



Getting Started:

[Testing Your Installation](#)

[The FaxMan Overview](#)

In General :

[FaxMan Server](#)

[Configuring the FaxMan Fax Server](#)

[Using the FaxMan Printer Driver](#)

[Creating Fax files for use with FaxMan](#)

[Banner & Coverpage Format Characters](#)

DLL Users :

[The DLL Tutorial](#)

[Message Reference](#)

[DLL Interface](#)

VBX / OCX Users:

[Using the VBX / OCX Custom Controls](#)

[VBX / OCX Interface](#)

Other Information:

[Distributing FaxMan with your Application](#)

[Troubleshooting Fax Problems](#)

[Common Problems, Symptoms and Fixes](#)

[Obtaining Technical Support](#)

[About FaxMan Help](#)

FaxMan Custom Control

[Properties](#) [Events](#)

Description

The FaxMan VBX / OCX control provides complete support for controlling the FaxMan server from your Visual Basic, Visual C++ and PowerBuilder applications.

[How to use the FaxMan VBX / OCX](#)

FaxMan On-line help
V 1.06
January 1996

FaxMan is a trademark of Data Techniques, Inc.

Troubleshooting Fax Problems

Using Windows For Workgroups and 16550A UARTS

When using Windows for Workgroups 3.11 system stability can be greatly enhanced by adding the proper settings to the windows system.ini file.

Be sure to add the following entries to the [386 Enh] section of the system.ini file:

COMxFifo = 2
COMxTXFifo = 1

where x is the com port number of your modem(s).

Using the most recent release of the serial.386 file can also increase system reliability. At this time the proper serial.386 file is dated 2-17-94 and is 10620 bytes. This file is available from Microsoft and is available on CompuServe in the Microsoft file libraries.

See the sections [Common Problems, Symptoms and Fixes](#) and [Technical Report Requests](#) for more information on troubleshooting, common problems and how to receive the fastest technical support.

Using the FaxMan Printer Driver

How it works

When an application begins to print to the FaxMan driver, the driver will look for the application associated with the driver. If the application is not running then the driver will attempt to launch it. Once the driver has determined the application is running it will send a message to the application to get the name of the fax file to be created. The application must supply a name otherwise the driver will attempt to locate a filename in its win.ini section. If the application doesn't respond and the filename isn't in the win.ini section then the driver will display a dialog box and terminate.

Assuming the driver obtained a filename, the print job will continue and upon completion the driver will send a message to the application signalling the end of the print job. At this point the application may display a dialog or send the fax.

[Overview - Print Driver](#)

[How to install the FaxMan Printer Driver](#)

How to install the Printer driver

Installing the printer driver requires the following steps:

1. Copy the driver files, fmfaxdrv.drv & im10xtif.del, to the windows system directory.
2. Decide on the name you wish to use for the driver and your application ID and add the following entries to the win.ini file:

In the [PrinterPorts] section add:

<Your Driver Name> = FMFAXDRV, <AppID>, 1, 1

In the [Devices] section add:

<Your Driver Name> =FMFAXDRV, <AppID>

where <AppID> is your application's ID string and <Your Driver Name> is the name you wish to use to identify the driver.

3. Add the following section to win.ini:
[FaxMan, AppID]
Application = <your application's executable path and filename>
4. Add the following line to the [Ports] section of win.ini:
<AppID>=
5. Restart Windows.

The printer driver should appear in applications and the control panel as: Your Driver Name on AppID.

Your application must use the same AppID as the driver for the driver to be able to send the printing notifications to your application. If you want the driver to create a fax file without your application running you'll need to add the fax filename to the win.ini file.

In the [FaxMan, AppID] section add the following line:

Filename = <output filename>

Distributing FaxMan with your Application

Required Files

The following files are required for applications that utilize the FaxMan Server:

faxman.exe
faxdll.dll
im10xfax.del
im10fax.dil
im10tif.dil
im10bmp.dil
im10pcx.dil
compobj.dll
storage.dll
class2.dat
class20.dat
class1.dat

faxman1.vbx - Only required for apps that use the FaxMan VBX control

fmfaxdrv.drv - Only required for apps that use the printer driver

All of these files should be installed in the Windows/System directory to ensure that they are available at application run time. All files have version resource information to allow your installation to be sure of installing the latest versions.

FaxMan Technical Support

You may obtain technical support for FaxMan by Phone, FAX, Internet email, CompuServe and our own StarMan Bulletin Board system. For communication problems with FaxMan you'll need all of the information listed in the [Technical Report Request](#) section, you should also consult the [Common Problems](#) section. For the fastest response please provide this information in your electronic communications or be ready to provide this information to telephone support.

Data Techniques, Inc.
300 Pensacola Road
Burnsville, NC 28714

Tech Support: 704-682-4111 9am to 5pm EST
Fax: 704-682-0025
BBS: 704-682-4356 (2400-28.8K)
CompuServe: GO DATATECH
Cserve ID: 74431,1412
Internet: support@data-tech.com
Web: www.data-tech.com

Custom Properties

| | | |
|---|--|--|
| <u>Action</u> | <u>AddDevice</u> | <u>AppName</u> |
| <u>ColName</u> | <u>ColWidth</u> | <u>DeviceCount</u> |
| <u>DeviceFlags</u> | <u>DeviceInit</u> | <u>DeviceReset</u> |
| <u>Devices</u> | <u>FaxBanner</u> | <u>FaxComments</u> |
| <u>FaxCompany</u> | <u>FaxCoverPage</u> | <u>FaxDate</u> |
| <u>FaxFiles</u> | <u>FaxID</u> | <u>FaxLogID</u> |
| <u>FaxName</u> | <u>FaxNumber</u> | <u>FaxResolution</u> |
| <u>FaxRetries</u> | <u>FaxRetryInterval</u> | <u>FaxSubject</u> |
| <u>FaxTime</u> | <u>FaxUserData</u> | <u>InputFiles</u> |
| <u>Log</u> | <u>LogComments</u> | <u>LogCompany</u> |
| <u>LogCoverPage</u> | <u>LogDate</u> | <u>LogDuration</u> |
| <u>LogEntries</u> | <u>LogFiles</u> | <u>LogFrom</u> |
| <u>LogHangupCode</u> | <u>LogID</u> | <u>LogIndex</u> |
| <u>LogNumber</u> | <u>LogPages</u> | <u>LogPagesSent</u> |
| <u>LogRemoteID</u> | <u>LogResolution</u> | <u>LogRetries</u> |
| <u>LogSelectedRow</u> | <u>LogSpeed</u> | <u>LogStatus</u> |
| <u>LogSubject</u> | <u>LogTime</u> | <u>LogToCompany</u> |
| <u>LogToName</u> | <u>LogUserData</u> | <u>RemoveDevice</u> |
| <u>ServerOption</u> | <u>ServerOptionSetting</u> | <u>Status</u> |
| <u>StatusConnectSpeed</u> | <u>StatusDate</u> | <u>StatusDestination</u> |
| <u>StatusFiles</u> | <u>StatusFrom</u> | <u>StatusID</u> |
| <u>StatusNumber</u> | <u>StatusPages</u> | <u>StatusPagesSent</u> |
| <u>StatusPercentage</u> | <u>StatusRecipient</u> | <u>StatusRemoteID</u> |
| <u>StatusResolution</u> | <u>StatusSubject</u> | <u>StatusTime</u> |
| <u>UserCompany</u> | <u>UserFaxNumber</u> | <u>UserName</u> |
| <u>UserVoiceNumber</u> | | |

Custom Events

[ColWidthChanged](#)

[FaxLogAdd](#)

[PrintComplete](#)

[Click](#)

[FaxReceived](#)

[PrintStart](#)

[DoubleClick](#)

[FaxStatus](#)

Action Property

[Examples](#)

Description

Setting this property causes the specified action to occur.

Usage

Visual Basic *FaxControl.Action* = setting%

Visual C++ *BOOL pFaxControl->SetNumProperty("Action", ISetting);*

Remarks

The Action property settings are:

| Setting | Description |
|---------|--|
| 0 | Send a Fax. |
| 1 | Enumerate fax devices |
| 2 | Cancel a fax. |
| 3 | Delete a Fax from the Log file |
| 4 | Create FMF Files from files set with InputFiles Property |
| 5 | Close the FaxServer and Remove from memory |

Note

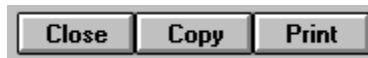
This property is write-only. When a log entry is deleted the associated internal File List is traversed, and all files matching *.FMF are deleted from the disk.

Access

Write-only

Data Type

Integer (Enumerated)



Action Property Example

' Send a Fax

'To create and send a the fax file fax.tif the following commands are required :

```
FaxMan1.FaxFiles = "c:\fax.tif"  
FaxMan1.FaxNumber = "1-404-555-1212"  
FaxMan1.Action = 0
```

'Create A Fax File

'To create a fax file called output. fmf from the files fax.bmp and f1.tif and then send 'the file output.fmf, do the following:

```
FaxMan1.InputFiles = "c:\fax.bmp+c:\images\tif\f1.tif"  
FaxMan1.FaxFiles = "c:\output.fmf"  
FaxMan1.Action = 4  
FaxMan1.FaxNumber = "1-404-555-1212"  
FaxMan1.Action = 0
```

Declare Function AddDevice% Lib "faxman1.vbx" (ByVal nPort%)

Adds the device specified by nPort as a faxman fax device.

You should place the Function declaration in the General/Declarations section of your VB application

AppName Property

Description

Specifies the name of the application using the FaxMan control.

Usage

Visual Basic *FaxControl*.**AppName**[= ApplicationName\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("AppName", lpcstrAppName);*
 CString pFaxControl->GetStrProperty("AppName");

Remarks

Access

Read/write

Data Type

String

Click Event

Description

Occurs when the user presses and then releases the left mouse button over an item in the control.

Syntax

Sub ctlname_Click(Index as Integer)

Visual C++ *Class::OnClick*(UINT, int, CWnd*, LPVOID)

Remarks

The Index argument uniquely identifies a control in a control array. The [LogSelectedRow](#) property indicates which row the user selected.

ColName Property

[Examples](#)

Description

Specifies which data is displayed in the specified column.

Usage

Visual Basic *FaxControl.ColName*(Index%)[= setting\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("ColName", lpcstrDataName, iIndex);*
 CString pFaxControl->GetStrProperty("ColName", iIndex);

Remarks

The control supports up to 14 Columns therefore the allowable values for idx are 0 to 13.

Note

The available data names are:

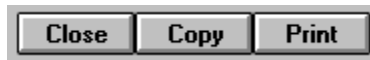
Date
Time
Pages
PagesSent
RemoteID
Status
Speed
Resolution
Duration
HangupCode
Fax ID
Subject
From
Phone

Access

Read/write

Data Type

String(Array)



ColName Property Example

```
' Set the first column to display the Date  
FaxMan1.ColName(0) = "Date"
```

ColWidth Property

Description

Specifies the width of the specified column in characters.

Usage

Visual Basic *FaxControl.ColWidth*(Index%)[= setting%]

Visual C++ *BOOL pFaxControl->SetNumProperty("ColWidth", lSetting, iIndex);*
 long pFaxControl->GetNumProperty("ColWidth", iIndex);

Remarks

To remove a column from the display, set the column width to zero.

Access

Read/write

Data Type

Integer(Array)

ColWidthChanged Event

Description

Occurs when the user resizes a column in the control.

Syntax

Sub ctlname_ColWidthChanged(Index as Integer, Column as Integer)

Visual C++ *Class::OnColWidthChanged*(UINT, int, CWnd*, LPVOID)

Remarks

The argument Index uniquely identifies a control in a control array. The Column argument specifies which column was resized.

Note

Refer to the ColWidth property to retrieve the new column width.

DeviceFlags Property

Description

Usage

Visual Basic *FaxControl.DeviceFlags(Index%)*

Visual C++ *long pFaxControl->GetNumProperty("DeviceFlags", iIndex);*

Remarks

The DeviceFlags property allows you to get the current status of the a particular device. The index number is the zero based index number of the Device wich can be found using the DeviceCount and Devices Properties.

Specifying the property with an array index will return the current status of the device. You can AND the return value with one or more of the status flags to determine the current status of the device.

Access

Read-only

Data Type

Integer(Array)

Devices Property

Description

Returns a description of each configured fax device including the port and modem type.

Usage

Visual Basic *FaxControl.Devices*(Index%)

Visual C++ CString *pFaxControl->GetStrProperty*("Devices", iIndex);

Remarks

Note

This property is Read-only.

Access

Read-only

Data Type

String(Array)

DeviceCount Property

Description

Returns the number of configured fax devices in the FaxMan server.

Usage

Visual Basic *FaxControl.DeviceCount*

Visual C++ `long pFaxControl->GetNumProperty("DeviceCount");`

Remarks

This property is Read-only. To add devices to the FaxMan system use either the AddDevice() function or set the Action property to a setting of 1 (ENUMERATE).

Access

Read-only

Data Type

Integer

DoubleClick Event

Description

Occurs when the user double clicks the left mouse button over a row in the control.

Syntax

Sub ctlname_DoubleClick(Index as Integer)

Visual C++ *Class::OnDoubleClick*(UINT, int, CWnd*, LPVOID)

Remarks

The argument Index uniquely identifies a control in a control array. The [LogSelectedRow](#) property specifies which row the user selected.

FaxBanner Property

Description

The FaxBanner is a line of text that may be added to the top of each fax page sent. The banner is intended to help the recipient keep the appropriate fax pages together by providing an identifier on each page. The FaxBanner may be formatted with [control characters](#) to determine information or spacing.

Usage

Visual Basic *FaxControl.FaxBanner* [= FaxBanner\$]

Visual C++ `BOOL pFaxControl->SetStrProperty("FaxBanner", lpcstrFaxBanner);`
 `CString pFaxControl->GetStrProperty("FaxBanner");`

Remarks

There is no requirement to have a Fax Banner, if none is set, then no banner is sent with the fax.

Access

Read/write

Data Type

String

FaxComments Property

Description

Specifies comments to be used when sending the fax. The comments may be displayed on the coverpage.

Usage

Visual Basic *FaxControl.FaxComments*[= Comment\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("FaxComments", lpcstrFaxComments);*
 CString pFaxControl->GetStrProperty("FaxComments");

Remarks

You may use this file in a variety of ways, including but not limited to using it for comments about a fax file, or transmission, entering comments on the fax cover page, and adding other fields to the FaxLog(using delimited strings).

Access

Read/write

Data Type

String

FaxCoverPage Property

Description

Specifies the coverpage to be used when sending a fax.

Usage

Visual Basic *FaxControl.FaxCoverPage*[= FaxCoverPageFileName\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("FaxCoverPage", lpcstrFaxCoverPageFileName);*
 CString pFaxControl->GetStrProperty("FaxCoverPage");

Remarks

This string specifies the coverpage file to be used when sending a fax, and must be in FaxMan coverpage format. The file name must include the complete path to the coverpage. If no Coverpage is specified then the fax will not include a coverpage.

Faxes which consist of just a coverpage can be sent by just scheduling a fax without any attached files. In this case the FaxComments field would be set to some valuable text.

Access

Read/write

Data Type

String

FaxDate Property

Description

Specifies the Date when a fax should be sent.

Usage

Visual Basic *FaxControl.FaxDate*[= setting\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("FaxDate", lpcstrDateString);*
 CString pFaxControl->GetStrProperty("FaxDate");

Remarks

This property defaults to the current day. Set this property to a date in the format of MM-DD-YY.

Note

Future releases of the control will support the date format specified in the control panel.

Access

Read/write

Data Type

String

FaxFiles Property

Description

Specifies the fax files to be sent or the name of a fax file to be created.

Usage

Visual Basic *FaxControl.FaxFiles*[= faxFiles\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("FaxFiles", lpcstrFileList);*
 CString pFaxControl->GetStrProperty("FaxFiles");

Remarks

When specifying files to send, multiple files may be sent in one fax by specifying multiple file names delimited with the plus sign, '+'. For instance:

```
FaxMan1.FaxFiles = "c:\fax1.fax+d:\faxes\fax2.fax"
```

Note

If the files to be sent are not in fax format an error will occur and the fax will not be scheduled.

Only one file name should be set for the creation of files, any additional file names will be ignored. If an error occurs in creation of the fax files an error code of 32005 will be generated.

Access

Read/write

Data Type

String

FaxID Property

Description

Specifies the Fax Identification string used when sending and receiving faxes.

Usage

Visual Basic *FaxControl.FaxID*[= setting\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("FaxID", lpcstrFaxID);*
 CString pFaxControl->GetStrProperty("FaxID");

Remarks

Access

Read/write

Data Type

String

FaxLogID Property

Description

After sending a fax, this returns the 'Fax ID' which uniquely identifies this fax job.

Usage

Visual Basic *FaxControl.FaxLogID* [= setting&]

Visual C++ *BOOL pFaxControl->SetNumProperty("FaxLogID", ISetting);*
 long pFaxControl->GetNumProperty("FaxLogID");

Remarks

To cancel a fax, set this property to the ID of the fax you wish to cancel, then cancel the fax by setting the [Action](#) property.

Access

Read/write

Data Type

Long

FaxNumber Property

Description

Specifies the phone number to which the fax should be sent.

Usage

Visual Basic *FaxControl.FaxNumber*[= setting\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("FaxNumber", lpcstrFaxNumber);*
 CString pFaxControl->GetStrProperty("FaxNumber");

Remarks

Access

Read/write

Data Type

String

FaxReceived Event

Description

Occurs when a new fax is received.

Syntax

Sub ctlname_FaxReceived(Index as Integer, RecvFileName as String)

Visual C++ *Class::OnFaxReceived*(UINT, int, CWnd*, LPVOID)

Remarks

The argument Index uniquely identifies a control in a control array. The RecvFileName argument specifies the name the file containing the fax.

Note

FaxName Property

Description

Specifies the name of the Fax recipient.

Usage

Visual Basic *FaxControl.FaxName* [= RecipientName\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("FaxName", lpcstrRecipientName);*
 CString pFaxControl->GetStrProperty("FaxName");

Remarks

Setting this property is optional.

Access

Read/write

Data Type

String

FaxResolution Property

Description

Specifies the resolution to be used when transmitting the fax..

Usage

Visual Basic *FaxControl.FaxResolution*[= setting%]

Visual C++ *BOOL pFaxControl->SetNumProperty("FaxResolution", lSetting);*
 long pFaxControl->GetNumProperty("FaxResolution");

Remarks

The allowable values for the FaxResolution property are:

Setting Description

| | |
|---|-----------------------------|
| 0 | Low Resolution (204 x 98) |
| 1 | High Resolution (204 x 196) |

Access

Read/write

Data Type

Integer(Enumerated)

FaxLogAdd Event

Description

Occurs when a fax is added to a log.

