_RXRESLO;
   value = DF_MMR;

   if (fx_setparm(dev,FC_RXCODING,(void *)&value) == -1) {
      printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
         ATDV_LASTERR(dev));
      if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
         /* Perform system error processing */
      }
   }
   break;

default:
   break;
}

/* Set initial state of FAX channel to RECEIVER. */
if (fx_initstat(dev,DF_RX) == -1) {
   printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
      ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
}

/*
 * Receive the fax data into "myfax.tif" file - synchronous
 * mode.
 */
if((fx_rcvfax(dev,"myfax.tif",rcvflag)
    == -1) {
   printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
       ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
   printf("Phase E status: %ld\n", ATFX_ESTAT(dev));
```

```
   /* Application specific error handling. */
   .
   .
}
```

Example 2: Receive Fax Data into Raw File - Synchronous

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int count = 0;
char faxname[30];

int voxdev;   /* Voice channel device handle. */
int dev;      /* Fax channel device handle. */

unsigned long rcvflag = DF_NOPOLL|DF_RAW|EV_SYNC;
unsigned short value;

/*
 * Open the channel using dx_open( ) to obtain the
 * VOICE channel device handle in voxdev. Use voxdev for
 * all Voice API calls.
 */
if ((voxdev = dx_open("dxxxB1C1", NULL)) == -1) {
   /* Error opening device. */
      /* Perform system error processing */
      exit(1);
}
/*
 * Open the channel using fx_open( ) to obtain the FAX
 * channel device handle in dev. Use dev for all Fax API
 * calls.
 */
if ((dev = fx_open("dxxxB1C1", NULL)) == -1) {
   /* Error opening device. */
   /* Perform system error processing */
   exit(1);
}
.
.
/*
 * Set channel on-hook using dx_sethook( ) in synchronous
 * mode.
 */
.
.
/*
 * Wait for 1 ring and go off-hook using dx_wtring( ).
 */
.
.

/* If this is a channel on a VFX/40SC (return type DFS_FAX40)
 * or VFX/40ESC (return type DFS_FAX40E),
 * a vertical resolution for the receive file can
 * be specified in rcvflag. For the VFX/40ESC, the
 * received data can be stored as MMR encoded data.
```

```
 */
switch (ATFX_CHTYPE(dev))  {

case DFS_FAX40:
   /* Store the received file in low vertical resolution. */
   rcvflag |= DF_RXRESLO;
   break;

case DFS_FAX40E:
   /*
    * Store the received file in low vertical resolution
    * and MMR encoding.
    */
   rcvflag |= DF_RXRESLO;
   value = DF_MMR;

   if (fx_setparm(dev,FC_RXCODING,(void *)&value) == -1) {
      printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
         ATDV_LASTERR(dev));
      if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
         /* Perform system error processing */
      }
   }
   break;

default:
   break;
}

/* Set initial state of the FAX channel to RECEIVER. */
if (fx_initstat(dev,DF_RX) == -1) {
   printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
      ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
}


do {
/* Receive each page into a separate file until the application
 * receives a DFS_EOP Phase D status value. fx_rcvfax( ) is
 * being used in synchronous mode.
 */
.
.
/*
 * Generate a file name in faxname, for example, rcv_pg0.raw,
 * rcv_pg1.raw, etc.
 */
.
.
   if(fx_rcvfax(dev,faxname,rcvflag) == -1)
      printf("Error - %s (error code %d)\n",
         ATDV_ERRMSGP(dev), ATDV_LASTERR(dev));
      if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
         /* Perform system error processing */
      }
      printf("Phase E status: %ld\n", ATFX_ESTAT(dev));
      /* Application specific error handling. */
      .
      .
```

```
   }
} while(ATFX_PHDCMD(dev) != DFS_EOP);

/* Show results. */
printf("Fax received: %ld pages\n",ATFX_PGXFER(dev));

/*
 * Note: The encoding scheme of the received RAW data is specified
 * in the variable 'value' used for setting the FC_RXCODING
 * parameter. If these RAW files have to be transmitted, the same
 * encoding scheme value will have to be specified in the DF_IOTT
 * entry.
 */
.
.
```

Example 3: Receive Fax Data using Asynchronous Programming Mode

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

#define MAXCHANS  24

int catchall( );
int recv_fax( );

/* Error routine - print error information. */

void print_err(dev)
   int dev;
{
   printf("Error - %s (error code %ld)\n",
      ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
      /* Perform system error processing */
   }
   return;
}

/*
 * main( ): Opens all channels and enables handler for
 * asynchronous operation. Channels go on-hook and wait for
 * rings. On receiving rings, the channel goes off-hook and
 * receives a fax.
 */

main( )
{
   int chan;
   char * chnamep;
   int mode = SR_STASYNC;

   int voxdev;   /* Voice channel device handle. */
   int faxdev;   /* Fax channel device handle. */

     /* Set SRL to turn off creation of internal thread */
```

```
     if( sr_setparm( SRL_DEVICE, SR_MODELTYPE, &mode ) == -1 ){
        printf( "Error: cannot set srl mode\n" );
        exit( 1 );
     }

   for (chan=0; chan < MAXCHANS; chan++) {

      /*
       * Set chnamep to the channel device name, e.g.,
       * dxxxB1C1, dxxxB1C2, etc.
       * Open the channel using dx_open( ) so that voxdev
       * has the VOICE channel device handle.
       * Open the channel using fx_open( ) so that faxdev
       * has the FAX channel device handle.
       */

     if(( voxdev = dx_open( chnamep, NULL )) == -1 ){
          printf( "Error: cannot open vox device\n" );
          exit( 1 );
     }

     if(( faxdev = fx_open( chnamep, NULL )) == -1 ){
          printf( "Error: cannot open fax device\n" );
          exit( 1 );
     }
       .
       .
       .
     /* enable a handler for all events on any devices */
     if( sr_enbhdlr( EV_ANYDEV, EV_ANYEVT, dx_handler ) == -1 ){
        printf( "Error: could not enable handler\n" );
        exit( 1 );
     }
       .
       .
       .

      /*
       * Place channel on-hook by calling dx_sethook( ) with
       * its mode field set to EV_ASYNC (asynchronous).
       */
     if( dx_sethook( voxdev, DX_ONHOOK, EV_ASYNC ) == -1 ){
        printf( "dx_sethook failed: error = %s\n", ATDV_ERRMSGP( voxdev ));
        exit( 1 );
     }

      .
      .
   }

   /*
    * All channels have been opened and a sethook function
    * issued to place the channels on-hook. Use sr_waitevt( )
    * to wait for completion events.
    * On receiving any completion event, control is transferred
    * to the catchall( ) handler function.
    */
    while(sr_waitevt(-1))
    .
    .

/* Event handler. */
```

```
/*
 * This routine is called when sr_waitevt( ) receives an event.
 * Maintain a state machine for every channel and issue the
 * appropriate function depending on the next action to be
 * performed on the channel, e.g., the application may wish
 * to wait for rings after an on-hook completion event and
 * start receiving a fax as soon as rings are received.
 */

int catchall( )

{

   int dev = sr_getevtdev( );
   char * fnamep;

   /* Determine the event. */
   switch(sr_getevttype( ))  {

   case TDX_SETHOOK:
      /*
       * If channel has gone off-hook, start receiving the
       * fax.
       */
      if (ATDX_HOOKST(dev) == DX_OFFHOOK) {
         /*
          * Set the fax state of the channel to DF_RX using
          * fx_initstat( ).
          */
         .
         .
         /*
          * Set up fnamep to point to TIFF/F file name.
          * Start receiving the fax.
          */
         if (fx_rcvfax(dev, fnamep, DF_TIFF|DF_NOPOLL|EV_ASYNC) == -1) {
            print_err(dev);
            printf("Phase E status: %ld\n",
              ATFX_ESTAT(dev));
            /* Application specific error handling here. */
            .
            .
         }
      } else {
         /*
          * Channel is on-hook. State machine dependent
          * action.
          */
         .
         .
      }
      break;

   case TDX_CST:
      /* Handle rings received event. */
      .
      .
      break;

   case TFX_FAXRECV:
      /* The document has been successfully received. */
```

```
      printf("Received %ld pages at speed %ld, resln %ld,
         width %ld\n", ATFX_PGXFER(dev), ATFX_SPEED(dev),
         ATFX_RESLN(dev), ATFX_WIDTH(dev));
      .
      .
      break;

   case TFX_FAXERROR:
      /* Error during the fax session. */
      print_err(dev);
      printf("Phase E status %d\n", ATFX_ESTAT(dev));
      /* Application specific error handling. */
      .
      .
      break;

   default:
      .
      .
      break;

   } /* End of switch. */

   return(0);
}
```

Example 4: Receive fax data using callback handler and setting SRL to operate in polled mode - Asynchronous (Linux only)

```
#include <stdio.h>
#include <errno.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

#define MAXCHANS 24

extern int errno;

int catchall( );
int recv_fax( );

/* Error routine - print error information. */

void print_err(dev)
            int dev;
{
            printf("Error - %s (error code %ld)\n",
                        ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
            if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
                        printf("errno = %d\n", errno);
            }
            return;
}

/*
 * main( ): Opens all channels and enables handler for
 * asynchronous operation. Channels go on-hook and wait for
 * rings. On receiving rings, the channel goes off-hook and
```

```
 * receives a fax.
 */

main( )
{
            int chan;
            char * chnamep;
            int mode = SR_POLLMODE;

            int voxdev; /* Voice channel device handle. */
            int faxdev; /* Fax channel device handle. */

            /* Set SRL to operate in POLLED Mode. */
            if (sr_setparm(SRL_DEVICE, SR_MODEID, &mode) == -1) {
                        printf("Cannot set SRL in polled mode\n");
                        .
                        .
            }

            for (chan=0; chan < MAXCHANS; chan++) {

                        /*
                         * Set chnamep to the channel device name, e.g.,
                         * dxxxB1C1, dxxxB1C2, etc.
                         * Open the channel using dx_open( ) so that voxdev
                         * has the VOICE channel device handle.
                         * Open the channel using fx_open( ) so that faxdev
                         * has the fax channel device handle.
                         */
                        .
                        .
                        /*
                         * Using sr_enbhdlr( ), set up handler to service all
                         * events.
                         */
                        if(sr_enbhdlr(voxdev, EV_ANYEVT, catchall) == -1) {
                                    printf("sr_enbhdlr( ) failed\n");
                                    print_err(dev);
                                    .
                                    .
                        }
                        if(sr_enbhdlr(faxdev, EV_ANYEVT, catchall) == -1) {
                                    printf("sr_enbhdlr( ) failed\n");
                                    print_err(dev);
                                    .
                                    .
                        }
                        /*
                         * Place channel on-hook by calling dx_sethook( ) with
                         * its mode field set to EV_ASYNC (asynchronous).
                         */
                        .
                        .
            }
             /* This will cause catchall() to be called when an event is available */

             for(;;)
             {
                   sr_waitevt(-1);
             }

}
```

```
/* Event handler. */

/*
 * This routine gets called by SRL on receiving any event.
 * Maintain a state machine for every channel and issue the
 * appropriate function depending on the next action to be
 * performed on the channel, e.g., the application may wish
 * to wait for rings after a on-hook completion event and
 * start receiving a fax as soon as rings are received.
 */

int catchall( )
{
            int dev = sr_getevtdev( );
            /* Determine the event. */
            switch(sr_getevttype( ))  {
            case TDX_SETHOOK:
                        /*
                         * If channel has gone off-hook, start receiving the
                         * fax.
                         */
                        if (ATDX_HOOKST(dev) == DX_OFFHOOK) {
                                    /*
                                     * Set the fax state of the channel to DF_RX using
                                     * fx_initstat( ).
                                     */
                                    .
                                    .
                                    /* Start receiving the fax. */
                                    if (fx_rcvfax(dev, fnamep,
                                    DF_TIFF|DF_NOPOLL|EV_ASYNC) == -1) {
                                                print_err(dev);
                                                printf("Phase E status: %ld\n",
                                                  ATFX_ESTAT(dev));
                                      /* Application specific error handling here. */
                                                .
                                                .
                                    }
                        } else {
                                    /*
                                     * Channel is on-hook. State machine dependent
                                     * action.
                                     */
                                    .
                                    .
                        }
                        break;
            case TDX_CST:
                        /* Handle rings received event. */
                        .
                        .
                        break;
            case TFX_FAXRECV:
                        /* The document has been successfully received. */
                        printf("Received %ld pages at speed %ld, resln %ld,
                                    width %ld\n", ATFX_PGXFER(dev), ATFX_SPEED(dev),
                                    ATFX_RESLN(dev), ATFX_WIDTH(dev));
                        .
                        .
                        break;
            case TFX_FAXERROR:
                        /* Error during the fax session. */
```

```
                    print_err(dev);
                    printf("Phase E status %d\n", ATFX_ESTAT(dev));
                    /* Application specific error handling. */
                    .
                    .
                    break;
          default:
                    .
                    .
                    break;
          } /* End of switch. */
          return(0);
}
```

■ Errors

In synchronous mode, this function returns a zero to indicate successful completion or a -1 to indicate an error.

In asynchronous mode, this function returns a zero to indicate successful invocation or a -1 to indicate an invocation error.

Errors that occur during reception generate a Standard Runtime Library event (TFX_FAXERROR). To access the error code, call the standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( ).** The latter returns a string describing the error. See *Appendix D* for a list of fax error codes.

If the **fx_rcvfax( )** function successfully completes, a TFX_FAXRECV Standard Runtime Library event is generated.

The fax extended attribute **ATFX_ESTAT( )**) provides additional error information for T.30 Phase E fax protocol.

System errors return an EDX_SYSTEM error On Linux, check the global variable errno for more information. On Windows, use **dx_fileerrno( )** to obtain error value**.** Refer to the **dx_fileerrno( )** function in the *Voice API Library Reference* for a list of the possible system error values.

■ See Also

- **ATFX_*name* functions**
- **fx_rcvfax2( )**
- **fx_getDCS( )**

- **fx_getDIS( )**
- **fx_getNSF( )**

Name:	int fx_rcvfax2(dev, fd, rcvflag)	
Inputs:	int dev	• fax channel device handle (to receive fax data)
	int fd	• receive file descriptor
	unsigned long rcvflag	• mode flag
Returns:	0 if success (on invocation in asynchronous mode)	
	-1 if failure (on invocation in asynchronous mode)	
Includes:	srllib.h	
	dxxxlib.h	
	faxlib.h	
Category:	receive fax	
Mode:	synchronous/asynchronous	
Platform:	DM3, Springware	

Description

The **fx_rcvfax2( )** function receives fax data (file descriptor argument) from an open channel device and stores it as a TIFF/F file or a raw file.

NOTE: A raw file stores fax data as a single page of unstructured, unformatted data.

The **fx_rcvfax2( )** function can be issued by the fax receiver or the fax transmitter. To stop a fax reception in progress, use **fx_stopch( )**.

The encoding scheme in which the incoming fax data may be stored (MH and/or MMR) is based on the capability of the fax product. For product capabilities, see *Section 2.3. Key Product Features* on *page 8*.

NOTES:
1. The **fx_rcvfax2( )** function uses a file descriptor argument (**fd**) to specify the receive file instead of a file name as in the **fx_rcvfax( )** function.
2. To receive a fax using user-definable I/O functions, you must issue **fx_rcvfax2( )** and logically "OR" the IO_UIO bit in the **rcvflag** argument.

For more information on setting up the channel device to receive fax data, see *Chapter 6. Implementing Receive Fax Capability*.

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.
fd	Specifies the file descriptor.
rcvflag	The **rcvflag** field is a logical OR bit mask. See the **fx_rcvfax( )** function for **rcvflag** field values.

■ Cautions

- The application must open the receive file and pass the file descriptor to **fx_rcvfax2( )**.
- The fax library does not close the receive file after the fax has been received or an error has occurred. The application must close the receive file.

■ Example

```
/*
 * The principal difference between fx_rcvfax( ) and
 * fx_rcvfax2( ) is that the application must open the
 * receive file and pass the file descriptor to the
 * fx_rcvfax2( ) function instead of the receive file name.
 * Example 1 from the function reference for fx_rcvfax( ) has
 * been modified for use with fx_rcvfax2( ) and included
 * below. The other examples in fx_rcvfax( ) can be modified
 * similarly.
 */

#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int voxdev;   /* Voice channel device handle. */
int dev;      /* Fax channel device handle. */

int rcvfd;
unsigned long rcvflag;

/*
 * Open the channel using dx_open( ) and obtain the
 * VOICE channel device handle in voxdev.
 */
.
.
/*
```

```
 * Open the channel using fx_open( ) and obtain the
 * FAX channel device handle in dev.
 */
.
.
/*
 * Set channel on-hook using dx_sethook( ) in synchronous
 * mode.
 */
.
.
/*
 * Wait for 1 ring and go off-hook using dx_wtring( ).
 */
.
.
/* If this is a channel on a VFX/40SC (return type DFS_FAX40)
 * or VFX/40ESC (return type DFS_FAX40E),
 * a vertical resolution for the receive file can
 * be specified in rcvflag. For the VFX/40ESC, the
 * received data can be stored as MMR encoded data.
 */
switch (ATFX_CHTYPE(dev))  {

case DFS_FAX40:
   /* Store the received file in low vertical resolution. */
   rcvflag |= DF_RXRESLO;
   break;

case DFS_FAX40E:
   /*
    * Store the received file in low vertical resolution
    * and MMR encoding.
    */
   rcvflag |= DF_RXRESLO;
   value = DF_MMR;

   if (fx_setparm(dev,FC_RXCODING,(void *)&value) == -1) {
      printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
         ATDV_LASTERR(dev));
      if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
         /* Perform system error processing */
      }
   }
   break;

default:
   break;
}

/*
 * Set the fax state of the channel to DF_RX using
 * fx_initstat( ).
 */
.
.

