Client/Server API Toolkit

The PMfax API

CDS Inc.
P.O. Box 25123
Woodbury, MN, USA 55125
http://www.cds-inc.com/

Table of Contents

Chapter 1 - Understanding the API

- Overview and System Requirements
- How API Clients and PMfax Servers Interact
- Index and Tag Values
- Session Handles
- Line Management
- Sending
- Receiving
- ASCII Conversion and Document Creation
- Import, Export, Conversion and Concatenation
- Status Management and Reporters
- LAN Routing and Notification
- T1 Lines and Dialers
- Other Features

Chapter 2 - Installing and Getting Started

- Installing the API
- Starting to Program with the API
- Getting Technical Support

Chapter 3 - Function Reference

- How to Use this Information

- Return Codes
- 16- and 32-Bit Entry Points and Definitions
- FxClose
- FxConvert
- FxDelete
- FxExport
- FxImport
- FxIndexToFile
- FxInfo
- FxKill
- FxLogCheck
- FxLogParse
- FxMessage
- FxNextIndex
- FxNextTag
- FxOCR
- FxOpen
- FxPortMode
- FxPrint
- FxPrintJob
- FxReceive
- FxReceiveMode
- FxRegisterDialer
- FxRegisterNotifier
- FxRegisterReporter
- FxRegisterRouter
- FxRoute
- FxSend
- FxServerStatus
- FxStatus
- FxTagToIndex
- FxTextToFax
- FxWakeupReceive

Chapter 4 - Dialog Boxes for 32-bit PM Programs

- FxWinSend
- FxWinPhoneBook

Chapter 1 - Understanding the API

Overview and System Requirements

Keller Group's *PMfax* products are fax products for the OS/2 operating system. These popular OS/2 products are available in single line, multiline and LAN versions. Keller's PMfax products are published by several vendors under several names, including *FaxWorks Pro for OS/2* (from Global Village Communication) and *OpenPort for OS/2* or *Corporate PMfax* (from NiteHawk/Open Port Technology). Provided that you have a current version of the products that supports the API interface, the features of the *v*arious products are the same. For convenience, this manual will refer to Keller's fax product as *PMfax*, but you can also use the API with *FaxWorks Pro for OS/2*, *OpenPort* or other licensed Keller fax products.

The API is hardware-independent and supports all configurations of the PMfax products. Applications that you write using the API will work with all fax hardware that is supported by the PMfax products, and will also work with the multiline and LAN versions of PMfax. Your application uses the PMfax engine as a "fax server", but the user can continue to use the end-user features of the PMfax product, too. Dedicated machines are not required.

The API is designed for OS/2 application programmers and provides extensive features for sending, receiving, status reporting, format conversion and other internal features of the PMfax engine.

The API is intended for programmers using the C language or other OS/2 languages that can call OS/2 DLL files. Programming experience is required.

[Note: If you are looking for advanced ways of creating and sending fax documents from DOS, Windows, OS/2 or command file programs without programming in C language, you should see the Printer Driver Toolkit. The Printer Driver Toolkit provides extensive features for fax document creation and spooled sending, but does not provide the level of direct control that is available through the API.]

The API can be used on OS/2 version 2.0 or later. As described below, the API allows you to communicate with a PMfax program to obtain fax services, so a compatible version of the PMfax program must be running. Some API features will be available only when used with appropriate LAN/multiline versions of the PMfax program or appropriate fax hardware.

The API is implemented as a 32-bit OS/2 Dynamic Link Library file (FxAPI.DLL) which takes advantage of OS/2 2.x features, but it provides both 16-bit and 32-bit entry points for all calls.

You can use compilers which generate either 16-bit or 32-bit code for creating your applications. All sample programs are tested with the IBM C Set++, Borland C++ for OS/2 and Microsoft C 6.0 compilers. Your applications must be run on OS/2 2.0 or later.

You may distribute the FxAPI.DLL file with your applications. There is no royalty for runtime use of the FxAPI.DLL file, but since the API requires the services of a PMfax program, a licensed copy of PMfax (or another licensed Keller product, such as *FaxWorks Pro for OS/2* from Global Village Communication) is required on the delivery system.

Only the FxAPI.DLL file can be distributed with your applications. Other files that come with the API Toolkit are used only for program development and may not be distributed to other parties.

How API Clients and PMfax Servers Interact

This is a client-server system. The PMfax program is the server. Your application is the client.

Your application makes requests by calling API entry points. The API entry points are provided by the FxAPI.DLL file. For most entry points, the FxAPI.DLL function will transact a named pipe to the PMfax program. In other words,

communication with the PMfax program (the fax server) is by OS/2 named pipe, but this is transparent to your application. Your application simply uses standard function calls to the API entry points to obtain fax services.

The PMfax program is fully functional at all times. The user can continue to use the PMfax program normally even while the PMfax program is providing services to your applications.

Multiple application programs, and even multiple threads in a program, can use fax services simultaneously. To use fax services, your program first opens a session with the fax server and obtains a handle which is used for subsequent calls. Multiple programs can maintain active sessions with a server.

In many cases, the PMfax program (the server) and your application program (the client) will be running on the same machine. However, provided that your LAN system supports OS/2 named pipes, the API calls also work across the LAN.

On a LAN that has multiple machines running PMfax, a program can maintain active sessions with multiple fax servers. For example, you can build a very large fax broadcast system by using several multiline copies of PMfax on a LAN, and you can use the API to write a control program that manages all the machines from a single "fax operator's console". Your control program, which can be a straightforward single-threaded OS/2 program, can distribute fax jobs across your network and balance the load among the available fax servers.

Index and Tag Values

When doing computer-based fax processing, it is important to keep track of fax documents and monitor the status of fax jobs. PMfax uses *index* values to efficiently deal with fax documents. The API uses *tag* values keep track of fax jobs. It is very important to understand the difference between the index and tag values.

The PMfax program tracks its internal fax documents using an index number. The index is displayed in the "Id" field in PMfax's fax log display (which is displayed by the **Fax Open log** command in PMfax). The PMfax program uses the index value to create the internal file name for the fax document. For example, the fax document for index "123" is stored in the internal file "fx000123.fax".

Using the PMfax program, it is easy to send the same document to many recipients (i.e., to "broadcast" the document). PMfax can keep a single copy of the fax document on disk even though you are sending the document to many different recipients. PMfax can dynamically add cover sheet and header line information to a fax document, so a single copy of the fax document can be shared even when each recipient is getting customized cover page and/or header line information. Since the index value identifies the fax document, there can be many records in the fax log with the same index value.

But what if you want to send a fax document and later check on whether or not it was successfully transmitted? If the index value can be used by many different send requests, how do you track the success or failure of *your* send request?

The answer is to use a *tag* value for your send request. By tagging a "job", you can use the tag value to monitor the status of the job. Tags are unique to a given send or receive request. By assigning a *tag* value when you make a **FxSend** or **FxReceive** call through the API, you can use the *tag* to monitor the status of your send or receive event.

Note that an index number is an "internal" name for a fax document file, just as a file name (like MyFax.TIF) is an "external" name for a fax document file. The API calls that deal with a fax document allow you to specify the document using either an index number or a file name. For many applications, you can either allow PMfax to keep the documents internally in the fax log and use the index as the document's "name", or you can export the fax document to your own file and refer to it by its file name.

Session Handles

To use fax services, your program first opens a session with the fax server by calling the **FxOpen** function. The **FxOpen** function returns a handle which is used for subsequent calls. When your program is finished, it should call **FxClose** to close the session with the fax server.

By default, **FxOpen** will look for the PMfax program on your local system. But when running on a LAN, you can specify the machine and fax data directory that you wish to use for the session. This allows your applications to attach to fax servers on the LAN.

The use of the handle allows multiple application programs, and even multiple threads in a program, to use fax services simultaneously without conflicting with each other. A given session is used by one active API call at a time, but a program can have multiple active sessions with a fax server. On a LAN with multiple fax servers, a program can also maintain active sessions with multiple fax servers.

Line Management

The PMfax program's **Options Ports** screen is used to select the lines that will be used for fax sending and receiving activities. In the single-line version of PMfax, you select one line as your *Send/Receive* line. In multiline versions, each line can be separately configured as *Send/Receive*, *Send*, *Receive*, *Standby* or *Off*. The API includes a **FxPortMode** call to change line status between the *Send/Receive*, *Send*, *Receive* and *Standby* modes.

All lines that are in *Send/Receive*, *Send* or *Receive* status are under the control of the PMfax program. When the user of the PMfax program sends a fax, the send request is added to the fax log and one of the *Send/Receive* or *Send* lines is used to transmit the fax. When the user of the PMfax program puts the PMfax program into receive mode, the program will answer incoming calls on the *Send/Receive* and *Receive* lines and add the received faxes to the fax log. These lines can also be used by "spooled" **FxSend** and **FxReceive** API calls (see below) because spooled sending and receiving uses the fax log and allows the PMfax program to control the lines.

Lines that are in *Standby* status (and only these lines) may be used for "direct" and "direct current call" sending or receiving with the API. *Standby* lines are under the control of the API. *Standby* lines are not used by the PMfax program for its normal sending or receiving activities. If your application requires the use of direct sending or receiving, it must use a line that is configured in *Standby* mode, or it must use the **FxPortMode** call to change an available line to *Standby* mode before doing the direct sending or receiving call.

Sending

After calling **FxOpen** to establish a session with the PMfax program (the fax server), you can call **FxSend** to send a fax. The **FxSend** call has a great deal of flexibility, and you can make several independent choices on how you use it:

1. You can specify the document that you wish to send by index value or file name, or you can specify a "cover sheet only" transmission. A file name must be a TIFF Class F (TIFF-F) file. You can use the PMfax printer driver to create TIFF-F files from ASCII text or by printing from other applications. You can also use the **FxImport** or **FxConvert** calls to convert various other formats to the TIFF-F format.
2. You can specify a method for sending: *spooled, direct,* or *current call direct.*
    - The *spooled* approach puts the send job in PMfax's fax log and let's the PMfax program manage the transmission, including the optional use of automatic retries and future date/time sending. The *spooled* approach allows PMfax to control the lines, so it can be used on line that are in *Send/Receive* or *Send* mode.
    - The *direct* approach immediately dials and sends the fax without putting the send job in the fax log. Your application controls the retries and status reporting for the job. The line must be in *Standby* mode.
    - The *current call direct* approach is like *direct,* but it assumes that call is already established and you want to immediately start sending (without dialing, etc.).