Syntax

Sub ctlname_FaxLogAdd(Index as Integer, LogType As Integer, FaxID As Long, LogIndex As Integer)

Visual C++ *Class::OnFaxLogAdd*(UINT, int, CWnd*, LPVOID)

Remarks

The argument Index uniquely identifies a control in a control array.

The LogType argument specifies the log into which the entry was added.

The FaxID argument specifies the fax job's ID.

The LogIndex specifies the index of the fax in the current log if the current log is the same log as the LogType parameter. If the value of LogIndex is -1 then the fax was added to a log that is not current.

Note

If the log into which the fax was added is not the current log, the FaxID property may be used to locate the fax once the VBX has changed to the proper log.

FaxStatus Event

Description

Occurs when the status of a fax device changes.

Syntax

Sub ctlname_FaxStatus(Index as Integer, Device As Integer, Status As Integer)

Visual C++ *Class::OnFaxStatus*(UINT, int, CWnd*, LPVOID)

Remarks

The argument Index uniquely identifies a control in a control array.

The Device argument specifies which device's status has changed.

The Status argument indicates the device's new status.

Note

FaxSubject Property

Description

Specifies the subject of the fax.

Usage

Visual Basic *FaxControl.FaxSubject*[= Subject\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("FaxSubject", lpcstrSubject);*
 CString pFaxControl->GetStrProperty("FaxSubject");

Remarks

Setting this property is optional.

Access

Read/write

Data Type

String

FaxTime Property

Description

Specifies the time at which the fax should be transmitted.

Usage

Visual Basic *FaxControl.FaxTime*[= setting\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("FaxTime", lpcstrFaxTime);*
 CString pFaxControl->GetStrProperty("FaxTime");

Remarks

The time should be specified in the format hh:mm where hh is 0-24 and mm is 0-59.

Access

Read/write

Data Type

String

InputFiles Property

Description

Specifies the file or files to be converted to a FaxMan FMF file ready to sent with FaxMan.

Usage

Visual Basic *FaxControl.InputFiles*[= file names\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("InputFiles", lpcstrFileNames);*
 CString pFaxControl->GetStrProperty("InputFiles");

Remarks

The input files should be valid BMP, TIF, PCX or ASCII text files. The FaxFiles Property should be set to the intended output file name prior to setting the Action Event to 4 to actually create the files. If the file creation fails a VB runtime error of 32005 will be created.

Access

Read/write

Data Type

String

Log Property

Description

Specifies the current FaxMan log.

Usage

Visual Basic *FaxControl.Log* [= setting%]

Visual C++ *BOOL pFaxControl->SetNumProperty("Log", ISetting);*
 long pFaxControl->GetNumProperty("Log");

Remarks

The allowable values for the Log property are:

Setting Description

| | |
|---|-------------|
| 0 | Pending Log |
| 1 | Sending Log |
| 2 | Sent Log |
| 3 | Failed Log |
| 4 | Receive Log |

Access

Read/write

Data Type

Integer(Enumerated)

LogComments Property

Description

Returns the comments that were associated with a sent fax.

Usage

Visual Basic *FaxControl.LogComments*(Index%)

Visual C++ CString *pFaxControl->GetStrProperty*("LogComments", *ilIndex*);

Remarks

The LogComments Property returns the Comments string that was used with a sent fax.

Access

Read-only

Data Type

String(Array)

LogCoverPage Property

Description

Returns the coverpage to used if any with a sent fax.

Usage

Visual Basic *FaxControl.LogCoverPage*(Index%)

Visual C++ *CString pFaxControl->GetStrProperty("LogCoverPage",iIndex);*

Remarks

The LogCoverPage property returns the coverpage that was used with the fax. If no coverpage was used a NULL string is returned.

Access

Read-only

Data Type

String(Array)

LogDate Property

Description

Returns the date the fax was sent.

Usage

Visual Basic *FaxControl.LogDate*(Index%)

Visual C++ CString *pFaxControl->GetStrProperty*("LogDate", *ilIndex*);

Remarks

This property is read--only.

Access

Read-only

Data Type

String(Array)

LogDuration Property

Description

Returns the duration of the fax transmission.

Usage

Visual Basic *FaxControl.LogDuration*(Index%)

Visual C++ *CString pFaxControl->GetStrProperty("LogDuration", iIndex);*

Remarks

This time is measured from the time the FCON is received till an FHNG is received.

Access

Read-only

Data Type

String(Array)

LogEntries Property

Description

Returns the number of entries in the current log.

Usage

Visual Basic *FaxControl.LogEntries*

Visual C++ `CString pFaxControl->GetStrProperty("LogEntries");`

Remarks

This property is Read-only.

Access

Read-only

Data Type

Integer

LogFiles Property

Description

Returns the names of the files sent.

Usage

Visual Basic *FaxControl.LogFiles*(Index%)

Visual C++ CString *pFaxControl->GetStrProperty*("LogFiles", *ilIndex*);

Remarks

Access

Read-only

Data Type

String(Array)

LogFrom Property

Description

Returns the sender's name of a fax.

Usage

Visual Basic *FaxControl.LogFrom*(Index%)

Visual C++ CString *pFaxControl->GetStrProperty*("LogFrom", *iIndex*);

Remarks

Access

Read-only

Data Type

String(Array)

LogHangupCode Property

Description

Returns the HangupCode returned by the faxmodem when the fax was transmitted.

Usage

Visual Basic *FaxControl.LogHangupCode*(Index%)

Visual C++ CString *pFaxControl->GetStrProperty*("LogHangupCode",*iIndex*);

Remarks

A HangupCode of zero (0) indicates a successful transmission. Any other value indicates an error.

[Hangup Status Codes](#)

Access

Read-only

Data Type

String(Array)

HangUp Codes

Code Description

0-9 Call Placement and Termination

| | |
|---|--|
| 0 | Normal and proper end of connection |
| 1 | Ring detected without successful handshake |
| 2 | Call aborted from either +FK or AN |
| 3 | No Loop Current |

10-19 Transmit Phase A & Misc. Errors

| | |
|----|-----------------------------|
| 10 | Unspecified Phase A error |
| 11 | No Answer (T.30 T1 timeout) |

20-39 Transmit Phase B Hangup Codes

40-49 Transmit Phase C Hangup Codes

| | |
|----|------------------------------------|
| 40 | Unspecified Phase C Transmit error |
| 43 | DTE to DCE data underflow |

50-69 Transmit Phase D Hangup Codes

| | |
|----|-------------------------------------|
| 50 | Unspecified Phase D transmit error |
| 51 | RSPREC error |
| 52 | No response to MPS repeated 3 times |
| 53 | Invalid response to MPS |
| 54 | No response to EOP repeated 3 times |
| 55 | Invalid response to EOM |
| 56 | No response to EOM repeated 3 times |
| 57 | Invalid response to EOM |
| 58 | Unable to continue after PIN or PIP |

70-89 Receive Phase B Hangup Codes

| | |
|----|---|
| 70 | Unspecified Phase B receive errors |
| 71 | RSPREC error |
| 72 | COMREC error |
| 73 | T.30 T2 timeout, expected page not received |
| 74 | T.30 T1 timeout, after EOM received |

90-99 Receive Phase C Hangup Codes

| | |
|----|-------------------------------------|
| 90 | Unspecified Phase C receive errors |
| 91 | Missing EOL after 5 seconds |
| 92 | Unused Code |
| 93 | DCE to DTE buffer overflow |
| 94 | Bad CRC or frame (ECM or BFT modes) |

100-119 Receive Phase D Hangup Codes

| | |
|-----|-------------------------------------|
| 100 | Unspecified Phase D receive errors |
| 101 | RSPREC invalid response received |
| 102 | COMREC invalid response received |
| 103 | Unable to continue after PIN or PIP |

120-255 Reserved Codes

LogID Property

Description

Returns the FaxMan ID of the fax.

Usage

Visual Basic *FaxControl.LogID*(Index%)

Visual C++ long *pFaxControl->GetNumProperty*("LogID",iIndex);

Remarks

The ID returned by this property is a unique number assigned by the FaxMan server when the fax is scheduled.

Access

Read-only

Data Type

Long(Array)

LogIndex Property

Description

Sets or Returns the index of the current log entry.

Usage

Visual Basic *FaxControl.LogIndex* [= Index%]

Visual C++ *BOOL pFaxControl->SetNumProperty("LogIndex", ISetting);*
 long pFaxControl->GetNumProperty("LogIndex");

Remarks

When using the FaxAction property to delete entries from a log, set this property to the index of the item to delete.

This property must be set to a value of zero to LogEntries-1.

Access

Read/write

Data Type

Integer

LogNumber Property

Description

Returns the phone number the fax was sent to.

Usage

Visual Basic *FaxControl.LogNumber*(Index%)

Visual C++ CString *pFaxControl->GetStrProperty*("LogNumber", *iIndex*);

Access

Read-only

Data Type

String(Array)

LogPages Property

Description

Returns the number of pages in the fax.

Usage

Visual Basic *FaxControl.LogPages*(Index%)

Visual C++ long *pFaxControl->GetNumProperty*("LogPages", iIndex);

Access

Read-only

Data Type

Integer(Array)

LogPagesSent Property

Description

Returns the number of pages sent.

Usage

Visual Basic *FaxControl.LogPagesSent*(Index%)

Visual C++ long *pFaxControl->GetNumProperty*("LogPagesSent", *ilIndex*);

Remarks

In the event of an error this property may be less than the LogPages property.

Access

Read-only

Data Type

Integer(Array)

LogRemoteID Property

Description

Returns the ID string of the remote Fax machine.

Usage

Visual Basic *FaxControl.LogRemoteID*(Index%)

Visual C++ CString *pFaxControl->GetStrProperty*("LogRemoteID", *ilIndex*);

Remarks

The maximum length of the remote ID is 20 characters.

Access

Read-only

Data Type

String(Array)

LogResolution Property

Description

Returns the resolution of the fax.

Usage

Visual Basic *FaxControl.LogResolution*(Index%)

Visual C++ *CString pFaxControl->GetStrProperty("LogResolution", iIndex);*

Remarks

The allowable values for the LogResolution property are:

0 - Low Resolution

1 - High Resolution

Access

Read-only

Data Type

String(Array)

LogRetries Property

Description

Returns the number of times transmission was attempted.

Usage

Visual Basic *FaxControl.LogRetries*(Index%)

Visual C++ long *pFaxControl->GetNumProperty*("LogRetries", *iIndex*);

Remarks

Note

Access

Read-only

Data Type

Integer(Array)

LogSelectedRow Property

Description

Returns the index of the row that was selected.

Usage

Visual Basic *FaxControl.LogSelectedRow*

Visual C++ long *pFaxControl->GetNumProperty("LogSelectedRow");*

Remarks

Note

Access

Read-only

Data Type

Integer

LogSpeed Property

Description

Returns the transmit speed of the fax

Usage

Visual Basic *FaxControl.LogSpeed*(Index%)

Visual C++ long *pFaxControl->GetNumProperty*("LogSpeed", *iIndex*);

Remarks

Access

Read-only

Data Type

String(Array)

LogStatus Property

Description

Returns the Status of the fax.

Usage

Visual Basic *FaxControl.LogStatus*(Index%)

Visual C++ CString *pFaxControl->GetStrProperty*("LogStatus", iIndex);

Remarks

The Status property may have the following values:

Completed

No Command Acknowledgement from Modem

Unsupported FaxModem

Error Initializing Modem

Bad FDIS Settings

Error Setting Local ID

Error Dialing

Error connecting to Remote fax

Bad FCSI string

Error receiving negotiated FDIS

Internal State Error

Line Busy

No Dialtone

Didn't get Connect message

User Cancelled

Bad or missing FPTs

Bad or missing FHNG

FDCS not found

General Modem Error

Invalid Fax Files

Incompatible DLL/Server releases

Access

Read-only

Data Type

String(Array)

LogSubject Property

Description

Returns the subject of the fax.

Usage

Visual Basic *FaxControl.LogSubject*[= setting%]

Visual C++ *CString pFaxControl->GetStrProperty("LogSubject", iIndex);*

Access

Read-only

Data Type

String(Array)

LogTime Property

Description

Returns the time the fax was sent.

Usage

Visual Basic *FaxControl.LogTime*(Index%)

Visual C++ CString *pFaxControl->GetStrProperty*("LogTime", *iIndex*);

Remarks

The time this property returns may not be the same as the time the fax was scheduled to be sent.

Access

Read-only

Data Type

String(Array)

PrintComplete Event

Description

Occurs when the printer driver completes a print job.

Syntax

Sub ctlname_PrintComplete(Index as Integer, Status as Integer)

Visual C++ *Class::OnPrintComplete*(UINT, int, CWnd*, LPVOID)

Remarks

The argument Index uniquely identifies a control in a control array. The Status argument returns the completion status of the print job.

Note

The Status argument may have the following values:

- 0 - Print job completed successfully
- 1 - Print job was aborted.

PrintStart Event

Description

Occurs when an application using the FaxMan printer driver starts a print job.

Syntax

Sub ctlname_PrintStart(Index as Integer, PrintFileName as String)

Visual C++ *Class::OnPrintStart*(UINT, int, CWnd*, LPVOID)

Remarks

The argument Index uniquely identifies a control in a control array. The PrintFileName argument is used to return a file name to the printer driver.

Note

The application should assign the name of the fax file to the PrintFileName argument. The file name should include a complete path.

If the application doesn't set the PrintFileName argument then the driver will attempt to get a file name from its section in the win.ini file.

In this release of the driver the application must not display a modal dialog box in this event. Displaying such a dialog will cause Windows to crash!

Declare Function RemoveDevice% Lib "faxman1.vbx" (ByVal nPort%)

Removes the port specified by nPort from the list of Faxmodems available for use by the faxman server.

Note: The declaration above should appear in your General/Declarations section in your VB project.

Status Property

Description

Returns the status of the specified fax device.

Usage

Visual Basic *FaxControl.Status*(Index%)

Visual C++ CString *pFaxControl->GetStrProperty*("Status", iIndex);

Remarks

This property is Read-only.

The returned values are:

"Error"
"Ok"
"Initializing Modem"
"Waiting to Send"
"Initializing Modem"
"Dialing"
"Waiting for Connect"
"Connected"
"Waiting for FCSI"
"Sending FCSI"
"Waiting for FDIS"
"Waiting for FDIS"
"Waiting to transfer page data"
"Negotiated parameters"
"Sending Page Data"
"End of Page"
"Port Closed"
"Aborting"
"Complete"
"Initializing modem for Receive"
"Waiting for call"
"Answering"
"Negotiating receive parameters"
"Receiving fax data"
"End of received page"

Access

Read-only

Data Type

String(Array)

StatusConnectSpeed Property

Description

Returns the transmit or receive speed if the device is in use.

Usage

Visual Basic *FaxControl.StatusConnectSpeed*(Index%)

Visual C++ long *pFaxControl->GetNumProperty*("StatusConnectSpeed", *ilIndex*);

Remarks

This property is Read-only.

Access

Read-only

Data Type

Long(Array)

StatusDate Property

Description

Returns the Date the current transmission began.

Usage

Visual Basic *FaxControl.StatusDate*(Index%)

Visual C++ CString *pFaxControl->GetStrProperty*("StatusDate", *iIndex*);

Remarks

This property is Read-only.

Access

Read-only

Data Type

String(Array)

StatusDestination Property

Description

Returns the Destination of the job currently being sent.

Usage

Visual Basic *FaxControl.StatusDestination*(Index%)

Visual C++ *CString pFaxControl->GetStrProperty("StatusDestination", iIndex);*

Remarks

This property is Read-only.

Access

Read-only

Data Type

String(Array)

StatusFiles Property

Description

Returns the name(s) of the file(s) being sent or received by the specified device.

Usage

Visual Basic *FaxControl.StatusFiles*(Index%)

Visual C++ *CString pFaxControl->GetStrProperty("StatusFiles", iIndex);*

Remarks

This property is Read-only.

Access

Read-only

Data Type

String(Array)

StatusFrom Property

Description

Returns the name of the sender for the fax being sent by the specified device.

Usage

Visual Basic *FaxControl.StatusFrom*(Index%)

Visual C++ *CString pFaxControl->GetStrProperty("StatusFrom", iIndex);*

Remarks

This property is Read-only.

Access

Read-only

Data Type

String(Array)

StatusID Property

Description

Returns the ID of the fax being sent or received by the specified device.

Usage

Visual Basic *FaxControl.StatusID*(Index%)

Visual C++ long *pFaxControl->GetNumProperty("StatusID", iIndex);*

Remarks

This property is Read-only.

Access

Read-only

Data Type

Long(Array)

StatusNumber Property

Description

Returns the phone number used for sending the fax being sent by the specified device.

Usage

Visual Basic *FaxControl.StatusNumber*(Index%)

Visual C++ CString *pFaxControl->GetStrProperty*("StatusNumber", *iIndex*);

Remarks

This property is Read-only.

Access

Read-only

Data Type

String(Array)

StatusPages Property

Description

Returns the number of pages in the fax being sent by the specified device.

Usage

Visual Basic *FaxControl.StatusPages*(Index%)

Visual C++ long *pFaxControl->GetNumProperty*("StatusPages", *ilIndex*);

Remarks

This property is Read-only.

Access

Read-only

Data Type

Integer(Array)

StatusPagesSent Property

Description

Returns the number of pages in a fax that have been sent or received by the specified device.

Usage

Visual Basic *FaxControl.StatusPagesSent*(Index%)

Visual C++ long *pFaxControl->GetNumProperty*("StatusPagesSent", *ilIndex*);

Remarks

This property is Read-only.

Access

Read-only

Data Type

Integer(Array)

StatusPercentage Property

Description

Returns the percentage of the current fax page that has been sent by the specified device.

Usage

Visual Basic *FaxControl.StatusPercentage*(Index%)

Visual C++ long *pFaxControl->GetNumProperty*("StatusPercentage", *iIndex*);

Remarks

This property is Read-only.

Access

Read-only

Data Type

Integer(Array)

StatusRecipient Property

Description

Returns the recipient name of the the fax being sent by the specified device.

Usage

Visual Basic *FaxControl.StatusRecipient*(Index%)

Visual C++ *CString pFaxControl->GetStrProperty("StatusRecipient", iIndex);*

Remarks

This property is Read-only.

Access

Read-only

Data Type

String(Array)

StatusRemoteID Property

Description

Returns the Remote ID of the fax machine connected to the specified device.

Usage

Visual Basic *FaxControl.StatusRemoteID*(Index%)

Visual C++ CString *pFaxControl->GetStrProperty*("StatusRemoteID", *ilIndex*);

Remarks

This property is Read-only.

Access

Read-only

Data Type

String(Array)

StatusResolution Property

Description

Returns the resolution being used for the fax being sent or received by the specified device.