/*
 * Open the file "myfax.tif" in preparation for receiving a
 * fax. Use dx_fileopen( ) to open the file.
 */
if ((rcvfd = dx_fileopen("myfax.tif", O_BINARY|O_WRONLY|O_CREAT|O_TRUNC,
```

```
   0666)) == -1) {
   /* Error opening file. */
   /* Perform system error processing */
   .
   .
}

/*
 * Receive the fax data into "myfax.tif" file - synchronous
 * mode.
 */
if((fx_rcvfax2(dev,rcvfd,rcvflag))
    == -1) {
   printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
       ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
   printf("Phase E status: %ld\n", ATFX_ESTAT(dev));

   /* Application specific error handling. */
   .
   .
}

/* Close the received file. */

if (dx_fileclose(rcvfd) == -1) {
      /* Error closing file. */
      /* Perform system error processing */
      .
      .
}
```

■ Errors

In synchronous mode, this function returns a zero to indicate successful completion or a -1 to indicate an error.

In asynchronous mode, this function returns a zero to indicate successful invocation or a -1 to indicate an invocation error.

Errors that occur during reception generate a Standard Runtime Library event (TFX_FAXERROR). To access the error code, call the standard attribute functions **ATDV_LASTERR( )** and **ATDV_ERRMSGP( ).** The latter returns a string describing the error. See *Appendix D* for a list of fax error codes.

If the **fx_rcvfax2( )** function successfully completes, a TFX_FAXRECV Standard Runtime Library event is generated.

The fax extended attribute **ATFX_ESTAT( )**) provides additional error information for T.30 Phase E fax protocol.

System errors return an EDX_SYSTEM error. On Linux, check the global variable errno for more information. On Windows, use **dx_fileerrno( )** to obtain error value. Refer to the **dx_fileerrno( )** function in the *Voice API Library Reference* for a list of the possible system error values.

See Also

- **fx_rcvfax( )**

Name:	int fx_rtvContinue(dev, continue)	
Inputs:	int dev	• fax channel device handle
	int continue	• TRUE or FALSE
Returns:	0 if success	
	-1 if failure	
Includes:	srllib.h	
	dxxxlib.h	
	faxlib.h	
Category:	miscellaneous	
Mode:	synchronous	
Platform:	Springware (Windows only)	

■ Description

Windows only. The **fx_rtvContinue( )** function is used for remote terminal verification. If you enable remote terminal verification (RTV), you must call this function after a PRE_PHASEB event is received by your application in order to proceed with the fax transfer. Otherwise, a firmware time-out occurs and the fax transfer is terminated.

To enable RTV, specify DF_ENABLE_RTV and DF_PHASEB in **fx_sendfax( )**. The RTV feature allows you to verify the recipient's identity and abort transmission if necessary before the firmware responds with a DCS message (digital command signal).

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.
continue	Specifies whether the application wishes to proceed with the fax protocol or not. Values are TRUE or FALSE.

■ Cautions

None

■ Example

```
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

DF_IOTT iott[10];

/* Handler for Phase B events. */
int phb_hdlr( );

main( )
{
   int voxdev;   /* Voice channel device handle. */
   int dev;   /* Fax channel device handle. */

   /*
    * Open the channel using dx_open( ) to obtain the
    * VOICE device handle in voxdev.
    * Open the channel using fx_open( ) to obtain the FAX channel
    * device handle in dev.
    */
   .
   .
   /*
    * Install handler using sr_enbhdlr( ) to service
    * TFX_PHASEB events.
    */
    if (sr_enbhdlr(dev, TFX_PHASEB, phb_hdlr) == -1) {
        printf("Failed to install Phase B handler \n");
        return;
    }

   /*
    * Call fx_sendfax( ) in asynchronous mode after setting
    * up the DF_IOTT array.  Set DF_ENABLE_RTV and DF_PHASEB bits
    * in mode field to enable generation of
    * remote terminal verification and Phase B events.
    */
    if (fx_sendfax(dev, iott, EV_ASYNC|DF_ENABLE_RTV|DF_PHASEB) == -1)
    {
     printf("Error: %s (error code %d)\n",
           ATFX_ERRMSGP(dev), ATFX_LASTERR(dev));
    }
.
.
.

    /*
     * Handler registered with SRL to handle TFX_PHASEB events.
     */
    int phb_hdlr( )
    {
         int dev = sr_getevtdev( );
         char szId[22], szValid[22];

         strcpy(szValid, "OK TERMINAL");
         if (sr_getevttype() == TFX_PHASEB)
     {
             if (fx_getparm(dev, FC_REMOTEID, szId) == -1)
```

```
            {
             printf("fx_getparm err: %s\n",
                  ATDX_ERRMSGP(dev));
             // Getting remote ID failed. Abort.
             fx_rtvContinue(dev, FALSE);
             }
             else
             {
             // Check the database here.
             If (!strcmp(szId, szValid))
                 fx_rtvContinue(dev, TRUE);
             else
                 fx_rtvContinue(dev, FALSE);
             }
    }
return 0;
}
```

■ Errors

None

■ See Also

- **fx_sendfax( )**

Name:	int fx_sendascii(faxname, phdcont)	
Inputs:	char * faxname	• ASCII filename
	unsigned short phdcont	• Phase D continuation value
Returns:	0 if success	
	-1 if failure	
Includes:	srllib.h	
	dxxxlib.h	
	faxlib.h	
Category:	convenience	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **fx_sendascii( )** function allows an application to send a single ASCII file at the default width, length, resolution, fonts and margins for ASCII data. For default ASCII information, see *Section 10.3. DF_ASCIIDATA – ASCII Data Description* on *page 119*.

This function is a convenience function and resides in *faxconv.c*. The **fx_sendascii( )** function calls **fx_sendfax( )** (see source code for **fx_sendascii( )**). The **fx_sendfax( )** function reference contains information on Phase D continuation values, status information and file error handling that applies to **fx_sendascii( )**.

The encoding scheme used in transmitting fax data varies by product; for more information see *Section 2.3. Key Product Features* on *page 8*. The preferred encoding scheme for transmission is determined by the value set in the FC_TXCODING parameter in **fx_setparm( )**.

Parameter	Description	
faxname	Specifies the name of the ASCII file to send.	
phdcont	Specifies the Phase D continuation value. This value defines the action to take at the end of the current DF_IOTT structure after the transfer of fax data. Valid values:	
	DFC_EOP	End of Procedure (T.30). Terminate current fax session; progress to Phase E; and disconnect fax call.
	DFC_MPS	Multi-Page Signal (T.30). End of current fax document page; next page is in same format as the current page; proceed directly to Phase C.
	DFC_EOM	End of Message (T.30). End of current fax document page; more fax data to follow at different resolution or width; return to Phase B and negotiate parameters for next fax document page.

■ Cautions

- Before calling **fx_sendascii( )**, you must open the channel using **fx_open( )** to obtain the fax channel device handle.
- If TDM bus routing is required, you must complete the routing before calling the convenience function.

Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int voxhandle;   /* Voice channel device handle. */
int devhandle;   /* Fax channel device handle. */


/*
 * Open the channel using dx_open( ) to obtain the
 * VOICE device handle in voxhandle.
 * Open the channel using fx_open( ) to obtain the FAX channel
 * device handle in devhandle.
 */

/*
 * Take channel offhook using dx_sethook( ) and perform outbound
 * dial using dx_dial( ).
 */
.
.

/*
 * Send the ASCII file. No more files to send (DFC_EOP).
 */
if (fx_sendascii("textdata.txt",DFC_EOP) == -1) {
   printf("Error - %s (error code %d)\n",
      ATDV_ERRMSGP(devhandle), ATDV_LASTERR(devhandle));
   if (ATDV_LASTERR(devhandle) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
   printf("Phase E status: %ld\n", ATFX_ESTAT(devhandle));

   /* Application specific error handling. */
   .
   .

}
```

■ Source Code for fx_sendascii( )

```
/*
 * NOTE: devhandle is a global variable of type int. Prior
 * to calling fx_sendascii( ), the channel is opened
 * using fx_open( ) to obtain the FAX channel device
 * handle in devhandle.
 */

DF_IOTT iott;

int fx_sendascii(faxname,phdcont)
   char * faxname;
   unsigned short phdcont
{
   int erc;

   /* Open the file as read-only. */
   if ((iott.io_fhandle = dx_fileopen(faxname,O_RDONLY|O_BINARY,0)) == -1) {
         return(-1);
   }
   /*
    * Set up the DF_IOTT structure as the default and then
    * change the necessary fields.
    */
   fx_setiott(&iott,iott.io_fhandle,DF_ASCII,phdcont);
   iott.io_type |= IO_EOT;

   erc = fx_sendfax(devhandle,&iott, EV_SYNC)

   dx_fileclose(iott.io_fhandle);

   return(erc);
}
```

■ Errors

See *Appendix D* for a list of common error codes that may be returned for this function.

■ See Also

- **ATFX_TERMMSK( )**
- **fx_sendfax( )**
- **fx_setiott( )**

Name:	int fx_sendfax(dev, iotp, sndflag)	
Inputs:	int dev	• fax channel device channel
	DF_IOTT *iotp	• pointer to fax transfer table
	unsigned long sndflag	• mode flag
Returns:	0 if success (on invocation in asynchronous mode)	
	-1 if failure (on invocation in asynchronous mode)	
Includes:	srllib.h	
	dxxxlib.h	
	faxlib.h	
Category:	send fax	
Mode:	synchronous/asynchronous	
Platform:	DM3, Springware	

■ Description

The **fx_sendfax( )** function transmits fax data as specified by a table of DF_IOTT data structures.

The **fx_sendfax( )** function can be issued by the fax transmitter or the fax receiver. You can stop a fax transfer in progress at any time by issuing **fx_stopch( )**.

You can also send faxes using fax convenience functions. See **fx_sendascii( )**, **fx_sendraw( )** and **fx_sendtiff( )**.

For more information on setting up the channel device to send fax data, see *Chapter 5. Implementing Send Fax Capability*.

Parameter	Description
dev	Specifies the channel device handle for the fax channel obtained when the channel was opened.
iotp	A pointer to the DF_IOTT table entries that describe the fax data.
sndflag	The **sndflag** field is a logical OR bit mask that indicates one or more conditions: • mode of operation for the function, synchronous or asynchronous • enable generation of Phase B events (T.30 pre-message procedure) • enable generation of Phase D events (T.30 post-message procedure) • enable accepting and issuing operator intervention (voice request) from remote station • enable transmitting all DF_IOTT entries at low or high resolution • enable transmitting of subaddress values The **sndflag** bit mask can have the following values: Mode bit (for more information, see *Section 5.7.1. Mode of Operation* on *page 69*):

Value	Description
DF_ENABLE_RTV	Enable remote terminal verification (RTV). You must set both DF_PHASEB and DF_ENABLE_RTV bits to enable RTV. This value is not supported on DM3 boards. If both bits are set, the Phase B event is sent to the application upon receipt of the DIS (digital identification signal) and the remote station ID. This allows the

Parameter	Description	
		application to verify the recipient's identity and abort transmission if necessary before the firmware responds with a DCS message (digital command signal). The application must then call **fx_rtvContinue( )** to indicate continuation or cancellation of the fax transfer. If both bits are not set, the Phase B event is received by the application after the firmware has already sent the DCS message to the recipient.
	EV_SYNC	Synchronous mode operation
	EV_ASYNC	Asynchronous mode operation
	Phase B, Phase D and Operator Intervention (Voice Request) enable bits. Set one or more of the following, where the default is disabled:	
	Value	**Description**
	DF_PHASEB	Enable Phase B event generation. When this bit is set, a TFX_PHASEB event is generated each time Phase B of the T.30 protocol is completed while **fx_sendfax( )** is transmitting fax data. For more information, see *Section 5.7.2. Enable Phase B Event Generation* on *page 70*.
	DF_PHASED	Enable Phase D event generation. When this bit is set, a TFX_PHASED event is generated each time Phase D of the T.30 protocol is completed during the send fax operation. A Phase D event

Parameter	Description	
		is generated for every page except for the last page. After the last page, if your application is running in synchronous mode **fx_sendfax( )** completes or in asynchronous mode a TFX_FAXSEND event occurs. For more information, see *Section 5.7.3. Enable Phase D Event Generation* on *page 71*.
	DF_ACCEPT_VRQ	Enable accepting voice request from remote station. This value is not supported on DM3 boards.
	DF_ISSUE_VRQ	Enable issuing voice request to remote station. This value is not supported on DM3 boards.

Vertical resolution of the fax transmission. The default is to transmit at the resolution specified in the file (TIFF/F) or DF_IOTT (raw and ASCII):

Value	Description
DF_TXRESLO	Transmit all DF_IOTT entries at low (coarse) vertical resolution.
DF_TXRESHI	Transmit all DF_IOTT entries at high (fine) vertical resolution.

Transmit subaddress enable bit:

Value	Description
DF_TXSUBADDR	Enable subaddress transmission. This value is not supported on DM3 boards.

■ Cautions

- You must declare the DF_IOTT structures passed as an argument to **fx_sendfax( )** as global or static.

- Do not modify the DF_IOTT structures until after the **fx_sendfax( )** function has completed. The DF_IOTT structures must exist for the duration of the fax transmission.

  NOTE: In asynchronous mode, the fax library needs to repeatedly access the DF_IOTT structure entries during the fax transmission, even though **fx_sendfax( )** has returned control to the application. Each channel controlled by the single process must have its own separate DF_IOTT structures.

- The **io_type** field of the last DF_IOTT structure entry must contain an IO_EOT to identify it as the last structure entry.

- On DM3 boards, all DF_IOTT structures are checked before any fax is sent, and a fax is not sent if a bad DF_IOTT structure is included anywhere in the **fx_sendfax( )** function. In such a case, a phase B event is not generated even for any initial good DF_IOTT structures.

■ Examples

Example 1 illustrates the use of **fx_sendfax( )** in synchronous mode with DF_IOTT structures as an array.

Example 2 shows **fx_sendfax( )** with DF_IOTT structures set for raw image merged with ASCII data followed by a multi-page TIFF/F file. The synchronous programming code fragments shown can be used in a multi-threaded application where the program creates a separate thread for every channel. Each thread can control a single channel using a synchronous mode of operation.

Example 3 illustrates the use of **fx_sendfax( )** in asynchronous mode. The asynchronous programming code fragments shown can be used in a multi-threaded application where the program creates multiple threads. Each thread can control a single channel or multiple channels using an asynchronous mode of operation. See the *Standard Runtime Library* documentation for information on programming modes and the SRL functions.

Example 4 shows how to use **fx_sendfax( )** to send two TIFF/F files with each file being routed to a different subaddress.

Example 1: Send Fax with Array-Based DF_IOTT - Synchronous

Notes for this example:

- By not defining **io_type** in the first DF_IOTT entry, the next DF_IOTT entry is a DF_IOTT array entry by default. Array entries must be contiguous (0, 1, 2, etc.). To explicitly state that the next DF_IOTT is contiguous, specify the IO_CONT value in the **io_type** field.
- To indicate that data is stored on a disk device, IO_DEV is specified in the **io_type** field of the **fx_setiott( )** function (see the **fx_setiott( )** function reference source code).
- In the last DF_IOTT entry of Example 1, the IO_EOT value in the **io_type** field indicates the last DF_IOTT entry in the table.
- The **fx_setiott( )** calls could specify DFC_AUTO for automatic Phase D continuation determination.