3. You can either wait for completion before returning from the call, or you can return immediately. If you wait, the call will return the status information for the send job. If you return immediately, you can use the tag value with the **FxStatus** call to check status later.
4. You can force a specified line to be used, or you can leave it up to the PMfax program to select any available line of the appropriate mode.
5. You can override the PMfax program's default TSI string, receive the recipient's CSI string, or specify special cover sheet and header line information. You can even provide a "callback" function for receiving real-time feedback on the progress of the call.

You can select the best sending method for your particular application. In some situations, you can use either spooled or direct sending.

For example, suppose that you wish to build a large fax broadcast system. You can install several multiline copies of PMfax on several machines on a LAN, and you can write a control program that distributes the send jobs to the various machines. This control program could work in several ways: 1) a single-threaded application that does spooled sending and monitors the backlog (using **FxServerStatus**) to ensure that all systems stay busy, 2) a single-threaded application that does direct/no-wait sending, or 3) a multi-threaded application that uses one thread per line and does direct/wait sending. In this situation, the first approach is probably the easiest to write, and it also provides the best line utilization since each PMfax program will manage its own lines.

Receiving

The **FxReceive** call also has a great deal of flexibility, and you can make several choices on how you use it:

1. You can receive the fax document into a named TIFF-F file, or refer to the fax document using its index value.
2. You can do *spooled*, *direct* or *current call direct* receiving.
    - *Spooled* receiving deals with fax documents that have been received by the PMfax program on *Send/Receive* or *Receive* lines and placed in the fax log. Each call will return the next received fax document, or tell you if all received faxes have been handled.
    - *Direct* receiving is used with your application wishes to directly control the answering of incoming calls and receiving of fax documents on *Standby* lines. The call will wait for a fax to be received, or you can use **FxWakeupReceive** from another thread to force a return.
    - *Current call direct* receiving is like *direct*, but it assumes that the call is already established and you want to immediately start receiving (without waiting for a ringing line, etc.).
3. You can deal with a specified line, or you can leave it up to the PMfax program to select any available line of the appropriate mode.
4. You can get the senders TSI string, and you can even provide a "callback" function for receiving real-time feedback on the progress of the call.

ASCII Conversion and Document Creation

The PMfax printer driver and PMfax application program provide powerful features for creating, "scanning" (by faxing in documents) and editing fax documents. Even if you are using API calls for most operations, you might also want to use the printer driver in your application. Or working off line, you may want to use the PMfax printer driver and application program for generating fax documents for fax broadcasting or fax-on-demand situations.

To convert ASCII text into a fax document, use the PMfax printer driver. The FxTextToFax API call provides a convenient way of using the PMfax printer driver from your applications, but you can also use the PMfax printer driver directly. As described in the PMfax Reference Manual and the Enhanced Printer Driver Developer's Manual, you can copy ASCII text to the printer driver's LPT device and the text will be converted into a fax document. You can use the >>FONT= command to select an Adobe font for the conversion. If you use the >>FILE= command, the fax document

will be written directly into a TIFF-F file that you specify. The Printer Driver Toolkit provides expanded printer driver features and documentation.

You can have your other DOS, Windows or OS/2 applications print directly to the PMfax printer driver to produce high-quality fax documents (see the User's Guide and Reference Manual). If you press the *Cancel* button on the printer driver's Send Fax pop-up screen (or have the printer driver configured for *Log entry* rather than *Send pop-up* action), the fax document will be placed in PMfax's fax log with a status of *Print*.

You can use a fax machine as a "scanner" for collecting fax documents for your application. Simply send the fax from a fax machine on one phone line and receive the fax with the PMfax program on another phone line. The received fax document will be in PMfax's fax log with a status of *Rcvd*. As described in the PMfax README.DOC file, you can also use various third-party scanner software to scan documents and have them converted to fax format by printing to the PMfax printer driver.

The PMfax program is a powerful fax editor. You can edit any fax document (including those from the printer driver, or those that you receive) and then save these documents for use by your fax application. The PMfax program includes the **Edit Crop page** command for cleaning up fax pages, removing header lines and resizing the page to a standard page height. You can use the **Fax Save file** command to save the fax document to TIFF-F files for use by the API calls, or use can directly reference the documents from the fax log using their index value (the *Id* value in the fax log display).

Import, Export, Conversion and Concatenation

The PMfax program currently uses the TIFF Class F (TIFF-F) file format as its native format for fax document files. But you can also convert between TIFF-F, DCX and PCX formats, and you can concatenate multiple fax documents to create a new fax document.

Note that TIFF-F and DCX formats are multipage formats, so all pages of the fax document are kept in one file. PCX is a single-page-per-file format, so PCX files should use an extension of ".001" to enable multipage sequencing, and the API will automatically sequence through the separate page files using extensions of ".001", ".002", ".003", etc.

The **FxConvert** call will convert a file from one format into a file of another type. You can also specify multiple input files (as a comma-separated list of file names) to concatenate multiple files.

The **FxImport** call will convert a file and place it in the fax log, returning the index and tag values for the log entry. You can specify the values for the log entry (status, name, company and notes). You can also specify multiple input files (as a comma-separated list of file names) to concatenate the multiple files into a single fax document.

The **FxExport** call copies a fax document from the fax log (you specify the document's index value) and saves it to a named file in a specified format.

You can also use the **FxImport** and **FxExport** calls to deal with the log's *voice message* (Wave), *text* and *data* formats in PMfax version 3 and later.

Status Management and Reporters

You can get status on your fax jobs through several different approaches:

1. If you call **FxSend** with wait=FALSE, you can call the **FxStatus** function to get current status information on your job. You can repeatedly call **FxStatus** to monitor status changes, and since you use the *tag* value to identify the job, you can monitor multiple jobs from a single thread.
2. If you call **FxSend** or **FxReceive** with wait=TRUE, you can provide your own callback function for monitoring status. The callback function is called when the fax is spooled, when fax activity starts, at each page transition and at completion.
3. You can register your own reporter function by using the **FxRegisterReporter** call. Your reporter function is called whenever a fax is sent or received. This can be used to control displays or provide your own special

processing of sent or received fax documents. For example, your reporter function could process the received fax documents and then tell the fax server to remove the fax document from the system.

LAN Routing and Notification

The LAN version of PMfax, when installed and used in its "private mode", provides extensive workstation support features including user privacy, routing of fax documents and user notification of received fax documents. These features are described in the LAN Installation Guide. The API provides several calls which allow you to exploit the routing and notification features of the LAN version.

You can route fax documents using the **FxRoute** call. Alternatively, you can use the **FxRegisterRouter** call to provide your own routing function. Your routing function will be called whenever a fax is received and the PMfax LAN redirector program is running, and your function can tell the redirector program how to route the fax document. This makes it easy for you to extend the features of the LAN version to include automatic OCR routing, automatic routing based on line number, or other value-added features.

You can also use the **FxRegisterNotifier** call to provide your own LAN notification function. Your function is called whenever the PMfax LAN version would normally notify the LAN user of a sent, received or routed fax event, and your function can choose to use normal notification, handle the notification itself, or take full responsibility for the fax document and remove it from the system. This can be used to provide customized notification systems (such as E-mail or message lights) or fax delivery systems (such as sending the entire fax via E-mail).

Using these API calls, you can build upon the base functionality of the PMfax LAN system and turn it into something that is even more powerful!

T1 Lines and Dialers

Most fax hardware uses normal "loop" telephone lines, and the control of the telephone line is usually done through the fax hardware. But you can adapt the PMfax program to work with T1 lines, E1 lines or other special telephone line interfaces. This can be used with MVIP™ configurations or other situations which require special processing for establishing telephone connections.

The API's **FxRegisterDialer** call allows you to provide your own dialer function. Whenever the PMfax program would normally dial, it will instead call your own dialer function. Your function can establish the line connection and then tell the PMfax program how it should proceed.

Your dialer function can even tell the PMfax program "*Don't bother... I already delivered it for you...*".

You can use this feature for adding your own fax transmission methods into PMfax. For example, based on the fax number or other information, you could deliver some documents via E-mail while allowing PMfax to send others normally.

Other Features

The API also includes additional utility functions that have not been mentioned above. Some of the utility functions include the following:

- **FxReceiveMode** - to change a PMfax program's receive mode
- **FxServerStatus** - to get information about the lines and jobs on a PMfax server
- **FxPrint** and **FxPrintJob** - to have PMfax print pages from a fax document
- **FxDelete** - to have PMfax delete log entries and/or fax documents

- **FxInfo** - to get information about a fax document
- **FxLogParse** - to extract information from a PMfax log record
- **FxNextIndex** - to obtain a unique document index value
- **FxNextTag** - to obtain a unique job tag value
- **FxTagToIndex** - to obtain the document index value that is associated with a job tag
- **FxIndexToFile** - to obtain the full document file name from the document index value
- **FxMessage** - to obtain a character string message for a FxAPI return code
- **FxOCR** - to use the PMfax program's OCR engine
- **FxLogCheck** - to clean the log in various ways (delete items based on date or age, etc.)

Chapter 2 - Installing and Getting Started

Installing the API

For program development, copy the FxAPI.DLL file into your C:\OS2\DLL directory (or another directory that is included in the LIBPATH in your CONFIG.SYS file). Copy the FxAPI.LIB file to your compiler library directory, and copy the FxAPI.h file and the example programs (*.C) to your development directory.

For delivery of runtime-only version of your programs that use the API, you must copy the FxAPI.DLL file into a LIBPATH directory on the delivery system. The other files in the API Toolkit are not needed when you deliver your applications.