Usage

Visual Basic *FaxControl.StatusResolution*(Index%)

Visual C++ *CString pFaxControl->GetStrProperty("StatusResolution", iIndex);*

Remarks

This property is Read-only.

Access

Read-only

Data Type

String(Array)

StatusSubject Property

Description

Returns the subject string of the fax being sent by the specified device.

Usage

Visual Basic *FaxControl.StatusSubject*(Index%)

Visual C++ CString *pFaxControl->GetStrProperty*("StatusSubject", *iIndex*);

Remarks

This property is Read-only.

Access

Read-only

Data Type

String(Array)

The FaxMan Server

The FaxMan fax server is by far the most complex part of the FaxMan system. In the server is all the logic required to manage a dynamic pool of faxmodems, keep track of a virtually unlimited number of fax events in various logs, send and receive multiple faxes simultaneously, and keep a potentially large number of applications informed as to what it's doing the whole time.

It's important to note from the outset that the server is exactly that - it is not a part of your application, nor does it in any way "belong" to your application. Think of it as a shared system resource which must be accessed through a prescribed interface, in much the same way you might access a hard drive.

Always keep in mind that other applications may be using the same resource at the same time, so your application should never assume that it is the only application using the FaxMan server. In practice, this means that you should make a point to follow the guidelines we set down as to installation and other practices to avoid a potential conflict.

[Installing YOUR FaxMan Application](#)

[FaxMan Server Command-line Options](#)

[FaxMan Server Configuration Options](#)

[The FaxMan Fax Logs](#)

[FaxModem Configuration](#)

[How FaxMan Communicates with Your Application](#)



Installing Your FaxMan Application

When your application is installed on your user's system, you should always install the parts of the FaxMan system in the Windows System directory. All of the FaxMan files contain version information, so the installation should only copy over any existing FaxMan files if the version you're installing is more recent than those already installed.

Installing in the system directory ensures that all applications using FaxMan will be using the same version. Failure to install in the system directory, on the other hand, could potentially cause difficult-to-trace problems related to having multiple FaxMan installations.

It is also suggested, though not required, that FaxMan be installed to load at Windows start time. This will allow the server to run continuously in the background and service any fax events more or less automatically (if you object to the FaxMan spinning world icon being shown, you can use the /H command-line option to hide the server entirely). If you do elect to install FaxMan as a start-up application, be sure to verify that it's not already installed that way before placing it in the start-up group.



FaxMan Server Command-line Options

There are several options you can utilize for starting the FaxMan server; these are specified on the command-line when invoking the application. These are as follows:

/H - starts the server in "hidden" mode. The server displays nothing whatsoever on screen, including its icon, but continues to execute in the normal manner. The only way to shutdown the server in this case is either to turn on the AutoShutdown configuration property or close the server manually (using the FaxCloseServer DLL function or the VBX/OCX Action property).

/D - starts the server in "debug" mode. In this mode, a debug panel is displayed which shows debug information. This mode is only useful for trying to track down problems with FaxMan.

/Lfilename - Used only in conjunction with /D, this mode causes all debug information to be logged to the file specified.



The FaxMan Server Configuration Options

In addition to the command-line options, there are several configuration options for the server which can be set and interrogated programmatically from your application. Be aware that these are global options, and should not be set arbitrarily from your application; rather, your application should provide some sort of user interface for displaying the current settings and allowing the user to alter them. In this way you can be sure that the configuration is consistently what the user has asked for, whether it was set from your FaxMan application or another on the system.

The server configuration options are as follows:

AutoShutdown - Default is OFF. When set to ON, the server will automatically shutdown sixty seconds after it goes idle. Use this option if FaxMan is installed on a system with severe memory constraints (the FaxMan system has a relatively small memory footprint).

ReceiveID - Default is blank. Set this to the ID that the user wants to identify this faxmodem with during receive events. (It is recommended that this item also be used as the LocalID when sending a fax, but that is left to your application's discretion. In any event, it would probably be nice for the user if you at least used the ReceiveID as the default LocalID for fax sending).

AutoReceive - Default is OFF. When set to ON, the server will prepare each receive-enabled faxmodem to receive faxes. The only practical side-effect to having this ON is that each port setup for reception will be inaccessible to any other application. Note that if this setting is changed from ON to OFF, all receive ports will be reset at the first opportunity, making the ports available (if a fax is actually being received when this is toggled, it will NOT be cancelled; the port will be freed when the receive process is complete).

ReceivePath - Default is the directory containing the FaxMan server (FAXMAN.EXE). This should be set to the path in which the user wants all received faxes to be placed. If the path is invalid, the faxes will be placed in the FaxMan server directory.



The FaxMan Server - Fax Log

Each fax sent or received using the FaxMan system is actually stored in two parts: the actual fax image is stored in a FaxMan fax file (*.FMF or *.FMP), while the control information associated with each fax (who the fax is from, who it's going to, the phone number to send it to, etc... [a SEND_FAX structure for you DLL interface programmers]) is stored separately in the FaxMan log file (the actual filename is FAXLOG, found in the server directory).

There are essentially five separate logs stored in this log file:

Pending Log - Those faxes which are scheduled for sending.

Sending Log - Faxes which are currently being sent. Remember that there may be multiple entries in this log if the system is using multiple faxmodems.

Completed Log - Faxes which were sent successfully. Note that received faxes will never appear here, this is only for successfully sent faxes.

Failed Log - Faxes which failed to send properly. As noted above, this applies only to faxes which were sent from this system, not received faxes.

Received Log - This log contains all faxes which were received, whether or not they were received successfully. A received fax is not placed into this log until the fax receive process ends; this means that faxes that are in the process of being received are not currently a part of any fax log!

A fax that goes through the sending process will normally pass through three fax logs: Pending when it is scheduled, Sending during the send attempt, and either Completed or Failed as a final destination, depending, of course, on the results of the send. A fax could also be automatically retried a number of times, depending on the settings your application provides. In this event, the fax will not be placed into the Failed log until all retries have been attempted.

These faxes will move from Pending Log to Sending Log to Pending Log continually until they are either sent successfully or the desired number of retries have been exhausted.

Each time a fax is added or removed from a given log, your application is notified in order that you might accurately display the faxes in each log. So when a fax is scheduled, for instance, your application will be notified when the fax is added to the Pending Log. When the fax is subsequently moved from Pending to Sending, your application will get two notifications: one when the fax is deleted from the Pending log and one when the fax is added to the Sending log. While developers using the VBX/OCX interfaces can utilize the nifty log interface we provide for keeping track of the logs, those developers utilizing the DLL interface will have to create their own log interface if they desire an up-to-date list of each log.



The FaxMan Server FaxModem Configuration

The FaxMan fax system works transparently with multiple faxmodems in any given computer system, allowing your application (and others) to send and receive multiple faxes simultaneously without requiring any intimate knowledge of the system's installed hardware.

As a part of this service, FaxMan provides your application with a programmatic method for automatically configuring the faxmodems installed on a given system, including the ability to selectively configure individual comm ports if desired.

These functions (the FaxAddDevice DLL function and the AddDevice VBX/OCX property) will cause the FaxMan server to detect and identify faxmodems on a specified port, or on all ports simultaneously. When the server has completed configuring a port, your application is then notified.

This allows your application to easily keep a list or display of the installed faxmodems and the salient information for each, including its class (either Class 1, Class 2, or Class 2.0) and which port it's installed on. In addition to the notifications provided when a faxmodem is identified by the system, your application is also notified each time the status of a faxmodem changes.

A faxmodem's status can be any of the following:

Idle - The faxmodem port is currently not in use by FaxMan. Note that this is the only faxmodem state in which the port is not being used by FaxMan.

Sending - The faxmodem is currently sending a fax.

Receiving - The faxmodem is currently receiving a fax or waiting for a call.

Your application can get an up-to-date listing of the installed devices, and their current status, at any time either by calling the FaxEnumDevices function (for DLL interface users) or referencing the Devices property (for VBX/OCX interface users). You should keep in mind that since there is no requirement that a faxmodem be installed in order to schedule a fax, unless your application uses the above-mentioned functions to verify that a valid device actually exists you could schedule a fax and have no valid devices on which to send it!



The FaxMan Server - How Does FaxMan Communicate With Your Application?

Throughout this overview of the server, we've been mentioning that FaxMan will send your application notifications for various events. It's time to explore exactly how this happens. This discussion really only applies to the DLL interface for FaxMan - those programmers using the VBX/OCX interface do not necessarily need to read this section, as your applications will be notified through a well-defined and documented event structure.

When your application is getting ready to start using the FaxMan system, the first thing it must do is call the FaxRegisterApp function, passing in a unique identifier for your application and a window for FaxMan to use for event notifications.

This window handle is FaxMan's only access to your application. For each notification event, FaxMan will use the SendMessage function to send the appropriate information to your application. Notice that this is specifically not a PostMessage function call; the reason for this is that any pointers FaxMan passes to your app are valid only for the duration of the SendMessage function call. This leads to the first rule concerning event notifications:

Your application should never store any pointers passed to it from the FaxMan server unless the documentation specifically states otherwise.

If you do store these pointers, your app will probably GPF when you try to reference them later. In any event, they are undefined pointers once the SendMessage call exits.

Additionally, since these are SendMessage function calls, you should keep in mind that the server is blocked waiting for your application to return from the SendMessage call. Which brings us to the second rule concerning event notification:

Your application should handle these event notifications as quickly as possible, since the server is not running while it's waiting for your application to respond.

This means that you should probably not perform any huge database searches while in the SendMessage handler. If you need to perform some longer operations, post a message to yourself (and make sure you don't hog the CPU, otherwise you'll achieve the same effect as not returning quickly from the SendMessage handler).

The order of messages you can expect for both sending and receiving events is documented in the DLL Message Reference section.

Creating Fax Files

There are three ways to create FaxMan fax files:

- Print from any Windows application using the [FaxMan printer driver](#) (FMFAXDRV.DRV.)
- Use of the FaxMan software (the [FaxCreate](#) Function or the [InputFiles](#) and Action property), to create a fax file by converting existing monochrome BMP, PCX, TIFF images or ASCII Text files.
- Use ImageMan to create fax files from any supported format.

Regardless of which method you choose to create files, FaxMan fax files must be created to meet the following specifications:

-
- The image must have a width of 1728 pixels
- The horizontal resolution must be 204 pixels/inch
- The vertical resolution must be 192 pixels/inch (for hi-res images) or 96 pixels/inch (for low-res images).
- Multipage files are supported (and encouraged).

[Creating FaxMan Fax Files with the DLL](#)

[Creating FaxMan Fax Files with the VBX or OCX](#)

FaxMan DLL Interface

[FaxAddDevice](#)
[FaxCancel](#)
[FaxCloseServer](#)
[FaxConfigureDevice](#)
[FaxCopy](#)
[FaxCreate](#)
[FaxEnumDevices](#)
[FaxGetConfig](#)
[FaxInitSendStruct](#)
[FaxLogDelete](#)
[FaxLogFind](#)
[FaxLogInit](#)
[FaxLogNext](#)
[FaxRegisterApp](#)
[FaxRemoveDevice](#)
[FaxSchedule](#)
[FaxSetConfig](#)
[FaxSubscribe](#)
[FaxUnregisterApp](#)

Structure Definitions

[FaxMan #DEFINES](#)
[SEND_FAX](#)
[FAXDEVICE](#)
[PRINTSTAT](#)
[SERVER_CONFIG](#)

PAPPINFO FaxRegisterApp(LPSTR lpAppName, WORD AppNotifyWnd)

Registers an application with the FaxMan system.

| <u>Parameter</u> | <u>Type/Description</u> |
|-------------------------|--|
| lpAppName | LPSTR A unique application identifier string. This identifier must be used consistently for this application or notifications will not be received. |
| AppNotifyWnd | WORD Window handle of the window to receive FaxMan notification messages. |

Return Value

The return value is a pointer to an APPINFO structure on success, a NULL pointer on failure.

Comments

All applications must register with the FaxMan system before calling any other FaxMan functions. The returned APPINFO pointer is used as a parameter to every other API call.

The lpAppName identifier must uniquely identify your application, so be original. Identifiers are limited to 25 characters in length (including the terminating 0). **THIS NAME MUST MATCH THE ENTRY IN THE WIN INI FILE** please see the section on [Using the FaxMan Printer Driver](#)

BOOL FaxUnregisterApp(PAPPINFO pApp)

Un-registers an application from the FaxMan system.

| <u>Parameter</u> | <u>Type/Description</u> |
|-------------------------|--|
| pApp | PAPPINFO Returned from FaxRegisterApp |

Return Value

The return value is always TRUE.

Comments

All applications should un-register via this function before terminating.

void FaxInitSendStruct(PAPPINFO pApp, PSEND_FAX pFax);

Initializes a SEND_FAX structure with valid default values.

| <u>Parameter</u> | <u>Type/Description</u> |
|------------------|---|
| pApp | PAPPINFO Returned from FaxRegisterApp |
| pFax | PSEND_FAX Pointer to a SEND_FAX structure to be initialized. |

Return Value

This function returns nothing.

Comments

Before using a SEND_FAX structure with the [FaxSchedule](#) function, you should call FaxInitSendStruct to initialize all the values to the proper defaults. This saves you from having to write tedious code to do the same thing, plus it helps eliminate coding errors resulting from uninitialized structures. After calling FaxInitSendStruct, your application is only required to fill in the list of files to send and the destination fax number (although you would probably want to fill in several other structures, like subject, they are not required).

DWORD FaxSchedule(PAPPINFO pApp, PSEND_FAX pFax)

Schedules a fax to be sent.

| <u>Parameter</u> | <u>Type/Description</u> |
|-------------------------|---|
| pApp | PAPPINFO Returned from FaxRegister function. |
| pFax | PSEND_FAX Pointer to a SEND_FAX structure. |

Return Value

The return value is a DWORD used to uniquely identify this fax within the FaxMan system. This id can be used in subsequent operations to obtain information on the status of this fax.

If the returned value is 0, the fax could not be scheduled. General causes of this are that the server could not be located/loaded or because the file list contained an invalid file.

Comments

FaxSchedule causes FaxMan to place the given fax into the send log. Note that you should be sure to call [FaxInitSendStruct](#) and then fill in the fax files and fax number field before calling this function.

Faxes can be scheduled for any time in the future by simply setting the time and date to send in the SEND_FAX structure.

BOOL FaxCancel(PAPPINFO pApp, DWORD dwID)

Cancels a send or receive operation that is currently in progress.

| <u>Parameter</u> | <u>Type/Description</u> |
|------------------|--|
| pApp | PAPPINFO Returned from FaxRegisterApp |
| dwID | DWORD Unique id of the fax to cancel. |

Return Value

Returns TRUE on successful cancel, FALSE otherwise.

Comments

The dwID is the return value from FaxSchedule for transmitted faxes. For received faxes, the dwID value is passed as part of the receive event notifications.

This function should only be used to cancel a fax that is in progress. If the fax is still in the pending log, you should use the LogDelete function.

If a sending fax is cancelled, it is moved immediately to the failed log, regardless of the number of retries requested.

NOTE

Cancelling a fax does not necessarily cause it to stop immediately. The faxmodem must be shut down in an orderly fashion, otherwise the phone line may be left off-hook. The process may take anywhere from a fraction of a second to several seconds, depending on the faxmodem and where in the faxing process the cancel was requested.

int FaxCloseServer(PAPPINFO pApp)

Closes the FaxMan Fax Server immediately, without regard to the current send or receive status of any device.

| <u>Parameter</u> | <u>Type/Description</u> |
|-------------------------|---------------------------------------|
| pApp | PAPPINFO Returned from FaxRegisterApp |

Return Value

Returns 0 if the Fax Server could not be located/loaded, otherwise returns 1

Comments

This function will cause the server to close all ports assigned to it and remove the server from memory. This function should be used with great caution as you can immediately terminate a device that is currently either sending or receiving a fax transmission.

NOTE

Use this function only when you need to take immediate control of all the communications devices, use FaxSetConfig() with a paramater of AutoShutDown in all other cases.

This function will take a moment to complete so that all the open ports may be shut down correctly.

PSEND_FAX FaxCopy(PSEND_FAX pFaxStruct)

Creates a Globally allocated "deep" copy of a pSend_Fax structure

| <u>Parameter</u> | <u>Type/Description</u> |
|------------------|---|
| pFaxStruct | PSEND_FAX The send fax structure to copy |

Return Value

A pointer to a globally allocated copy the original pSend_Fax structure.

Comments

The function globally allocates a block of memory for the new SEND_FAX structure and copies the the original structure into it. This "deep" copy, copies all of the associated strings in the structure into the memory block, including those which are not normally needed or available.

NOTE

You MUST call GlobalFreePtr() when you finish with this structure to free the memory that was allocated by the function call.

int FaxAddDevice(PAPPINFO pApp, int nPort)

Prompts the FaxMan server to attempt to configure a given port, or all ports, for use with the system.

| <u>Parameter</u> | <u>Type/Description</u> |
|-------------------------|--|
| pApp | PAPPINFO Returned from FaxRegisterApp |
| int | int Port number to add to the FaxServer or 0 to Autoconfigure |

Return Value

Returns 0 if the Fax Server could not be located/loaded, otherwise returns 1

Comments

This function will cause the server to interrogate the port specified by nPort and determine if it can be used for faxing operations. If the nPort parameter is 0, the server will attempt to configure all installed ports simultaneously.

Setting nPort to 0 is the preferred method for initially configuring the FaxMan system; this would usually be done during application installation, but can be done at any time.

NOTE

When a port is successfully configured, it defaults to being a send and receive port. If you wish to change this, your app should reconfigure the port using the [FaxConfigureDevice](#) function.

int FaxConfigureDevice(PAPPINFO pApp, PFAXDEVICE pFaxDev)

Configures a single Fax Device for use by the FaxMan Server.

Parameter

pApp
pFaxDev

Type/Description

PAPPINFO Returned from FaxRegisterApp
PFAXDEVICE a pointer to a FAXDEVICE structure

Return Value

Returns 0 if the Fax Server could not be located/loaded, otherwise returns 1

Comments

This function allows you to reconfigure a specific device for use with the FaxMan Server according to the settings contained in the [FAXDEVICE structure](#).

int FaxRemoveDevice(PAPPINFO pApp, int nPort)

Removes a single Fax Device the FaxMan Server devices.

Parameter

pApp
int

Type/Description

PAPPINFO Returned from FaxRegisterApp
int Port number to remove from the FaxMan Server

Return Value

Returns 0 if the Fax Server could not be located/loaded, otherwise returns 1

Comments

This function allows you to remove a specific device from use with the FaxMan server. See the related functions [FaxConfigureDevice](#) and [FaxAddDevice](#) for adding and configuring devices.

int FaxEnumDevices(PAPPINFO pApp)

This function causes the FaxServer to iterate through each faxmodem under it's control and send a message to your application concerning the state of that device as described in the [FAXDEVICE](#) structure.

| <u>Parameter</u> | <u>Type/Description</u> |
|------------------|--|
| pApp | PAPPINFO Returned from FaxRegisterApp |

Return Value

This function returns 0 if it was unable to load/find the server and returns 1 otherwise.