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>

#define NUMDOC 3

/* Need a DF_IOTT entry for each document to send. */
DF_IOTT iott[NUMDOC];
int rawfd,tifd1,tifd2;

int voxdev;   /* Voice channel device handle. */
int dev;      /* Fax channel device handle. */
unsigned short value;
.
.
/*
 * Open the channel using dx_open( ) to obtain the
 * VOICE channel device handle in voxdev.
 */
if ((voxdev = dx_open("dxxxB1C1", NULL)) == -1) {
   /* Error opening device. */
      /* Perform system error processing */
      exit(1);
}

/*
 * Open the channel using fx_open( ) to obtain the FAX
 * channel device handle in dev.
 */
if ((dev = fx_open("dxxxB1C1", NULL)) == -1) {
   /* Error opening device. */
   /* Perform system error processing */
   exit(1);
}
.
.
/*
 * Take channel offhook using dx_sethook( ) and perform
 * outbound dial using dx_dial( ). Use voxdev as
 * channel device handle for Voice API functions.
 */
.
.
/* Required -- Set initial state of FAX channel to TRANSMITTER. */
if (fx_initstat(dev,DF_TX) == -1) {
   printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
      ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
}
.
.
/*
 * Enable automatic Phase D messaging for TIFF/F file inter-page Phase D
```

```
 * value. (NOTE: Specific Phase D messaging could have been used for each
 * DF_IOTT structure if required for the application.
 */
value = DFC_AUTO;

if (fx_setparm(dev,FC_SENDCONT,(void *)&value) == -1) {
   printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
      ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
}

/* Open raw and TIFF/F files to transmit. */
rawfd = dx_fileopen("coversht.raw",O_RDONLY|O_BINARY, NULL);
tifd1 = dx_fileopen("document1.tif",O_RDONLY|O_BINARY, NULL);
tifd2 = dx_fileopen("document2.tif",O_RDONLY|O_BINARY, NULL);

/*
 * Set DF_IOTT structure (using fx_setiott( )).Send Phase D
 * continuation value MPS after the raw format file
 * coversheet.
 */
fx_setiott(&iott[0],rawfd,DF_RAW,DFC_MPS);

/*
 * Set next DF_IOTT structure in the array. Send Phase D
 * continuation value EOM after the first TIFF/F document:
 * more pages to follow; renegotiate Phase B.
 */
fx_setiott(&iott[1],tifd1,DF_TIFF,DFC_EOM);

/*
 * Set the next DF_IOTT structure in the array. Send Phase D
 * continuation value EOP after the final TIFF/F document.
 * Send 2 pages, start at document page 3 (Note: TIFF/F
 * documents begin with document page zero).
 */
fx_setiott(&iott[2],tifd2,DF_TIFF,DFC_EOP);
iott[2].io_type |= IO_EOT;
iott[2].io_firstpg = 2L;
iott[2].io_pgcount = 2L;

/*
 * Set the fax state of the channel to DF_TX using
 * fx_initstat( ).
 */

/* Send all fax data now - synchronous mode. */
if (fx_sendfax(dev,iott,EV_SYNC) == -1) {
   printf("Error code: %ld Error message: %s\n",
      ATDV_LASTERR(dev), ATDV_ERRMSGP(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
   printf("Phase E status: %ld\n", ATFX_ESTAT(dev));
   /* Further error processing - application specific. */
   .
}
```

Example 2: Send Fax of Raw Image Merged with ASCII Data Followed by a Multi-Page TIFF/F File - Synchronous

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>

#define NUMDOC 4

/* Need a DF_IOTT entry for each document to send. */
DF_IOTT iott[NUMDOC];
DF_ASCIIDATA asciidata[2];
int rawfd,tiffd,txtfd1,txtfd2;
unsigned long sndflag = EV_SYNC;
unsigned short value;

int voxdev;   /* Voice channel device handle. */
int dev;      /* Fax channel device handle. */
.
.
/*
 * Open the channel using dx_open( ) to obtain the
 * VOICE channel device handle in voxdev.
 */
if ((voxdev = dx_open("dxxxB1C1", NULL)) == -1) {
   /* Error opening device. */
      /* Perform system error processing */
      exit(1);
}
/*
 * Open the channel using fx_open( ) to obtain the FAX
 * channel device handle in dev.
 */
if ((dev = fx_open("dxxxB1C1", NULL)) == -1) {
   /* Error opening device. */
   /* Perform system error processing */
   exit(1);
}
.
.
/*
 * Take the channel offhook using dx_sethook( ) and
 * perform outbound dial using dx_dial( ). Use voxdev
 * as channel device handle for Voice API functions.
 */
.
.
/* Set initial state of FAX channel to TRANSMITTER. */
if (fx_initstat(dev,DF_TX) == -1) {
   printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
      ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
}
```

```
.
.
/*
 * If this is a channel for a VFX/40SC (return type DFS_FAX40)
 * or VFX/40ESC (return type DFS_FAX40E),
 * a resolution for sending the entire DF_IOTT can
 * be specified in sndflag.
 */
if ((ATFX_CHTYPE(dev) == DFS_FAX40) ||
      (ATFX_CHTYPE(dev) == DFS_FAX40E))  {
   /* Set the transmit resolution to coarse (low). */
   sndflag |= DF_TXRESLO;
}

/*
 * Enable automatic Phase D messaging for TIFF/F file inter-page Phase D
 * value. (NOTE: Specific Phase D messaging could have been used for each
 * DF_IOTT structure if required for the application.
 */

value = DFC_AUTO;

if (fx_setparm(dev,FC_SENDCONT,(void *)&value) == -1) {
   printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
      ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
}
/* Open raw and TIFF/F files to transmit. */
rawfd = dx_fileopen("logo.raw",O_RDONLY|O_BINARY, NULL);
tiffd = dx_fileopen("document.tif",O_RDONLY|O_BINARY, NULL);
txtfd1 = dx_fileopen("ascii1.txt",O_RDONLY|O_BINARY, NULL);
txtfd2 = dx_fileopen("ascii2.txt",O_RDONLY|O_BINARY, NULL);

/*
 * Set DF_IOTT structure. The first fax page is to be created
 * by merging a raw image file with 2 ASCII text files on to
 * a single page.  Set io_phdcont  to DFC_MPG to cause the next
 * DF_IOTT entry's image to be appended to the same page.
 */
fx_setiott(&iott[0],rawfd,DF_RAW,DFC_MPG);

/* The raw file is at low resolution */
iott[0].io_resln = DF_RESLO;

/*
 * Set next DF_IOTT structure.  Set io_phdcont to DFC_MPG to
 * cause the next DF_IOTT entry's image to be appended to the
 * same page. This is the first ASCII file to be appended to the
 * raw image on a single page.
 *
 */
fx_setiott(&iott[1],txtfd1,DF_ASCII,DFC_MPG);

/* Set the Margins and other ASCII graphical attributes
 * in the DF_ASCIIDATA structure for the ASCII sub-page.
 */
asciidata[0].unit = DF_UNITS_IN10;     /* 1/10th inch units */
asciidata[0].leftmargin = 10;         /* 1" margins */
asciidata[0].rightmargin = 10;
asciidata[0].font = DF_FONT_0;         /* use normal font */
```

```
asciidata[0].linespace = DF_SINGLESPACE;
asciidata[0].tabstops = 0;

/* These fields will apply to all subsequent ASCII sub-pages */
asciidata[0].topmargin = 10;           /* 1" margins */
asciidata[0].botmargin = 10;
asciidata[0].pagelength = 110;         /* length of page */
asciidata[0].pagepad = DF_PAD;         /* pad to end of page */

/* Link the DF_ASCIIDATA to the DF_IOTT */
iott[1].io_datap = (void *)&asciidata[0];

/*
 * Set next DF_IOTT structure.  Send a Phase D continuation
 * of MPS  after this ASCII sub-page. This DF_IOTT entry completes
 * the MPG chain with the last ASCII sub-page merged with the images
 * defined by the previous DF_IOTT.
 */
fx_setiott(&iott[2],txtfd2,DF_ASCII,DFC_MPS);

/* Set the Margins and other ASCII graphical attributes
 * in the DF_ASCIIDATA structure for the ASCII sub-page.
 * Note that the Top/Bottom margins and Page Length/Page pad
 * will take effect from the first ASCII sub-page.
 */
asciidata[1].unit = DF_UNITS_IN10;     /* 1/10th inch units */
asciidata[1].leftmargin = 15;          /* 1.5" margins */
asciidata[1].rightmargin = 15;
asciidata[1].font = DF_FONT_0;         /* use normal font */
asciidata[1].linespace = DF_SINGLESPACE;
asciidata[1].tabstops = 0;

/* Link the DF_ASCIIDATA to the DF_IOTT */
iott[2].io_datap = (void *)&asciidata[1];

/*
 * Set last DF_IOTT structure in the chain. Send Phase D
 * continuation value EOP for the final document (TIFF/F
 * format); send 2 pages, starting at document page 3 (Note:
 * TIFF/F documents begin with document page zero).
 */
fx_setiott(&iott[3],tiffd,DF_TIFF,DFC_EOP);
iott[3].io_type |= IO_EOT;
iott[3].io_firstpg = 2L;
iott[3].io_pgcount = 2L;

/* Send all fax data now - synchronous mode. */
if (fx_sendfax(dev,&iott[0],sndflag) == -1) {
   printf("Error code: %ld Error message: %s\n",
      ATDV_LASTERR(dev), ATDV_ERRMSGP(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
   /* Further error processing - application specific. */
   .
   .
}
```

Example 3: Send Fax - Asynchronous Programming Mode

```
#include <stdio.h>
#include <string.h>

#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>

#define MAXCHANS 12

/* Global variables. */

int catchall( );
int fax_send( );

/* Error routine - print error information. */

void print_err(dev)
   int dev;
{
   printf("Error - %s (error code %d)\n",
      ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
      /* Perform system error processing */
   }
   return;
}

/*
 * main( ): Opens all channels and enables handler for
 * asynchronous operation. Channels go off-hook, dial the
 * appropriate number and send the fax document.
 */

main( )
{
   int chan;
   char * chnamep;
   int mode = SR_STASYNC;

   int voxdev;   /* Voice channel device handle. */
   int faxdev;   /* Fax channel device handle. */

     /* Set SRL to turn off creation of internal thread */
     if( sr_setparm( SRL_DEVICE, SR_MODELTYPE, &mode ) == -1 ){
        printf( "Error: cannot set srl mode\n" );
        exit( 1 );
     }

   for (chan=0; chan < MAXCHANS; chan++) {

      /*
       * Set chnamep to the channel device name, e.g.,
       * dxxxB1C1, dxxxB1C2, etc.
       * Open the channel using dx_open( ) such that voxdev
       * has the VOICE channel device handle.
```

```
       * Open the channel using fx_open( ) such that faxdev
       * has the FAX channel device handle.
       */

     if(( voxdev = dx_open( chnamep, NULL )) == -1 ){
          printf( "Error: cannot open vox device\n" );
          exit( 1 );
     }

     if(( faxdev = fx_open( chnamep, NULL )) == -1 ){
          printf( "Error: cannot open fax device\n" );
          exit( 1 );
     }
       .
       .
       .
   /* enable a handler for all events on any devices */
     if( sr_enbhdlr( EV_ANYDEV, EV_ANYEVT, dx_handler ) == -1 ){
         printf( "Error: could not enable handler\n" );
         exit( 1 );
     }
       .
       .
       .
   /*
   * Place channel on-hook by calling dx_sethook( ) with
   * its mode field set to EV_ASYNC (asynchronous).
   */
     if( dx_sethook( voxdev, DX_ONHOOK, EV_ASYNC ) == -1 ){
         printf( "dx_sethook failed: error = %s\n", ATDV_ERRMSGP( voxdev ));
         exit( 1 );
     }

      .
      .
      /*
       * Enable automatic Phase D messaging for TIFF/F file inter-page
       * Phase D value by setting FC_SENDCONT to DFC_AUTO.
       */
      .
      .
   }

   /*
    * All channels have been opened and a sethook function
    * issued to place the channels on-hook. Use sr_waitevt( )
    * to wait for completion events. On receiving any
    * completion event, control is transferred to the
    * catchall( ) handler function.
    */
    while(sr_waitevt(-1))
    .
    .

/* Event handler. */

/*
 * This routine gets called when sr_waitevt( ) receives any event.
 * Maintain a state machine for every channel and issue the
 * appropriate function depending on the next action to be
 * performed on the channel, e.g., the application may wish
 * to perform an outbound dial after receiving an offhook
```

```
 * completion event.
 */

int catchall( )

{

   int dev;
   char * fnamep;
   long phdcmd, phdrpy;

   dev = sr_getevtdev( );

   /* Determine the event. */
   switch(sr_getevttype( ))  {

   case TDX_SETHOOK:
      .
      .

      break;

   case TDX_DIAL:
      /* Dial complete. */
      .
      .
      /*
       * Connection has been established with remote
       * receiver. Prepare to send fax. Call fax_send( ) -
       * fnamep is the name of the file (TIFF/F) containing
       * the document to be sent.
       */
      if (fax_send(dev, fnamep,DF_TIFF) == -1) {
         /*
          * Application specific error handling here;
          * fax_send( ) prints out error information.
          */
         .
         .
      }

      break;

   case TFX_FAXSEND:
      /* The document has been successfully sent. */
      printf("Sent %ld pages at speed %ld, resln %ld,
         width %ld\n", ATFX_PGXFER(dev), ATFX_SPEED(dev),
         ATFX_RESLN(dev), ATFX_WIDTH(dev));

      /* Set channel on-hook; fax session completed. */
      .
      .
      break;

   case TFX_FAXERROR:
      /* Error during the fax session. */
      print_err(dev);
      printf("Phase E status %ld\n", ATFX_ESTAT(dev));
      /* Application specific error handling. */
      .
      .
      break;
```

```
   default:
      .
      .
      break;
   } /* End of switch. */

   return(0);
}


/*
 * This routine is called from the catchall( ) event handler
 * after an outbound dial has successfully completed and a
 * fax document has to be sent. The fax_send( ) routine will
 * perform the necessary initialization of the DF_IOTT
 * structure and call fx_sendfax( ) to send the document.
 */

int fax_send(dev, filenamep, datatype)
   int dev;
   char * filenamep;
   int datatype;
{
   int fhandle;

   /*
    * Set the Local ID using fx_setparm( ) and set the
    * initial state of the channel to be a transmitter
    * (DF_TX) using fx_initstat( ).
    */
    .

   /*
    * Set up the DF_IOTT structure to send the required
    * document.
    */
   if((fhandle = dx_fileopen(filenamep, O_RDONLY|O_BINARY, NULL))==-1) {
      printf("Unable to open send file %s\n",filenamep);
      return(-1);
   }

   fx_setiott(&iott, fhandle, datatype, DFC_EOP);
   iott.io_type |= IO_EOT;

/*
 * Set the fax state of the channel to DF_TX using
 * fx_initstat( ).
 */
.

   if (fx_sendfax(dev, &iott, EV_ASYNC) == -1) {
      printf("Error issuing sendfax\n");
      print_err(dev);
      .
      .
      return(-1);
   }
   return(0);

}
```

Example 4: Send Fax with two TIFF/F files, Each File to a Different Subaddress

```
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

#define NUMDOC 2

/* Need a DF_IOTT entry for each document to send. */
DF_IOTT iott[NUMDOC];
int tifd1,tifd2;

int dev;          /* Fax channel device handle */
int voxdev;       /* Voice channel device handle */
unsigned short phdcmd;
char *subaddr1 "3865";
char *subaddr2 "3923";

.
.
/*
 * Open the channel using dx_open( ) to obtain the
 * voice channel device handle in voxdev.
 */

if ((voxdev = dx_open("dxxxB1C1", NULL)) == -1) {
   /* Error opening device */
   /* Perform system error processing */
   exit(1);
}
/*
 * Open the channel using fx_open( ) to obtain the fax
 * channel device handle in dev.
 */
if ((dev = fx_open("dxxxB1C1", NULL)) == -1) {
   /* Error opening device */
   /* Perform system error processing */
   exit(1);
}
.
.
/*
 * Take channel offhook using dx_sethook( ) and perform
 * outbound dial using dx_dial( ). Use voxdev as
 * channel device handle for Voice API functions.
 */
.
.
/* Required -- set initial state of channel to Transmitter */
if (fx_initstat(dev,DF_TX) == -1) {
   printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
      ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
}
/*
 * Change the default FC_SENDCONT parameter value (DFC_EOM) to DFC_AUTO.
 * Note: The FC_SENDCONT parameter controls the Phase D command sent between
 * pages of a multi-page TIFF/F file. When subaddress fax routing is specified in
 * fx_sendfax( ), DFC_AUTO sets the FC_SENDCONT parameter value to DFC_MPS for the
```

```
 * fax transmission. If the FC_SENDCONT value is left at the default (DFC_EOM) when
 * subaddress fax routing is specified, the DFC_EOM value would indicate that each
 * page of the multi-page TIFF/F file should be sent to a different subaddress.
 * With DFC_EOM, renegotiation of Phase B would take place after each TIFF/F page.
 */

phdcmd = DFC_AUTO;

if (fx_setparm(dev,FC_SENDCONT,(void *)&phdcmd) == -1) {
   printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
      ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
}

/* Open TIFF/F files to transmit. */
tifd1 = dx_fileopen("file3865.tif",O_RDONLY | O_BINARY);
tifd2 = dx_fileopen("file3923.tif",O_RDONLY | O_BINARY);

/*
 * To allow the application to route the second TIFF/F file to a different
 * subaddress, set the first TIFF/F file's DF_IOTT data structure's io_phdcont
 * field value to DFC_EOM. DFC_EOM will force a Phase B negotiation after all
 * specified pages of the first TIFF/F file are sent (the io_type field value
 * should be set to IO_EOT). The second subaddress is sent to the receiver
 * during the second Phase B negotiation. Note: This only needs to be done when
 * sending fax data to more than one subaddress. If all fax data is to be sent to
 * one subaddress, set the parameter once and send the entire fax.
 */
fx_setiott(&iott[0],tifd1,DF_TIFF,DFC_EOM);
iott[0].io_type |= IO_EOT;

/*
 * Set the next DF_IOTT structure in the array. Send Phase D
 * continuation value DFC_EOP after the final TIFF/F document.
 */
fx_setiott(&iott[1],tifd2,DF_TIFF,DFC_EOP);
iott[1].io_type |= IO_EOT;

/* Set the subaddress parameter for the first TIFF/F file. */
if ((rc = fx_setparm(dev, FC_TXSUBADDR, subaddr1)) == -1) {
   printf("\nTXSUBADDR setparm  Error : %s", ATDV_ERRMSGP(dev)
   .
   .
}
/* Send the first file. */
if (fx_sendfax(dev, &iott[0] , DF_TXSUBADDR|EV_SYNC) == -1) {
   printf("Error code: %ld Error message: %s\n",
      ATDV_LASTERR(dev), ATDV_ERRMSGP(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
   printf("Phase E status: %ld\n", ATFX_ESTAT(dev));
   /* Further error processing - application specific. */
   .
   .
}
/* Update the subaddress parameter for the second TIFF/F file. */
if ((rc = fx_setparm(dev, FC_TXSUBADDR, subaddr2)) == -1) {
   printf("\nTXSUBADDR setparm  Error : %s", ATDV_ERRMSGP(dev)));
   .
```

```
}
.
.
/* Send the second file. */
if (fx_sendfax(dev, &iott[1] , DF_TXSUBADDR|EV_SYNC) == -1) {
   printf("Error code: %ld Error message: %s\n",
      ATDV_LASTERR(dev), ATDV_ERRMSGP(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
   printf("Phase E status: %ld\n", ATFX_ESTAT(dev));
   /* Further error processing - application specific. */
   .
   .
}
.
.
```