In all cases, a licensed copy of PMfax (or FaxWorks OS/2, or another Keller fax product) must be used to provide the fax services. The API supports client/server programming, with your application as the client and a running copy of PMfax as the fax server.

Follow the instructions in your PMfax Reference Manual and/or PMfax LAN Installation Guide to install and use the PMfax program. Be sure that you can send and receive with the PMfax program before attempting to use the API calls.

Starting to Program with the API

First read Chapter 1 in this manual to understand the API design. It is important to have a general understanding of the API clients and the fax servers.

Then look at the example programs. In the next chapter, each API call lists the example programs where you can find working examples that demonstrate the use of the API call.

The best way to start programming with the API is to study the example programs, observe the API calls and API data structures that are used in the example programs, and look at the reference information in the next chapter to learn more about the API calls and data structures.

We strongly encourage you to use the example programs and to modify them to create your own programs. Several example programs are provided, and there is probably an example program that can serve as a good starting point for your application.

Make sure that you can compile the example program and get it to work on your system, then develop your own fax program by gradually modifying the working program. By making incremental changes to the working program rather than starting a new program from scratch, you can more easily identify programming problems.

The example programs in the API Toolkit include the following (your API Toolkit files may also contain additional example programs that aren't listed here):

- **Send_Spl.C -** Spooled send example program - FxSend, FxStatus, FxDelete
- **Send_Dir.C** - Direct send example program - FxSend, FxPortMode
- **MegaSend.C** - Multi-server send example program - FxSend, FxServerStatus
- **Recv_Log.C** - Receive from log example program - FxReceive, FxReceiveMode
- **Recv_Dir.C** - Direct receive example program - FxReceive, FxPortMode
- **CVT.C** - File conversion example program - FxConvert, FxInfo
- **RCV.C** - Receive mode example program - FxReceiveMode
- **SRV.C** - Server status example program - FxServerStatus
- **DLR.C** - Dialer callback function example - FxRegisterDialer
- **NFY.C** - Notifier callback function example - FxRegisterNotifier
- **RPT.C** - Reporter callback function example - FxRegisterReporter
- **RTE.C** - Router callback function example - FxRegisterRouter
- **Send_Pop.C** - PM example - Send/Phone book popups - FxWinSend, FxWinPhoneBook

Getting Technical Support

Before you attempt to use the API Toolkit, you are expected to know how to develop OS/2 programs using your compiler. Please contact your compiler manufacturer with general questions about how to use your compiler to create OS/2 programs.

To receive support for questions about the API, the PMfax program or product licensing, please contact us by e-mail. Try to include sufficient information to allow us to recreate a problem or research the question. We will contact you by phone, fax or e-mail with a response.

For problems with the API, we encourage you to provide source code which demonstrates the problem.

For fax sending or receiving problems with your fax hardware, please use the -V parameter when starting the PMfax or FaxWorks program to capture debugging information. This is described in the **Startup Parameters** section in the Reference Manual. If you start the program with the -V parameter, recreate the sending or receiving problem and then exit the program, the VOUT file will contain several pages of detailed information about the program, your fax hardware, and the commands and responses between the program and your fax hardware. Please fax or e-mail the entire VOUT file for analysis.

Chapter 3 - Function Reference

How to Use this Information

This is a reference section. The API calls are listed in alphabetical order. Information about special data structures and return codes for the API call is included on the page where the API call is described.

See the FxAPI.h file for the latest information on all API functions, return codes, etc. The information in FxAPI.h will reflect any new features or changes to the API.

Where applicable, the "Example programs" are noted for the API call. The example programs (included with your API Toolkit files) show how to use the API call.

Return Codes

The API calls use return values to indicate success or failure. API calls return a negative value if the API call fails or a non-negative value if the API call succeeds. But for sending or receiving fax documents, the success of the API call does not necessarily mean that the fax document was successfully sent or received. A fax error code may be returned to show that the API call succeeded but the fax activity failed.

It is important for your application to examine return codes and take appropriate action. For example, a **FxSend** call may return the FXRET_BUSY return code which means that the **FxSend** call worked fine but the fax document could not be sent because the recipient's fax line was busy.

The list of all API return codes is shown here, but see FxAPI.h for the latest list. The reference page for each API call may elaborate upon the meaning of normal return codes which are important to the API call. Use symbolic names for the error codes in your programs since numeric values may change.

Fax Error Codes - API call succeeded, but fax send/receive failed:

```
#define FXRET_NODRV        38    device driver not found or not available
#define FXRET_NOISE        37    bad data, probably noisy line
#define FXRET_BAD_SEND     36    general failure during send
#define FXRET_BAD_RECEIVE  35    general failure during receive
#define FXRET_BUSY         34    fax recipient's line was busy
#define FXRET_CONFIG_ERROR 33    fax hardware is improperly configured
#define FXRET_DIALTONE     32    dial tone detected after dialing
#define FXRET_USER_KILLED  31    fax was killed by user or FxKill
#define FXRET_MODEM_ERROR  30    fax hardware command response error
#define FXRET_NO_ANSWER    29    fax recipient's line was not answered
#define FXRET_NO_CARRIER   28    call was answered, but no fax detected
#define FXRET_NO_DIALTONE  27    no dial tone was detected on fax line
#define FXRET_NO_MEMORY    26    memory allocation error during fax
#define FXRET_NO_TRAIN     25    fax hardware couldn't train with remote fax
#define FXRET_SYSTEM_ERROR 24    file error during fax
#define FXRET_TIMEOUT      23    no fax hardware response to command
#define FXRET_VOICE        22    voice detected after dialing
#define FXRET_HANGUP       21    fax modem detected line hangup
#define FXRET_ROUTE_ERROR  20    route attempted to unknown user
```

Normal Return Codes - see each function for details:

```
#define FXRET_ASYNC        12    asynch return from dialer function
#define FXRET_OLD_RECORD   11    deleted, will be removed by garbage collection
#define FXRET_EDIT         10    log status value
#define FXRET_PRINT         9    log status value
#define FXRET_ROUTE         8    log status value
#define FXRET_SENT          7    fax transmission completed successfully
#define FXRET_RECEIVED      6    fax was successfully received
#define FXRET_SPOOLED       5    fax activity not started yet
#define FXRET_ACTIVE        4    fax send or receive in progress
#define FXRET_DIALED        3    application dialer handled dialing
#define FXRET_DELETE        2    application reporter wants fax deleted
#define FXRET_WAKEUP        1    receive was interrupted by FxReceiveWakeup
#define FXRET_OK            0    successful api function return
```

API Error Codes - API call failed:

```
#define FXRET_NO_SERVER       -1    fax server not there or not responding
```

```
#define FXRET_SERVER_BUSY     -2    resource already allocated by fax server
#define FXRET_SERVER_TIMEOUT  -3    fax server response timeout
#define FXRET_NO_FILE         -4    specified file does not exist
#define FXRET_FILE_ERROR      -5    file is corrupt or not of specified type
#define FXRET_PARAMETER_ERROR -6    incompatible parameters were specified
#define FXRET_FAILED          -7    general failure in API
#define FXRET_TAG_NOT_FOUND   -8    tag unknown by server
#define FXRET_NO_LINE         -9    specifed fax line doesn't exist or in use
#define FXRET_MISMATCH       -10    version mismatch - API versus Program
```

16- and 32-Bit Entry Points and Definitions

FxAPI.h works for both 16- and 32-bit compilers. FxAPI.DLL has both 16- and 32-bit entry points (but must be run under OS/2 2.x).

The FxAPI.h file provides the necessary #define values for 16-bit programming as shown below. As long as you include FxAPI.h in your source files, these #defines will automatically provide the appropriate entry points to your application program.

```
#ifndef APIENTRY16
/* 16-bit os2def.h, so switch to 16-bit entry names */
# define FXPTR              _far *
typedef ULONG               BOOL32;
typedef BOOL32              FXPTR PBOOL32;
# define FxOpen             Fx16Open
# define FxClose            Fx16Close
# define FxSend             Fx16Send
# define FxReceive          Fx16Receive
# define FxStatus           Fx16Status
# define FxPortMode         Fx16PortMode
# define FxReceiveMode      Fx16ReceiveMode
# define FxServerStatus     Fx16ServerStatus
# define FxWakeupReceive    Fx16WakeupReceive
# define FxImport           Fx16Import
# define FxExport           Fx16Export
# define FxConvert          Fx16Convert
# define FxPrint            Fx16Print
# define FxRoute            Fx16Route
# define FxDelete           Fx16Delete
# define FxNextIndex        Fx16NextIndex
# define FxInfo             Fx16Info
# define FxLogParse         Fx16LogParse
# define FxNextTag          Fx16NextTag
# define FxTagToIndex       Fx16TagToIndex
# define FxIndexToFile      Fx16IndexToFile
# define FxMessage          Fx16Message
# define FxRegisterDialer   Fx16RegisterDialer
# define FxRegisterReporter Fx16RegisterReporter
# define FxRegisterRouter   Fx16RegisterRouter
# define FxRegisterNotifier Fx16RegisterNotifier
# define FxDialerCompleted  Fx16DialerCompleted
# define FxKill             Fx16Kill
# define FxTextToFax        Fx16TextToFax
```

```
# define FxPrintJob         Fx16PrintJob
# define FxOCR              Fx16OCR

#else /* 32-bit */
# define FXPTR        *
#endif

#define FXENTRY         EXPENTRY
#define FXRET           LONG
typedef PVOID           FXHANDLE;
typedef FXHANDLE        FXPTR PFXHANDLE;

/* char array lengths */
#define FXLEN_LOGLINE   4096
#define FXLEN_ID        (20+1)
#define FXLEN_TEXT      (40+1)
#define FXLEN_HEADER    (80+1)
#define FXLEN_PATH      (256+1)
#define FXLEN_COMMENT   (1000+1)
```

FxClose

Description:

The FxClose call will close the session with the fax server. When your program is finished with the fax session that was established by calling FxOpen, it should call FxClose to release the handle and free the session's resources.

```
/* Close the session */
FXRET FXENTRY FxClose(
    FXHANDLE handle );
```

Return codes:

```
FXRET_OK        successful api function return
  or
API Error Code
```

Example programs:

FxClose is used in all example programs.