Comments

The server will call the SendMessage function to report the current state of all installed faxmodems, meaning that your application should be ready to process the FAXMODEMSG messages which are sent to your application.

The FAXMODEMSG will pass as the lParam a pointer to a [FAXDEVICE](#) structure.

WORD FaxLogFind(PAPPINFO pApp, PSEND_FAX *ppFax, DWORD dwID)

Retrieves information for a given fax event.

| <u>Parameter</u> | <u>Type/Description</u> |
|-------------------------|---|
| pApp | PAPPINFO Returned from FaxRegisterApp |
| ppFax | PSEND_FAX * Pointer to a PSEND_FAX to receive fax information. |
| dwID | DWORD ID of the fax to get information for. |

Return Value

Return values are as follows:

- 0 if the fax server could not be loaded to process the request.
- 1 if the fax event information was successfully retrieved.
- 2 if the fax event was not located in the logs.

Comments

The PSEND_FAX pointer returned to the calling application must be freed via a call to GlobalFreePtr(). Failure to free this pointer will result in a memory leak for your application.

WORD FaxLogInit(PAPPINFO pApp, WORD wLog)

Used in conjunction with FaxLogNext. Allows an application to iterate through the entries in a given fax log.

| <u>Parameter</u> | <u>Type/Description</u> |
|-------------------------|--|
| pApp | PAPPINFO Returned from FaxRegisterApp |
| wLog | WORD Log to begin iterating through. |

Return Value

Returns a LOGERR value (as defined in faxdll.h) if unable to begin iteration. Returns 0 if unable to load the FaxMan server.

Comments

FaxLogInit initializes iteration for the given log. If this function is successful, FaxLogNext is called to actually iterate through the log entries.

WORD FaxLogNext(PAPPINFO pApp, PSEND_FAX pFax)

Used in conjunction with FaxLogInit. Allows an application to enumerate fax log entries.

| <u>Parameter</u> | <u>Type/Description</u> |
|-------------------------|--|
| pApp | PAPPINFO Returned from FaxRegisterApp |
| pFax | PSEND_FAX Pointer to a SEND_FAX structure to receive information on the next log entry. |

Return Value

The return value is LOGERR_CONTINUEIT if there are more log entries to enumerate. Any other return value should cause an application to halt enumeration.

Comments

Once a successful call to FaxLogInit has been made, an application should call FaxLogNext repeatedly until it returns 0. Each call will fill in the pFax structure with information for a fax log entry.

WORD FaxLogDelete(PAPPINFO pApp, DWORD dwID, WORD wLog)

Deletes a fax event and all associated FMF files from the requested log and disk.

| <u>Parameter</u> | <u>Type/Description</u> |
|-------------------------|---|
| pApp | PAPPINFO Returned from FaxRegisterApp |
| dwID | DWORD Unique id for the fax to delete. |
| wLog | WORD Log to remove the fax from. This must be specified. |

Return Value

- 0 if the server is not found or could not be loaded
- 1 if the item is deleted successfully
- 2 if the item was not found in the log.

Comments

This function will fail if you do not specify the correct log to remove the fax from. When a log entry is deleted the associated fax files which have .FMF extensions are deleted from the disk. This means that if you want to have persistent fax files, you should give them an extension other than FMF.

int FaxSubscribe(PAPPINFO pApp, WORD wNotify)

Allows an application to receive message notification for selected server events.

| <u>Parameter</u> | <u>Type/Description</u> |
|-------------------------|--|
| pApp | PAPPINFO Returned from FaxRegisterApp |
| wNotify | WORD ID of the notification stream your application wishes to subscribe to. |

Valid subscription identifiers are :

SUBSCRIBE_LOG_PENDING
SUBSCRIBE_LOG_SENDING
SUBSCRIBE_LOG_COMPLETE
SUBSCRIBE_LOG_FAILED
SUBSCRIBE_LOG_RECEIVE
SUBSCRIBE_LOG_ALL
SUBSCRIBE_FAXMODEM
SUBSCRIBE_SERVER_CONFIG

Return Value

Returns 0.

Comments

You may subscribe to any combination of notification streams, see [Message Subscription](#) for information on Message responses.

Stages of Fax Transmission

WORD FaxCreate(PAPPINFO, LPSTR lpFiles, LPSTR lpFileName, DWORD dwFlags)

Allows an application to convert monochrome BMP, PCX, or TIFF images into valid FaxMan fax files.

| <u>Parameter</u> | <u>Type/Description</u> |
|-------------------------|--|
| pApp | PAPPINFO Returned from FaxRegisterApp |
| lpFiles | LPSTR List of files to convert into a single fax file (separated by "+"). |
| lpFileName | LPSTR The Name of fmf file to create |
| dwFlags | DWORD Flags to use when creating the fax file. |

Return Value

The return value is 0 on success. On failure, the return value is the (1-based) index of the first file which failed to convert.

Comments

Allows an application to convert monochrome BMP, PCX, DCX, TIFF and ASCII Text file into valid FaxMan fax files.

This function is used to create faxable files from existing monochrome images. The files to be converted are listed as a single string of files separated by pluses (for example, "c:\images\covrldr.bmp+c:\pg1.tif+c:\pg5.txt"). The resultant file is a single multi-page faxman fax file.

Note

ASCII text files are defined as any file which is not a BMP, PCX, DCX, or TIFF file and which does not contain any NULL data.

That this function will scale images that are wider than 1728 pixels down to 1728 pixels. Images that are narrower than 1728 pixels will have white space added to pad them to 1728 pixels.

int FaxSetConfig(PAPPINFO pApp, LPCSTR Ipkey, LPCSTR IpValue)

Sets the profile of a given INI key under the FaxGeneral Section of the FaxMan Fax Server INI file.

| <u>Parameter</u> | <u>Type/Description</u> |
|-------------------------|--|
| pApp | PAPPINFO Returned from FaxRegisterApp |
| IpKey | LPCSTR The Profile Key Name of the item to be Configured |
| IpValue | LPCSTR The Profile Key Value of the item to be Configured |

Return Value

0 if the Fax Server could not be located/loaded,
1 if the configuration item was set successfully, and
2 if unable to confirm the new setting.

Comments

This function allows you to configure the FaxMan Server by setting the Profile ints in the FaxMan.INI file.

int FaxGetConfig(PAPPINFO pApp, LPCSTR lpkey, LPSTR lpBuf, WORD wBufSize)

Gets the profile of a given key name for the FaxMan Fax Server.

| <u>Parameter</u> | <u>Type/Description</u> |
|-------------------------|--|
| pApp | PAPPINFO Returned from FaxRegisterApp |
| lpKey | LPCSTR The Profile Key Name of the item to be Configured |
| lpBuf | LPSTR A pointer to the buffer to be filled with the servers configuration |
| wBufSize | WORD The size of the allocated buffer in lpBuff above |

Return Value

Returns 0 if the Fax Server could not be located/loaded, returns 1 otherwise.

Comments

This function allows you to get the current configuration of the FaxMan server. Please note the functions will only write wBufSize characters to the buffer and the the buffer must be that length at a minimum.

Defined FaxMan Macros

```
#define WM_FAXMSG WM_USER+1000

// These identifiers should be responded to by the client apps in their FaxMan message window.
#define FAXSENDMSG 0
#define FAXPRINTMSG 1
#define FAXGETFILENAME 2
#define FAXLOGADD 3
#define FAXLOGREMOVE 4
#define FAXSCHEDULEMSG 5

// FAXCLASS is an enumeration of the possible types of faxmodems installed and
// usable by FaxMan
typedef enum {
    FAX_0 = 0,
    FAX_1,
    FAX_2,
    FAX_20
} FAXCLASS;

// FAXSTATE is an enumeration of the possible fax states. During fax transmission, an
// application
// will receive notifications which contain the current state of the operation, as defined below.
typedef enum {
    FAXST_ERROR= 0,
    FAXST_OK,
    FAXST_INIT,
    FAXST_WAITFORSEND,
    FAXST_SEND_INIT, //initializing a send operation
    FAXST_SEND_DIALING, //we're dialing...
    FAXST_SEND_WAIT_FCON, //waiting for connect message
    FAXST_SEND_FCON, //Connected!
    FAXST_SEND_WAIT_FCSI, //waiting for FCSI
    FAXST_SEND_FCSI, //Receiver's ID string
    FAXST_SEND_WAIT_FDIS, //waiting for remote +FDIS
    FAXST_SEND_FDIS, //Remote T.30 parameters
    FAXST_SEND_WAIT_CONNECT, //wait for connect
    FAXST_SEND_FDCS, //negotiated T.30 parms
    FAXST_SENDING, //actually sending data - no longer in command mode
    FAXST_PAGE_END, //end of page
    FAXST_PORTSHUT, //comm port deleted
    FAXST_ABORT, //abort the current FAX operation and reset faxmodem
    FAXST_COMPLETE, //we're done, and can dismantle our faxing apparatus
    // Receiving fax states
    FAXST_INITRX, //initialize faxmodem for receive
    FAXST_WAITFORRX, //waiting for RING
    FAXST_ANSWERING, //answering the phone
    FAXST_RX_NEGOTIATE, //negotiating RX parms
}
```

```

    FAXST_RXDATA,           //receiving fax data
    FAXST_RX_PAGE_END,     //end of rx page

} FAXSTATE;

// FAXERROR is a list of error codes possible during transmission
typedef enum {
    FAXERR_OK,
    FAXERR_ACK,             // Command wasn't acknowledged w/OK
    FAXERR_BADFAXMODEM,    // Faxmodem doesn't support FaxModem-level ops
    FAXERR_INIT,           // error initializing faxmodem
    FAXERR_FDIS,           // bad FDIS settings
    FAXERR_FLID,           // some kind of error setting local ID
    FAXERR_DIAL,           // error dialing
    FAXERR_FCON_ERR,       // error trying to connect to remote fax
    FAXERR_FCSI,           // bad FCSI string
    FAXERR_NEG_FDIS,       // error receiving Negotiated +FDIS
    FAXERR_BADSTATE,       // tried to do something at wrong time
    FAXERR_BUSY,           // line is busy
    FAXERR_NODIALTONE,     // no dialtone found when dialing
    FAXERR_NOCONNECT,      // didn't get expected CONNECT msg
    FAXERR_CANCEL,         // user cancel
    FAXERR_FPTS,           // bad or no FPTS when expected
    FAXERR_FHNG,           // bad FHNG exit code or no FHNG when expected
    FAXERR_FDCS,           // +FDCS: not found when expected
    FAXERR_ERROR,          // general unspecific error w/faxmodem
    FAXERR_FILE,           // invalid fax file or files specified
    FAXERR_VERSION,        // incompatible DLL/Server versions
    FAXERR_TIMEOUT         // Command timed out
    FAXERR_NO_MPS_RESP     //No response to MPS repeated 3 times
    FAXERR_NO_EOP_RESP     //No Response to EOP repeated 3 times
    FAXERR_NOTRIAN         // Could not train the remote system
} FAXERROR;

// Log identifier flags
#define SUBSCRIBE_LOG_PENDING 0x01
#define SUBSCRIBE_LOG_SENDING 0x02
#define SUBSCRIBE_LOG_COMPLETE 0x04
#define SUBSCRIBE_LOG_FAILED 0x08
#define SUBSCRIBE_LOG_RECEIVE 0x10
#define SUBSCRIBE_LOG_ALL 0xff

```

SEND_FAX Structure Definition

```
typedef struct tagSendFaxStruct {
    // FaxMan uses these first 3 items for security
    // Don't change them unless you know what to do with 'em
    char        sentinel[4];    //should be 'FMAN'
    WORD        wStructRev;     //revision # of structure
    WORD        wStructSize;    //size of structure, including variable
strings
    LPSTR       szDestName;
    LPSTR       szDestFax;
    LPSTR       szSubject;
    LPSTR       szFromLine;
    LPSTR       szBanner;      //defines the banner line for this fax
    LPSTR       szCover;      //file containing cover page
    LPSTR       szComments;     //comments for use in banner or on cover
page
    LPSTR       szFromCompany;   //Company sending the fax
    LPSTR       szToCompany;     //Company receiving the fax
    LPSTR       szFromFax;       //Senders Fax Number
    LPSTR       szFromPhone;     //Senders Voice Number
    LPSTR       szUserDefined;   //User defined string
    char        szLocalID[FAXIDLEN]; //Fax ID of sending station
***LocalID***
    SHORT       nTimeout;       //Timeout val
    SHORT       nRetries;       //# of retries
    SHORT       nRetryDelay;    //seconds before a retry
    SHORT       nYear;         //year to send (1970-2038)
    SHORT       nMonth;        //month to send (1-12)
    SHORT       nDay;          //Day to send
    SHORT       nHour;         //hour to send (0-23)
    SHORT       nMin;          //minute to send (0-59)
    SHORT       nSecond;       //second to send (0-59)
    SHORT       nSendRes;      //desired send resolution
    LPSTR       szFileList;     //list of files to fax
    // items below here are filled in by FaxMan - clients shouldn't
change them.
    WORD        hWndNotify;     //notification window
    char        szAppName[APPNAMELEN]; //name of app that sent the fax
    char        szRemoteID[FAXIDLEN]; //String ID of the fax we're
connected to
    DWORD       dwSpeed;        //Connect speed
    SHORT       nFaxRes;        //Fax resolution (1 = LOW, 2 = HIGH)
    SHORT       nTotPages;      //# of pages in this fax (total)
    SHORT       nTotCurPage;    //current page (total)
    SHORT       nCurPages;      //# of pages in current file (faxfile)
    SHORT       nCurPage;       //currently sending page in current file
    SHORT       nPagesSent;      //actual # of pages sent (may be
different from nTotPages)
    SHORT       nPercent;       //percent complete on current page
    DWORD       dwID;           //FaxMan Generated "handle" to this fax
    SHORT       nHangCode;      //+FHNG result (it's int 'cause it's
```



```

initialized to -1)
    SHORT    nPort;                //comm port
    SHORT    nRetryCnt ;           //actual # of retries attempted by server
    SHORT    nSendYear;            // time of actual transmission
    SHORT    nSendMonth;           //
    SHORT    nSendDay;             //
    SHORT    nSendHour;            //
    SHORT    nSendMin;             //
    SHORT    nSendSecond;          //
    DWORD    dwDuration;           //duration of send event
    FAXCLASS faxclass;             //type of faxmodem we're dealin' with
    WORD      wLog;                //current log this fax is in
    FAXSTATE fs;                  //current fax state (dialing, waiting,
connected, etc...)
    FAXERROR fe;                  //current fax error state (if applicable)
    char      szStrs[1];           //strings
} SEND_FAX, FAR *PSEND_FAX;

```

sentinel

wStructRev

wStructSize

szDestName

szSubject

szFromLine

szComments

szFromCompany

szToCompany

szFromFax

szFromPhone

FAXDEVICE Structure

```
typedef struct tagDevice {  
    int      nPort;  
    WORD     wFlags;  
    char     szInit;  
    char     szReset;  
} FAXDEVICE, FAR *PFAXDEVICE;
```

nPort is the comm port the faxmodem is installed on.
wFlags is a set of flags which give the current status of the faxmodem.

wFlags is composed by combining the following flags:

PORT_SEND

Set if this port is available to send faxes.

PORT_RX

Set if this port is available to receive faxes.

PORT_SENDING

Set if this port is currently sending a fax.

PORT_RECEIVING

Set if this port is currently receiving a fax.

PORT_CLS1

PORT_CLS2

PORT_CLS20

These flags are set if the faxmodem is of the specified type (i.e., Class 1, Class 2, or Class 2.0)

PORT_BLOCKED

This flag is set if the port is unusable for some reason. The faxmodem may be turned off, or there may be a problem with the hardware.

PORT_BUSY = (PORT_SENDING | PORT_RECEIVING)

Use this flag to indicate if a port is currently active.

szInit is the string that is used to initialize the port for sending. This string has a maximum length of 128 characters including the terminating \0.

szReset is the string that is used to reset the device. This sting has a maximum length of 128 cahracters including the terminating \0.

PRINTSTAT structure

```
typedef struct tagPrintStat {  
    WORD nPages;                                // # of pages printed  
    char  file name[MAXPATH]; // Name of file created by driver  
    PRN_STATUS pStat;                // status of print job  
} PRINTSTAT, FAR *PPRINTSTAT;
```

Fax Banner and Coveragepage Format Characters

The banner strings can consist of any of 3 parts: a left-justified portion, center-justified portion, and right-justified portion. The justification is created by use of a vertical bar character (|).

The following control characters can be inserted in the banner or the coveragepage string to included some common fax information:

- %d = date (dd-mmm-yyyy format, e.g. 03-Feb-1995)
- %t = send time (24-hour format, e.g. 6:00pm = 18:00:00)
- %p = total # of pages in the fax
- %c = current page
- %r = fax recipient
- %s = fax senders name
- %i = sender's fax ID
- %u = subject of fax
- %f = Fax Number
- %o = Comments
- %m = From Company
- %y = To Company
- %e = User defined string
- %h = Senders Voice Number
- %x = Senders Fax Number

Example:

To create a banner line that looks like this:

From: Jane Doe@MYCO (centered)01-Jan-200 (right)Page 1 of 3

you would create a banner string like so:

"From: %s|%d|Page %c of %p"

To create a centered banner line of Date and Time and Page counts you would create the following string:

"|%d %t Page %c of %p"

Notes

The string is parsed character by character. If the string contains an invalid character following the % token, the token and the following character will be ingored. i.e. a Banner string of "From % s" would omit the Fax Sender information and would print as "From s".

The string can be as many characters wide as you like, however, only one line of text will print as the banner. There is the potential that the center string could overwrite the left string and the right string could overwrite the both the left and center strings due to the justification. As a general rule the sender would not be aware of this unless the recipient were to inform them.

Message Subscription

By subscribing to FaxMan message streams, your application can be notified for any events that may be of interest to your application. For instance, if you subscribe to the SUBSCRIBE_LOG_SENDING message stream, your application can use the messages sent by FaxMan to display the status of any currently sending faxes, including such items as send speed, ID of the remote fax, and which page is currently being sent.