■ Example 5: Send fax using callback handler and setting SRL to operate in polled mode - Asynchronous (Linux only)

```
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

#define MAXCHANS 12

/* Global variables. */
extern int errno;
DF_IOTT iott;

int catchall( );
int fax_send( );

/* Error routine - print error information. */

void print_err(dev)
            int dev;
{
            printf("Error - %s (error code %d)\n",
                        ATDV_ERRMSGP(dev),ATDV_LASTERR(dev));
            if (ATDV_LASTERR(dev)==EDX_SYSTEM) {
                        printf("errno = %d\n", errno);
            }
            return;
}

/*
 * main( ): Opens all channels and enables handler for
 * asynchronous operation. Channels go off-hook, dial the
 * appropriate number and send the fax document.
 */

main( )
```

```
{
            int chan;
            int voxdev; /* Voice channel device handle. */
            int faxdev; /* Fax channel device handle. */
            char * chnamep;
            int mode = SR_POLLMODE;

            /* Set SRL to operate in POLLED Mode. */
            if(sr_setparm(SRL_DEVICE, SR_MODEID, &mode) == -1) {
                        printf("Cannot set SRL in polled mode\n");
                        .
                        .
            }

            for (chan=0; chan < MAXCHANS; chan++) {

                        /*
                         * Set chnamep to the channel device name, e.g.,
                         * dxxxB1C1, dxxxB1C2, etc.
                         * Open the channel using dx_open( ) such that voxdev
                         * has the VOICE channel device handle.
                         * Open the channel using fx_open( ) such that faxdev
                         * has the fax channel device handle.
                         */
                        .
                        .
                        /*
                         * Using sr_enbhdlr( ), set up handler to service all
                         * events.
                         */
                        if(sr_enbhdlr(voxdev, EV_ANYEVT, catchall) == -1) {
                                    printf("sr_enbhdlr( ) failed\n");
                                    print_err(dev);
                                    .
                                    .
                        }
                        if(sr_enbhdlr(faxdev, EV_ANYEVT, catchall) == -1) {
                                    printf("sr_enbhdlr( ) failed\n");
                                    print_err(dev);
                                    .
                                    .
                        }
                        /*
                         * Place channel on-hook by calling dx_sethook( ) with
                         * its mode field set to EV_ASYNC (asynchronous).
                         */
                        .
                        .
                        /*
                         * Enable automatic Phase D messaging for TIFF/F file inter-page
                         * Phase D value by setting FC_SENDCONT to DFC_AUTO.
                         */
                        .
                        .
            }

             /* This will cause catchall() to be called when an event is available */

             for(;;)
             {
                   sr_waitevt(-1);
```

```
            }

}

/* Event handler. */

/*
 * This routine gets called by SRL on receiving any event.
 * Maintain a state machine for every channel and issue the
 * appropriate function depending on the next action to be
 * performed on the channel, e.g., the application may wish
 * to perform an outbound dial after receiving an offhook
 * completion event.
 */

int catchall( )
{
            int dev;
            char * fnamep;
            long phdcmd, phdrpy;
            dev = sr_getevtdev( );
            /* Determine the event. */
            switch(sr_getevttype( ))  {
            case TDX_SETHOOK:
                        .
                        .
                        break;
            case TDX_DIAL:
                        /* Dial complete. */
                        .
                        .
                        /*
                         * Connection has been established with remote
                         * receiver. Prepare to send fax. Call fax_send( ) -
                         * fnamep is the name of the file (TIFF/F) containing
                         * the document to be sent.
                         */
                        if (fax_send(dev, fnamep,DF_TIFF) == -1) {
                                    /*
                                     * Application specific error handling here;
                                     * fax_send( ) prints out error information.
                                     */
                                    .
                                    .
                        }
                        break;
            case TFX_FAXSEND:
                        /* The document has been successfully sent. */
                        printf("Sent %ld pages at speed %ld, resln %ld,
                                    width %ld\n", ATFX_PGXFER(dev), ATFX_SPEED(dev),
                                    ATFX_RESLN(dev), ATFX_WIDTH(dev));
                        /* Set channel on-hook; fax session completed. */
                        .
                        .
                        break;
            case TFX_FAXERROR:
                        /* Error during the fax session. */
                        print_err(dev);
                        printf("Phase E status %ld\n", ATFX_ESTAT(dev));
                        /* Application specific error handling. */
                        .
                        .
```

```
                        break;
                default:
                        .
                        .
                        break;
                } /* End of switch. */
                return(0);
}

/*
 * This routine is called from the catchall( ) event handler
 * after an outbound dial has successfully completed and a
 * fax document has to be sent. The fax_send( ) routine will
 * perform the necessary initialization of the DF_IOTT
 * structure and call fx_sendfax( ) to send the document.
 */

int fax_send(dev, filenamep, datatype)
                int dev;
                char * filenamep;
                int datatype;
{
                int fhandle;
                /*
                 * Set the Local ID using fx_setparm( ) and set the
                 * initial state of the channel to be a transmitter
                 * (DF_TX) using fx_initstat( ).
                 */
                 .
                /*
                 * Set up the DF_IOTT structure to send the required
                 * document.
                 */
                if((fhandle = open(filenamep, O_RDONLY))==-1) {
                        printf("Unable to open send file %s\n",filenamep);
                        return(-1);
                }
                fx_setiott(&iott, fhandle, datatype, DFC_EOP);
                iott.io_type |= IO_EOT;
/*
 * Set the fax state of the channel to DF_TX using
 * fx_initstat( ).
 */
.
                if (fx_sendfax(dev, &iott, EV_ASYNC) == -1) {
                        printf("Error issuing sendfax\n");
                        print_err(dev);
                        .
                        .
                        return(-1);
                }
                return(0);
}
```

■ Errors

See *Appendix D* for a list of error codes that may be returned for this function. See *Section 11.3. Error Handling* on *page 148* for more information on Standard Runtime Library events generated.

- If errors occur during transmission, a Standard Runtime Library event (TFX_FAXERROR) is generated. The error code is accessible by issuing the standard attribute function **ATDV_LASTERR( )**. The standard attribute function **ATDV_ERRMSGP( )** returns a string describing the error.
- If **fx_sendfax( )** returns an error, you can locate the DF_IOTT structure processed when the error occurred by using the fax extended attribute **ATFX_LASTIOTT( )**.
- If **fx_sendfax( )** successfully completes, a Standard Runtime Library event (TFX_FAXSEND) is generated.
- System errors generate an EDX_SYSTEM error code. On Linux, check the global variable errno for more information. On Windows, use **dx_fileerrno( )** to obtain error value. Refer to the **dx_fileerrno( )** function in the *Voice API Library Reference* for a list of possible system error values.

See Also

- **ATFX_*name* functions**
- **fx_getDCS( )**
- **fx_getDIS( )**
- **fx_getNSF( )**
- **fx_sendascii( )**
- **fx_sendraw( )**
- **fx_sendtiff( )**
- **fx_setiott( )**
- **fx_setuio( )**

Name:	int fx_sendraw(faxname, width, resln, phdcont)	
Inputs:	char * faxname	• name of raw file to send
	unsigned short width	• carriage width to send the data
	unsigned char resln	• vertical resolution to send data
	unsigned char phdcont	• Phase D continuation value
Returns:	0 if success	
	-1 if failure	
Includes:	srllib.h	
	dxxxlib.h	
	faxlib.h	
Category:	convenience	
Mode:	synchronous	
Platform:	DM3, Springware	

Description

The **fx_sendraw( )** function allows an application to send a single page of raw fax data, unformatted MH Group 3 data at the width and resolution specified by the function parameters.

This function is a convenience function and is resident in *faxconv.c*. The **fx_sendraw( )** function calls **fx_sendfax( )** as illustrated in the source code for **fx_sendraw( )**. For detailed information on Phase D continuation values, status information and file error handling that applies to **fx_sendraw( )**, see the **fx_sendfax( )** function reference.

The encoding scheme used in transmitting fax data varies by product; for more information, see *Section 2.3. Key Product Features* on *page 8*. The preferred encoding scheme for transmission is determined by the value set in the FC_TXCODING parameter in **fx_setparm( )**.

Parameter	Description	
faxname	Specifies the name of the raw file to send.	
width	Specifies the carriage width. Valid values:	
	DF_WID1728	1728 pixels per line
	DF_WID2048	2048 pixels per line
	DF_WID2432	2432 pixels per line
resln	Specifies the vertical data resolution. Valid values:	
	DF_RESHI	High (fine) resolution (196 lpi)
	DF_RESLO	Low (coarse) resolution (98 lpi)
phdcont	Specifies the Phase D continuation value. Valid values:	
	DFC_EOP	End of Procedure (T.30). Terminate current fax session; progress to Phase E; and disconnect fax call.
	DFC_MPS	Multi-Page Signal (T.30). End of current fax document page; next page is in same format as the current page; proceed directly to Phase C.
	DFC_EOM	End of Message (T.30). End of current fax document page; more fax data to follow at different resolution or width; return to Phase B and negotiate parameters for next fax document page.

■ Cautions

- Before calling **fx_sendraw( )**, you must open the channel using **fx_open( )** to obtain the fax channel device handle.
- If TDM bus routing is required, the routing must be completed before calling the convenience function.

Example

```
#include <stdio.h>

#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int voxhandle;   /* Voice channel device handle. */
int devhandle;   /* Fax channel device handle. */

/*
 * Open the channel using dx_open( ) to obtain the
 * VOICE device handle in voxhandle.
 * Open the channel using fx_open( ) to obtain the FAX channel
 * device handle in devhandle.
 */

/*
 * Take channel offhook using dx_sethook( ) and perform outbound
 * dial using dx_dial( ).
 */
.
.

/*
 * Set the fax state of the channel to DF_TX using
 * fx_initstat( ).
 */
.
.

/*
 * Transmit raw document at page width 1728 pixels per line
 * at low (coarse) vertical resolution and disconnect when
 * finished.
 */
if (fx_sendraw("document.raw",DF_WID1728,DF_RESLO,DFC_EOP) == -1 {
   printf("Error code: %ld Error message: %s\n",
      ATDV_LASTERR(devhandle), ATDV_ERRMSGP(devhandle));
   if (ATDV_LASTERR(devhandle) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
   printf("Phase E status: %ld\n", ATFX_ESTAT(devhandle));
}
```

Source Code for fx_sendraw( )

```
/*
 * NOTE: devhandle is a global variable of type int. Prior to
 * calling fx_sendraw( ), the channel is opened using
 * fx_open( ) to obtain the FAX channel device handle in
 * devhandle.
 */
   DF_IOTT iott;

int fx_sendraw(faxname,width,resln,phdcont)
   char * faxname;
   unsigned short width;
   unsigned char resln;
   unsigned char phdcont
{
   int erc;

   /* Open the file as read-only. */
   if ((iott.io_fhandle = dx_fileopen(faxname,O_RDONLY|O_BINARY,0)) == -1) {
         return(-1);
   }

   /*
    * Set up the DF_IOTT structure as the default and then
    * change the necessary fields.
    */
   fx_setiott(&iott,iott.io_fhandle,DF_RAW,phdcont);
   iott.io_type |= IO_EOT;
   iott.io_width = width;
   iott.io_resln = resln;

   erc = fx_sendfax(devhandle,&iott, EV_SYNC)

   dx_fileclose(iott.fhandle);

   return(erc);

}
```

Errors

See *Appendix D* for a list of error codes that may be returned for this function.

See Also

- **ATFX_TERMMSK( )**
- **fx_sendfax( )**
- **fx_setiott( )**

Name:	int fx_sendtiff(faxname, firstpg, pgcount, phdcont)	
Inputs:	char * faxname	• pointer to name of TIFF/F file to send
	unsigned long firstpg	• first page to send (0 = first page in file)
	unsigned long pgcount	• number of consecutive pages to send (-1 = send all remaining pages in file)
	unsigned short phdcont	• Phase D continuation value
Returns:	0 if success -1 if failure	
Includes:	srllib.h dxxxlib.h faxlib.h	
Category:	convenience	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **fx_sendtiff( )** function allows an application to send pages of a single TIFF/F file at the width and resolution set in the TIFF/F.

This function is a convenience function and is resident in *faxconv.c.* The **fx_sendtiff( )** function calls **fx_sendfax( )** as illustrated in the source code for **fx_sendtiff( )**. For detailed information on Phase D continuation values, status information and file error handling that apply to **fx_sendtiff( )**, see the **fx_sendfax( )** function reference.

See *Appendix A* for a list of TIFF/F tags and values.

The encoding scheme used in transmitting fax data varies by product; for more information, see *Section 2.3. Key Product Features* on *page 8*. The preferred encoding scheme for transmission is determined by the value set in the FC_TXCODING parameter in **fx_setparm( )**.

Parameter	Description	
faxname	Pointer to the name of the TIFF/F file to send.	
firstpg	Specifies the document page number of the first page to send. 0 = first document page in file	
pgcount	Specifies the number of consecutive pages to send. -1 = send all remaining pages in the file	
phdcont	Specifies the Phase D continuation value. Valid values:	
	DFC_EOP	End of Procedure (T.30). Terminate current fax session; progress to Phase E; and disconnect fax call.
	DFC_MPS	Multi-Page Signal (T.30). End of current fax document page; next page is in same format as the current page; proceed directly to Phase C.
	DFC_EOM	End of Message (T.30). End of current fax document page; more fax data to follow at different resolution or width; return to Phase B and negotiate parameters for next fax document page.

■ Cautions

- Before calling **fx_sendtiff( )**, you must open the channel using **fx_open( )** to obtain the fax channel device handle.
- If TDM bus routing is required, the routing must be completed before calling the convenience function.

Example

```
#include <stdio.h>

#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int voxhandle;   /* Voice channel device handle. */
int devhandle;   /* Fax channel device handle. */


/*
 * Open the channel using dx_open( ) to obtain the
 * VOICE device handle in voxhandle.
 * Open the channel using fx_open( ) to obtain the FAX channel
 * device handle in devhandle.
 */

/*
 * Take channel offhook using dx_sethook( ) and perform outbound
 * dial using dx_dial( ).
 */
.
.
/*
 * Set the fax state of the channel to DF_TX using
 * fx_initstat( ).
 */
.
.
/*
 * Send 2 pages starting at page number 4, disconnect when
 * finished.
 */
if (fx_sendtiff("document.tif",4L,2L,DFC_EOP) == -1) {
   /* Process error. */
   printf("Error - %s (error code %d)\n",
      ATDV_ERRMSGP(devhandle), ATDV_LASTERR(devhandle));
   if (ATDV_LASTERR(devhandle) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
   printf("Phase E status: %ld\n", ATFX_ESTAT(devhandle));

   /* Application specific error handling. */
   .
   .

}
```

Source Code for fx_sendtiff( )

```
/*
 * NOTE: devhandle is a global variable of type int. Prior to
 * calling fx_sendtiff( ), the channel is opened using
 * fx_open( ) to obtain the FAX channel device handle in
 * devhandle.
 */
   DF_IOTT iott;

int fx_sendtiff(faxname,firstpg,pgcount,phdcont)
   char * faxname;
   unsigned long firstpg;
   unsigned long pgcount;
   unsigned short phdcont
{
   int erc;

   /* Open the file as read-only. */
   if ((iott.io_fhandle = dx_fileopen(faxname,O_RDONLY|O_BINARY,0)) == -1) {
         return(-1);
   }

   /*
    * Set up the DF_IOTT structure as the default and then
    * change the necessary fields.
    */
   fx_setiott(&iott,iott.io_fhandle,DF_TIFF,phdcont);
   iott.io_type |= IO_EOT;
   iott.io_firstpg = firstpg;
   iott.io_pgcount = pgcount;

   erc = fx_sendfax(devhandle,&iott, EV_SYNC)

   dx_fileclose(iott.io_fhandle);

   return(erc);

}
```

Errors

See *Appendix D* for a list of error codes that may be returned for this function.

See Also

- **ATFX_TERMMSK( )**
- **fx_sendfax( )**
- **fx_setiott( )**

Name:	void fx_setiott(iotp,fhandle,dtype,cont)	
Inputs:	DF_IOTT *iotp	• pointer to DF_IOTT
	int fhandle	• file descriptor
	unsigned short dtype	• type of fax data
	unsigned short cont	• Phase D continuation value
Returns:	None	
Includes:	srllib.h	
	dxxxlib.h	
	faxlib.h	
Category:	initialize DF_IOTT	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **fx_setiott( )** function sets up a DF_IOTT structure with default values for the specified type of fax data.

Use this function to initialize the DF_IOTT structure before setting specific DF_IOTT field values.

The DF_IOTT structure consists of fields describing the fax data for one fax document to be transmitted. A linked list or array of DF_IOTT structures can be created to specify multiple fax documents for transmission. The structure defines raw, TIFF/F and ASCII data.

The default setting for the **fx_setiott( )** function is to send fax data from a **file** with the next DF_IOTT entry contiguous in memory. See the **fx_setiott( )** Source Code section and the examples in the **fx_sendfax( )** function reference.

Parameter	Description	
iotp	Pointer to DF_IOTT structure. For more information on this structure, see *Section 10.6. DF_IOTT – Fax Transmit Data Description* on *page 128*.	
fhandle	Specifies the file descriptor.	
dtype	Specifies the type of fax data to be transmitted. Valid values:	
	DF_TIFF	TIFF/F structured formatted fax data
	DF_RAW	Raw, unformatted fax data
	DF_ASCII	ASCII text file data
cont	Specifies the Phase D continuation value. Valid values:	
	DFC_AUTO	Automatic Phase D Messaging. The fax library automatically determines the T.30 Phase D continuation value based on the width, resolution and position of the current fax page, next fax page and the remote receiver's capability. Possible values automatically assigned are DFC_EOM, DFC_EOP and DFC_MPS.
	DFC_MPG	Merge-page. The data specified for the DF_IOTT entry directly following the current DF_IOTT entry is concatenated to the same page.
	DFC_EOP	End of Procedure (T.30). Terminate current fax session, progress to Phase E and disconnect fax call.
	DFC_MPS	Multi-Page Signal (T.30). End of current fax document page; next fax document page is in the same format as the current page; proceed directly to Phase C.

Parameter	Description	
	DFC_EOM	End of Message (T.30). End of current fax document page; more fax data to follow at different resolution or width; return to Phase B and negotiate parameters for next fax document page.

NOTE: DFC_MPG and DFC_AUTO are fax library terms, not T.30 protocol terminology.

■ Details

For usage information on DF_IOTT, see *Section 5.5. Specifying Fax Data for Transmission in a DF_IOTT Table Entry* on *page 50.* For reference information on DF_IOTT, see *Section 10.6. DF_IOTT – Fax Transmit Data Description* on *page 128.*

Connecting DF_IOTT Entries

When the next DF_IOTT entry is contiguous in memory, the **io_type** logical OR field specifies IO_CONT, and **io_nextp** and **io_prevp** specify NULL.