FxConvert

Description:

The FxConvert call will convert a file from one format into a file of another type. You can specify multiple input files (as a comma-separated list of file names) to concatenate multiple input files into one output file. You can force the output file to be fine resolution (200x200 dpi, rather than the 200x100 dpi of normal resolution), which is useful for ensuring that PCX or DCX images will have a square aspect ratio.

TIFF-F and DCX formats are multipage formats, so all pages of the fax document are kept in one file. If the "finecvt" argument is false and FxConvert is given a TIFF-F file (or a list of TIFF-F files) as the "file" argument, the output file will match the resolution of the first input file. If a file list contains mixed resolutions, the data from those files which do

not already match the resolution of the first file will be converted. The FxConvert call is optimized so that no decode/encode of the TIFF-F data is done if all files in the file list and the output file are TIFF-F format of the same resolution.

PCX is a single-page-per-file format, so PCX files should use an extension of ".001" to enable multipage sequencing, and the API will automatically sequence through the separate page files using extensions of ".001", ".002", ".003", etc. The PCX format which you provide to this call must be a "fax style" (Intel SatisFAXtion-compatible) PCX format. Specifically, it must be monochrome and 1728 pixels wide. The FxConvert call does not provide generalized PCX conversion features. If necessary, you can use the Edit/Paste/PCX command in the fax application for pasting other types of PCX files onto fax pages.

```
FXRET FXENTRY FxConvert(
    FXHANDLE handle,
    PCHAR file,         /* file (or comma-separated file list) to read */
    ULONG filetype,     /* type of file to read (FXFT_*) */
    PCHAR file2,        /* file to create */
    ULONG filetype2,    /* type of file to create (FXFT_*) */
    BOOL32 finecvt );   /* convert file on copy to fine (if normal res) */

/* file types */
#define FXFT_TIFFF      0
#define FXFT_DCX        1
#define FXFT_PCX        2
/* PCX with sequencing file extensions as File.001, File.002, ... */
```

Return codes:

```
FXRET_OK        successful api function return
  or
API Error Code
```

Example programs:

CVT.C File conversion example program

FxDelete

Description:

The FxDelete call will delete log entries and fax files from the fax system.

If you specify a document index value but tag = 0, the internal fax document file is deleted and all log entries which reference that fax document file are removed from the fax log. This is done with a single pass through the fax log. Deleting by document index is therefore a way to ensure that a fax document is completely removed from the fax system.

If you specify a job tag value, the job's log entries are removed, and unless the job's fax document is shared by other jobs that are still in the fax log, the job's internal fax document file is also deleted. Deleting by job tag value is therefore a good (and safe) way to remove your fax job from the fax system when you no longer need to keep it around. If a tag value is specified but index = 0, this operation requires two passes through the log file. If both the tag value and the index value are provided, this operation is done with a single pass through the log file, and it is the caller's responsibility to ensure that the document index value is correct for the specified job tag value.

```
FXRET FXENTRY FxDelete(
    FXHANDLE handle,
    ULONG tag,      /* job tag to be deleted, or 0 if no tag specified */
```

```
    ULONG index );  /* document index to be deleted, or 0 if not specified */
```

Return codes:

```
FXRET_OK        successful api function return
  or
API Error Code
```

Example programs:

Send_Spl.C Spooled send example program

FxExport

Description:

The **FxExport** call copies an interal fax document from the fax log (identified by the document index value) and saves it to a named file in a specified format. You can use this call to extract a copy of the fax document from the fax system. This call can also be used with the log's *voice message* (Wave), *text* and *data* items in PMfax version 3 and later.

Note that TIFF-F and DCX formats are multipage formats, so all pages of the fax document are kept in one file. PCX is a single-page-per-file format, so PCX files should use an extension of ".001" to enable multipage sequencing, and the API will automatically sequence through the separate page files using extensions of ".001", ".002", ".003", etc.

```
FXRET FXENTRY FxExport(
    FXHANDLE handle,
    ULONG index,       /* log doc-id to export */
    PCHAR file,        /* file to export to */
    ULONG filetype,    /* type of file to create (FXFT_*) */
    BOOL32 finecvt );  /* convert file on export to fine (if normal res) */

/* file types */
#define FXFT_TIFFF  0    fax - TIFF Class F multipage
#define FXFT_DCX    1    fax - DCX multipage
#define FXFT_PCX    2    fax - sequencing file ext: File.001, .002...
#define FXFT_MSG    3    voice message - Wave format
#define FXFT_TXT    4    text - ASCII format
#define FXFT_DAT    5    data - any format
```

Return codes:

```
FXRET_OK        successful api function return
  or
API Error Code
```

Example programs:

FxImport

Description:

The FxImport call converts a file and places it in the internal fax log, returning the index and tag values for the log entry. You can specify the values for the log entry (status, name, company and notes). You can also specify multiple input files (as a comma-separated list of file names) to concatenate the multiple files into a single fax document. This call can also be used with the log's *voice message* (Wave), *text* and *data* items in PMfax version 3 and later.

Note that TIFF-F and DCX formats are multipage formats, so all pages of the fax document are kept in one file. PCX is a single-page-per-file format, so PCX files should use an extension of ".001" to enable multipage sequencing, and the API will automatically sequence through the separate page files using extensions of ".001", ".002", ".003", etc.

```
FXRET FXENTRY FxImport(
    FXHANDLE handle,
    PCHAR file,         /* file (or comma-separated file list) to import from */
    ULONG filetype,     /* type of file to read (FXFT_*) */
    PCHAR status,       /* log status for log (Edit, Print, Rcvd, Read, Route) */
    PCHAR name,         /* nonNULL=from_name field in log entry */
    PCHAR company,      /* nonNULL=from_company field in log entry */
    PCHAR notes,        /* nonNULL=notes field in log entry */
    PULONG tag,         /* nonNULL: return assigned tag for new entry */
    PULONG index );     /* nonNULL: return log doc-id of fax */

/* file types */
#define FXFT_TIFFF  0    fax - TIFF Class F multipage
#define FXFT_DCX    1    fax - DCX multipage
#define FXFT_PCX    2    fax - sequencing file ext: File.001, .002...
#define FXFT_MSG    3    voice message - Wave format
#define FXFT_TXT    4    text - ASCII format
#define FXFT_DAT    5    data - any format
```

Return codes:

```
FXRET_OK        successful api function return
  or
API Error Code
```

Example programs:

FxIndexToFile

Description:

The FxIndexToFile call converts a document index value into the pathname of the internal fax document file. Normally, you do not need to use this call since the storage of the internal fax documents is an internal issue which is automatically managed by the API and the fax server. Rather than directly manipulating the internal files, it is usually better to use API calls like FxExport and FxExport.

```
FXRET FXENTRY FxIndexToFile(
    FXHANDLE handle,
    ULONG index,    /* log doc-id of a fax */
    PCHAR file );   /* return file name (_MAX_PATH sized buffer) */
```

Return codes:

```
FXRET_OK        successful api function return
  or
API Error Code
```

Example programs:

FxInfo

Description:

The FxInfo call accepts a fax document (identified by either a job tag value, a document index value, or a TIFF-F document file name) and provides information about the fax document. Given a job tag value, this call can return the fax log record which can then be passed to FxLogParse to obtain additional information.

```
FXRET FXENTRY FxInfo(
    FXHANDLE handle,
    ULONG tag,      /* non0=tag of fax */
    ULONG index,    /* non0=log doc-id (tag=0) */
    PCHAR tifff,    /* nonNULL=TIFF class-F file (tag=0, index=0) */
    PULONG pages,   /* nonNULL: return number of pages */
    PBOOL32 fine,   /* nonNULL: return TRUE if finemode */
    PCHAR logline );/* nonNULL: return entire Fax.Log line (for tagged fax)
                       (requires FXLEN_LOGLINE length buffer for worst case)*/
```

Return codes:

```
FXRET_OK        successful api function return
  or
API Error Code
```

Example programs:

CVT.C File conversion example program

FxKill

Description:

The FxKill call accepts a job tag value (for a job that is either spooled or currently sending) and aborts the job. The job is left in the log in "Killed" status.

```
FXRET FXENTRY FxKill(
    FXHANDLE handle,
    ULONG tag,              /* tag of fax to kill (does not delete fax) */
    BOOL32 spooled );       /* TRUE=fax was spooled, else direct send */
```

Return codes:

```
FXRET_OK        successful api function return
  or
API Error Code
```

FxLogCheck

Description:

The Fx:LogCheck call provides access to log clean up services which are like those provided by the "Utilities Maintain log" command in the product's user interface.

OLDDATE removes all log entries with a date that is older than a specified date or a specified number of days. If no remaining log entry shares the associated document, it also removes the document itself. For example, if you specify a value of "90", all entries that are more than 90 days old are removed.

DELSENT automatically removes all log entries that show a status of "Sent" and, if no remaining log entry shares the associated document, removes the document itself.

DELINFO removes "informational" log entries (from failed attempts which were retried) but keeps "final status" entries. Log entries that don't have any document or cover sheet data, such as errors from "wrong number" calls to your line when in receive mode, will also be deleted. (Same as log's "Clean" command.)

DELNOFAX removes all log entries for which the associated document cannot be found. If you manually delete any *.FAX or *.MSG files, you can use this to remove log entries for the deleted files.

ADNOENT creates new log entries for *.FAX, *.MSG, *.TXT and *.DAT files (i.e., fax documents and voice message files) that are not associated with a log entry. Ordinarily, each file is associated with one or more log entries, and the file is automatically deleted when its last associated log entry is removed.

If MOVEPATH is used, log entries and their document files will be moved to your specified archive directory rather than deleted. The directory will be created if necessary, and the directory will be a complete "data directory" which contains a fax.log file and the document files for the log entries.

The ERRBUF string reports the number of items "deleted" (a log entry and a file deleted), "removed" (a log entry removed, but no file deleted), and "added" (a log entry added) as a result of the call or an error message.