After subscribing to a message stream, your application will receive messages from the Server when certain events occur. The messages will have the following structure

HWND hWnd, WORD wMsg, WORD wParam, LONG lParam

| <u>Parameter</u> | <u>Type/Description</u> |
|------------------|--|
| hWnd | HWND This is the window specified by your application in the FaxRegisterApp function. |
| wMsg | WORD The Message ID will be WM_FAXMSG |
| wParam | WORD A valid Message Identifier as listed below |
| lParam | long A pointer to a structure based on the wParam |

wParams

FAXSENDMSG - sent for current sending/receiving notifications
FAXPRINTMSG - sent by the printer driver to notify your application of print events
FAXGETFILENAME - sent by printer driver to request filename for print output file
FAXLOGADD - sent when a fax is added to a log
FAXLOGREMOVE - sent when a fax is removed from a log
FAXMODEMSG - sent when the status of a faxmodem is changed or when a new device is added
FAXSERVERCONFIGMSG - sent when the server configuration is changed

The following table lists the value of lParam for each message identifier:

| <u>wParam</u> | <u>lParam - Pointer to this type of Structure</u> |
|--------------------|---|
| FAXSENDMSG | Far Pointer to a FAXSEND structure |
| FAXPRINTMSG | Far Pointer to a PRINTSTAT structure |
| FAXGETFILENAME | LPSTR to be filled in with the file name |
| FAXLOGADD | Far Pointer to a SEND_FAX structure |
| FAXLOGREMOVE | Far Pointer to a SEND_FAX structure |
| FAXMODEMSG | Far Pointer to a FAXDEVICE structure |
| FAXSERVERCONFIGMSG | Far Pointer to a SERVER_CONFIG structure |

To subscribe to a message stream use the FaxSubscribe() function with a valid subscription identifier as listed below.

SUBSCRIBE_LOG_PENDING
SUBSCRIBE_LOG_SENDING
SUBSCRIBE_LOG_COMPLETE
SUBSCRIBE_LOG_FAILED
SUBSCRIBE_LOG_RECEIVE
SUBSCRIBE_LOG_ALL

Log subscriptions relate information regarding to changes in the specified log. Your app will be notified when an item is removed or added to the specified log. In addition, the SUBSCRIBE_LOG_SENDING messages will send up-to-the-second notifications of any currently occurring fax activity, allowing your

application to display pertinent information for all send and receive events. (Note that this works for both send and receive events, but received faxes aren't really a part of any log until the reception is complete, at which point they are added to the receive log).

SUBSCRIBE_FAXMODEM

Causes your application to receive FAXMODEMMSG messages.

SUBSCRIBE_SERVER_CONFIG

Causes your application to receive FAXSERVERCONFIGMSG messages.

SERVER_CONFIG Structure Definition

```
typedef struct tagSERVER_CONFIG {  
    LPSTR lpKey;  
    LPSTR lpValue;  
} SERVER_CONFIG, FAR *PSERVER_CONFIG;
```

Technical Report Request

Contacting Tech Support

When reporting a FaxMan problem or bug, please include the following information:

A detailed textual description of the problem. If you can't provide details here, the odds of us determining the problem are greatly reduced.

The location of all copies of the following FaxMan files:

FAXMAN.EXE FAXDLL.DLL
FAXMAN1.VBX FAXMAN.INI
CLASS1.DAT CLASS2.DAT
CLASS20.DAT FAXDLL.H

Please be sure to actually search your system and any network drives on your path for these files, don't just try to remember where they are installed. One of the first things our tech support people will do is ask you to search for these files again, so make sure you know where every copy is. The most common problems stem from either having multiple copies of some of these files or not having the server (FAXMAN.EXE) on your path.

A debug log illustrating the problem, if applicable. (To generate a debug log, start FaxMan with the /D /Llog-filename parameters, then run FaxMan to the point where the problem manifests itself.)

A copy of your FAXMAN.INI file (located in the same location as the FAXMAN.EXE file).

The brand and model of your faxmodem.

A report printed from MSD.EXE which lists all relevant sections of your system (including SYSTEM.INI, WIN.INI, COM ports, Windows Information, and environment strings).

The name and version of any applications which have problems printing from the FaxMan printer driver. You should also include copies of any files which won't print correctly.

Please enclose these items in a ZIP file and e-mail it to us, either via our CompuServe account or our BBS. You may alternately fax these items to us, but it can quite easily be 20+ pages of information. Please do not post these items to our CompuServe forum, EMail them directly to our account.

Common Problems, Symptoms and Fixes

FaxMan doesn't work.

This description is guaranteed to take the longest time to resolve. If you can't get the server to work, we need as much detailed information as you can possibly find for us to look at. Just saying "It don't work" doesn't help anyone, least of all yourself. See the section [troubleshooting](#) or on [Technical Report Requests](#) to find out what things to look for, then put all the information into a fax or e-mail posting that we can print-out and pore over.

Server auto-configuration (using FaxAddDevice with a port of 0) doesn't detect any faxmodems when I know they're installed.

This is probably caused by having a non-faxmodem device of some kind that causes the system to "hang" for a second or so when FaxMan tries to initialize it. Since FaxMan configures all devices in parallel, this "hang" can cause all of the other ports to timeout, thus causing FaxMan to assume that there are no ports installed. To get around this, try auto-configuring each device individually, using FaxAddDevice and a specific port number. This should keep the "hang" from affecting the other configuration processes.

NOTE that you should wait for the return message notification (FAXMODEMMSG) for the current port before calling FaxAddDevice for the next port.

Printing a file to the FaxMan Printer Driver generates the message "Unable To Get Filename" and doesn't generate an output file.

This is caused by one of two things:

1. The printer driver is not properly installed. In this case, check your WIN.INI settings to verify that the driver is installed properly. See the section [Using the Faxman Printer Driver.](#)
2. The application you've installed for use with the printer driver uses the VBX and the application and server aren't loaded at print time. If you can print just fine when your app is already loaded, then this is the culprit. This is caused by a problem with Visual Basic not firing events as documented. We're still looking for a work-around for this VB bug. The only sure way to make printing work in this case is to make sure your app (and the server) are loaded before printing is initiated.

Sometimes I schedule a fax and I get no reaction. Nothing shows up in the pending log, it doesn't start sending it...It's like it never happened.

This will happen when one of the fax files you've scheduled is not a valid FaxMan fax file. If possible you should check the files you're sending by trying to view them. If you can't do this, then upload the files to us and we'll tell you what the problem with them is.

DeviceInit Property

Description

Determines the string used to initialize a faxmodem

Usage

Visual Basic *FaxControl.DeviceInit*(Index%)[= InitString\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("DeviceInit", lpstrInitString, iIndex);*
 CString pFaxControl->GetStrProperty("DeviceInit", iIndex);

Remarks

The DeviceInit property allows you to get or set the current initialization string for the particular device. The index number is the zero based index number of the Device wich can be found using the DeviceCount and Devices Properties.

The initialization string is sent to the faxmodem prior to sending a fax and in the process of setting up for the receipt of faxes.

Access

Read/write

Data Type

String(Array)

DeviceReset Property

Description

Determines the string used to reset a faxmodem

Usage

Visual Basic *FaxControl.DeviceReset*(Index%)[= ResetString\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("DeviceReset", lpstrResetString, iIndex);*
 CString pFaxControl->GetStrProperty("DeviceReset", iIndex);

Remarks

The DeviceReset property allows you to get or set the current reset string for the particular device. The index number is the zero based index number of the Device wich can be found using the DeviceCount and Devices Properties.

The reset string is sent to the faxmodem prior to sending a fax and in the process of setting up for the receipt of faxes.

Access

Read/write

Data Type

String(Array)

FaxCompany Property

Description

The FaxCompany is the name of the Company to which the fax is being sent.

Usage

Visual Basic *FaxControl.FaxCompany*[= FaxCompany\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("FaxCompany", lpcstrFaxCompany);*
 CString pFaxControl->GetStrProperty("FaxCompany");

Remarks

There is no requirement to have a FaxCompany.

Access

Read/write

Data Type

String

FaxRetries Property

Description

The number of times the server should try to send a failed fax before placing it in the failed log.

Usage

Visual Basic *FaxControl.FaxRetries*[= NumberOfTries%]

Visual C++ *BOOL pFaxControl->SetNumProperty("FaxRetries", lTries);*
 long pFaxControl->GetNumProperty("FaxRetries");

Remarks

This item must be set prior to sending a fax and may not be changed after the fax has attempted to send.

Access

Read/write

Data Type

Integer

FaxRetryInterval Property

Description

The number of seconds between times the server should wait before attempting to resend a failed fax.

Usage

Visual Basic *FaxControl.FaxRetryInterval* [= Seconds%]

Visual C++ *BOOL pFaxControl->SetNumProperty("FaxRetryInterval", lInterval);*
 long pFaxControl->GetNumProperty("FaxRetryInterval");

Remarks

This item must be set prior to sending a fax and may not be changed after the fax has attempted to send.

Access

Read/write

Data Type

Integer

FaxUserData Property

Description

The FaxUserData is a user defined field for fax information.

Usage

Visual Basic *FaxControl.FaxUserData*[= FaxUserData\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("FaxUserData", lpcstrUserData);*
 CString pFaxControl->GetStrProperty("FaxUserData");

Remarks

There is no requirement to have fill in this field. The field may be used for any type of user defined data for a fax.

Access

Read/write

Data Type

String

LogCompany Property

Description

Gets the FaxCompany field of a sent fax which should contain the sending company.

Usage

Visual Basic *FaxControl.LogCompany*(Index%)

Visual C++ *CString pFaxControl->GetStrProperty("LogCompany", iIndex);*

Remarks

The LogCompany gets the FaxCompany field from a sent fax. The index number is the FaxID number of the sent fax.

Access

Read-only

Data Type

String(Array)

LogToCompany Property

Description

Gets the UserCompany field of a sent fax which should contain the senders Company Name.

Usage

Visual Basic *FaxControl.LogToCompany*(Index%)

Visual C++ CString *pFaxControl->GetStrProperty*("LogToCompany", *ilIndex*);

Remarks

The LogToCompany gets the UserCompany field from a sent fax. The index number is the FaxID number of the sent fax.

Access

Read-only

Data Type

String(Array)

LogToName Property

Description

Gets the FaxName field of a sent fax which should contain the recipients name.

Usage

Visual Basic *FaxControl.LogToName*(Index%)

Visual C++ CString *pFaxControl->GetStrProperty*("LogToName", *ilIndex*);

Remarks

The LogToName gets the FaxName field from a sent fax. The index number is the FaxID number of the sent fax.

Access

Read-only

Data Type

String(Array)

LogUserData Property

Description

Gets the UserData field of a sent fax which contains user defined data.

Usage

Visual Basic *FaxControl.LogUserData*(Index%)

Visual C++ CString *pFaxControl->GetStrProperty*("LogUserData", *iIndex*);

Remarks

The LogUserData gets the UserData field from a sent fax. The index number is the FaxID number of the sent fax.

Access

Read-only

Data Type

String(Array)

UserFaxNumber Property

Description

The UserFaxNumber is the Fax Number of the Company which is sending the fax.

Usage

Visual Basic *FaxControl.UserFaxNumber* [= FaxNumber\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("UserFaxNumber", lpcstrFaxNumber);*
 CString pFaxControl->GetStrProperty("UserFaxNumber");

Remarks

There is no requirement to have a UserFaxNumber.

Access

Read/write

Data Type

String

UserCompany Property

Description

The UserCompany is the name of the Company which is sending the fax.

Usage

Visual Basic *FaxControl.UserCompany*[= FaxCompany\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("UserCompany", lpcstrUserCompany);*
 CString pFaxControl->GetStrProperty("UserCompany");

Remarks

There is no requirement to have a UserCompany.

Access

Read/write

Data Type

String

UserName Property

Description

The UserName is the name of the person which is sending the fax.

Usage

Visual Basic *FaxControl.UserName*[= SendersName\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("UserName", lpcstrUserName);*
 CString pFaxControl->GetStrProperty("UserName");

Remarks

There is no requirement to have a UserName.

Access

Read/write

Data Type

String

UserVoiceNumber Property

Description

The UserVoiceNumber field is for the voice number of the person which is sending the fax.

Usage

Visual Basic *FaxControl.UserVoiceNumber*[= SendersVoice\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("UserVoiceNumber", lpcstrUserVoiceNumber);*
 CString pFaxControl->GetStrProperty("UserVoiceNumber");

Remarks

There is no requirement to have a UserVocieNumber field.

Access

Read/write

Data Type

String

StatusTime Property

Description

Gets the time the current transmission started.

Usage

Visual Basic *FaxControl.StatusTime*(Index%)

Visual C++ CString *pFaxControl->GetStrProperty*("StatusTime", *ilIndex*);

Remarks

The StatusTime gets the time the current transmission (send or receive) started. The index number is the FaxID number of the fax.

Access

Read-only

Data Type

String(Array)

ServerOption Property

Description

Determines the Option key to use in subsequent calls to the ServerOptionSetting Property.

Usage

Visual Basic *FaxControl.ServerOption*[= OptionKey\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("ServerOption", lpcstrOptionKey);*
CString pFaxControl->GetStrProperty("ServerOption");

Remarks

The ServerOption gets or sets the option key for subsequent calls to the ServerOptionSetting property. Current option keys are stored in the server INI file under the FaxGeneral section. Currently valid keys are AutoShutDown, Receive, ReceiveID, and ReceivePath, although others may be added in the future. The ServerOption property must be set before calling the ServerOptionSetting property of an error will be generated. Using the combination of ServerOption and ServerOptionSetting you can change any of the server settings.

Access

Read/write

Data Type

String

ServerOptionSetting Property

Description

Determines the Option value of the server Key specified in the last call to the ServerOption Property.

Usage

Visual Basic *FaxControl.ServerOptionSetting*[= OptionValue\$]

Visual C++ *BOOL pFaxControl->SetStrProperty("ServerOptionSetting", lpcstrOptionValue);*
 CString pFaxControl->GetStrProperty("ServerOptionSetting");

Remarks

The ServerOptionSetting gets or sets the option value for the key last specified with the ServerOption property. Current option keys are stored in the server INI file under the FaxGeneral section. Currently valid keys are AutoShutDown, Receive, ReceiveID, and ReceivePath, although others may be added in the future. The ServerOption property must be set before calling the ServerOptionSetting property of an error will be generated. Using the combination of ServerOption and ServerOptionSetting you can change any of the server settings.

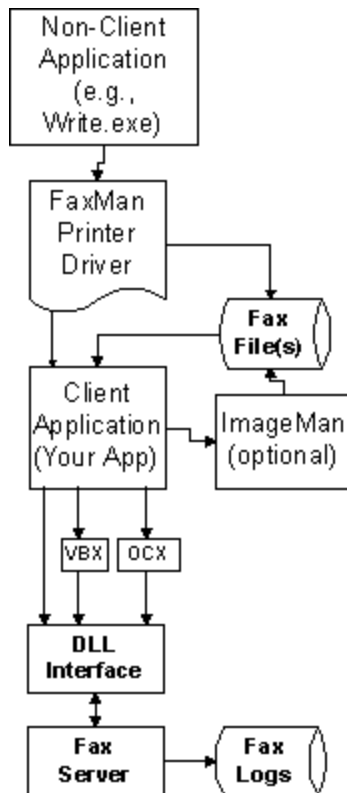
Access

Read/write

Data Type

String

FaxMan Overview



[Installing FaxMan](#)

[Testing the FaxMan Installation](#)

[FaxMan Fax Server](#)


[Interface Layer](#)

[Print Driver](#)

[ImageMan \(a shameless plug\)](#)

[Your Application](#)

[Fax Files](#)

To Follow the Overview in order click on this  on each page

Overview - FaxMan Fax Server

At the heart of this system is the fax server, FAXMAN.EXE. This program is responsible for all communications with the faxmodems in a given machine. The server also sends various notifications to each of the client applications during fax send and receive events.

In addition, the server maintains all fax logs for a given fax system. Each client application must register itself with the server before calling any fax functions (for VBX/OCX users this is handled automatically).

It is critical that the server (FAXMAN.EXE) be located on the path, since its services may be required at any time by any application. The server can be configured through the use of several command-line parameters and configuration settings.



Overview - Interface Layer

The interface layer resides between your application and FaxMan. It actually consists of one or more pieces, depending on which interface your application is using (i.e., direct DLL function calls, VBX, or OCX).

At the most basic level, the interface consists solely of the FAXDLL.DLL library. This is the only interface that is required in every configuration, and in fact the other available interfaces communicate with FaxMan through the FAXDLL library. The server and print driver also make calls into this library, so it will be loaded for pretty much every conceivable FaxMan function.

The FaxMan architecture allows for the replacement of the basic FAXDLL library with a network version which can communicate with a FaxMan server anywhere on the network. This is how the network version of FaxMan is implemented, making it easy for your application to work with either a standalone version of FaxMan or the network version, since both use identical interfaces.

For DLL Users See:

[Tutorial - using the DLL Interface](#)
[The DLL Interface](#)

For VBX/OCX Users See:

[Using The Custom Controls](#)
[The VBX / OCX Interface](#)



Overview - Print Driver

The FaxMan print driver allows your users to create faxes from virtually every Windows application by simply printing. The print driver notifies your application when printing starts and when printing is completed, giving your application the opportunity to set the output path and filename for every printed document.

The trick to all of this lies in the way the print driver is specified in the WIN.INI file; for each application which wishes to be a "print receiver" (i.e., an application which wants to get notified when print events occur) there must be corresponding entries in the [devices] and [ports] sections of WIN.INI. These entries link the print driver selected by the end-user to your application (as you can see from the entries we've placed in WIN.INI during installation to support the FAXTEST.EXE and CPPTST.EXE applications.)

Finally, you should note that the application which receives print notifications does not necessarily have to be the same application which sends the faxes; it is entirely conceivable that you could have several applications communicating with FaxMan (and eachother) simultaneously. The neat thing about FaxMan's server architecture is that it makes all of this extremely easy from an application programmer's (that's you) point of view, since you never need to be concerned with which ports are currently in use or which have faxmodems installed.



Overview - ImageMan (a shameless plug)

Although not technically a part of the FaxMan system, and certainly not a requirement for sending and receiving faxes, using ImageMan in conjunction with FaxMan makes it a snap to perform the high-speed image manipulation that is the hallmark of a good faxing application. The idea to produce FaxMan was inspired by our many ImageMan customers who are using ImageMan for document processing systems that just cried out for good fax support also, so the two products are really designed to be extremely complementary. ImageMan provides the capabilities to:

- Display images in all popular raster formats and some vector formats
- Scale images (with optional antialiasing, what WinFax calls Image Enhancement)
- Rotate images (because a lot of faxes are sent upside down)
- Pan and zoom images
- Store images (in FaxMan fax format and others)
- Convert between different file formats
- And much, much more.