When the next DF_IOTT entry is linked to the current entry, after the current entry's call to **fx_setiott( )**, the **io_type** logical OR field specifies IO_LINK and **io_nextp** specifies a pointer to the next entry. For sample code, see the examples in the **fx_sendfax( )** function reference.

TIFF/F File Entry Defaults

If **fx_setiott( ) dtype** parameter specifies DF_TIFF, the descriptor defines TIFF/F formatted data. The DF_IOTT default values for TIFF/F specify that all pages in the TIFF/F are transmitted:

io_firstpg = 0L;
io_pgcount = -1L;

If **io_phdcont** specifies DFC_MPG, the **io_pgcount** field is set to 1. Specify the starting page of the TIFF/F file to send in **io_firstpg**. For sample code, see the examples in the **fx_sendfax( )** function reference.

Raw File Entry Defaults

If **fx_setiott( ) dtype** parameter specifies DF_RAW, the descriptor defines raw fax data that includes no other formatting. The DF_IOTT default values for raw data specify that the following is sent: a disk file of raw, Group 3 MH-encoded data with no offset at the standard carriage width (8.5") and at fine (high) resolution:

io_offset = 0L;
io_length = -1L;
io_width = DF_WID1728;
io_resln = DF_RESHI;
io_coding = DF_MH;

For sample code, see the **fx_setiott( )** Source Code section.

ASCII File Entry Defaults

If **fx_setiott( ) dtype** parameter specifies DF_ASCII, the descriptor defines an ASCII text file. The DF_IOTT default values for ASCII text file specify that the following is sent: ASCII data at standard carriage width (8.5") and coarse (low) resolution. The NULL pointer passed to **io_datap** in place of the DF_ASCIIDATA structure results in default values being used for margins and other graphical attributes.

io_offset = 0L;
io_length = -1L;
io_width = DF_WID1728;
io_resln = DF_RESLO;
io_datap = (void *) NULL;

For sample code, see the **fx_setiott( )** Source Code section.

■ Example

See the examples in the **fx_sendfax( )** function reference.

■ Source Code for fx_setiott( )

```
void fx_setiott(iotp,fhandle,dtype,cont)
   DF_IOTT *iotp;
   int fhandle;
   unsigned short dtype;
   unsigned short cont;
   {

      /* Data in file, next entry contiguous. */
      iotp->io_type = IO_DEV;
      iotp->io_fhandle = fhandle;
      iotp->io_nextp = (DF_IOTT *)NULL;
      iotp->io_prevp = (DF_IOTT *)NULL;

      iotp->io_datatype = dtype;
      iotp->io_phdcont = cont;

      switch (dtype) {
         /* For TIFF/F, set up firstpg and pgcount to send all pages. */
         case DF_TIFF:
            iotp->io_firstpg = 0L;
            iotp->io_pgcount = (cont == DFC_MPG) ?1 : -1L;
            break;
         /*
          * For raw file, set up to send complete file at default width and
          * resolution.
          */
         case DF_RAW:
            iotp->io_offset = 0L;
            iotp->io_length = -1L;
            iotp->io_width  = DF_WID1728;
            iotp->io_resln  = DF_RESHI;
            iotp->io_coding = DF_MH;
            break;
         /*
          * For ASCII file, set up to send complete file at default width and
          * resolution.
          */
         case DF_ASCII:
            iotp->io_offset = 0L;
            iotp->io_length = -1L;
            iotp->io_width  = DF_WID1728;
            iotp->io_resln  = DF_RESLO;
            iotp->io_datap  = (void *)NULL;
            break;
         default:
         break;
      }
      return;
   }
```

■ Errors

None

■ See Also

- **ATFX_BADIOTT( )**
- **ATFX_BADPAGE( )**
- **ATFX_LASTIOTT( )**
- **ATFX_TFBADTAG( )**
- **ATFX_TFNOTAG( )**
- **ATFX_TFPGBASE( )**
- **fx_sendfax( )**
- **fx_sendascii( )**
- **fx_sendraw( )**
- **fx_sendtiff( )**

Name:	int fx_setparm(dev, parm, valuep)	
Inputs:	int dev	• fax channel device handle
	unsigned long parm	• parameter to set
	void *valuep	• pointer to parameter value
Returns:	0 if success	
	-1 if failure	
Includes:	srllib.h	
	dxxxlib.h	
	faxlib.h	
Category:	configuration	
Mode:	synchronous	
Platform:	DM3, Springware	

Description

The **fx_setparm( )** function sets the fax parameter of an open fax channel device.

Parameter	Description
dev	Specifies the device handle for the fax channel obtained when the channel was opened.
parm	Specifies the define for the parameter ID to be set (see the alphabetical list of fax parameters and values on the pages that follow).
valuep	Points to the location where the **parm** value is to be stored.

Many of the same parameter IDs are available for **fx_setparm( )** and **fx_getparm( )**; any differences are noted. These functions allow you to configure an open fax channel device and retrieve the parameters set for that fax channel device.

The parameters are used to set (or read) the following categories of information:

- TIFF/F file:
  - delimiters to store fax in multiple TIFF/F files
  - Phase D continuation value for multi-page TIFF/F files
  - base page numbering scheme for TIFF/F file
  - level of TIFF/F tag checking
- fax page header:
  - graphical attributes, such as bold, underline, and format
  - date and time format
  - starting page number
  - user-defined text
- encoding scheme for data transmission and reception
- baud rate for data transmission and reception
- local and remote ID, phone number used for fax transmission and reception
- number of retry attempts for unsuccessful fax transmission
- percentage of acceptable bad scan lines
- subaddress routing information, phone number/extension used in routing a fax

The following summarizes the types of parameter IDs and their purpose. The defines for parameter IDs are described in more detail following this table, in alphabetical order.

TIFF/F file settings:

FC_ENDDOC	* delimiters for multiple TIFF/F files
FC_SENDCONT	* Phase D value for multi-page TIFF/F
FC_TFPGBASE	* base page numbering scheme
FC_TFTAGCHECK	* level of TIFF/F tag checking

Fax page header settings:

FC_HDRATTRIB	* graphical attributes and format
FC_HDRDATEFMT	* date format
FC_HDRTIMEFMT	* time format
FC_HDRDATETIME	* user-defined date and time format
FC_HDRSTARTPAGE	* starting page number
FC_HDRUSER	* user-defined text
FC_HDRUSER2	* user-defined text

Encoding scheme settings:

FC_TXCODING	* encoding scheme for transmission
FC_RXCODING	* encoding scheme for reception

Baud rate settings:

FC_TXBAUDRATE	* baud rate for transmission
FC_RXBAUDRATE	* baud rate for reception

Local and remote ID settings:

FC_LOCALID	* phone number used for transmission
FC_REMOTEID	* phone number used for reception

Number of retry attempts setting:

FC_RETRYCNT	* number of retries for unsuccessful transmit

Percentage of acceptable bad scan lines:

FC_RTN	* percent before RTN is returned
FC_RTP	* percent before RTP is returned

Subaddress routing setting:

FC_TXSUBADDR	* phone number/ext. for fax routing
FC_REMOTESUBADDR	* contents of T.30 SUB message

Font handles for ASCII to fax conversion:

FC_FONT0	* font applied to ASCII text file
FC_FONT3	* font applied to header

Other:

FC_TXNSF	* pointer to customized NSF message

Define	Description
FC_ENDDOC	**Bytes:** 2 **Default:** DFS_EOP Value set for **fx_rcvfax( )** and **fx_rcvfax2( )** to return when DFS_EOP, DFS_EOM or DFS_MPS is received in Phase D. Used to set delimiters to store pages of incoming fax data into more than one TIFF/F file. The default (DFS_EOP) stores incoming fax data in a single TIFF/F file. For more information, see *Section 6.2.2. Storing Incoming Fax Data - Storing in Multiple TIFF/F Files*. Valid values for FC_ENDDOC: DFS_ALL Receive function returns all FC_ENDDOC values; that is, selects all valid FC_ENDDOC values as delimiters when receiving fax data in a TIFF/F file. DFS_EOM End of Message. Stores individual pages of a multi-page fax into separate TIFF/F files. Execute **fx_rcvfax( )** or **fx_rcvfax2( )** in a loop, using a different file to store each page every time the receive fax function is called. DFS_EOP End of Procedure (default). Stores all incoming pages in a single multi-page TIFF/F file. DFS_MPS Multi-page Signal.

Define	Description
	Stores individual pages of a multi-page fax in separate TIFF/F files.
	DFS_REMOTESUBADDR
	Remote subaddress fax routing.
	Stores fax data for each subaddress in separate files. This value is not supported on DM3 boards.
FC_FONT0	**Bytes:** 4 **Type:** HFONT **Default:** similar to OEM_FIXED_FONT (12 point)
	(Windows only)
	Specifies the font handle to be used by **fx_sendfax( )** or **fx_sendascii( )** to render ASCII documents in the default font. This define is not supported on DM3 boards.
	This font handle is used to render ASCII documents in the font specified by DF_FONT_0 in the DF_ASCIIDATA structure or as specified by the control character F0 in the ASCII document itself.
	This font handle specifies one of two fonts that can be active at one time on a fax channel device.
	For further information on font handles, see *Section 7.4. Overriding Fonts Set with fx_setparm( ).*
	For details on how to obtain other font handles or character sets, see your Software Development Kit documentation.
	Note the following:
	• The font handle must remain valid during the fax transmission; that is, the font handle must not be deleted until the fax transmission has completed. You must delete the handle when it is no longer

Define	Description
	needed. • During multiple fax transmissions, if one transmission uses a different set of fonts and you want to restore the fonts to their original or default values, you must use **fx_getparm( )** to obtain the default font handles for FC_FONT0 and FC_FONT3 before using **fx_setparm( )** to redefine handles. Valid values are any Windows font handle including font handles created from font files supplied by Intel:

Font Files	Description
DEFAULT0.FON	English language character set, 12 point
DEFAULT3.FON	English language character set, 9 point
KATAKNA0.FON	Japanese Katakana language character set, approximately 12 point
KATAKNA3.FON	Japanese Katakana language character set, approximately 9 point

FC_FONT3	**Bytes:** 4 **Type:** HFONT **Default:** similar to OEM_FIXED_FONT (9 point) (Windows only) Specifies the font handle to be used by **fx_sendfax( )** or **fx_sendascii( )** to render transmitted ASCII documents in the font specified by DF_FONT_3 in the DF_ASCIIDATA structure or as specified by the control character F3 in the ASCII document itself. This define is not supported on DM3 boards. This font handle specifies one of two font handles

Define	Description
	that can be active at one time on a fax channel device. FC_FONT3 also specifies the font for the header. For further information on font handles, see *Section 7.4. Overriding Fonts Set with fx_setparm( )* on *page 99*. For details on how to obtain other font handles or character sets, see your Software Development Kit documentation. Valid values are the same as for FC_FONT0. See the NOTES for FC_FONT0 for additional information.
FC_HDRATTRIB	**Bytes:** 2 **Default:** DF_HDRINSERT and DF_HDRFMT1 Bitmap parameter that indicates specific graphical attributes applied to the fax header and the selection of a page header format. Valid values for FC_HDRATTRIB: DF_HDRBOLD Header text is bold. This value is not supported on DM3 boards. DF_HDRDISABLE Fax header is not included on the image page. DF_HDRFMT1 Enable header format 1 (default format). DF_HDRFMT2 Enable header format 2. DF_HDRINSERT Header text is inserted before the image. This increases the page length by the number of scan lines in the header (default).

Define	Description
	DF_HDRUNDERLINE Header text is underlined. This value is not supported on DM3 boards. If the **DF_HDRFMT1** bit is set (default), the following information is included in the fax page header: <date> <time> <user field> From: <local ID> To: <remote ID> Page: <page #> **<date> <time>:** specified in FC_HDRDATEFMT and FC_HDRTIMEFMT. You can override this date and time string using FC_HDRDATETIME. **<user field>:** specified in FC_HDRUSER. Optional. **<local ID>:** specified in FC_LOCALID. The local station ID is implemented as a fax parameter that can be set at the user level. **<remote ID>:** specified in FC_REMOTEID. The remote station ID is automatically inserted, when available. This is a read-only parameter. **<page #>:** specified in FC_HDRSTARTPAGE. The page number is automatically inserted. There are 4 spaces between each field in the fax page header text. When the **DF_HDRFMT2** bit is set, the user header information specified for the FC_HDRUSER2 parameter is placed in the fax page header. See the FC_HDRUSER2 parameter for details.
FC_HDRDATEFMT	**Bytes:** 2 **Default:** DF_HDRDATEFMT_2 Fax header date format. The date appears before the time specified in FC_HDRTIMEFMT. Both date and time are generated internally in the

Define	Description
	specified format. **NOTE:** You can override this format using FC_HDRDATETIME. When overriding this format, you must set either FC_HDRDATEFMT to DF_HDRDATEFMT_0 or FC_HDRTIMEFMT to DF_HDRTIMEFMT_0. Valid values for FC_HDRDATEFMT: DF_HDRDATEFMT_0 Disable internal date generation; use user text string set in FC_HDRDATETIME DF_HDRDATEFMT_1 Date appears as: MM-DD-YYYY DF_HDRDATEFMT_2 Date appears as: MM/DD/YYYY DF_HDRDATEFMT_3 Date appears as: DD-MM-YYYY DF_HDRDATEFMT_4 Date appears as: DD/MM/YYYY DF_HDRDATEFMT_5 Date appears as: YYYY-DD-MM DF_HDRDATEFMT_6 Date appears as: YYYY/DD/MM DF_HDRDATEFMT_15 Date appears as: day month dd YYYY (Example: Fri Sep 13 1999)

Define	Description
FC_HDRDATETIME	**Default:** NULL User-defined text for fax header date/time string - 27 character maximum null-terminated ASCII string (26 characters + NULL; left-justified). Overrides the default date/time format specified in FC_HDRDATEFMT and FC_HDRTIMEFMT. To enable this, you must set either FC_HDRDATEFMT to DF_HDRDATEFMT_0 or FC_HDRTIMEFMT to DF_HDRTIMEFMT_0. See these parameter descriptions for more information.

FC_HDRSTARTPAGE

Bytes: 2

Default: 1

Starting page number that appears in the fax page header for the first transmitted fax page. Valid values for FC_HDRSTARTPAGE:

≥ 1	Starting page number to display in the fax page header.

When the fax page header DF_HDRFMT1 or DF_HDRFMT2 bit is set in the FC_HDRATTRIB parameter, at the beginning of a fax transmission the FC_HDRSTARTPAGE parameter is used to determine the page number to be displayed in the fax page header for the first transmitted fax page. On each subsequent page of the fax transmission, the page number is automatically increased by one.

NOTE: If the page number is greater than 99, only the last 2 digits of the page number are placed in the header.

To set a starting page number other than the default (1), set FC_HDRSTARTPAGE before issuing the first **fx_sendfax( )** of the fax transmission. It is up to the application to reset FC_HDRSTARTPAGE to the default (1) after the fax transmission is complete.

For example, if a 5-page fax transmission is interrupted after sending only 3 pages, redial, set FC_HDRSTARTPAGE to 4, then issue **fx_sendfax( )** to send the last 2 pages of the fax. The receiver would receive a 2-page fax with the pages numbered 4 and 5.

FC_HDRTIMEFMT

Bytes: 2

Default: DF_HDRTIMEFMT_1

Fax header time format. The time appears after the date specified in FC_HDRDATEFMT. Both date and time are generated internally in the specified format.

You can override this format using FC_HDRDATETIME. When overriding this format, you must set either FC_HDRTIMEFMT to DF_HDRTIMEFMT_0 or FC_HDRDATEFMT to DF_HDRDATEFMT_0.

Valid values for FC_HDRTIMEFMT:

DF_HDRTIMEFMT_0

Disable internal time generation; use user text string set in FC_HDRDATETIME

DF_HDRTIMEFMT_1

Time appears as: HH:MM am/pm (12-hour clock)

DF_HDRTIMEFMT_2

Time appears as: HH:MM (24-hour clock)

FC_HDRUSER

Default: NULL

User-defined text that appears in the fax header of every transmitted fax page - 32 character maximum null-terminated ASCII string (31 characters + NULL; centered).

If the string is greater than 31 characters, it is truncated. If it is less than 31 characters, it is centered.

On DM3 boards, the user information is not displayed because of a character number limitation.

FC_HDRUSER2

Default: NULL

User-defined text for fax header - 133 character maximum null-terminated ASCII string (132

Define	Description
	characters + NULL; centered). Allows your application to control the entire contents of the fax page header. To enable this feature, set the DF_HDRFMT2 bit in the FC_HDRATTRIB parameter. When enabled, the contents of FC_HDRUSER2 are placed in the fax page header instead of the contents of the default header format (DF_HDRFMT1). Two special escape sequences can be used in the FC_HDRUSER2 field: • **%R** This sequence is replaced with the 20 character remote ID, if available. If the remote ID is not available and %R is specified, the %R is replaced with 20 spaces. If the %R is within the last 20 bytes of the FC_HDRUSER2 field, the remote ID is truncated to fit into the field. If you include the %R sequence, the maximum number of characters in the FC_HDRUSER2 field is 115 (114 characters + NULL). • **%P** This sequence is replaced with the current fax page number. Only the last 2 digits are placed in the header. The current page number is automatically inserted for each page of the fax transmission (see FC_HDRSTARTPAGE). %% is interpreted as % and placed in the header. When using **fx_getparm( )** to read FC_HDRUSER2, you must provide a 133 byte buffer (132 characters + NULL).