Note: If the software is started with the *"-L pathname"* parameter, it will use the pathname as its data directory (so you can view, print, delete, save or drag/drop log entries from your archive). To switch back to your current data directory, start the software with the *"-L pathname"* parameter again and specify the pathname of your current data directory. The program remembers the last data directory, so you only need to use the -L parameter when you want to change to a different directory.

```
FXRET FXENTRY FxLogCheck(
    FXHANDLE handle,
    PCHAR olddate,   /* nonNULL: number of days or date to delete before */
    BOOL32 delsent,  /* TRUE=delete 'sent' entries */
    BOOL32 delinfo,  /* TRUE=delete informational log entries */
    BOOL32 delnofax, /* TRUE=delete entries missing files */
    BOOL32 addnoent, /* TRUE=add 'found' entries for files w/no log entry */
    PCHAR movepath,  /* nonNULL: move files/log to path instead of delete */
    PCHAR errbuf );  /* nonNULL: return message string */
    /* returns FXRET_OK or FXRET_FAILED */
```

Return codes:

```
FXRET_OK        successful api function return
  or
FXRET_FAILED
```

FxLogParse

Description:

The FxLogParse call accepts a logline string (the fax log record) and extracts the fields for your use. While the fax log records is a string of comma-separated values, you should use this function to extract the values since the internal format of log records may change in future releases.

```
FXRET FXENTRY FxLogParse(
    FXHANDLE handle,
    PCHAR logline,  /* line from Fax.Log file */
    PULONG tag,     /* nonNULL: return tag (0 if none) */
```

```
    PULONG index,   /* nonNULL: return log doc-id */
    PDATETIME dt,   /* nonNULL: return date/time of sendrecv */
    PULONG elapsed, /* nonNULL: return elapsed time of sendrecv in seconds */
    PULONG line,    /* nonNULL: return line used for sendrecv */
    PCHAR fax,      /* nonNULL: return fax number of send */
    PFXINFO info,   /* nonNULL: return fax info of send */
    PBOOL32 final   /* nonNULL: return TRUE = final status (no more retries)*/
    PCHAR remoteid);/* nonNULL: return CSI/TSI (20+1 characters) */


/* FXINFO structure is used by FxSend to specify cover sheet information
 * and by FxLogParse to retrieve the cover sheet information.
 */
typedef struct _FXINFO {
    /* cover sheet */
    ULONG cover_flag;
    CHAR cover_bitmap[ FXLEN_TEXT ];
    CHAR name[ FXLEN_TEXT ];
    CHAR company[ FXLEN_TEXT ];
    CHAR from_name[ FXLEN_TEXT ];
    CHAR from_company[ FXLEN_TEXT ];
    CHAR from_phone[ FXLEN_TEXT ];
    CHAR from_fax[ FXLEN_TEXT ];
    CHAR comment[ FXLEN_COMMENT ];
    /* page heading */
    ULONG heading_flag;
    CHAR heading[ FXLEN_TEXT ];
    /* log notes field */
    CHAR notes[ FXLEN_TEXT ];
    /* owner id field (intermediate and final status routed to this user */
    CHAR owner_id[ FXLEN_TEXT ];
} FXINFO;
typedef FXINFO FXPTR PFXINFO;


/* cover sheet modes (for cover_flag value) */
#define FXCS_OFF           0
#define FXCS_ON            1
#define FXCS_ON_FULLSIZE   2

/* page header modes (for heading_flag value) */
#define FXPH_OFF           0
#define FXPH_ON            1


/* char array lengths */
#define FXLEN_LOGLINE      4096
#define FXLEN_ID           (20+1)
#define FXLEN_TEXT         (40+1)
#define FXLEN_COMMENT      (1000+1)
```

Return codes:

```
FXRET_SENT
FXRET_RECEIVED
  or
Fax Error Code
  or
API Error Code
```

Example programs:

FxMessage

Description:

The FxMessage call converts a numeric API return code into a character string which describes the return status. You can use this call for obtaining textual return values for your error messages and displays.

```
PCHAR FXENTRY FxMessage(
    FXRET retcode );     /* FxAPI return code */
```

Return codes:

```
    pointer to character string message
```

Example programs:

FxMessage is used in all example programs.

FxNextIndex

Description:

The FxNextIndex call returns a unique document index value. If you are using index values for special purposes in your application, you can use this call to get a new index value for your use. Normally, you do not need to use this call since the fax server automatically generates the index values for the fax documents in the fax log.

```
FXRET FXENTRY FxNextIndex(
    FXHANDLE handle,
    PULONG index ); /* return next 6-digit document id from Fax.Idx
            file for manually creating Fx######.Fax files
            and Fax.Log entries (10-999999) */
```

Return codes:

```
FXRET_OK        successful api function return
  or
API Error Code
```

Example programs:

FxNextTag

Description:

The FxNextTag call returns a unique job tag value. If you are using tag values for special purposes in your application, you can use this call to get a new tag value for your use. Normally, you do not need to use this call since the FxSend and

FxReceive calls automatically generate and return a unique tag value.

You might want to use this call to get a job tag value for passing to the Enhanced Printer Driver's >>TO command. The >>TO command is described in the Enhanced Fax Printer Driver Developer's Manual. This would allow you to spool a fax document using the printer driver but then check the job status using the API calls.

```
FXRET FXENTRY FxNextTag(
    FXHANDLE handle,
    PULONG tag );   /* return next 9-digit number from Fax.Tag file
            (similar to Fax.Idx file) for generating
            tags (1-999999999) */
```

Return codes:

```
FXRET_OK        successful api function return
  or
API Error Code
```

Example programs:

FxOCR

Description:

If the fax program has the OCR option installed, you can call FxOCR to have the program do OCR processing on a fax file or bitmap. You can process a bitmap image, a page from a TIFF-F fax file, or an entire TIFF-F fax file. The result is written to a text file which you specify.

```
FXRET FXENTRY FxOCR(
    FXHANDLE handle,
    PCHAR tifff,        /* nonNULL=fax file to OCR, else to bitmap */
    ULONG pageno,       /* non0=page to OCR, else do whole file */
    HBITMAP hbm,        /* nonNULL=handle to bitmap to OCR (tifff=NULL) */
    PCHAR txtfile );    /* output text filename */
```

Return codes:

```
FXRET_OK        successful api function return
  or
API Error Code
```

Example programs:

FxOpen

Description:

To use fax services, your program must first call FxOpen to establish a session with a fax server and obtain a session handle which is used for subsequent calls. The fax server can be on your local machine, or it can be across a LAN which supports OS/2 named pipes.

The use of the session handle allows multiple application programs, and even multiple threads in a program, to use fax services simultaneously without conflicting with each other. On a LAN with multiple fax servers, your program can also maintain active sessions with multiple fax servers.

A program can have multiple active sessions, but a given session should only be used by one simultaneously active API

call at a time. (The exception to this is the FxReceiveWakeup call.)

IMPORTANT: Each thread of a program accessing the API needs its own session handle.

(This includes registered functions such as notifiers and reporters.)

```
/* Open a session with the fax server, determines log directory, ... */
FXRET FXENTRY FxOpen(
    PFXHANDLE handle,
    PCHAR server,      /* NULL=local server, else server computer name */
    PCHAR logdir );    /* NULL=local server, else server log directory */
```

Return codes:

```
FXRET_OK            successful api function return
FXRET_NOFILE        FAX.LOG file not found in LogDir
  or
API Error Code    other errors usually indicate named pipe error
                  when communicating with server
```

Example programs:

FxOpen is used in all example programs.

FxPortMode

Description:

The FxPortMode call is used to query or change the status of a line on the fax server. You can change the line status between the *Send/Receive*, *Send*, *Receive* and *Standby* modes.

```
FXRET FXENTRY FxPortMode(
    FXHANDLE handle,
    LONG line,         /* line to change mode on */
    ULONG mode,        /* FXPM_ new port mode for any line */
    PULONG prevmode,   /* nonNULL: return old port mode */
    PBOOL32 active );  /* nonNULL: return TRUE if currently busy */
    /* returns FXRET_OK or error */

/* port modes */
#define FXPM_QUERY          0    query only, don't change the port mode
#define FXPM_STANDBY        1
#define FXPM_SEND           2
#define FXPM_RECEIVE        3
#define FXPM_SENDRECV       4
```

Return codes:

```
FXRET_OK        successful api function return
  or
API Error Code
```

Example programs:

Send_Dir.C Direct send example program

Recv_Dir.C Direct receive example program

FxPrint

Description:

You can call FxPrint to have the fax server print pages from a fax document. Also see FxPrintJob.

```
FXRET FXENTRY FxPrint(
    FXHANDLE handle,
    PCHAR tifff,        /* nonNULL=TIFF class-F file (index=0) */
    ULONG index,        /* non0=log doc-id (tifff=NULL) */
    ULONG start,        /* non0=1st page to print, else start at beginning */
    ULONG end,          /* non0=last page to print, else print to end */
    PCHAR queue );      /* NULL=use server default, else specify queue name */
```

Return codes:

```
FXRET_OK        successful api function return
  or
API Error Code
```

Example programs:

FxPrintJob

Description:

You can call FxPrintJob to have the fax server print pages from a fax document. Also see FxPrint.

Unlike FxPrint which prints based on *document index*, FxPrintJob prints based on *job tag*. A job may have a cover sheet, and FxPrintJob can print the cover sheet. A job also has a status, and the status will be shown in the print header if the "Add time-stamp header line" option is being used in the fax program. FxPrintJob also includes the delete option so that you can have it automatically delete the job after printing.