Overview - Your Application

So, what does your application have to do to exist in this world? As it turns out, remarkably little. Here are the things you're responsible for doing:

- Making sure your application is installed correctly for FaxMan operations
- Scheduling faxes
- Displaying any desired status information during send/receive events
- Displaying a list of installed fax devices (if desired)
- Determining the name of any files generated by the FaxMan print driver
- Displaying faxes (if desired)
- Displaying the fax logs in your application (if desired)



Overview -Fax Files

The fax files used by FaxMan are multi-page files (i.e., a single file can contain an unlimited number of fax pages) which are stored in a slightly altered TIFF Group 3 file. We alter the file signature slightly to make things a little less ambiguous; there are a seemingly infinite number of possible variations on the TIFF file format, while we use only one of these for faxing.

To avoid the problem of having developers complain when FaxMan can't fax a 24-bit color image directly ("Well you said FaxMan could send TIFF images!") we opted for our own (basically TIFF) file format called the FaxMan Fax File Format.

This format uses two default extensions: FMF for temporary fax files, and FMP for permanent fax files. The only difference between the two as far as FaxMan is concerned is that when a log entry is deleted from the fax logs, all the FMF files associated with it are also deleted from the system. This helps to keep such transitory things as cover pages, etc., from clogging up the system. If you want to create permanent attachment files, on the other hand, they can simply be given an FMP file extension, and they will remain on disk until you (or your user!) intentionally removes them.

It is important to note that FaxMan can accept any number of files in a single fax event. The files are simply "chained" together at schedule time by separating them with a "+" character, like this: "cover.fmf+attach1.fmp+test.fmf". This makes it extremely easy to keep your sales literature, for example, in separate attachment files (say one file for each product you sell) and then chain together a single fax event which contains information on all the products the client is interested in.

[How to Create Fax Files](#)

[Creating FaxMan Fax Files with the DLL](#)

[Creating FaxMan Fax Files with the VBX or OCX](#)

This completes the Faxman overview tour, you should have a good idea of how the system is designed to work and how much you can do with it. If you'd like to continue exploring the FaxMan system chose one of the topics below.

For DLL Users :

[Tutorial - using the DLL Interface](#)

[The DLL Interface](#)

For Custom Control Users:

[Using The Custom Controls](#)

[The VBX / OCX Interface](#)

Overview - Installing the FaxMan System

To install FaxMan, place the diskette in the drive (we're assuming drive A: here) and type:
a:setup <enter>

The install process will allow you to select where to install the FaxMan development system. You should note that this location must be placed on your path for proper operation. We recommend installing all portions of the FaxMan system at this time, although you are free to install only those portions you wish.



Overview - Testing your FaxMan Installation

After running the installation program, you should perform a simple test to make sure that all of the parts of the FaxMan system have been installed correctly. First, though, make sure you've added the directory containing FaxMan to your system's path and restarted your system. Don't just copy the FaxMan executables to a directory that is on your path. We know it's tempting, but having multiple copies of these files is almost guaranteed to create problems later. Just bite the bullet and reset your system's path. Trust us, it's better this way.

After restarting Windows, start up the Faxtest application shipped with your FaxMan system. Starting this will, in turn, force the FaxMan server to be loaded as an icon (it looks like a globe). Select the Modem option on the Setup menu to Auto Configure the fax server; this will cause the server to inspect your system for installed faxmodems and to configure itself to work with what it finds (you'll notice that the server's globe icon appears to rotate for a second or two during configuration; this rotation occurs whenever the server is communicating with a faxmodem). If this process causes your machine to "hang", there's probably some kind of problem with your Windows communications setup. In this case, see the section about getting tech support and we'll help you track down the problem.

Ok, now let's try to print a document from Windows Notepad using the FaxMan printer driver we installed earlier. To do this, start Notepad and select Print Setup from the File menu. You should have a selection that says "FMFAXDRV on FAXTEST." If you don't, then either you didn't install the printer driver or there was some problem during installation. Try installing it again - if there's still a problem, give our Tech Support line a call and we'll straighten it out for you.

Once you've setup for printing, try printing a document. You should see the following things happen, in this order:

- 1) The Notepad print status dialog will display itself.
- 2) The Send A Fax dialog will be displayed in the Faxtest application

At this point, you can send the fax. Just fill in the destination phone number plus any of the other fields shown in this dialog, then press the Send button. You should see the entry briefly in the Faxtest pending log, then it will disappear and the FaxMan globe icon will begin rotating as the fax is sent. You should also see a dialog box which displays the status of the sending fax (we should point out that this dialog box is displayed by the faxtest application, not the FaxMan fax server. The fax server displays nothing except the spinning globe icon, and even this can be hidden if desired). Upon completion of the fax, it will show up in the completed log or failed log, depending on the results.



Using the DLL Interface - Tutorial

This tutorial section will instruct you on the proper usage of the FaxMan DLL interface. The specific areas to be covered are:

[Registering your Application with FaxMan](#)

[Sending a Fax \(and the many varied options you have\)](#)

[How to get status information from FaxMan](#)

[Examining the FaxMan server from your application](#)

[How to track the status of the FaxMan system, including responding to sending and receiving notifications.](#)

[How to work with FaxLogs](#)

[How to create faxes](#)

Before starting this tutorial, we recommend reading the DLL Function Reference to get an idea of the available functions and a background on how they might be used. When you're done with this tutorial, you should be able to use the FaxMan server in a variety of applications without much trouble. It's really a pretty simple system to work with.

One more note: As you're reading through the code fragments presented here, be sure to read all the comments in the code. In some situations, the code to handle certain aspects of the system will be spread over a couple of locations in your code. The samples clearly document when this is happening, so pay careful attention to them.

Well, without further ado, let's get started!



DLL Tutorial - Registering your Application with the FaxMan system

Before your application can call any functions in the FaxMan system, it must register itself with the server using the [FaxRegisterApp](#) function. Registration is required for a couple of reasons:

Your application needs to specify a unique identification string so the server can find it in order to send all of the event notifications you need. This identification string must be persistent; i.e., it must be identical each time your application is run. If it's not, then your app may not be properly activated to handle print messages, and any faxes scheduled for a future time won't be able to locate your app to send the event notifications. **(NOTE that your application will receive no event notifications unless it specifically requests them by calling the [FaxSubscribe](#) function!).**

The following code fragment illustrates how to register your application:

```
PAPPINFO  pInfo;  
pInfo = FaxRegisterApp("MyUniqueAppID", (WORD)hMyWnd);
```

That's all there is to it. One important point here is that the PAPPINFO returned by the FaxRegisterApp function should be declared as a global or in such a way as to remain available for the duration of your FaxMan usage. Every other FaxMan function call you make will require the PAPPINFO returned from FaxRegisterApp.

The flip side to registering your application before using the FaxMan system is, of course, to un-register it when you're done using the FaxMan system. This is done quite easily also through use of the [FaxUnregisterApp](#) function call. Failure to call the FaxUnregisterApp function before terminating your application could result in significant memory and resource leaks, so please be careful about this.



DLL Tutorial - Sending a Fax

This is the main event; you're now actually ready to send a fax. You'll be amazed, first at how little is required of your application to send a fax, then at the ease with which you can create and send complex groupings of faxes.

Sending a fax is as simple as initializing (via [the FaxInitSendStruct](#) function) and filling out a SEND_FAX structure and passing it to the [FaxSchedule](#) function, like this:

```
SEND_FAX sf;  
FaxInitSendStruct(pInfo, &sf);  
sf.szDestFax = "1-555-555-5555"; //set the destination fax number  
sf.szFileList = "myfax.fmf";      //set the list of fax files to send  
FaxSchedule(pInfo, &sf);          //schedule it!
```

It is a requirement that your application call the FaxInitSendStruct function to initialize the [SEND_FAX structure](#) before filling in any of the SEND_FAX fields.

The only things your application is required to fill in are the destination phone number (szDestFax) and the list of files you wish to send (szFileList), although as you'll discover later you don't even need a file list as long as you include a cover page.

OK, you see what is required to send a fax: not much at all. Now let's start exploring the different options your application has when scheduling a fax. The key to all of this, of course, is the SEND_FAX structure, so let's start by exploring it in detail



DLL Tutorial - The SEND_FAX Structure and You

There is a small section which details the structures used when working with the FaxMan DLL interface. Take a minute to jump over there and examine the [SEND_FAX structure](#), then we'll cover the interesting members of the structure in some detail. OK, now that you've been there and back, we'll get started. The first thing you should notice is that this structure is really divided into two sections:

The top section, which contains the items your application can set before calling FaxSchedule. The bottom part, which contains members filled in by FaxMan during the fax sending process in order to keep your application informed as to the status of the send or receive event. **Your application will never alter any of the items in the bottom half of this structure they are for informational purposes only!**

Now, let's start investigating the structure members you should be concerned with.

[szDestFax](#)
[szFileList](#)
[nYear, nMonth, nDay, nHour, nMinute, nSecond](#)
[nRetries, nRetryDelay](#)
[szLocalID](#)
[nSendRes](#)

The fields above all have some kind of impact on the transmission of the fax. The remaining fields exist only for your application's use in such things as banners and cover pages. They are described briefly below:

[szBanner](#)
[szCover](#)
szDestName - The name of the person is addressed to.
szSubject - The subject of the fax.
szFromLine - The name of the sender of the fax
szComments - Any comments concerning the fax
szFromCompany - The name of the company sending the fax
szToCompany - The name of the company the fax is being sent to
szFromFax - The fax number of the sending fax machine
szFromPhone - The phone number of the person sending the fax
szUserData - A field to be used for whatever purpose your app desires

You should note that all of these fields are virtually unlimited in length, although for practical usage in banners or cover pages you will probably want to limit them to reasonable lengths for your application. Just what constitutes a "reasonable length" will probably differ from application to application.

Now we'll take a look at how your application can communicate with the FaxServer



You've already encountered this member - it's just the phone number of the fax machine you wish to send to. This string can contain any of the numbers you would use to dial a phone number, including such things as commas to create pauses, "*70," to turn off call waiting, and others.

Put simply, this is the list of files you wish to include in this fax. This list can consist of just a single filename, as in the previous tutorial example we showed, or it can be a list of as many files as you wish to chain together. And each of these files can contain as many pages as you like. Effectively, there's no limit to the number of pages a single fax can contain. The following example shows how you can use the "+" character to string together a chain of fax files:

```
SEND_FAX sf;  
FaxInitSendStruct(pInfo, &sf);    //set the destination fax number  
  
sf.szDestFax = "1-555-555-5555";  
sf.szFileList = "coverltr.fmf+myfax.fmf+saleinfo.fmp";  
FaxSchedule(pInfo, &sf);    //schedule it!
```

One thing to be careful of is trying to be sure that each of the files you submit to FaxSchedule is a valid FaxMan fax file. If you try to schedule a file list that contains an invalid file, then the fax will not be scheduled (See the section on the ways to create faxes for more info on what constitutes a valid fax file and how to make them).

Finally, the only time that this member can be left NULL is when you're sending a fax which consists solely of a cover page. The rule is that there must be at least one page to fax in order to be able to schedule a fax successfully, so if there's not at least a cover page then you can't send a fax. (Note that you don't always have to include a cover page, but you must always include either a cover page or fill in the szFileList).

These fields specify the time to send the fax. The year ranges from the present to 2038, nMonth ranges from 1-12, nDay ranges from 1-31 (depending on the month, of course), nHour ranges from 0-23, nMinute and nSecond range from 0-59. As long as the server is running when the time to send the fax arrives, the fax will be sent. Your application does not need to be running in order for the fax to be sent.

These two members, `nRetries` and `nRetryDelay`, specify the number of times to retry the fax on failure and the length of time to wait (in seconds) before attempting to send the fax again. Note that if you set `nRetries` to 3, the fax will be attempted up to 4 times.

This field is used to identify your faxmodem to the receiving fax machine. This string will generally be displayed by the receiving fax machine during transmission. We recommend that you set this field to the same string used as the ReceiveID for consistency, but it is certainly not a requirement. You may fill in up to 20 characters for the zLocalID.

This field specifies the resolution you wish to use to send the fax, either high or low (standard or enhanced). To send in high resolution, set nSendRes to 1; for low resolution, set it to 0. The default is high resolution.

This is used to define the one-line header at the top of each fax page. This line generally contains such mundane items as the date and time of the fax transmission and the Local ID field. This is discussed in more detail in a subsequent section.

This field is used to define a cover page file to be prepended to the transmitted fax. Details on this are also presented in a subsequent section.

DLL Tutorial - Setting Up to Communicate With FaxMan

Example

Before your application can really start working with FaxMan, you'll want to add code to handle the event notifications FaxMan uses to keep your application informed. These events come to your app in the form of Windows messages sent to the HWND you registered with the server in the [FaxRegisterApp](#) function. To deal with these event messages, you'll need to add some code to the window handler for the registered window.

The notification messages sent to your application from the FaxMan server break down into the following groups:

- [Changes in Faxmodem Status \(FAXMODEMSG\)](#)
- [Additions and Deletions from the fax logs \(FAXLOGADD / FAXLOGREMOVE\)](#)
- [Changes in Server Configuration \(FAXSERVERCONFIGMSG\)](#)
- [Requests for output filename at print start time \(FAXGETFILENAME\)](#)
- [Print status messages at print end time \(FAXPRINTMSG\)](#)

Your application won't receive any of these messages unless it first calls the [FaxSubscribe](#) function to notify FaxMan that you want them. In the example is a general idea of the code you would incorporate into your message handler if your application were subscribing to all of the available message types.

Now let's see about checking the system



FAXMODEMMSG messages are sent to notify your app when a given faxmodem changes state (i.e., idle to sending a fax) or when a faxmodem is added to or removed from the FaxMan system.

For example, when a scheduled fax transitions from the pending log to the sending log, your app will be notified of it's removal from the pending log (FAXLOGREMOVE) and its addition to the sending log (FAXLOGADD).

When the fax server changes state, for instance when it goes from Auto Receive ON to Auto Receive OFF (possibly as the result of the actions of another application), your app is notified with one of these messages indicating the server configuration setting which changed and its new value.

When an application starts printing to the FaxMan print driver that's "hooked" into your application, the first notification your app will receive will be this message requesting your app to provide an output filename for the generated fax file.

When printing is completed, your app will receive this message which reports the outcome of the printing process.


```

// First call FaxSubscribe to subscribe to the notifications.
// This can be done anywhere in your code, as long as you do it
// *before* you want to start getting notification messages. Most
// applications will call this as part of their FaxMan
// initialization code (probably right after FaxRegisterApp)
// as shown below...
//
PAPPINFO pInfo;
pInfo = FaxRegisterApp("MyGroovyFaxApp", hMyWnd);
FaxSubscribe(pInfo, SUBSCRIBE_LOG_ALL | //log subscriptions
              SUBSCRIBE_FAXMODEM |     //FAXMODEMSG subscriptions
              SUBSCRIBE_SERVER_CONFIG); //FAXSERVERCONFIGMSGs ...
//
// The print messages will arrive whether you ask for them
// or not if you've configured this application as a
// "print receiver" for the FaxMan print driver.
//
// The following is the text of a window procedure you
// might create to handle the FaxMan notification messages.
//

LONG FAR PASCAL MyWndProc(HWND hWnd, WORD wParam, WORD lParam, LONG
lParam)
{
    PSEND_FAX pFax;
    PPRINTSTAT ps;
    FAXDEVICE *pDev;
    SERVER_CONFIG *pConfig;

// You'll probably include the WM_FAXMSG identifier as part
// of your outermost switch statement. Here we just return
// if the wParam isn't WM_FAXMSG, since we assume this window
// exists only to service FaxMan messages.

if (wParam != WM_FAXMSG) return 0;
switch (wParam) {
    case FAXSERVERCONFIGMSG:
        //lParam points to a SERVER_CONFIG structure
        pConfig = (PERVER_CONFIG)lParam;
        ...
        break;
    case FAXMODEMSG:
        //lParam points to a FAXDEVICE structure
        pDev = (PFAXDEVICE)lParam;
        ...
        break;
    case FAXSENDMSG:
        //lParam points to a SEND_FAX structure
        pFax = (PSEND_FAX)lParam;
        ...
        break;
    case FAXPRINTMSG:

```

```

        //lParam points to a PRINTSTAT structure
        ps = (PPRINTSTAT)lParam;
        ...
        break;
case FAXGETFILENAME:
    //lParam points to a string buffer to be filled in
    //with the output filename.
    break;
case FAXLOGADD:
    //lParam points to a SEND_FAX structure
    pFax = (PSEND_FAX)lParam;
    switch (pFax->wLog) {
        //pFax->wLog tells us which log we're adding to
        case SUBSCRIBE_LOG_PENDING:
            break;
        case SUBSCRIBE_LOG_SENDING:
            break;
        case SUBSCRIBE_LOG_COMPLETE:
            break;
        case SUBSCRIBE_LOG_FAILED:
            break;
        case SUBSCRIBE_LOG_RECEIVE:
            break;
    }
    ...
    break;
case FAXLOGREMOVE:
    //lParam points to a SEND_FAX structure
    pFax = (PSEND_FAX)lParam;
    switch (pFax->wLog) {
        //pFax->wLog tells us which log we're removing from
        case SUBSCRIBE_LOG_PENDING:
            break;
        case SUBSCRIBE_LOG_SENDING:
            break;
        case SUBSCRIBE_LOG_COMPLETE:
            break;
        case SUBSCRIBE_LOG_FAILED:
            break;
        case SUBSCRIBE_LOG_RECEIVE:
            break;
    }
    ...
    break;
}
return 0L;
}

```

As you can see, setting up your application to receive these notification messages is a very straightforward process - it's just like dealing with any other Windows messages. Just remember that you must subscribe to them before you'll receive them.

DLL Tutorial - Examining Your FaxMan System

Now that you know how to register with the FaxMan system and how to handle FaxMan's notification messages, the next step is to do a little exploring. This section will illustrate how to examine your FaxMan system to determine:

- How many fax devices are installed and ready for use (if any!)
- Whether the server is setup for automatic fax reception
- Where received faxes will be stored

You'll also find out how to add fax devices, which you will have to do at least once before FaxMan will work (We recommend adding fax devices as part of your application's installation, but you can just as easily make it a part of your application proper).



DLL Tutorial - Checking for installed Fax Devices

Example

One of the first things your application should do is to call the [FaxEnumDevices](#) function to determine if there are any faxmodems currently configured for use with FaxMan.

Calling FaxEnumDevices causes the FaxMan server to immediately send several FAXMODEMMSG notification messages to your application, one message for each installed device. Note that when the FaxEnumDevices function returns, your application will have already received and processed all of these messages, so it will know which faxmodems are installed before the next line in your application is processed. See the example code fragment that indicates how this might look in your application.