Define	Description
FC_LOCALID	**Default:** NULL Local identification - 21 character null-terminated ASCII string (20 characters + NULL, left-justified). This is the phone number your fax application uses for transmission. It is included in the data portion of the T.30 Call Subscriber Identification (CSI), Transmitting Subscriber Identification (TSI) and Calling Subscriber Identification (CIG) messages (the CIG message is used when polling is specified). The FC_LOCALID parameter value is specified by your application. If the length of the string exceeds the maximum value, it is truncated. If the length of the string is less than the maximum value, the string is left-justified and padded with blanks to equal 20 characters.
FC_REMOTEID	**Default:** NULL (Windows only) Remote identification - 21 character null-terminated ASCII string (20 characters + NULL). When the length of the string is less than the maximum value, the string is left-justified and padded with blanks to equal 20 characters. **This is a read-only parameter**, used by **fx_getparm( )** only. It specifies the phone number used by the remote fax machine to transmit a fax to your fax application. It is included in the data portion of the T.30 Call Subscriber Identification (CSI), Transmitting Subscriber Identification (TSI) and Calling Subscriber Identification (CIG) messages (the CIG message is used in a polling fax transmission).

Define	Description
FC_REMOTESUBADDR	**Bytes:** 2 **Default:** NULL (Windows only) Subaddress information sent by the remote transmitter – 21 character NULL-terminated ASCII string (20 characters + NULL, left-justified). **This is a read-only parameter**, used by **fx_getparm( )** only. This define is not supported on DM3 boards. It specifies the contents of the T.30 SUB message if the T.30 SUB message is included as part of the fax transmission. The SUB message is sent by the transmitter during Phase B negotiations. The FC_REMOTESUBADDR parameter is updated after each Phase B negotiation. The contents of the parameter are valid from the completion of a Phase B negotiation that contains a T.30 SUB message until the next Phase B negotiation. The application can use the subaddress information to route fax data to specified subaddresses. If the subaddress is less than 20 characters, the field is padded with spaces.
FC_RETRYCNT	**Bytes:** 2 **Default:** DF_NORETRY The number of retry attempts for an unsuccessfully transmitted fax page. This define is ignored on DM3 boards. Valid values: DF_NORETRY No retries DF_RETRY1 One retry DF_RETRY2 Two retries

Define	Description
	DF_RETRY3 Three retries DF_RETRYDCN After specified number of retries, disconnect ("OR" this value with DF_RETRYn). To disconnect after the specified number of retry attempts, logically OR the retry count number value with the DF_RETRYDCN value (see the example in the **fx_setparm( )** function reference). After the specified number of retry attempts, the *send* function fails with a -1 and the transmitter disconnects the fax call.
FC_RTN	**Bytes:** 2 **Default:** 15 Percent of bad scan lines acceptable for a fax page before an RTN (Retrain Negative) message is returned to the transmitter at the completion of the fax page (in Phase D). Valid values are integers between 1 and 100. This define is not supported on DM3 boards.
FC_RTP	**Bytes:** 2 **Default:** 5 Percent of bad scan lines acceptable for a fax page before an RTP (Retrain Positive) message is returned to the transmitter at the completion of the fax page (Phase D). Valid values are integers between 1 and 100**.** This define is not supported on DM3 boards. The following activities occur when FC_RTP and FC_RTN are set to their default values, where FC_RTP = 5 and FC_RTN = 15, and the level of bad scan lines received per page is as noted: • Between 0 and 5 percent, the MCF (Message Confirmation) message is sent to the remote

Define	Description
	station. • Between 5 percent and 15 percent, the RTP (Retrain Positive) message is sent to the remote station. The next page is received after a training sequence. • Between 15 percent and 100 percent, the RTN (Retrain Negative) message is sent to the remote station, requesting retransmission of the current page. To determine the total number of RTN pages received (that is, the number of pages that required retransmission), call **ATFX_RTNPAGES( )**.
FC_RXBAUDRATE	**Bytes:** 2 **Default:** DF_MAXBAUD Maximum preferred baud rate for incoming fax data. Capability varies by product; see *Section 2.3. Key Product Features* on *page 8* for more information. Valid values: DF_MAXBAUD Maximum baud rate value for reception. DF_14400BAUD 14400 baud reception DF_9600BAUD 9600 baud reception DF_7200BAUD 7200 baud reception DF_4800BAUD 4800 baud reception Set this parameter to one of the supported baud rates to receive fax transmissions at a lower baud rate than the default. This parameter is useful when receiving fax over known noisy lines and you wish

Define	Description
	to explicitly set a lower initial baud rate; this saves time that would be taken in negotiating a lower baud rate. **NOTE:** If an invalid value is specified, the maximum receive baud rate for the channel is used.
FC_RXCODING	**Bytes:** 2 **Default:** DF_MH If supported, the encoding scheme in which the incoming fax data is stored in TIFF/F or raw file(s). Reset to MH when the channel is first opened. Valid values: DF_MH Modified Huffman (default) DF_MR Modified Read DF_MMR Modified Modified Read Calls to **fx_rcvfax( )** or **fx_rcvfax2( )** result in the storage of the incoming fax image data in the encoding scheme specified in FC_RXCODING. This value is in effect until it is reset. **NOTE:** If you set the FC_RXCODING parameter value to DF_MMR, Intel recommends that you reset it to DF_MH (default) before exiting the program. This ensures encoding scheme compatibility with other fax application programs that expect all incoming fax data to be stored in the MH encoding scheme. If this parameter is set on an unsupported product, **ATDV_LASTERR( )** returns an error, EFX_UNSUPPORTED.

FC_SENDCONT

Bytes: 2

Default: DFC_AUTO (DM3 boards)
Default: DFC_EOM (Springware boards)

Phase D value to be used by the transmitter between pages of a multi-page TIFF/F file. Valid values:

DFC_AUTO	Automatic Phase D Messaging. The fax library automatically determines the T.30 Phase D continuation value to be sent between pages of a multi-page TIFF/F file. This setting forces negotiation of Phase B when a page of a different width and/or resolution is found. If the following page has the same format as the current page, this setting bypasses Phase B negotiation for each page and saves transmit time.
DFC_EOM	End of Message (T.30). End of current fax document page; more fax data to follow at a different resolution or width; return to Phase B and negotiate parameters for next fax document page. To maintain backward compatibility, this is the default setting. This setting forces the negotiation of Phase B after each page.
DFC_MPS	Multi-Page Signal (T.30). End of current fax document page; next page is in the same format as the current page; proceed directly to Phase C. This setting bypasses Phase B negotiation for each page

Define	Description
	and saves transmit time.
FC_TFPGBASE	**Bytes:** 2 **Default:** TF_BASE0 Base page numbering scheme set for the TIFF/F file to send. This define is ignored on DM3 boards. This parameter accommodates TIFF/F file utilities that may not adhere to TIFF/F specifications which require the first page to be page zero. For most cases, this parameter may remain at the default setting. To determine the base page numbering scheme of the transmitted TIFF/F file, call the fax extended attribute **ATFX_TFPGBASE( )** after the file transmission is completed. Valid values for FC_TFPGBASE: TF_AUTOPG — Automatic fax library adjustment for each zero or one base numbered TIFF/F file to be transmitted. Use this setting when you are unsure of the base page number scheme of the TIFF/F file. This setting may introduce additional disk I/O access time. TF_BASE0 — Zero base page numbered TIFF/F file. TF_BASE1 — One base page numbered TIFF/F file. Use this setting when you know the page number scheme of the TIFF/F to be transmitted is base 1. **NOTE:** Always use zero base page numbering when describing TIFF/F file pages to send in a DF_IOTT structure array. The fax

Define	Description
	library internally adjusts the page number depending on the TIFF/F file's page numbering scheme when FC_TFPGBASE is set to TF_BASE1 or TF_AUTOPG. To send TIFF/F files of both zero and one base page numbering scheme during the same fax session, do one of the following: • Set FC_TFPGBASE to TF_AUTOPG before issuing **fx_sendfax( )**. The fax library automatically determines the base page numbering scheme for each TIFF/F file described in the DF_IOTT structures. • Return control to the application to set the FC_TFPGBASE parameter value each time the TIFF/F file page numbering scheme changes. To do so, set the DF_IOTT structure **io_type** field to IO_EOT and the **io_phdcont** field to DFC_EOM for the **last** DF_IOTT TIFF/F file entry of a page numbering scheme type. Change the FC_TFPGBASE parameter to the appropriate value for the TIFF/F file described in the next DF_IOTT structure, then issue another **fx_sendfax( )** function for the file(s) with the new page numbering scheme while the fax session is still active. For more information on DF_IOTT, see *Section 10.6. DF_IOTT – Fax Transmit Data Description*.

FC_TFTAGCHECK

Bytes: 2

Default: TF_MAXTAGS

Level of TIFF/F tag checking. This define is ignored on DM3 boards. Valid values:

TF_MAXTAGS	Check all mandatory tags; see *Appendix A* for TIFF/F tags.
TF_MINTAGS	Check essential subset of mandatory tags. This accommodates TIFF/F files created by utilities that may not strictly adhere to TIFF/F requirements.

FC_TXBAUDRATE

Bytes: 2

Default: DF_MAXBAUD

Preferred maximum baud rate for fax transmission. Capability varies by product; see *Section 2.3. Key Product Features* on *page 8* for more information. Valid values:

DF_MAXBAUD

Maximum baud rate value for transmission.

DF_14400BAUD

14400 baud transmission

DF_9600BAUD

9600 baud transmission

DF_4800BAUD

4800 baud transmission

DF_2400BAUD

2400 baud transmission

Set this parameter to one of the supported baud rates if you wish to transmit at a lower baud rate than the default (DF_MAXBAUD).

Although the fax channel automatically steps down to a lower baud rate if the remote station is

Define	Description
	incapable of receiving at the default baud rate, specifying a lower transmit baud rate saves time that would be taken in negotiating a lower baud rate. Setting a lower baud rate is also useful when transmitting over noisy lines. Once the value is set, the selected preferred baud rate remains in effect until it is set again. **NOTE:** When DF_MAXBAUD is set, the value returned in **fx_getparm( )** reflects the maximum transmission baud rate value based on the capability of the fax product.
FC_TXCODING	**Bytes:** 2 **Default:** DF_MMR If supported, the preferred encoding scheme for data transmission over the phone line. For more information on FC_TXCODING and ECM, see *Section 5.6.2. Specifying a Preferred Encoding Scheme for Transmission* on *page 65*. Valid values for FC_TXCODING: DF_MH — Modified Huffman DF_MR — Modified Read DF_MMR — Modified Modified Read DF_ECM — Use ECM switch; ECM can be explicitly specified for Phase B negotiation in fax transmission. Use of ECM is determined by the receiver's capability. On DM3 boards, DF_ECM is automatically implied with DF_MMR, DF_JPEG_COLOR, and DF_JPEG_GREY. DF_JPEG_COLOR — color fax image (JPEG)

Define	Description
	DF_JPEG_GREY greyscale fax image (JBIG) The FC_TXCODING value is used during Phase B negotiations with the remote receiver. The capabilities of the remote receiver determine the data transmission encoding scheme used over the phone line. The fax image data specified in the DF_IOTT structures is automatically converted to the negotiated encoding scheme at the time of transmission. Once the FC_TXCODING value is set, it remains in effect until it is set again. If FC_TXCODING is set on an unsupported product, **ATDV_LASTERR( )** returns an EFX_UNSUPPORTED error. If an invalid **fx_setparm( )** parameter value is specified, **ATDV_LASTERR( )** returns an EFX_BADPARM error. You can specify the explicit use of ECM for fax data transmission when the remote station receiver has ECM capabilities. To do so, "OR" the DF_ECM bit flag with DF_MH, DF_MR, or DF_MMR when setting the FC_TXCODING parameter.
FC_TXNSF	**Default:** NULL Pointer to customized non-standard facilities (NSF) message sent by the transmitter in Phase B. This define is only supported on DM3 boards; it is not supported on Springware boards. See DF_TXNSF data structure description for more information on setting and getting a customized NSF message.
FC_TXSUBADDR	**Default:** NULL Subaddress information sent by the transmitter - 21 character NULL-terminated ASCII string (20 characters + NULL, right-justified).

Define	Description
	This define is not supported on DM3 boards. This parameter sets the telephone or extension numbers to be used as fax routing subaddresses. If the parameter value is less than 20 characters, the parameter value is padded with spaces to equal 20 characters. Valid characters are digits zero (0) through nine (9), and the asterisk (*). Extensions are separated by a single pound sign (#). Phone numbers are separated by two pound signs (##). If the first parameter in the subaddress field is a phone number, it should be preceded by a single pound sign (#). Examples of FC_TXSUBADDR values:

Value	Description
3765	single extension
3765#3978#3767	multiple extensions
#5551212#2767	phone number followed by an extension
#5551234##5554321	phone number followed by another phone number
#5551234#2767#2018	phone number followed by multiple extensions

■ Cautions

- You must pass the value of the parameter to be set in a variable cast as (void *) as shown in the function example.
- Do not use the voice driver library function **dx_setparm( )** to set fax parameter values.

■ Example

Example 1: fx_setparm( ) and FC_RETRYCNT, FC_HDRDATEFMT, FC_HDRTIMEFMT and FC_HDRUSER

```
#include <stdio.h>

#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int dev;
unsigned short value;
char *coname = "ABCDE Company";
/*
 * Open device using fx_open( ). Obtain fax device
 * handle in dev.
 */
.
.
/*
 * Set retry count parameter to 2, disconnect after specified
 * number of retries. FC_RETRYCNT uses 2 bytes (the variable value
 * is of type unsigned short).
 */
value = DF_RETRYDCN|DF_RETRY2;

if (fx_setparm(dev,FC_RETRYCNT,(void *)&value) == -1) {
   printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
      ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
}

/*
 * Set the following fax page header parameters:
 * Date format: MM-DD-YYYY
 * Time format: HH:MM (24 hour)
 * User text: ABCDE Company
 */
value = DF_HDRDATEFMT_1;

if (fx_setparm(dev,FC_HDRDATEFMT,(void *)&value) == -1) {
   printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
      ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
}
value = DF_HDRTIMEFMT_2;

if (fx_setparm(dev,FC_HDRTIMEFMT,(void *)&value) == -1) {
   printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
      ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
```

```
}

if (fx_setparm(dev,FC_HDRUSER,(void *)coname) == -1) {
   printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
      ATDV_LASTERR(dev));
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
}
```

Example 2: fx_setparm( ) and FC_FONT0 (Windows only)

```
#include <stdio.h>

#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

int dev;
unsigned short value;

HFONT hMyFont;
LOGFONT lFont;

/*
 * Open device using fx_open( ). Obtain fax device
 * handle in dev.
 */
.
.
/*
 * Use Windows API to get a font handle.  See the Microsoft
 * Win32 API Programmer's Reference for other ways of getting
 * font handles from resources.
 */

ret= AddFontResource("Roman.fon");
memset(&lFont,0,sizeof(lFont));
lFont.lfCharSet = DEFAULT_CHARSET;
lFont.lfHeight = 24;
lFont.lfWidth = 20;
lFont.lfEscapement = 0;
lFont.lfOrientation =0;
lFont.lfWeight = FW_NORMAL;
lFont.lfItalic = FALSE;
lFont.lfUnderline = FALSE;
lFont.lfStrikeOut = FALSE;
lFont.lfOutPrecision = OUT_DEFAULT_PRECIS;
lFont.lfClipPrecision = CLIP_DEFAULT_PRECIS;
lFont.lfQuality = DEFAULT_QUALITY;
lFont.lfPitchAndFamily = DEFAULT_PITCH|FF_DONTCARE;
strcpy(lFont.lfFaceName,"Roman");
hMyFont = CreateFontIndirect(&lFont);

/* pass the handle to the fax library as one of the 2 internal fonts. */


if (fx_setparm(dev,FC_FONT0,(void *)&hMyFont) == -1) {
   printf("Error - %s (error code %d)\n", ATDV_ERRMSGP(dev),
      ATDV_LASTERR(dev));
```

```
   if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
      /* Perform system error processing */
   }
}


/*
 * when you use fx_sendfax() or fx_sendascii() to
 * send an ASCII document, the control character <ESC>F0
 * will use the font handle hMyFont to render the ASCII text.
 */

.
.
.

fx_sendfax( ) ;  /* after specifying the font, send the fax */

.
.
.

deleteObject(hMyFont) ; /* delete the font handle */
```

■ Errors

See *Appendix D* for a list of error codes that may be returned for this function.

If you issue the function for a parameter that is not supported by your fax hardware channel, **ATDV_LASTERR( )** returns an EFX_UNSUPPORTED error code. On DM3 boards, specifying an unsupported parameter results in the EFX_INVALARG error.

■ See Also

- **fx_getparm( )**

Name:	void fx_setuio(df_uio)	
Inputs:	DF_UIO df_uio	• DF_UIO structure
Returns:	none	
Includes:	srllib.h dxxxlib.h faxlib.h	
Category:	miscellaneous	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **fx_setuio( )** function registers user-defined I/O functions, replacements for the standard I/O functions **read( )**, **write( )** and **lseek( )** with the fax library. This function is useful for applications requiring access and/or storage of data from, for example, a network device that requires the use of specific I/O functions.

Parameter	Description
df_uio	Specifies the DF_UIO structure.

Your application provides the addresses of user-defined **read( )**, **write( )** and **lseek( )** functions (with pointers to the user-defined **read( )**, **write( )** and **lseek( )** functions) by initializing the DF_UIO structure. The application then installs the user-defined functions by issuing the **fx_setuio( )** function.

If you specify the user-defined I/O mode during a fax send or receive, the fax library uses the I/O functions registered by **fx_setuio( )** instead of the standard I/O functions provided by the operating system. The user-defined I/O functions are passed the same arguments as the standard **read( )**, **write( )** and **lseek( )** I/O functions.

When issuing the **fx_setuio( )** function to receive a fax, you must provide a user-defined **write( )** function. When issuing this function to send a fax, you must provide a user-defined **read( )** function.

NOTE: The application can override the standard I/O functions on a file-by-file basis; see *Section 10.8. DF_UIO – User-Defined I/O* (*page 134*) and the **fx_rcvfax( )** and **fx_sendfax( )** function references.