If the job has a cover sheet, the cover sheet will be page 1. For example, a job which sends a two-page document with a cover sheet will have three printable pages, so the *start* and *end* arguments could be 1 (cover sheet), 2 or 3. That same document if sent as a job without a cover sheet will have just two printable pages (1 and 2).

```
FXRET FXENTRY FxPrintJob(
    FXHANDLE handle,
    ULONG tag,          /* tag of fax job to print (w/ status&cover if any)*/
    ULONG start,        /* non0=1st page to print, else start at beginning */
    ULONG end,          /* non0=last page to print, else print to end */
    BOOL32 delete,      /* TRUE=delete after printing */
    PCHAR queue );      /* NULL=use server default, else specify queue name */
```

Return codes:

```
FXRET_OK        successful api function return
  or
API Error Code
```

Example programs:

FxReceive

Description:

After calling FxOpen to establish a session with the PMfax program (the fax server), you can call FxReceive to receive a fax. Be sure to read the section called "Receiving" in Chapter 1 for a discussion of the power of the FxReceive call. The call supports "spooled", "direct" or "current call direct" modes of receiving. You can return immediately if using spooled mode, or you can wait for a direct mode operation to complete before returning. If you wait, you can provide a callback function that will be called as status information changes. You can select any-line or specified-line receiving.

```
FXRET FXENTRY FxReceive(
    FXHANDLE handle,
    LONG line,      /* 0=log check for receive on any line (wait=FALSE)
                       +n=log check for receive on specific line (wait=FALSE)
                       +n=direct receive on specific line (wait=TRUE)
                       -n=direct receive current call (wait=TRUE) */
    PCHAR tifff,    /* NULL=put received fax into fax log,
                           nonNULL=receive into non-log file */
    PULONG tag,     /* nonNULL: return assigned tag */
    PULONG index,   /* nonNULL: return log doc-id (tifff=NULL) */
    PCHAR remoteid, /* nonNULL: return TSI (20+1 characters) */
    PFXSTATUS func, /* NULL=no callback, else called for status (wait=TRUE) */
    PVOID apparg,   /* application defined argument to pass to func */
    BOOL32 wait );  /* TRUE=wait for standby line receive
                       FALSE=check log for non-standby 'Rcvd' */


/* FXSTATUS callback function template is used by FxSend and
 * FxReceive to specify a function to receive active fax status.
 * The function is called when fax is spooled, fax activity starts,
 * at each page transition, and at completion.  The status function
 * can only be used on 'wait' type calls.  For no-wait, the same
 * information can be polled with the FxStatus function or
 * completion will be indicated by a call to a registered FXREPORT
 * function.
 */
typedef FXRET (FXENTRY FXSTATUS)(
    PVOID apparg,      /* application defined argument */
    ULONG tag,         /* fax-being-statused tag */
    ULONG index,       /* log doc-id number */
    ULONG line,        /* line number being used */
    ULONG pageno,      /* 0=pre- or post-fax, n=page faxing */
    BOOL32 final,      /* TRUE=final status of fax (no more updates) */
    PCHAR remoteid,    /* CSI/TSI (20+1 characters) (post-fax) */
    FXRET retcode );
    /* retcode FXRET_SPOOLED, ACTIVE, SENT, RECEIVED or fax error or error */
    /* returns from application supplied status function:
     *    FXRET_OK          proceed
     *  FXRET_USER_KILLED   terminate activity
     */
typedef FXSTATUS FXPTR PFXSTATUS;
```

Return codes:

```
FXRET_RECEIVED  fax was received
FXRET_OK        no fax, but API call was OK
  or
Fax Error Code
  or
API Error Code
```

Example programs:

Recv_Dir.C Direct receive example program

Recv_Log.C Receive from log example program

FxReceiveMode

Description:

The FxReceiveMode call allows you to change a fax server's receive mode. You can use this to make the fax program turn receiving off so that a modem is available for a data call, or for other purposes that may require you to change the current state of the PMfax program's receive mode.

The HOLD and RELEASE modes can also be used to change the hold status of the PMfax program.

```
FXRET FXENTRY FxReceiveMode(
    FXHANDLE handle,
    ULONG mode );   /* FXRM_ new receive mode for non-standby lines value */


/* receive modes */
#define FXRM_OFF           0
#define FXRM_CURRENTCALL   1
#define FXRM_ONECALL       2
#define FXRM_ALLCALLS      3
#define FXRM_HOLD          4
#define FXRM_RELEASE       5
```

Return codes:

```
FXRET_OK        successful api function return
  or
API Error Code
```

Example programs:

RCV.C Receive mode example program (FxReceiveMode)

Recv_Log.C Receive from log example program

FxRegisterDialer

Description:

The FxRegisterDialer call allows you to provide your own dialer function. Whenever the PMfax program would normally pick up the line an dial a fax call, it will instead call your own dialer function. Your function can establish the line connection and then tell the PMfax program how it should proceed. This allows you to adapt the PMfax program to

work with special telephone line interfaces (T1, E1, MVIP, etc.) or other situations which require special processing for establishing telephone connections. The program calling this function must be on the same system as the fax server (PMfax) program.

```
/* DIALERS:
 * A function can be registered to receive dial information
 * and can then control the dialing process.  This can be used
 * with MVIP setups for handling line connection and dialing or
 * other line connection or fax delivery custom requirements.
 * The following is the callback function template to receive
 * dial information.  The function is called whenever a fax is
 * about to be sent.  Only ONE dialer can be registered at a
 * time.
 */
typedef FXRET (FXENTRY FXDIALER)(
    ULONG tag,          /* fax-being-dialed tag (0 if no tag) */
    PCHAR cover,        /* prepared cover page (NULL if none) */
    PCHAR header,       /* prepared page headers (NULL if none) */
    PCHAR file,         /* fax file */
    ULONG line,         /* line number being used */
    PCHAR fax,          /* fax number being dialed */
    PCHAR logline );    /* line from Fax.Log file */
    /* returns from application supplied dialer:
     *    FXRET_OK         line connected, proceed with dial
     *    FXRET_DIALED     line dialed, proceed with call progress
     *    FXRET_ACTIVE     fax connected, proceed as current call
     *    FXRET_SENT       fax connected and document delivered
     *  or any of the FXRET_ fax send failures (BUSY, USER_KILLED, ...)
     */
typedef FXDIALER FXPTR PFXDIALER;


FXRET FXENTRY FxRegisterDialer(
    PFXDIALER func ); /* nonNULL=dialer function, NULL=deregister */
```

Return codes:

```
FXRET_OK            successful api function return
FXRET_SERVER_BUSY   in use - can't register the function now
  or
API Error Code
```

Example programs:

DLR.C Dialer callback function example program (FxRegisterDialer)

FxRegisterNotifier

Description:

If you are using a LAN version of PMfax as your fax server (in its "private mode", with the PMfax and FxRdr programs), you can register your own notification function. Your function is called whenever the PMfax LAN version would normally notify the LAN user of a sent, received or routed fax event, and your function can choose to use normal notification, handle the notification itself, or take full responsibility for the fax document and remove it from the system.

This can be used to provide customized notification systems (such as E-mail or message lights) or fax delivery systems (such as sending the entire fax via E-mail). The program calling this function must be on the same system as the FxRdr program.

IMPORTANT: Each thread of a program needs its own session handle from FxOpen.

```
/* NOTIFIERS:
 * A function can be registered to receive notify information
 * whenever a fax is routed.  This can be used to setup custom
 * notification systems (such as E-mail or message lights).
 * The following is the callback function template to receive
 * user information.  The function is called whenever the
 * user's log file has been updated with final send status or
 * a fax has been received or routed.  (FXRET_ROUTE status
 * indicates a user-to-user route, other statuses are the status
 * from the log record, such as: FXRET_RECEIVED, FXRET_BADRCV,
 * or even FXRET_EDIT or FXRET_PRINT for API controlled routes).
 * The function can return FXRET_DELETE to cause the router to
 * remove the fax from the system (This could be used if the
 * E-mail notifier sent the entire image via mail).
 * Only ONE notifier can be registered at a time.
 * (FxLogParse can be used to further parse the logline.)
 */
typedef FXRET (FXENTRY FXNOTIFIER)(
    PCHAR userid,   /* user id of user being notified */
    FXRET status,   /* FXRET_* status */
    PCHAR file,     /* fax file */
    PCHAR logline );/* line from Fax.Log file */
    /* return:
        FXRET_OK go ahead and do default notification,
        FXRET_SENT notification sent,
        FXRET_DELETE to remove from log and delete file */
typedef FXNOTIFIER FXPTR PFXNOTIFIER;


FXRET FXENTRY FxRegisterNotifier(
    PFXNOTIFIER func ); /* nonNULL=notifier function, NULL=deregister */
```

Return codes:

```
FXRET_OK            successful api function return
FXRET_SERVER_BUSY   in use - can't register the function now
  or
API Error Code
```

Example programs:

NFY.C Notifier callback function example program (FxRegisterNotifier)

FxRegisterReporter

Description:

You can register your own reporter function by using the FxRegisterReporter call. Your reporter function is called

whenever a fax is sent or received. This can be used to control displays or provide your own special processing of sent or received fax documents. For example, your reporter function could process the received fax documents and then tell the fax server to remove the fax document from the system. The program calling this function must be on the same system as the fax server (PMfax) program.

IMPORTANT: Each thread of a program needs its own session handle from FxOpen.

This includes registered functions such as your registered reporter function.

```
/* REPORTERS:
 * A function can be registered to receive status information
 * whenever a fax is sent or received.  This can be used to
 * control displays and database applications for custom setups.
 * The following is the callback function template to receive
 * fax report information.  The function is called whenever a
 * a fax has been sent or received.  The function can return
 * FXRET_DELETE to cause the server to remove the fax from the
 * system.
 * Only ONE reporter can be registered at a time.
 * (FxLogParse can be used to further parse the logline.)
 */
typedef FXRET (FXENTRY FXREPORT)(
    ULONG tag,       /* non0=tag */
    ULONG index,     /* log doc-id */
    ULONG line,      /* line number used */
    FXRET status,    /* FXRET_* status */
    PCHAR logline ); /* line from Fax.Log file */
    /* return FXRET_OK, FXRET_DELETE to remove from log and delete file */
typedef FXREPORT FXPTR PFXREPORT;


FXRET FXENTRY FxRegisterReporter(
    PFXREPORT func ); /* nonNULL=report function, NULL=deregister */
```

Return codes:

```
FXRET_OK            successful api function return
FXRET_SERVER_BUSY   in use - can't register the function now
  or
API Error Code
```

Example programs:

RPT.C Reporter callback function example program (FxRegisterReporter)

FxRegisterRouter

Description:

If you are using a LAN version of PMfax as your fax server (in its "private mode", with the PMfax and FxRdr programs), you can register your own routing function. Your routing function will be called when a fax is received and the PMfax LAN redirector program (FxRdr) is running and FxRdr cannot automatically route the fax, and your function can then tell the redirector program how to route the fax document. This allows you to extend the features of the LAN version to include additional automatic routing modes, such as OCR routing, routing based on line number, etc. The program calling this function must be on the same system as the FxRdr program.