It's a pretty straightforward concept once you get used to the idea of FaxEnumDevices causing FaxMan to call SendMessage for each installed faxmodem. By calling FaxEnumDevices at the beginning of your application, and then responding to all FAXMODEMMSG notification messages, your app can easily keep an up to date list of all installed devices, including which port they're on and what they're currently doing (these things are stored in the FAXDEVICE structure passed in the lParam of the message).



```

//
// We assume that the application has already registered via
// FaxRegisterApp and setup the proper subscriptions using
// FaxSubscribe, as illustrated above.

// The following code would be added to handle the FAXMODEMSG
// send by the FaxMan server to your application. For the purposes
// of this example, imagine it being part of the window proc setup
// to handle FaxMan messages.
//
// The variable nFaxModems referenced below is declared as a
// global int.
//

case FAXMODEMSG:
    pDev = (PFXDEVICE)lParam;
    nFaxModems++;
    break;

//
// Once you've setup your message handler (as above), the
// rest is pretty simple...
//

nFaxModems = 0;      //reset this global variable
FaxEnumDevices(plInfo);    //enumerate the devices

//
// At this point, nFaxModems should contain the number
// of installed faxmodems.
//

if (!nFaxModems) {
    MsgBox(NULL, "No Installed Faxmodems!", NULL, MB_OK);
}else {
    // we have at least one faxmodem...
}

```

DLL Tutorial - Installing Specific Fax Devices

When FaxMan is initially installed, there will be no devices configured for its use. Your application can fix this little problem quite easily, though, through the use of the [FaxAddDevice](#) function.

FaxAddDevice is used to add a faxmodem to the FaxMan system. Your application calls this function and passes it the port number you wish to add the device on. FaxMan will then query the faxmodem on that port and configure it for future use, this process normally takes about 1 to 2 seconds to complete.

If the device on the specified port is not a valid faxmodem, or if some other condition causes FaxMan to be unable to configure the port, then FaxMan is smart enough to exclude the port from use. This means that your application can attempt to add faxmodems on any port it wishes without fear of adding an invalid device for FaxMan's use.



DLL Tutorial - Auto Configuration of All Faxmodems

As an added bonus, the [FaxAddDevice](#) function can also be used to cause the FaxMan server to automatically detect and configure all installed faxmodems. This is accomplished by passing a 0 as the port number to add. Doing this causes the server to inspect each installed comm port to detect any faxmodems. Plus, FaxMan can configure all these devices simultaneously, so it should still only take a second or two to complete (In our testing, we've been able to configure up to 9 comm ports within two seconds using a multi-port card).

As FaxMan adds each port to the system, it will notify your application with a FAXMODEMSG for the port. Unlike the [FaxEnumDevices](#) function call, however, the configuration process will continue after the FaxAddDevice function returns; in other words, after calling the FaxAddDevice function, you'll have to wait until configuration is completed to be sure that all devices have been reported to your application (as we stated above, this normally takes a couple of seconds).



DLL Tutorial - Inspecting the Server Configuration

[Get Example](#)

[Set Example](#)

There are a several other server configuration items your application may want to check out during initialization or other times. These are accessed and changed through the [FaxGetConfig](#) / [FaxSetConfig](#) functions. The configuration items currently used by the server are:

AutoReceive - When set to ON, your FaxMan system will automatically begin listening for incoming calls on all faxmodems that have been configured to receive faxes.

ReceiveID - This is the character string used to identify your faxmodem when receiving faxes. This string, which is limited to a maximum of 20 characters, will be transmitted to the calling fax machine during the initial negotiation phase of the fax process.

AutoShutDown - When set to ON, the server (FAXMAN.EXE) will automatically close itself about a minute after all faxmodems enter the idle state. Note that if AutoReceive is ON and any faxmodems are configured for reception then this state will never be reached, and the server will never automatically shutdown. This feature allows you to minimize memory usage when FaxMan isn't required (although FaxMan does not have a large memory footprint).

ReceivePath - This item specifies the path for received fax files to be placed in. If this string is not present or specifies an invalid path then all faxes will be placed in the same directory as the FaxMan server (FAXMAN.EXE).

To retrieve the setting of any of these configuration items, your app can call the FaxGetConfig function. Setting the configuration options is just as straightforward and is done with the FaxSetConfig function. See the Get Example and Set Example code fragments on how to access these settings.




```

char buf[255];

//
// If the first FaxGetConfig function returns 0, that means
// we couldn't load the FaxMan server for some reason, and
// we should report an error.
//

if (FaxGetConfig(pInfo, "AutoReceive", buf, 255)) {
    // now we can check the other config items without
    // fear of not finding the server

    FaxGetConfig(pInfo, "ReceiveID", buf, 255);
    FaxGetConfig(pInfo, "AutoShutDown", buf, 255);
    FaxGetConfig(pInfo, "ReceivePath", buf, 255);

    // In the real world, of course, I'd store these values
    // somewhere, but this is just an example, so I don't have to!
}
else {
    //report an error
}

```

This is a pretty straightforward operation, similar to extracting the settings from an INI file.

```
//Here's a sample function that will toggle the current AutoReceive state:
```

```
int ToggleAutoRx(void){  
    char buf[255];  
    if (!FaxGetConfig(pInfo, "AutoReceive", buf, 255)) return 0;  
    if (!strncmpi(buf, "ON")) FaxSetConfig(pInfo, "AutoReceive",  
"OFF");  
        else FaxSetConfig(pInfo, "AutoReceive", "ON");  
    return 1;  
}
```

```
// As you can see, there's nothing complex about altering  
// the server configuration settings.
```

DLL Tutorial - Banners & Coverpages

Banners

FaxMan allows your application to specify a banner, or single line of text at the top of each fax page, for the purposes of placing information such as the date and time of the fax transmission or the ID of the fax sender on the fax.

Most of the fields in the SEND_FAX structure can be placed in this banner string through the use of place markers similar to those you would encounter in a C printf command. For a detailed description of how to format the banner string, see the section on [Banner & Cover page Format Characters](#).

Cover Pages

The FaxMan system also allows your application to specify a cover page file which FaxMan uses to generate a cover page for the fax. A unique cover page file can be used for each fax sent. The format of the cover page file allows for the inclusion of all the fields that can be included in a banner. See the appendix on cover page files for more information.



DLL Tutorial - Tracking the Status of Faxes

OK, so you've filled in all the SEND_FAX fields to your liking and scheduled a fax. Now how do you know what's happened to it? Well, as you've already seen, you'll need to subscribe to the proper notification messages (using [FaxSubscribe](#)) to keep up with the situation. The notification messages you should subscribe to are the SUBSCRIBE_LOG_SENDING messages.

The reference section describing the FAXSENDMSG notification message gives a good account of what order these messages are sent in and how to write code to handle them.



DLL Tutorial - Dealing with Fax Logs

Example

The FaxMan server keeps all of the information contained in the [SEND_FAX structure](#) for each fax sent or received through the server. These items are stored in the fax log, a file called FAXLOG (with no extension) in the same directory as the server file (FAXMAN.EXE). As described in the section covering the FaxMan server, there are five logical log groupings within this file:

[Pending Log](#)
[Sending Log](#)
[Completed Log](#)
[Failed Log](#)
[Received Log](#)

Any time a fax is added or removed from any of these logs, your application will get a notification message (assuming, of course, that it has subscribed to this event message using [FaxSubscribe](#)). So, for example, when an item goes from the Pending Log into the Sending Log, your app will get two notifications: one to tell it that the item was removed from the Pending Log and one to tell it that an item was added to the Sending Log. For more information on these notification messages, see the reference section on the FAXLOGADD/FAXLOGREMOVE notification messages.

OK, so you've got your app all primed and ready to receive these wonderful log notification messages, and your application is now loaded-how does your application know what the current state of the logs is, before any notification messages are sent? The answer to this is, of course, to use the log iteration functions, [FaxLogInit](#) and FaxLogNext. These will allow your application to step through each log one item at a time and retrieve its' log entries. The Example code fragment illustrates how to do this for the sending log; the code for the other logs is identical except for the FaxLogInit function which starts off the whole process.

The remaining log functions are straightforward: [FaxLogFind](#) will return the information associated with a given fax ID from the FaxMan logs and [FaxLogDelete](#) will delete a log entry from the log. Note that you must pass the proper log for FaxLogDelete to work properly; this information can be retrieved via FaxLogFind.



The Pending Log contains faxes that have been scheduled but not sent. This also includes faxes that have been rescheduled due to some type of problem with transmission.

The Sending Log contains faxes that are currently sending. Note that there may be more than one of these at any given time, depending on the number of faxmodems installed in your system.

The Completed Log contains faxes that were successfully sent.

The Failed log contains faxes that were not successfully transmitted.

The received Log - all received faxes are placed in this log, whether they were successfully received or not.

```
SEND_FAX    tmpsf;
if (FaxLogInit(pInfo, SUBSCRIBE_LOG_SENDING) < MAXITERATORS) {
    while (FaxLogNext(pInfo, &tmpsf) == LOGERR_CONTINUEIT) {
        //add the fax info to our log here
    }
}
```

Note that the FaxLogNext function assumes that the FaxLogInit function has been called previously to setup the iteration. Never call FaxLogNext unless you've first called FaxLogInit to properly initialize the iteration.

DLL Tutorial - How to Create Faxes

OK, now that you know all about sending and receiving faxes, you're probably beginning to wonder how to create them. Well, there are actually three ways of creating fax files which will work with FaxMan, and we'll cover each in detail.

[Use the FaxCreate function](#)

[Use ImageMan to save a fax file directly](#)

[Use the FaxMan printer driver to create faxes by printing from any Windows application](#)



DLL Tutorial - Creating Fax File with FaxCreate

Example

The FaxCreate function allows your application to easily convert monochrome TIFF, PCX/DCX, BMP, and even text files into FaxMan fax format. You can combine as many of these files as you want in whatever order you wish into a single output file.

The FaxCreate function will handle rasterizing text files into faxable pages, scaling down monochrome images that are too large to be faxed, and adding whitespace to the edges of any monochrome images that are too small to be faxed. It will also include all pages of a multiple-page input image in the output image.

To combine multiple files, simply separate them with a '+' character, as shown in the example code.



```
WORD wResult;  
wResult = FaxCreate(pInfo, "salesfig.txt+mychart.tif+myscan.pcx",  
"outfax.fmf", 0L);  
if (wResult) {  
    // the FaxCreate failed. The return value wResult contains the  
    // index of the input file which caused the problem.  
}
```

NOTE: The FaxCreate function takes as its last parameter a DWORD. This is currently not used, and should be set to 0.

DLL Tutorial - Creating Fax Files With ImageMan

If you're using the ImageMan imaging library with FaxMan, you can use it to convert images of all types (not just monochrome) into FaxMan fax files. You can also use ImageMan to give your app access to a wider range of images for input, and a wider range of images for output also. Rotation, color reduction, color inversion, and scaling are some of the image manipulations possible with the ImageMan library. With ImageMan your application can create a comprehensive fax viewer and editor, giving you the ability to load, alter, and store fax images.



DLL Tutorial - How to Create Fax Files with the FaxMan Printer Driver

Example

This is the option your users will probably utilize most often for creating faxes; it is by far the easiest to work with. You should take a moment to review the section on the [FaxMan printer driver](#) before continuing with this explanation.

To enable an application to work with the FaxMan printer driver, your application needs to do several things, as follows:

- 1) Install the printer driver for use with Windows
- 2) Install the proper entries in the WIN.INI file to enable the printer driver to communicate with your application, including the ability to start your application if it is currently not running.
- 3) Add code to handle the communication with the printer driver.

Since the communication will take place through the use of Windows SendMessage functions, you'll be adding code to the WM_FAXMSG handler in your app. Since the first two items have previously been covered in this manual, we'll devote our attention to the third: how should your application respond to the print messages sent to your app by the printer driver?

The print driver will send notification messages to your application (if it's registered properly) at both the start and end of printing. The first message will simply ask your application to provide an output filename for the fax file generated from the print job. The second message will notify your application of the status of the print job, including the number of pages generated and whether it completed successfully (the actual structure passed to your application is a PRINTSTAT structure). The Example code shows how an application might respond to these messages.

Like most FaxMan programming situations, this is a pretty straightforward affair. The one thing you should **note: Do not display a dialog box of any kind in response to the FAXGETFILENAME message. Due to re-entrancy problems with the Windows dialog management code, doing so will crash Windows. Seriously.** If you wish to display a dialog box, do so in response to the FAXPRINTMSG, as shown in the Example code fragment.

As you can see, FaxMan gives your application considerable control over the creation and sending of faxes. If you have any further questions regarding this tutorial, please contact our technical support line. This concludes the DLL Interface Tutorial.


```

//
// It assumed that this code is a part of your WM_FAXMSG handler.
// The case statements are part of a switch (wParam) statement.
//

case FAXGETFILENAME:
    //this message is sent at print start time
    //
    // This code uses the GetTempFileName Windows API
    // function to generate an output filename...
    //
    // lParam is an LPSTR which points to a buffer which your
    // app must fill in with the name of the output file.
    //

    GetTempFileName(0, "FAX", 0, (LPSTR)lParam);
    nLen = STRLEN((LPSTR)lParam);
    ((LPSTR)lParam)[nLen - 3] = 0;
    STRCAT((LPSTR)lParam, "fmf");
    STRCPY(cFileBuf, (LPSTR)lParam);
    SetDlgItemText(IDC_FAXFILE, cFileBuf);
    break;

case FAXPRINTMSG:
    // this message is sent at print end time
    //
    // The following code is extracted from the CPPTST C++
    // application. It uses the PRINTSTAT structure members
    // to display a dialog which allows the user to send the
    // fax immediately if they desire.
    //

    ps = (PPRINTSTAT)lParam;
    if (ps) {
        SendDlg sd(AfxGetMainWnd());
        sd.sf.szFileList = new char[STRLEN(ps->FileName)+1];
        STRCPY(sd.sf.szFileList, ps->FileName);

        // It's the user's responsibility to delete the pointer
        // passed by the driver
        GlobalFreePtr(ps);
    }
    break;

```

Message Reference - Introduction

The Message Reference documents the messages sent from FaxMan to your application. Each message is identified by its value as defined in FAXDLL.H.

These messages are sent to your application through the SendMessage API function call. The wParam parameter is always WM_FAXMSG, and the lParam corresponds to the identifiers documented in this reference section. The lParam will differ according to the wParam message identifier.

You should be aware that any pointers passed to your application in these notification messages are for reference only. Do not alter them in any way!

Finally, remember that you won't receive any of these notification messages unless you specifically request them using the [FaxSubscribe](#) API function call.

[FAXGETFILENAME](#)
[FAXLOGADD / FAXLOGREMOVE](#)
[FAXMODEMSG](#)
[FAXPRINTMSG](#)
[FAXSENDMSG](#)
[FAXSERVERCONFIGMSG](#)



Message Reference - FAXGETFILENAME

IParam - LPSTR which points to the buffer to place the output filename in.

The FAXGETFILENAME message is sent when any Windows application begins printing using the FaxMan printer driver configured to communicate with your application. It is your application's responsibility to place a valid output filename in the buffer pointed to by IParam. Although it may be desirable to display a dialog at this time which would allow your application's users to select the output filename or set other parameters, you should note:

Due to re-entrancy problems with the Windows dialog box functions, your application cannot display anything at this point! You must wait until you receive the FAXPRINTMSG message after printing is completed to display any kind of dialog box.

Since there may possibly be many applications installed which can function with the FaxMan printer driver, you may be interested in how FaxMan determines which application to communicate with when printing begins.

The process begins when the end-user uses the Print Setup option of the application they're using (which may be your application or any other Windows app) to select the output device they desire. If they make no changes using Print Setup then the current Windows default printer, as defined in Control Panel, will be used. Entries which use the FaxMan printer driver will appear as

FMFAXDRV on AppName

Where AppName is the name used to identify the application in the FaxRegisterApp function call.

For instance, to support the CPPTST sample application provided with FaxMan, the following lines are placed in the WIN.INI File:

```
[FaxMan,CPPTST]
Application=c:\faxman\cpptst\cpptst.exe
Y Resolution=98
```

To print using FaxMan and the CPPTST application, you would select the output device labeled FMFAXDRV on CPPTST

When printing begins, the FaxMan printer driver first checks to see if the application CPPTST is registered with the FaxMan system (i.e., is the application we need to communicate with currently loaded?). If not, the driver then looks in the WIN.INI file for an entry matching [FaxMan,CPPTST], gets the path to the application's executable file from the 'Application=' entry, then loads the application. Once loaded (and registered with FaxRegisterApp!) the driver can then communicate with the application by sending the FAXGETFILENAME message at print start time and the FAXPRINTMSG at print end time.



Message Reference - FAXLOGADD / FAXLOGREMOVE

IParam - PSEND_FAX The wLog member of the SEND_FAX struct indicates the log we're adding/removing from.

Example

This message is sent to your application when entries are added or removed from any of the FaxMan logs. This message will allow your application to track fax items as they move from log to log, enabling you to keep an up to date log display on-screen if desired.

The Example code fragment illustrates how use the wLog member to determine which log is being altered.



```
//This code is assumed to be in response to a  
//FAXLOGADD/FAXLOGREMOVE notification message.
```

```
PSEND_FAX pFax;  
pFax = (PSEND_FAX)lParam;  
switch (pFax->wLog) {  
    case SUBSCRIBE_LOG_PENDING:  
        break;  
    case SUBSCRIBE_LOG_SENDING:  
        break;  
    case SUBSCRIBE_LOG_COMPLETED:  
        break;  
    case SUBSCRIBE_LOG_FAILED:  
        break;  
    case SUBSCRIBE_LOG_RECEIVED:  
        break;  
}
```

```
// you would place the code for your application's desired  
// response to this message in each case statement.
```

DLL Message Reference - FAXMODEMSG

IParam - PFAXDEVICE which describes the device that has been added/alterd/removed.

Example

This message is sent to your application when a faxmodem is added to or removed from the FaxMan system, or when the status of an installed faxmodem changes. Your application can use this message to keep an up to date display containing a list of all currently installed faxmodems and their status.

When the FaxMan server is loaded, it will send one of these messages for each installed faxmodem. Your application can use the [FaxEnumDevices](#) function to force the server to send a list of installed faxmodems, and their current status, at any time. The Example code illustrates how an application could respond to the FAXMODEMSG.