■ Example

```
#include <stdio.h>
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>

DF_UIO userio;

/* User read function (note: same arguments as read( )) */
int user_read(filedes, buf, size)
   int filedes;
   char * buf;
   unsigned size;

{
   /* Application specific read( ) function. */

      .
      .
}

/* User write function (Note: Same arguments as write( )). */
int user_write(filedes, buf, size)
   int filedes;
   char * buf;
   unsigned nbyte;
{
   /* Application specific read( ) function. */
      .
      .
}

/* User lseek function (Note: Same arguments as lseek( )). */
long user_lseek(filedes, offset, whence)
   int filedes;
   long offset;
   int whence;
{
   /* Application specific lseek( ) function. */
      .
      .
}

main( )
{
   .
   .
   userio.u_read = user_read;
   userio.u_write = user_write;
   userio.u_seek = user_lseek;

   /* Register these functions with the FAX library. */
```

```
    fx_setuio(userio);
    .
    .
}
```

Name:	int fx_stopch(dev, mode)	
Inputs:	int dev	• valid fax channel device handle
	unsigned short mode	• synchronous/asynchronous mode bitmap
Returns:	0 if success -1 if failure	
Includes:	srllib.h dxxxlib.h faxlib.h	
Category:	resource management	
Mode:	synchronous/asynchronous	
Platform:	DM3, Springware	

■ Description

The **fx_stopch( )** function forces termination of a fax send or receive on a fax channel device. It forces a fax channel in the busy state to become idle.

If the fax channel specified in **dev** is already idle, **fx_stopch( )** has no effect and returns a success.

A **fx_stopch( )** function issued on a channel executing the T.30 fax protocol for either send or receive will disconnect the fax transfer and enter Phase E of the T.30 fax protocol to terminate the fax transfer. The active *send* or *receive* function returns a -1. **ATDV_LASTERR( )** returns EFX_DISCONNECT and **ATFX_ESTAT( )** returns EFX_ABORTCMD.

NOTE: It may take a few seconds after **fx_stopch( )** returns before termination takes effect. The timing depends on the phase of the fax transfer at the time **fx_stopch( )** is issued.

Parameter	Description	
dev	Specifies the valid fax channel device handle obtained when the channel was opened.	
mode	Specifies the mode of operation.	
	EV_SYNC	Synchronous mode (default). The function does not return to the application until the fax channel device is idle.
	EV_ASYNC	Asynchronous mode. The function immediately returns after initiating a stop on the fax channel device. If you issue the **fx_stopch( )** function from an event handler, you must call **fx_stopch( )** in asynchronous mode (EV_ASYNC). See the *Standard Runtime Library* documentation for information on event handlers.

■ Cautions

- It is recommended that you use **fx_stopch( )** to stop fax I/O only after a fax *send* or *receive* function has been issued.
- On Linux operating system only. To ensure that fax T.30 protocols executing on the channels are properly terminated, it is highly recommended that processes running fax applications have appropriate signal handlers installed to handle process kill or exit. The signal handler must issue **fx_stopch( )** set to operate in asynchronous mode (EV_ASYNC) to the channels executing fax calls.

■ Example

```
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>
```

```
main( )
{
   int dev;   /* Fax channel device handle.*/

   /* Open the FAX channel device. */
   if ((dev = fx_open("dxxxB1C1", NULL)) == -1) {
      /* Error opening device. */
      /* Perform system error processing */
      exit(1);
   }
   /* Use the FAX channel device to send or receive faxes. */
   .
   .
   /*
    * Issue a stop to force the termination of the fax session
    * if necessary.
    */
   if (fx_stopch(dev, EV_ASYNC) == -1) {
      /* Error stopping device. */
      printf("Error stopping channel\n");
       printf("Error - %s (error code %d)\n",
          ATDV_ERRMSGP(dev), ATDV_LASTERR(dev));
      if (ATDV_LASTERR(dev) == EDX_SYSTEM) {
         /* Perform system error processing */
      }
      exit(1);
   }
   .
   .
}
```

■ Errors

If this function returns -1 to indicate failure, use **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to retrieve one of the following error reasons:

EDX_BADPARM	Invalid fax parameter
EDX_SYSTEM	Operating system error. On Linux, check the global variable errno for more information. On Windows, use **dx_fileerrno**( ) to obtain error value.

■ See Also

- **ATFX_TERMMSK( )**
- **fx_rcvfax( )**
- **fx_rcvfax2( )**
- **fx_sendfax( )**

Name:	int fx_unlisten(dev)	
Inputs:	int dev	• fax channel device handle
Returns:	0 if successful -1 if error	
Includes:	srllib.h dxxxlib.h faxlib.h	
Category:	TDM bus routing	
Mode:	synchronous	
Platform:	DM3, Springware	

■ Description

The **fx_unlisten( )** function disconnects fax receive channel from TDM bus.

Calling the **fx_listen( )** function to connect to a different TDM bus time slot will automatically break an existing connection. Thus, when changing connections, you need not call the **fx_unlisten( )** function.

NOTE: TDM bus convenience function **nr_scunroute( )** includes **fx_unlisten( )** functionality. See the *Voice API Library Reference* for more information on nr_ convenience functions.

Parameter	Description
dev	Specifies the fax channel device handle obtained when the channel was opened using **fx_open**( ).

■ Cautions

This function will fail when an invalid fax channel device handle is specified.

■ Example

```
#include <srllib.h>
#include <dxxxlib.h>
#include <faxlib.h>
```

```
main( )
{
     int dev;          /* Fax channel device handle. */
     .
     .
     /* Open the FAX channel resource. */
     if ((dev = fx_open("dxxxB7C1", NULL)) == -1) {
          /* Error opening device. */
          /* Perform system error processing */
          exit(1);
     }
          .
     .
     /*
      * Disconnect the FAX channel device from "listening" to an
      * TDM bus transmit time slot.
      */
     if (fx_unlisten(dev) == -1) {
          printf("Error message = %s", ATDV_ERRMSGP(dev));
          exit(1);
     }
     .
     .
```

■ Errors

If this function returns -1, use **ATDV_LASTERR( )** and **ATDV_ERRMSGP( )** to retrieve one of the following error reasons:

Equate	Returned When
EDX_BADPARM	Parameter error
EDX_SH_BADCMD	Command is not supported in current bus configuration
EDX_SH_BADEXTTS	TDM bus time slot is not supported at current clock rate
EDX_SH_BADINDX	Invalid Switch Handler index number
EDX_SH_BADLCLTS	Invalid channel number
EDX_SH_BADMODE	Function not supported in current bus configuration
EDX_SH_BADTYPE	Invalid channel type (voice, analog, etc.)

Equate	Returned When
EDX_SH_CMDBLOCK	Blocking command is in progress
EDX_SH_LCLDSCNCT	Channel already disconnected from TDM bus
EDX_SH_LIBBSY	Switch Handler library busy
EDX_SH_LIBNOTINIT	Switch Handler library uninitialized
EDX_SH_MISSING	Switch Handler is not present
EDX_SH_NOCLK	Switch Handler clock failback failed
EDX_SYSTEM	System Error

See also

- **fx_listen( )**

Appendix A
TIFF/F Tags and Values

Overview

This appendix presents the TIFF/F tags and values required in TIFF/F files for a successful fax transfer. The first section describes the required tags and values for TIFF/F files received by the fax library. The second section describes the tags and values for TIFF/F files written by the fax library.

Input to the Library from Disk Storage

The fax library accepts TIFF/F files that have valid tag values as shown in the following table, where *x* equals a number.

NOTE: An asterisk (*) after a TIFF/F tag field name indicates a subset of the TIFF/F tags; see the **fx_setparm( )** FC_TFTAGCHECK parameter.

Except for PageNumber, all tags listed are mandatory. All tags listed are checked by default. If the PageNumber tag is absent, the fax library assumes there is only one page in a single page file. Page numbers can be zero base or one base. Note that zero-based page numbering adheres to the TIFF/F standard.

Table 18. TIFF/F Tags Input to Library

Field	Valid Values	Description
BitsPerSample	1	one bit per sample
Compression*	3	Group 3 - MH or MR encoding
	4	Group 4 - MMR encoding
FillOrder*	1	most significant bit first
	2	least significant bit first
ImageWidth*	1728	number of pixels per line
	2048	number of pixels per line
	2432	number of pixels per line
PageNumber	*x/x*	page number/number of pages
SamplesPerPixel	1	one sample per pixel
StripByteCounts*	*x*	byte count in image (as appropriate)
StripOffsets*	*x*	byte offset of image (as appropriate)
T4Options	*x/x*	Group 3 - MH encoding, EOL not padded/padded
T6Options	*x*	Group 4 - MMR encoding
Yresolution*	≤ 150	coarse (normal)
	>150	fine

If T4Options or T6Options tag is absent, the following values are assumed:

Field	Valid Values	Description
T4Options	0	Group 3 - MH encoding, EOL not padded
	4	Group 3 - MH encoding, EOL padded
T6Options	0	Group 4 - MMR encoding

Output from the Library to Disk Storage

The TIFF/F files written by the fax library have the following tags with valid values, where *x* equals a number:

Table 19. TIFF/F Tags Output from Library

Field	Valid values	Description
BadFaxLines	x	number of bad scan lines received; this is a measure of quality of the image received from the remote
BitsPerSample	1	one bit per sample
CleanFaxData	0	data in file does not contain bad scan lines
		Bad Scan Line Replacement (BLR) restores MH encoded scan line to the correct pixel count, as indicated by CleanFaxData set to zero; however, the integrity of the repaired scan line image may be impaired. To determine the error content of the stored data, use BadFaxLines.
Compression	3	Group 3 - MH encoding
	4	Group 4 - MMR encoding
DateTime		YYYY:MM:DD HH:MM:SS
FillOrder	2	least significant bit first
ImageWidth	1728	number of pixels per line
	2048	number of pixels per line
	2432	number of pixels per line
ImageLength	*x*	number of scan lines in image
NewSubFileType	2	single page in multi-page image

Field	Valid values	Description
Orientation	1	first row = top left
		first column = top left
PageNumber	*x/x*	page number/number of pages
PhotometricInterpretation	0	gray scale/bilevel: zero = white
ResolutionUnit	2	inch
RowsPerStrip	*x*	number of scan lines in image; if present, RowsPerStrip must equal ImageLength.
SamplesPerPixel	1	one sample per pixel
Software		Intel Dialogic TIFF/F Library version *x.xx*
StripByteCounts	*x*	byte count in image (as appropriate); only one value for StripOffsets and StripByteCounts must be present (the image is considered to be one large strip). Multiple strips per image is not currently supported.
StripOffsets	*x*	byte offset of image (as appropriate)
T4Options	0	Group 3 - MH encoding, EOL not padded
T6Options	0	Note: Replaces T4Options if MMR
Xresolution	204	number of pixels per resolution unit X
Yresolution	98	coarse (normal) (number of pixels per resolution unit Y)
	196	fine

Appendix B

Fax Phase D Status Values

T.30 Phase D (post-message procedure) status values indicate the status of a fax transmission and reception. This appendix lists the Phase D command values returned from the transmitter to the receiver, and the reply values returned from the receiver to the transmitter.

NOTE: DFS_ALL and DFS_REMOTESUBADDR are Intel Dialogic fax library terms, not T.30 protocol terminology.

To obtain Phase D status values, use the following fax extended attribute functions after a TFX_PHASED event or after the completion of the last page of a fax *send* or *receive* function.

- **ATFX_PHDCMD( )** – returns the Phase D command, which specifies the next phase of the fax session to the receiver.
- **ATFX_PHDRPY( )** – returns the Phase D reply from the receiver, which indicates the quality of the received transmission.

Table 20. Phase D Command Values - Transmitter to Receiver

Value	Description
DFS_EOP	End of Procedure – Terminate fax session. Progress to Phase E and disconnect fax call.
DFS_MPS	Multi-page Signal – End of current fax document page, more fax data to follow. Next fax document page is in the same format as the current page, so proceed directly to Phase C.
DFS_EOM	End of Message – End of current fax document page, more fax data to follow. Return to Phase B and negotiate parameters for next fax document page.
DFS_POLL	A poll request was sent.
DFS_PRI_EOP	Request for operator intervention sent (PRI_EOP).
DFS_PRI_MPS	Request for operator intervention sent (PRI_MPS).
DFS_PRI_EOM	Request for operator intervention sent (PRI_EOM).

Most fax machines wait for the completed transmission of the final page of a fax session before sending an operator intervention request (PRI_EOP). The fax library responds to operator intervention requests from the remote station after receiving one of the following messages: PRI_EOP, PRI_MPS or PRI_EOM.

Table 21. Phase D Reply Values - Receiver to Transmitter

Value	Description
DFS_MCF	Message confirmation – valid fax image received, ready for more pages.
DFS_RTN	Retrain negative – bad fax image received, retrain and resend image.
DFS_RTP	Retrain positive – valid fax image received but retraining required (transmitter determines whether lower baud rate is needed for continued successful reception).
DFS_PIP	Procedure interrupt positive – operator intervention request.
DFS_PIN	Procedure interrupt negative – operator intervention request.

Appendix C

Fax Phase E Status Values

The Phase E (fax call release) status values indicate errors during the course of a fax transmission/reception. To obtain Phase E status values, use the fax extended attribute function **ATFX_ESTAT( )**.

Table 22. General Phase E Status Values

Value	Description
EFX_ABORTCMD	Command stopped by stop_fax firmware command
EFX_BUSYCHN	Request to start fax while channel is currently busy
EFX_CEDTONE	Remote CED (Called Station Identification) tone exceeds 5 seconds
EFX_CHIPNORESP	Fax modem is not responding
EFX_HDLCCARR	Excessive HDLC carrier
EFX_OPINTFAIL	Operator intervention failed

Table 23. Phase E Status Values Returned to the Transmitter

Value	Description
EFX_BADDCSTX	Received bad response to DCS (Digital Command Signal) or training
EFX_BADPGTX	Received a DCN (Disconnect) from remote after sending a page
EFX_COMMERRTX	Transmit communication error
EFX_ECMPHDTX	Invalid ECM response received from receiver
EFX_ECMRNRTX	Timer T5 expired, receiver not ready
EFX_GOTDCNTX	Received a DCN (Disconnect) while waiting for a DIS (Digital Identification Signal)
EFX_INVALMMRTX	Invalid input MMR data
EFX_INVALRSPTX	Invalid response after sending a page
EFX_NODISTX	Received other than DIS (Digital Identification Signal) while waiting for DIS
EFX_NOFINERECTX	Remote cannot receive fine resolution
EFX_NOISETX	Too much noise at 2400 bps
EFX_NOWIDTHTX	Remote cannot receive at this width
EFX_NXTCMDTX	Timed out waiting for next send_page command from driver
EFX_PHBDEADTX	Received no response to DCS (Digital Command Signal), training or TCF (Training Check)
EFX_PHDDEADTX	No response after sending a page
EFX_RXCOMP	Remote site is not receive compatible
EFX_T1EXPTX	Timed out while waiting for a message

Table 24. Phase E Status Values Returned to the Receiver

Value	Description
EFX_COMMERRRX	Receiver communication error
EFX_DCNDATARX	Unexpected DCN (Disconnect) while waiting for fax data
EFX_DCNFAXRX	Unexpected DCN (Disconnect) while waiting for EOM (End Of Message), EOP (End Of Procedure) or MPS (Multi-page Signal)
EFX_DCNNORTNRX	DCN (Disconnect) after requested retransmission
EFX_DCNPHDRX	Unexpected DCN (Disconnect) after EOM (End Of Message) or MPS (Multi-page Signal) sequence
EFX_DCNRRDRX	Unexpected DCN (Disconnect) after RR/RNR sequence
EFX_ECMPHDRX	Invalid ECM response received from transmitter
EFX_GOTDCSRX	DCS (Digital Command Signal) received while waiting for DTC
EFX_INVALCMDRX	Unexpected command after page received
EFX_NOCARRIERRX	Lost carrier during fax receive
EFX_NOEOLRX	Timed out while waiting for EOL (End Of Line)
EFX_NOFAXRX	Timed out while waiting for first line
EFX_NXTCMDRX	Timed out waiting for next receive page command
EFX_PNSUCRX	High speed training success not returned by modem during receive
EFX_T1EXPRX	Timed out while waiting for a message
EFX_T2EXPDCNRX	Timed out while waiting for DCN (Disconnect)
EFX_T2EXPDRX	Timed out while waiting for Phase D
EFX_T2EXPFAXRX	Timed out while waiting for fax page
EFX_T2EXPMPSRX	Timed out while waiting for next fax page

Value	Description
EFX_T2EXPRRRX	Timer T2 expired waiting for RR command
EFX_T2EXPRX	Timed out waiting for NSS (Non-standard set-up), DCS (Digital Command Signal) or MCF (Message Confirmation)
EFX_TXCOMP	Remote site is not transmit compatible
EFX_WHYDCNRX	Received unexpected DCN (Disconnect) while waiting for DCS (Digital Command Signal) / DIS (Digital Identification Signal)

Appendix D
Fax Error Codes

Fax Error Code Overview

The following table lists errors for fax.

- To access the error code values, use the **ATDV_LASTERR( )** function of the Standard Runtime Library.
- To return a string describing the error, use the **ATDV_ERRMSGP( )** function of the Standard Runtime Library.

For more information on these functions, see the *Standard Runtime Library API Library Reference*. For a list of TIFF/F tags and values required for successful fax transfer, see *Appendix A*.

Fax Error Code Listing

Table 25. Fax Error Codes

Value	Description and Suggested Action
EFX_BADIOTT	Invalid data in DF_IOTT entry. No data sent. Use **ATFX_BADIOTT**( ) to return a pointer to the DF_IOTT entry with invalid data. Verify that the appropriate Phase D continuation value is used. For example, an error occurs if the DF_IOTT entry specifies DFC_EOP but it is not the last entry in the table.
EFX_BADPAGE	Invalid or missing TIFF/F PageNumber tag. No data sent. Use **ATFX_BADPAGE**( ) to find the page offset from the first page described by the DF_IOTT entry.
EFX_BADPARM	Invalid value for fax parameter.
EFX_BADPHASE	Unexpected Phase transition (internal).