Note: If FxRdr can find a matching ID value (from DID, DTMF, T.30 Subaddress, etc.) in the WS.INI file, then it automatically routes the fax to that user and your registered routing function is not called. If you are using such auto-routing features and wish to process all received fax documents, then you should remove the ID values from the WS.INI file and manage them with your own function.

IMPORTANT: Each thread of a program needs its own session handle from FxOpen.

```
/* ROUTERS:
 * A function can be registered to receive route information
 * when a fax is received and cannot be auto-routed with normal
 * means.  This can be used for OCR routing, line number
 * routing, etc.  The following is the callback function template
 * to receive fax route information.  The function
 * copies a user id (as setup by the redirector) to the
 * specified pointer to cause a route, otherwise does nothing
 * for default route (to the fax administrator).
 * Only ONE router can be registered at a time.
 * (FxLogParse can be used to further parse the logline.)
 */
typedef FXRET (FXENTRY FXROUTER)(
    PCHAR file,     /* fax file being routed */
    PCHAR logline,  /* line from Fax.Log file */
    PCHAR userid ); /* ""=default route, else return user id to route to */
    /* return FXRET_OK */
typedef FXROUTER FXPTR PFXROUTER;

FXRET FXENTRY FxRegisterRouter(
    PFXROUTER func ); /* nonNULL=router function, NULL=deregister */
```

Return codes:

```
FXRET_OK            successful api function return
FXRET_SERVER_BUSY   in use - can't register the function now
  or
API Error Code
```

Example programs:

RTE.C Router callback function example program (FxRegisterRouter)

FxRoute

Description:

If you are using a LAN version of PMfax as your fax server (in its "private mode", with the PMfax and FxRdr programs), you can use FxRoute to tell the FxRdr program to route fax documents to users of the LAN fax system.

```
FXRET FXENTRY FxRoute(
    FXHANDLE handle,
    ULONG tag,      /* tag of fax to route */
    PCHAR userid,   /* redirector user id to route to */
    PCHAR notes,    /* nonNULL=info for user's log notes field */
    BOOL32 keep );  /* TRUE=keep an entry server's log */
```

Return codes:

```
FXRET_OK        successful api function return
  or
API Error Code
```

Example programs:

FxSend

Description:

After calling FxOpen to establish a session with the PMfax program (the fax server), you can call FxSend to send a fax. Be sure to read the section called "Sending" in Chapter 1 for a discussion of the power of the FxSend call. The call supports "spooled", "direct" or "current call direct" modes of sending. You can return immediately and use FxStatus to monitor status, or you can wait for the fax send operation to complete before returning. If you wait, you can provide a callback function that will be called as status information changes. You can select any-line or specified-line sending.

```
FXRET FXENTRY FxSend(
    FXHANDLE handle,
    PCHAR tifff,    /* TIFF class-F file to send (index=0) */
    ULONG index,    /* log doc-id to send (tifff=NULL) */
    PCHAR number,   /* NULL=current call (direct only), else fax number */
    LONG tries,     /* 0=spooled send with server default retries,
                       +n=spooled send (n=1st+retries),
                       -1=direct send on standby line */
    LONG line,      /* 0=spooled send on anyline,
                       +n=spooled send on specific line
                       +n=direct send on specific line */
    PDATETIME dt,   /* NULL=now, else spooled send future date/time */
    PFXINFO info,   /* NULL=no cover sheet/page headers, else specified
                       (info!=NULL and index=0 and tifff=NULL is cover only)*/
    PCHAR localid,  /* NULL=server default, else specified TSI */
    PULONG tag,     /* nonNULL: return assigned tag */
    PCHAR remoteid, /* nonNULL: return CSI (20+1 characters) */
    PFXSTATUS func, /* NULL=no callback, else called for status (wait=TRUE) */
    PVOID apparg,   /* application defined argument to pass to func */
    BOOL32 wait );  /* FALSE=no-wait, TRUE=wait for completion */


/* cover sheet modes (for cover_flag value in FXINFO structure) */
#define FXCS_OFF           0
#define FXCS_ON            1
#define FXCS_ON_FULLSIZE   2

/* page header modes (for heading_flag value in FXINFO structure) */
#define FXPH_OFF           0
#define FXPH_ON            1

/* char array lengths */
#define FXLEN_LOGLINE      4096
#define FXLEN_ID           (20+1)
#define FXLEN_TEXT         (40+1)
#define FXLEN_HEADER       (80+1)
#define FXLEN_PATH         (256+1)
```

```
#define FXLEN_COMMENT      (1000+1)

/* FXINFO structure is used by FxSend to specify non-default
 * cover sheet information (and by FxLogParse to retrieve it).
 */
typedef struct _FXINFO {
    /* cover sheet */
    ULONG cover_flag;
    CHAR cover_bitmap[ FXLEN_TEXT ];   /* "*" = server default */
    CHAR name[ FXLEN_TEXT ];
    CHAR company[ FXLEN_TEXT ];
    CHAR from_name[ FXLEN_TEXT ];      /* "*" = server default */
    CHAR from_company[ FXLEN_TEXT ];   /* "*" = server default */
    CHAR from_phone[ FXLEN_TEXT ];     /* "*" = server default */
    CHAR from_fax[ FXLEN_TEXT ];       /* "*" = server default */
    CHAR comment[ FXLEN_COMMENT ];     /* "*" = server default */
    /* page heading */
    ULONG heading_flag;
    CHAR heading[ FXLEN_TEXT ];        /* "*" = server default */
    /* log notes field */
    CHAR notes[ FXLEN_TEXT ];
    /* owner id field (intermediate and final status routed to this user */
    CHAR owner_id[ FXLEN_TEXT ];
    BOOL32 priority;                   /* FALSE=normal, TRUE=high */
} FXINFO;
typedef FXINFO FXPTR PFXINFO;

/* FXSTATUS callback function template is used by FxSend and
 * FxReceive to specify a function to receive active fax status.
 * The function is called when fax is spooled, fax activity starts,
 * at each page transition, and at completion.  The status function
 * can only be used on 'wait' type calls.  For no-wait, the same
 * information can be polled with the FxStatus function or
 * completion will be indicated by a call to a registered FXREPORT
 * function.
 */
typedef FXRET (FXENTRY FXSTATUS)(
    PVOID apparg,      /* application defined argument */
    ULONG tag,         /* fax-being-statused tag */
    ULONG index,       /* log doc-id number */
    ULONG line,        /* line number being used */
    ULONG pageno,      /* 0=pre- or post-fax, n=page faxing */
    BOOL32 final,      /* TRUE=final status of fax (no more updates) */
    PCHAR remoteid,    /* CSI/TSI (20+1 characters) (post-fax) */
    FXRET retcode );
    /* retcode FXRET_SPOOLED, ACTIVE, SENT, RECEIVED or fax error or error */
    /* returns from application supplied status function:
     *    FXRET_OK         proceed
     * FXRET_USER_KILLED   terminate activity
     */
typedef FXSTATUS FXPTR PFXSTATUS;
```

Return codes:

```
If wait = TRUE:


FXRET_SENT        successful send
  or
Fax Error Code
  or
API Error Code


If wait = FALSE:


FXRET_OK        successful api function return
  or
API Error Code
```

Example programs:

Send_Dir.C Direct send example program

Send_Spl.C Spooled send example program

MegaSend.C Multi-server send example program

FxServerStatus

Description:

FxServerStatus returns information about the overall status of a fax server, including the number and configuration of its fax lines and the status of its fax jobs. Your application can use this information to make intelligent decisions about balancing the load between multiple fax servers on a LAN, to decide how to best use the available fax lines, to reset the status windows counts on the fax server's screen, or for other purposes.

The total number of lines in the server (not counting any lines that have been set to "off" mode) is equal to the sum of send_lines, receive_lines, sendrecv_lines and standby_lines.

```
FXRET FXENTRY FxServerStatus(
    FXHANDLE handle,
    BOOL32 reset,    /* TRUE=reset status window counters after call      */
    PULONG pending,  /* nonNULL: return number of faxes in 'Spool' state  */
    PULONG sent,     /* nonNULL: return 'sent' count on status window     */
    PULONG received, /* nonNULL: return 'received' count on status window */
    PULONG errors,   /* nonNULL: return 'error' count on status window    */
    PULONG send_lines,    /* nonNULL: return number of lines set for send   */
    PULONG receive_lines, /* nonNULL: return number of lines set for receive*/
    PULONG sendrecv_lines,/* nonNULL: return number of lines set for both   */
    PULONG standby_lines, /* nonNULL: return number of lines set for standby*/
    PBOOL32 receive_on ); /* nonNULL: return TRUE if receive-all-calls on   */
```

Return codes:

```
FXRET_OK        successful api function return
  or
API Error Code
```

Example programs:

SRV.C Server status example program (FxServerStatus)

MegaSend.C Multi-server send example program

FxStatus

Description:

If you call FxSend with wait=FALSE, you can later call FxStatus to get the current status of your job. You use the tag value that was returned by the FxSend call to identify the job. You can repeatedly call FxStatus to monitor status changes, and since you use the tag value to identify the job, you can monitor multiple jobs from a single thread.

```
FXRET FXENTRY FxStatus(
    FXHANDLE handle,
    ULONG tag,      /* fax-to-check's tag */
    PULONG index,   /* nonNULL: return log doc-id */
    PULONG line,    /* nonNULL: return line number used (0 if pre-fax) */
    PULONG pageno,  /* nonNULL: return 0=pre- or post-fax, n=page faxing */
    PBOOL32 final,  /* nonNULL: return TRUE=final status (no more updates) */
    PCHAR remoteid);/* nonNULL: return CSI/TSI (20+1 characters) (post-fax) */
```

Return codes:

```
FXRET_SENT       send is completed
FXRET_RECEIVED   receive is completed
FXRET_SPOOLED    spooled, but not yet active
FXRET_ACTIVE     active, currently in process
  or
Fax Error Code
  or
API Error Code
```

Example programs:

Send_Spl.C Spooled send example program

FxTagToIndex

Description:

FxTagToIndex obtains the document index value that is associated with a job tag. You can then use the index value for other API calls which manipulate fax documents.

```
FXRET FXENTRY FxTagToIndex(
    FXHANDLE handle,
    ULONG tag,      /* tag of a fax */
    PULONG index ); /* return log doc-id of fax */
```

Return codes:

```
FXRET_OK        successful api function return
  or
API Error Code
```

Example programs:

FxTextToFax

Description:

The FxTextToFax call will convert a text file to a fax file using the FxPrint printer driver. The FxPrint printer driver must be installed on your system. The result is written to a fax file which you specify.