Note that if the wFlags member of the FAXDEVICE structure is 0, then FaxMan is removing this faxmodem from the faxmodem pool.



```

// This code simply outputs a string displaying the current
// status of the given faxmodem, including whether it's sending
// or receiving, the comm port it's installed on, whether it's
// configured for sending, receiving, or both, and the
// initialization and reset strings defined for this device.

PFAXDEVICE pDev;
char jbuf[100];
LPSTR p1, p2, p3;
...
    case FAXMODEMSG:
        pDev = (FAXDEVICE *)lParam;
        if (!pDev->wFlags) {
            sprintf(jbuf, "Com%d\tDeleting!\r\n", pDev->nPort);
        } else {
            if (pDev->wFlags & PORT_SENDING)
                p1 = "SENDING";
            else if (pDev->wFlags & PORT_RECEIVING)
                p1 = "RECEIVING";
            else p1 = "IDLE";

            if ( (pDev->wFlags & PORT_SEND) && (pDev->wFlags &
PORT_RX) )
                p2 = "Send/Receive";
            else if (pDev->wFlags & PORT_SEND)
                p2 = "Send";
            else p2 = "Receive";

            if (pDev->wFlags & PORT_CLS20)
                p3 = "Class 2.0";
            else if (pDev->wFlags & PORT_CLS1)
                p3 = "Class 1";
            else p3 = "Class 2";

            wsprintf(jbuf, "Com%d\t%s\t%s\t%s\t%s\t%s\r\n",
pDev->nPort, p1, p2, p3, pDev->szReset, pDev->szInit);
        }
        OutputDebugString(jbuf);
    }
}

```

Message Reference - FAXPRINTMSG

IParam - PPRINTSTAT which contains the status of the completed print job.

Example

When the FaxMan printer driver has completed printing a document, your application will receive a FAXPRINTMSG containing the results of the printing process. The PRINTSTAT structure pointed to by the IParam passed with this message contains the number of pages printed, the full path and filename of the output file generated, and the status of the print job.

The Example code fragment shows how to work with the FAXPRINTMSG message

NOTE

You **MUST** call GlobalFreePtr() when you finish to free the memory that was allocated.




```
PPRINTSTAT ps;
ps = (PPRINTSTAT)lParam;
switch (ps->pStat) {
    case PRN_OK:
        //printing completed OK
        break;
    case PRN_ABORT:
        //aborted by user - output file is incomplete
        break;
    case PRN_DISKFULL:
        //full disk error - output file is incomplete
        break;
    case PRN_ERROR:
        //some other undefined error
        break;
}
```

Message Reference - FAXSENDMSG

IPParam - PSEND_FAX which points to a SEND_FAX structure containing the up to date information regarding this fax.

Example

The FAXSENDMSG is sent to your application during fax send/receive events to keep your app informed of a fax's progress. By subscribing to this message stream, your application can easily display the progress of every fax sent and received. Each time the FAXSENDMSG is sent, the underlying SEND_FAX structure will have changed to reflect the new status of the fax. The easiest way to follow the progress of a fax is to follow the SEND_FAX's fs member, which reflects the current FAXSTATE. The Example code shows how this is done.

The next question you might ask is "Sure, it gives me the status events, but how do I know what has changed at each event, and what order can I expect them to come in?" Well, the next two tour stops should give you a better idea what's happening with each status event (these entries correspond to the FAXSTATUS enumeration defined in FAXDLL.H).



```
PSEND_FAX pFax;
pFax = (PSEND_FAX)lParam;
switch (pFax->fs) {
    case FAXST_ANSWERING:
        break;
    case FAXST_RX_NEGOTIATE:
        break;
    case FAXST_RXDATA:
        break;
}

// you can include case statements for all of the faxstates
// you wish to track...
```

Message Reference - FAXSENDMSG - Sending Status Event

FAXST_INIT - FaxMan is initializing the sending structures. Can be ignored.

FAXST_SEND_INIT - The faxmodem is being initialized. If an error occurs at this stage, the faxmodem is probably not responding at all. It is probably a good idea to present a dialog to your users asking them to verify that the faxmodem is turned on and working properly.

FAXST_SEND_DIALING - The phone is off-hook and dialing.

FAXST_SEND_WAIT_FCON - Dialing is completed and we're waiting for a response from the remote faxmodem.

FAXST_SEND_FCSI - We're connected to the remote fax and have received its ID string. At this point, the szRemoteID field is filled in with the remote fax's ID string.

FAXST_SEND_FDCS - We've successfully negotiated with the remote fax machine and have established this session's parameters. The fields dwSpeed and nFaxRes now reflect the values for this fax session.

FAXST_SENDING - We are now actually transmitting the fax data.

FAXST_PAGE_END - We have completed sending a page of data. At this point, depending on whether there are more pages to be sent, we will receive either FAXST_COMPLETE or return to FAXST_SENDING.

FAXST_COMPLETE - The fax has been completed.

FAXST_PORTSHUT - The faxmodem has been freed and all activity for this fax is now finished. The comm port is not released until this notification is received. You should also check the FAXERROR member, fe, to determine whether the fax went through successfully or not. If it did not, the nHangCode member will generally indicate what happened.



Message Reference - FAXSENDMSG - Receive Status Event

FAXST_INITRX - The faxmodem is being initialized for receiving a fax. Until the FAXST_ANSWERING notification is received the faxmodem has not yet received a call. If there is an error at this point, the faxmodem is probably not responding at all, and the end-user should be instructed to make sure the faxmodem is turned on and is installed properly, etc...

FAXST_WAITFORRX - The faxmodem was initialized successfully and is not waiting patiently for a call.

FAXST_ANSWERING - The faxmodem is now receiving a call.

FAXST_RX_NEGOTIATE - The faxmodem has successfully negotiated the session parameters. The dwSpeed and nSendRes structure members are now valid, as is the szRemoteID string.

FAXST_RXDATA - We're receiving page data from the remote fax.

FAXST_RX_PAGE_END - We've completed page data reception. At this point we may return to the FAXST_RXDATA stage or we may continue to the FAXST_COMPLETE stage, depending on whether there are more pages to be sent.

FAXST_COMPLETE - We're done with the receive session.

FAXST_PORTSHUT - We've now closed the faxmodem and freed the comm port. You should check the FAXERROR member, fe, to determine if the fax was successful. If it was not, the nHangCode member will probably indicate why it was not.



Message Reference - FAXSERVERCONFIGMSG

IParam - PSERVER_CONFIG which contains the changed server configuration item and its new value.

This message is sent to your application when a server configuration option is changed. You may use these messages to maintain a display of the current server configuration.

This completes the Message Reference Tour ->[Contents](#)

Using the VBX / OCX Custom Controls

The FaxMan VBX and OLE controls provide an easy way to harness the functionality of the FaxMan server from environments that support VBXs or OCXs. These controls provide the same functionality and are different only in their underlying architectures. The properties and events are the same so we will refer to both types of controls as the FaxMan control.

The controls provide two basic functions, first, they provide a visual log interface which displays the FaxMan log entries and they also provide a programmatic interface for creating, scheduling, receiving faxes and configuring the FaxMan server.

[Adding the Custom Controls to your Application](#)

[Setting up the Control](#)

[Configuring and Displaying FaxModems](#)

[Sending a Fax](#)

[Displaying Status Information](#)

[Specifying a Fax page Banner](#)

[Converting Files to Fax Format](#)

[Configuring the FaxMan Server Options](#)

[Receiving Faxes](#)

[Using the Print Driver](#)

[Configuring the Log Display](#)

[Accessing Log Information](#)

[Deleting Log Entries](#)

[Runtime Errors](#)



Custom Controls - Adding

Adding the FaxMan VBX control to your application

The name of the FaxMan VBX control is FAXMAN1.VBX and it should be located in your Windows system directory. The FAXMAN1.LIC file must also be located in your Windows system directory for the control to be used in development mode. The license file is not required for running your application in runtime mode and may not be distributed with your applications.

Adding the FaxMan OCX control to your application

The OCX control should have been installed and registered on your system during the installation process.



Custom Controls - Setting up the Control

Once you have added the control to your project, you'll need to set some basic properties. The [AppName](#) property should be set to an ID string which uniquely identifies your application. This string is used primarily when you'll be creating faxes using the FaxMan printer driver. This ID allows the printer driver to notify your application when a print job has started so you can provide a filename for the print job. See the [FaxMan Printer Driver](#) section for more information on using it. This property should be set at design time.

The [FaxID](#) property should be set to a string which identifies your Fax system when sending and receiving faxes. The string may be up to 20 characters in length.



Custom Controls - Configuring and Displaying FaxModems

The first time you run your application after installing FaxMan, you'll need to configure one or more faxmodems for FaxMan to use. There are two ways to add devices to the FaxMan server. The easiest method is to have the FaxMan server query a system's installed comm ports looking for supported modems. Alternatively, you can specify that the FaxMan server add a modem on a specific comm port.

To have FaxMan query the installed faxmodems, set the [Action](#) property of the FaxMan control to a value of 1(FAX_ENUMERATE). This will cause the server to look for faxmodems on all the installed comm ports. After setting this property you may query the DeviceCount property to see how many modems where found. If DeviceCount is zero then no modems where found. If this happens and you are sure a faxmodem is installed, check Appendix B in the printed manual or [Common Problems, Symptoms and Fixes](#) for hints on how to correct the problem.

FaxModems may also be configured for use by FaxMan by calling the [AddDevice](#) method. This function requires the number of the comm port on which the faxmodem is installed.

Once you have configured one or more faxmodems on the server you may use the [Devices](#) property to obtain information about them. The Devices property is an array of strings which describes each configured modem. The returned string will display the Modem Class and the commport on which it is installed, for instance, 'Class 1 Modem on Port 1.'

To display a list of all the configured devices you can use a loop like this:

```
Dim I%For I=0 to FaxMan1.DeviceCount-1
    Print FaxMan1.Devices(I)
Next I
```



Custom Controls - Sending a Fax

Once the FaxMan server has been configured you are ready to send faxes. Faxes may be created in two ways, using either the FaxMan printer driver or importing TIFF, BMP, PCX, DCX or TXT files.

After creating a fax using either method you are ready to send it by setting the appropriate properties in the control.

At a minimum you'll need to set: the [FaxNumber](#) and either [FaxFiles](#) or [FaxComments](#) properties. You may optionally set these properties:

| | |
|---------------------------------|-------------------------------|
| FaxBanner | FaxCoverpage |
| FaxDate | FaxTime |
| FaxSubject | FaxName |
| FaxCompany | UserName |
| UserCompany | UserFaxNumber |
| UserVoiceNumber | |

Once you have set the required properties you can schedule the fax by setting the [Action](#) property to a value of 0 (FAX_SEND). This will call the server to schedule the fax and will return a unique ID for this fax in the [FaxLogID](#) property. If the fax cannot be scheduled a runtime error will be generated. If you didn't use the FaxDate or FaxTime properties to change the Date and Time it should be sent, the fax server will attempt to send the fax immediately. For instance, to send a fax immediately:

```
FaxMan1.FaxNumber = "1-704-682-0025"  
FaxMan1.FaxFiles = "faxtest.fmf"  
FaxMan1.Action = 0
```

To send a fax consisting of just a coverpage your application will need to specify a coverpage file in the FaxCoverpage property and also set the FaxComments property.

The FaxMan server is also capable of sending multiple fax files in one fax, this is done by setting the FaxFiles property to the names of the faxfiles delimited with '+' character. To send the files, fax1.fmf and fax2.fmf, you'd set the FaxFiles property to 'fax1.fmf+fax2.fmf'. Note that the files you specify must already be in fax format otherwise the fax will not be scheduled.



Custom Controls - Displaying Status information

The FaxStatus event is fired by the control when the status of any fax device changes. You can use this information to display a dialog when faxing is occurring or to maintain a realtime fax status display. These properties contain information about the current status of the specified fax device. The Status properties are:

- [Status](#)
- [StatusConnectSpeed](#)
- [StatusDate](#)
- [StatusDestination](#)
- [StatusFiles](#)
- [StatusFrom](#)
- [StatusID](#)
- [StatusNumber](#)
- [StatusPages](#)
- [StatusPagesSent](#)
- [StatusPercentage](#)
- [StatusRecipient](#)
- [StatusRemoteID](#)
- [StatusResolution](#)
- [StatusSubject](#)
- [StatusTime](#)

Each of these properties is indexed from 0 to the value of the DeviceCount property minus one. The Device parameter of the event can be used as an index when accessing these properties.



Custom Controls - Specifying a Fax page Banner

The [FaxBanner](#) property allows you to specify the text that is printed on the top of each fax page. This text generally specifies the sender, date/time and perhaps the page number.

The FaxMan server allows you to specify exactly what information you want to print in the page banner. This is done by setting the FaxBanner property to a string which indicates the data you wish to print and its placement. You can specify data be printed at the left side, center or right side of the page.

You may specify two types of data, actual text or a field to be printed. The allowable fields are:

| Placeholder | Description |
|-------------|----------------------|
| %d | Date |
| %t | Time |
| %r | Fax Recipient's Name |
| %y | Recipient's Company |
| %s | Fax Sender's Name |
| %i | Sender's Fax ID |
| %u | Subject of Fax |
| %f | Fax Number |
| %o | Comments |
| %u | Subject |
| %x | Senders Fax # |
| %h | Senders Voice # |
| %m | Senders Company |
| %c | Current Page Number |
| %p | Total Pages in Fax |

To print the Recipients name, Senders name & date on each page you can set the FaxBanner property like this:

```
FaxMan1.FaxBanner = 'To: %r|From: %s|%d'
```

The vertical bar characters '|' are used to specify how the text is located. The text to the left of the first vertical bar will be left justified, the text between the two vertical bars will be centered and the remaining text will be right justified. You may specify text for any or all of the three areas.

Setting the FaxBanner property is optional and the default is no page header.



Custom Controls - Converting Files to Fax Format

The FaxMan Controls support converting files in TIFF, PCX, BMP, DCX and ASCII TXT formats into FaxMan fax files. Only black and white image files are supported, if you need to fax color files you can use our [ImageMan](#) product to load and convert them.

To convert files you must set the InputFiles property to the names of the input files, specify the fax filename in the [FaxFiles](#) property and then set the [Action](#) property to a value of 4 (FAX_IMPORT). You may specify multiple files in the InputFiles property by delimiting the filenames with '+' signs. Only one filename may be specified in the FaxFiles property though when converting.

```
FaxMan1.InputFiles = "c:\sample.tif+c:\cover.tif"  
FaxMan1.FaxFiles = "c:\faxes\temp.fmf"  
FaxMan1.Action = 4
```



Custom Controls - Configuring the FaxMan Server Options

By using the `ServerOption` and `ServerOptionSetting` properties you can query and set all the FaxMan server option settings such as `AutoReceive`, `ReceivePath` and `AutoShutdown`.

To query a setting, set the `ServerOption` property to the name of the option you wish to query and then read the value from the `ServerOptionSetting` property.

To set an option, set the `ServerOption` property to the name of the option you wish to set, then set the `ServerOptionSetting` property to the new value.

For instance to set the `AutoReceive` option to On, your code would look like this:

```
FaxMan1.ServerOption = "AutoReceive"  
Faxman1.ServerOptionSetting = "ON"
```

To query the `ReceivePath` option, you code would look like this:

```
FaxMan1.ServerOption = "ReceivePath"  
Print "The ReceivePath is ", FaxMan1.ServerOptionSetting
```



Custom Controls - Receiving Faxes

The FaxMan server default for the AutoReceive option is OFF. To enable fax receiving you'll need to set this option to ON using the [ServerOption](#) and [ServerOptionSetting](#) properties.

Once the server has been configured for receiving, the fax control will generate [FaxReceived](#) events specifying that a fax was received and its filename. A log entry will be create for each received fax in the Receive log. Once the entry is in the log, your application can use the LogXXX properties to gather information about the fax.



Custom Controls - Using the Print Driver

Using the FaxMan printer driver to create fax files is just a matter of responding to the [PrintStart](#) event and providing a filename to the driver for the fax file. **In the current implementation of the VBX control you must not display a modal dialog during the PrintStart event.** You may display a dialog during the [PrintComplete](#) event.

The following code responds to the PrintStart event:

```
Sub ctlName_PrintStart( Index as Integer, PrintFileName as String )      PrintFileName = 'c:  
\printfile.fmf'  
End
```

You must set the AppName property in the control at development time to the same name as the device the printer is on. Be sure to read the [FaxMan Printer Driver](#) section for more information on how to configure the printer driver in Windows.



Custom Controls - Configuring the log display

The Visual Log display in the FaxMan control is highly configurable. You can programmatically specify which columns are displayed, their order and widths. Your end users can also use a mouse to drag and drop columns and resize the columns. Right clicking the mouse over the column headers will also allow end-users to select which columns are displayed. The [ColWidthChanged](#) event will be fired when the user resizes a column in the control. The [ColumnName](#) and [ColumnWidth](#) properties control the order and width of the columns which are displayed.



Custom Controls - Accessing the log Information programmatically

In addition to providing a visual display of the fax logs, the FaxMan controls allow programmatic access to the log data as well. By specifying which log is to be queried by setting the Log property, the LogXXX properties listed below can be queried. These properties are arrays which are indexed from zero to the value of the LogEntries property minus 1.

LogXXXProperties

| | |
|-------------------------------------|--------------------------------------|
| <u>LogComments</u> | <u>LogCompany</u> |
| <u>LogCoverPage</u> | <u>LogDate</u> |
| <u>LogDuration</u> | <u>LogFiles</u> |
| <u>LogFrom</u> | <u>LogHangupCode</u> |
| <u>LogID</u> | <u>LogNumber</u> |
| <u>LogPages</u> | <u>LogPagesSent</u> |
| <u>LogRemoteID</u> | <u>LogResolution</u> |
| <u>LogRetries</u> | <u>LogSpeed</u> |
| <u>LogStatus</u> | <u>LogSubject</u> |
| <u>LogTime</u> | <u>LogToCompany</u> |
| <u>LogToName</u> | <u>LogUserData</u> |

This code will display the phone numbers of all the failed faxes:

```
Dim I%  
FaxMan1.Log = 3  
` Set to the Failed Log  
For I = 0 to FaxMan1.LogEntries-1  
    Print FaxMan1.LogFaxNumber(I)  
Next I
```



Custom Controls - Deleting Log entries

Entries in the FaxMan log are not deleted automatically. Your application can delete log entries by selecting the log you wish to delete entries in using the [Log](#) property, then setting the [LogIndex](#) property to the entry you wish to delete and setting the [Action](#) property to a value of 3 (FAX_DELETE) to delete the entry.

When a log entry is deleted, any fax files with an .FMF extension included in the [LogFiles](#) property will be deleted by the control.



Custom Controls - Runtime Errors

When an error occurs the Faxman controls will generate a runtime error that can be processed by your applications. Most all errors will occur when setting the Action property since these operations can fail due to bad files, etc.

The possible runtime errors are:

| | |
|-------|--|
| 32000 | Unsupported Action property setting |
| 32001 | Error Scheduling Fax |
| 32002 | Unable to register application with FaxMan Server |
| 32003 | Error cancelling Fax |
| 32004 | Error creating Fax File |
| 32005 | Internal Error - Bad internal pointer |
| 32006 | Set Server Configuration failed |
| 32007 | Error attempting to shutdown server |
| 32008 | No Option name set when setting ServerOptionSetting. |

This completes the Custom Control Tour ->[Contents](#)