Value	Description and Suggested Action
EFX_BADSTATE	Invalid initial state value specified. Following T.30 protocol, the initial state of the caller station must be transmitter (DF_TX) and the initial state of the called station must be receiver (DF_RX). See **fx_initstat( )**.
EFX_BADTAG	Incorrect values for TIFF/F tags. No data sent. Use **ATFX_TFBADTAG( )** to return the tag number associated with the invalid tag value.
EFX_BADTIF	Incorrect TIFF/F format. No data sent. Verify that the TIFF/F file contains all mandatory TIFF/F tags (or subset). Use **ATFX_TFNOTAG( )** to return the number of the missing tag. You can select the level of TIFF/F tag checking by setting FC_TFTAGCHECK in **fx_setparm( ).**
EFX_BADTFHDR	Bad TIFF/F header - incorrect values in fields. No data sent. Verify TIFF/F header values.
EFX_CMDDATA	Last command contained invalid data.
EFX_COMPAT	DF_IOTT entry problem. I/O file type, image width and resolution defined are not compatible with the receiver's hardware. No data sent. Use **ATFX_BADIOTT( )** to return a pointer to the DF_IOTT entry that caused the error.
EFX_DSPERROR	DSP Fax resource error.
EFX_DISCONNECT	Fax call disconnected by other station. Use **ATFX_ESTAT( )** to find out why disconnection occurred during a fax session. For example, some reasons may be receiver incompatibility or an invalid poll request by the transmitter.
EFX_DRVERROR	Error occurred in driver (internal).
EFX_FWERROR	Firmware error.
EFX_INUSE	Channel is in use (failed **fx_open( )** or **fx_close( )** using a DSP Fax resource).
EFX_INVALARG	Illegal argument to function.
EFX_INVALFUNC	Illegal call to function.

Value	Description and Suggested Action
EFX_LIBERROR	Error in library state machine (internal)

Table 25. Fax Error Codes (cont.)

Value	Description/Suggested Action
EFX_MAXCHAN	Maximum channel capacity reached. Capacity varies by product. See *Table 1. Key Fax Features and Specifications*.
EFX_NODATA	Data requested is not available (NSF, DIS, DCS). A function is called before completion of the initial Phase B or the message was not sent by the remote station.
EFX_NOFAX	No fax capability on this device
EFX_NOMEM	Cannot allocate memory for more pages
EFX_NOPAGE	Requested TIFF/F page not found. Check the base page numbering scheme used.
EFX_NOPOLL	Poll not accepted. For example, if the transmitter issues a poll request and the receiver's poll bit is set to DF_NOPOLL, this error code is generated.
EFX_NORESOURCE	A DSP Fax resource is not available for assignment/sharing through **fx_open( )**.
EFX_NOSTATE	Initial state value not set
EFX_NOTIDLE	Channel is not idle. Some functions require that the channel be in idle state before invocation. Use **ATFX_STATE( )** to determine the state of the channel.
EFX_NOTIMESLOT	No time slot assigned
EFX_NSFBUFF	Buffer length supplied to **fx_getNSF( )** is less than 2 bytes
EFX_NXTCMDRX	Time out while waiting for next receive fax operation
EFX_RDFWER	Error reading firmware version
EFX_RETRYDCN	Disconnected after specified retries

Value	Description/Suggested Action
EFX_UNSUPPORTED	Unsupported feature

Appendix E
Fax Event Codes

The following table lists the fax event code values.

In synchronous mode operation, an event handler must be enabled to detect Phase B and Phase D events. For event handler details, see the **sr_enbhdlr( )** function in the *Standard Runtime Library API Library Reference.*

Table 26. Fax Event Codes

Value	Description
TFX_FAXERROR	Error in fax transmission or fax reception
TFX_FAXRECV	Fax reception successfully completed
TFX_FAXSEND	Fax send successfully completed
TFX_PHASEB	Phase B event
TFX_PHASED	Phase D event

Appendix F
ASCII to Fax Tables

Overview

This appendix contains the following ASCII to Fax information:

- ASCII to Fax command set table
- Proprietary Extended ASCII Character Set
  (Modified ASCII 437 character set)
- Katakana Character Set (Japanese Characters)
  (Modified ASCII 437 character set)

The information in this appendix does not apply to DM3 boards.

ASCII to Fax Command Set

An ASCII data file may contain embedded escape sequences as shown in *Table 27. ASCII to Fax Command Set*. These embedded sequences control certain graphical attributes such as font and line spacing from within the file and override the values specified at the application level in the DF_ASCIIDATA structures.

Insert ASCII escape sequences before the ASCII text you wish to change. The format you specify using an escape sequence is in effect until a new escape sequence is inserted.

Table 27. ASCII to Fax Command Set

Format	Selection	Hex	ASCII
Font	10 pitch (default)	1B 46 30	<Esc>F0
	17 pitch	1B 46 33	<Esc>F3
Line Spacing	2.4 lines/inch	1B 4C 33	<Esc>L3
	3 lines/inch	1B 4C 32	<Esc>L2
	4 lines/inch	1B 4C 31	<Esc>L1
	6 lines/inch (default)	1B 4C 30	<Esc>L0
	8 lines/inch	1B 4C 34	<Esc>L4
	Single spacing (font height)	1B 44 32	<Esc>D2
	Double spacing (font height)	1B 44 34	<Esc>D4
	Triple spacing (font height)	1B 44 36	<Esc>D6
Attributes *	Underline off (default)	1B 55 30	<Esc>U0
	Underline on	1B 55 31	<Esc>U1
	Boldface off (default)	1B 42 30	<Esc>B0
	Boldface on	1B 42 31	<Esc>B1
Line Wrap	Line Wrap off	1B 54 30	<Esc>T0
	Line Wrap on (default)	1B 54 31	<Esc>T1
Tabs **	Tab stop setting (default: n=8)	1B 09 n	<Esc><Tab>n
Control Characters	New line	0D	<CR>
		0D 0A	<CR><LF>
	New Page	0C	<FF>
		0D 0C	<CR><FF>
	Tabs	09	<Tab>
	End of File	1A	<EOF>

* Attributes format not recommended for Katakana.
** Tabs format available for DSP Fax only. n = number of tab stops. Default at position 1, 9, 17, 25 and so on.

On some Intel® Dialogic® fax products, you can:

- Use the tilde (~) instead of the <ESC> sequence character to format ASCII documents. For example, both <ESC>F0 and ~F0 are valid control characters in an ASCII document.

  To print a tilde itself in an ASCII document, use two consecutive tildes (~~).

- Specify italicized text using <ESC>I1 or ~I1. To disable it, use <ESC>I0 or ~I0.

For product support, see *Table 1. Key Fax Features and Specifications*.

Example

Following is a sample sequence to specify bold ASCII characters:

```
<Esc>B1Dialogic<Esc>B0
```

Follow your ASCII editor instructions to insert ASCII code 27 <Escape>.
Use uppercase letters with no spaces in the escape sequence.

Figure 1. Proprietary Extended ASCII Character Set (Modified ASCII 437 Character Set)

HEX	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
00				0	@	P	`	p	Ç	É	á	▒	└	╨	α	≡
01			!	1	A	Q	a	q	ü	æ	í	▓	┴	╤	ß	±
02			"	2	B	R	b	r	é	Æ	ó	▓	┬	╥	Γ	≥
03			#	3	C	S	c	s	â	ô	ú	┃	├	╙	π	≤
04			$	4	D	T	d	t	ä	ö	ñ	┤	─	╘	Σ	⌠
05			%	5	E	U	e	u	à	ò	Ñ	╡	┼	╒	σ	⌡
06			&	6	F	V	f	v	ª	û	ª	╢	╞	╓	µ	÷
07			'	7	G	W	g	w	ç	ù	º	╖	╟	╫	τ	≈
08			(	8	H	X	h	x	ê	ÿ	¿	╕	╚	╪	Φ	°
09	TAB		)	9	I	Y	i	y	ë	Ö	⌐	╣	╔	┘	Θ	•
0A		EOF	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Ω	·
0B			+	;	K	[	k	{	ï	¢	½	╗	╦	█	δ	√
0C			,	<	L	\	l	|	î	£	¼	╝	╠	▄	∞	ⁿ
0D			-	=	M	]	m	}	ì	¥	¡	╜	═	▌	φ	²
0E			.	>	N	^	n	~	Ä	₧	«	╛	╬	▐	Є	■
0F			/	?	O	_	o	⌂	Å	ƒ	»	┐	╧	▀	∩	blank

= Not a printable or control character. These ASCII codes are parsed out and ignored.

For ASCII codes ranging from 00H to 1FH, only the control characters outlined in the above figure are supported.

Figure 2. Katakana Japanese Character Set (Modified ASCII 437 Character Set)

HEX	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
00				0	@	P	`	p				ー	タ	ミ	年	社
01			!	1	A	Q	a	q			。	ア	チ	厶	月	頁
02			"	2	B	R	b	r			「	イ	ツ	メ	日	番
03			#	3	C	S	c	s			」	ウ	テ	モ	時	号
04			$	4	D	T	d	t			、	エ	ト	ヤ	分	発
05			%	5	E	U	e	u			・	オ	ナ	ユ	午	信
06			&	6	F	V	f	v			ヲ	カ	ニ	ヨ	前	
07			'	7	G	W	g	w			ァ	キ	ヌ	ラ	後	□
08			(	8	H	X	h	x			ィ	ク	ネ	リ	火	
09	TAB		)	9	I	Y	i	y			ゥ	ケ	ノ	ル	水	
0A		EOF	*	:	J	Z	j	z			ェ	コ	ハ	レ	木	
0B			+	;	K	[	k	{			ォ	サ	ヒ	ロ	金	
0C			,	<	L	¥	l	|			ャ	シ	フ	ワ	土	
0D			-	=	M	]	m	}			ュ	ス	ヘ	ン	株	
0E			.	>	N	^	n	―			ョ	セ	ホ	゛	式	
0F			/	?	O	_	o				ッ	ソ	マ	°	会	blank

= Not a printable or control character. These ASCII codes are parsed out and ignored.

For ASCII codes ranging from 00H to 1FH, only the control characters outlined in the above figure are supported.

Appendix G
Acronyms List

Table 28. Acronyms Translated lists acronyms used in this document.

Table 28. Acronyms Translated

Acronym	Meaning
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
DCN	Disconnect message
DCS	Digital Command Signal
DIS	Digital Identification Signal
ECM	Error Correction Mode
EOL	End of Line
EOM	End of Message
EOP	End of Procedure
ITU-T	International Telecommunication Union - Telecommunications
LSB	Least Significant Bit
MCF	Message Confirmation
MH	Modified Huffman
MMR	Modified Modified Read (Modified Modified Relative Element Address Differentiation Code)
MPS	Multi-page Signal

Acronym	Meaning
MR	Modified Read (Modified Relative Element Address Differentiation Code)
RTN	Retrain Negative
RTP	Retrain Positive
SRL	Standard Runtime Library
TIFF/F	Tagged Image File Format - Class F

Glossary

asynchronous function A function that allows program execution to continue without waiting for a task to complete. To implement an asynchronous function, an application-defined event handler must be enabled to trap and process the completed event. See synchronous function.

CCITT International Telegraph and Telephone Consultative Committee. See *ITU-T*.

configuration file An unformatted ASCII file that stores initialization information for an application.

data structure Programming term for a data element consisting of fields, where each field may have a different type definition and length. A group of data structure elements usually share a common purpose or functionality.

delimiting The ability to return control to the application at the end of each fax page or group of fax pages allowing the application to store these groups into separate files.

ECM Error Correction Mode. An *ITU-T* T.30 recommendation for Group 4 fax, now also used for Group 3 fax. ECM provides more efficient error handling for noisy or distorted fax transmissions. Encapsulated data within HDLC frames gives the receiver an opportunity to check for and request retransmission of garbled data.

extended attribute functions Class of functions that take one input parameter (a valid device handle) and return device-specific information. For instance, a voice device's extended attribute function returns information specific to the voice devices. Extended attribute function names are case-sensitive and must be in capital letters. The fax extended attribute functions return information specific to fax resources. See also Standard Runtime Library.

facsimile Fax. Transmitting and recording a scanned document to produce a copy of the original via phone lines.

fax See facsimile.

fax extended attribute functions See extended attribute functions.

fax session A fax session refers to the five phases of a fax call as defined by the ITU-T T.30 recommendation. The phases are Phase A (set up fax call),

Phase B (pre-message procedure), Phase C (transmit message), Phase D (post-message procedure) and Phase E (release fax call).

Group 3 T.4 standards recommendations for digital facsimile devices developed by CCITT, adopted in 1980, and modified in 1984 and 1988. A Group 3 digital fax transmission of an 8.5 by 11 inch page (or A4) at 9,600 bps is completed in 15 to 30 seconds using PSTN phone lines.

Group 4 T.6 standards recommendations for digital facsimile devices developed by CCITT and adopted in 1984. Using Public Data Networks or modified PSTN, the data is transmitted using ECM (Error Correction Mode) which essentially ensures error-free reception.

ID Refers to the telephone number of the fax. As per the ITU-T T.30 recommendations, the information to be included in the local ID 20-character data fields is the international telephone number with + in position 1, followed by the telephone country code, area code and the subscriber number.

ITU-T (formerly CCITT) International Telecommunication Union-Telecommunications. A United Nations agency based in Geneva whose three main aims are developing and recommending international telecommunications standards, regulating the use of radio frequency spectrum, and advancing telecommunications development around the world.

Modified Huffman (MH) code One-dimensional run length digital encoding scheme used to compress fax data for transmission in Group 3 fax devices. For example, a white line with no text, called a run, extending across an 8.5" page equals 1728 bits. MH code compresses the 1728 bits into a 17-bit code word. The lengths for all possible white runs are grouped together into 92 binary codes that will handle any white run length from 0 to 1728.

Modified Modified Read (MMR) code An optional, Group 4 facsimile two-dimensional digital encoding scheme with improved transfer speed over Modified Read encoding. This encoding scheme is now available on Group 3 fax devices. MMR makes use of the high degree of vertical correlation between each scan line in the fax image to achieve a higher compression than MH.

Modified Read (MR) code An optional, Group 3 facsimile two-dimensional digital encoding scheme with improved transfer speed over Modified Huffman encoding. MR makes use of the high degree of vertical correlation between each scan line in the fax image to achieve a higher compression than MH.

normal fax transmission A fax document transmitted from the called to caller application, as in the transmission of a fax document between two fax machines.

pel Picture element containing black and white information. A single point in a facsimile transmission.

Phase A One of five phases in a fax call, as defined by the T.30 protocol recommendation. This is the fax call setup phase. Communication is established between two stations: caller and called.

Phase B One of five phases in a fax call, as defined by the T.30 protocol recommendation. This is the pre-message procedure phase of a fax call. The two stations (caller and called) negotiate parameters for a fax transfer, such as receiver and transmitter state, transmission speed, resolution and so on.

Phase C One of five phases in a fax call, as defined by the T.30 protocol recommendation. This is the message transmission phase of a fax call. Transmits based on the parameters negotiated in Phase B.

Phase D One of five phases in a fax call, as defined by the T.30 protocol recommendation. This is the post-message procedure of a fax call where the Phase D continuation value indicates how the data just transmitted is connected to the next data transmitted.

Phase E One of five phases in a fax call, as defined by the T.30 protocol recommendation. This is the fax call release phase: disconnect call.

pixel Picture element containing levels of gray information. A single point in a facsimile transmission.

polling fax transmission Also known as fax on demand. A caller application requests that a fax be transmitted from the called to caller application. If polling is valid, the caller and called applications switch transmitter and receiver roles. The fax is then transmitted from called to caller application.

PSTN Public Switched Telephone Network.

raw fax data Unstructured fax data that does not conform to TIFF/F or other formats. A raw file stores fax data as a single page of unstructured, unformatted data. The raw file uses MH coding for transmission.

SCbus Signal Computing Bus. Third generation TDM (Time Division Multiplexed) resource sharing bus that allows information to be transmitted and received among resources over multiple data lines.

SRL See Standard Runtime Library.

standard attribute functions Class of functions that take one input parameter (a valid device handle) and return generic information about the device. For instance, standard attribute functions return IRQ and error information for all device types. Standard attribute function names are case-sensitive and must be written in uppercase letters. Standard attribute functions for all devices are contained in the SRL. See Standard Runtime Library.

Standard Runtime Library A software resource containing event management and standard attribute functions, and data structures used by all Intel Dialogic devices, but which return data unique to the device.

stand-alone configuration A hardware configuration where a single board contains all the hardware components (i.e., processing, data reception/transmission) necessary to implement an application. In a stand-alone configuration, the board's channels are not routed through SCbus time slots. NOTE: Some Intel Dialogic hardware products can be used in a stand-alone configuration, or as a component in an SCbus bus configuration.

subaddress A T.30 message protocol that allows a fax to be routed to one or more telephone numbers (or extensions) once it is received by the fax station.

sub-page addressing A method in which a single page of fax data is formed from images stored in different sources. Each stored image is considered a sub-page.

synchronous function A function that blocks program execution until a value is returned by the device. Also called a blocking function. See asynchronous function.

T.30 An ITU-T recommendation that specifies a fax communications protocol for Group 3 fax. This recommendation describes how to establish and terminate communications between Group 3 fax machines. The five phases of a fax session are Phase A, Phase B, Phase C, Phase D and Phase E.

TIFF/F Tagged Image File Format Class F. TIFF is a tag-based general purpose raster format used to exchange image data between application programs. Class F indicates specific format information for fax applications.

time out In telephone networks, an event which occurs at the end of a predetermined interval of time.

turnaround polling fax transmission At different times during a single fax call, the caller and called applications switch transmitter and receiver roles to enable both applications to send and receive a fax document during the fax call.

Index

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W