The text may include printer driver commands (>> commands) to control the font, page layout and other features of the resulting fax document. These printer driver commands are documented in the PMfax Reference Manual and the Enhanced Printer Driver Toolkit Developer's Manual.

The current printer queue settings are used, so long lines will wrap if the printer is configured for >>FONT emulation and will truncate if in other emulations (PCL5 or Proprinter).

The queue is the print queue name, as shown as the "Physical Name" on the first page of a printer object's Settings notebook, and defaults to FxPrint if no queue is specified. This command can actually be used to do a "print to file" operation through any queue (not just those using FxPrint.DRV).

You can produce the same fax file by copying the text file to the LPT port which is attached to a suitably-configured FxPrint printer object and including the >>FILE= command in the text, but that uses the OS/2 spooler which makes it difficult for your program to know when the file has been completed. With the FxTextToFax call, your program knows that the file is available as soon as the call returns.

```
FXRET FXENTRY FxTextToFax(
    FXHANDLE handle,
    PCHAR textf,        /* input text file */
    PCHAR tifff,        /* output TIFF class-F file */
    PCHAR queue );      /* NULL=use FxPrint, else specify queue name */
```

Return codes:

```
FXRET_OK        successful api function return
  or
API Error Code
```

Example programs:

FxWakeupReceive

Description:

FxWakeupReceive is used from a thread in your application to force another thread to return from a "direct receive" call to FxReceive. In other words, if you've called FxReceive and told it to wait to receive a fax call, you can force it out of its wait state by calling FxWakeupReceive. The waiting FxReceive call that you "wake up" will return a FXRET_WAKEUP return value.

```
FXRET FXENTRY FxWakeupReceive(
    FXHANDLE handle );
```

Return codes:

```
FXRET_OK        successful api function return
  or
API Error Code
```

Example programs:

Chapter 4 - Dialog Boxes for 32-bit PM Programs

The calls in this section provide pop-up dialog boxes for entering cover sheet and header line information and selecting phone numbers from PMfax phone books. The pop-up dialog boxes are like the "Send Fax" and "Phone Book/Phone Book Edit" dialog boxes in the PMfax program.

Your application can call the Phone Book pop-up to allow the user to select recipients from phone book files. The user can also add and edit the phone book records through the dialog boxes. The selected destinations are returned to your application as a linked list.

Your application can call the Send Fax pop-up to allow the user to enter or confirm the information for the cover sheet, header line and date/time of transmission. The user can also confirm the destination list, or use the Phone book or Manual buttons to specify additional destinations. Using the Phone book button gets the user into the Phone Book pop-up (like in the PMfax product). If desired, you can disable the Phone book button if you only want to allow manual entry of recipient information.

The PMfax phone books intentionally use a simple format, so in addition to using the PMfax product for building or maintaining phone books, you can directly manipulate phone book files either manually (with a text editor) or with your application program. The phone book format is documented in an Documents area of the Keller Group web site, but this information is also repeated below for your convenience.

Phone Book File Format

Any file in the fax directory that has the extension .PBK is assumed to be a phone book file and will automatically appear in the Phone book pull-down list in the Phone book dialog box.

A phone book file is stored in "Comma-separated values" (CSV) text file format. This format can be imported and exported by various applications. For example, Excel 3.0 can open phone book files (use the Text button on Excel's Open dialog box to specify a "Comma" column delimiter) and write phone book files (use the Options button on Excel's Save As dialog box to specify a file format of CSV). Since the phone book files contain only ASCII text, you can also create and edit them with a text editor.

Each phone book entry is a line in the file. Each line consists of four elements representing the Name, Company, Fax number and Group. Elements are separated by commas. If the element contains comma characters or double quotation marks, the element is enclosed in double quotation marks. Double quotation marks within quoted elements are doubled. Examples of simple phone book lines are shown below. Newer versions of the program may be adding additional elements to the end of the line, so examine your current phone book file to determine the use of additional (optional) fields:

```
Version 1.x phone book record elements:

name, company, faxnumber, group


Initial version 2.x phone book record elements:

lastname, company, faxnumber, group, firstname, voicenumber

Note that you can place the entire name in the first element (as was done in
version 1.x), or split the name between the lastname and firstname elements. Some
examples:

Jim Smith,,1 123 555-5555
```

```
Jim Smith,JimCo Inc.,1 (123) 555-5555,Group1
Smith,,1 (123) 555-5555,Group1,Jim,123/555-5556
Smith,"JimCo, Inc.",1 123/555-5555,Group1,Jim
"Jim ""JimBo"" Smith",,1 (123) 555-5555,Group1,,(123) 555-5556
"Smith","JimCo","1 (123) 555-5555","Group1","Jim","x345"
```

FxWinSend

Description:

This is a 32-bit call for use from Presentation Manager programs.

The FxWinSend call displays a standard "Fax Send" dialog box. Your application can use this to have the user enter or confirm the information for the cover sheet, header line and date/time of transmission. The user can also confirm the destination list, or use the Phone book or Manual buttons to specify additional destinations. Using the Phone book button gets the user into the Phone Book pop-up (like in the PMfax product, or like calling the FxWinPhoneBook call from your application). If desired, you can disable the Phone book button if you only want to allow manual entry of recipient information.

If a *pbk_dir* is specified, the user can select any of the phone books in the specified directory in addition to the phone books in the fax server's log directory (i.e., all the phone books from both directories will show up in the pull-down list on the Phone Book screen). You can use this feature to augment the standard phone books with additional user or application phone books.

If *default_info* is true, the fax server's default cover sheet and header line values will be automatically used as the default values for the fields on the Fax Send dialog box. The default values will be displayed in the fields if the fax server is local, or asterisk ('*') values will be displayed in the fields if the fax server is remote (across the LAN).

The caller is responsible for freeing the "to" list (the list of phone book selections) using the FxToFree call. See the FxWinPhoneBook documentation and the Send_Pop.C example program for details on the phone book list.

```
FXRET FXENTRY FxWinSend(    /* create 'to' list from send popup */
    HAB hab,
    HWND hwnd,
    FXHANDLE handle,
    PCHAR pbk_dir,          /* 2nd pbk directory (with log dir) */
    PDATETIME dt,           /* date/time to/from send screen */
    PFXINFO info,           /* cover sheet info to/from send screen */
    ULONG pages,            /* number of pages in fax for send scrn */
    BOOL default_info,      /* TRUE=display default send info */
    BOOL disable_pbk,       /* TRUE=no phonebook access from screen */
    PFXTO FXPTR to );       /* linked list of destinations */
```

Return codes:

```
FXRET_OK            successful api function return
  or
FXRET_USER_KILLED   user pressed the Cancel button
  or
API Error Code
```

Example programs:

Send_Pop.C PM example - Send/Phone book popups

FxWinPhoneBook

Description:

This is a 32-bit call for use from Presentation Manager programs.

The FxWinPhoneBook call displays the PMfax Phone Book dialog box to allow the user to select destinations from phone book files. The user can also add and edit the phone book records through the dialog boxes. The destinations that the user selects are returned to your application as a linked list.

If a *pbk_dir* is specified, the user can select any of the phone books in the specified directory in addition to the phone books in the fax server's log directory (i.e., all the phone books from both directories will show up in the pull-down phone book list on the Phone Book screen). You can use this to augment the shared phone books with additional user or application phone books.

The caller is responsible for freeing the "to" list (the list of phone book selections) using the FxToFree call (see the Send_Pop.C example program).

```
FXRET FXENTRY FxWinPhoneBook(   /* create 'to' list from phonebook dlg */
    HAB hab,
    HWND hwnd,
    FXHANDLE handle,
    PCHAR pbk_dir,              /* 2nd pbk directory (with log dir) */
    PFXTO FXPTR to );           /* linked list of destinations */


typedef struct _FXTO {
    struct _FXTO FXPTR next;
    CHAR name[ FXLEN_TEXT ];
    CHAR company[ FXLEN_TEXT ];
    CHAR fax[ FXLEN_TEXT ];
} FXTO;
typedef FXTO FXPTR PFXTO;


PFXTO FXENTRY FxToAlloc(        /* allocate one 'to' and link to end */
    PFXTO FXPTR to,
    PCHAR name,
    PCHAR company,
    PCHAR fax );
    /* returns pointer to 'to' structure or NULL on error */


VOID FXENTRY FxToFree(          /* free to list */
    PFXTO to );
    /* no return value */
```

Return codes:

```
FXRET_OK               successful api function return
or FXRET_USER_KILLED   user pressed the Cancel button
or API Error Code
```

Example programs:

Send_Pop.C PM example - Send/Phone book popups

CDS Inc.
P.O. Box 25123
Woodbury, MN, USA 55125
(651) 730-4156
FAX: (651) 730-4161
sales@cds-inc.com
http://www.cds-inc.com/