Adobe

FDK 7.2 Platform Guide, Windows

Adobe FrameMaker 7.2 FDK Platform Guide for Windows

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA

Contents

Using This Manual

The Frame® Developer's Kit (FDK) provides developer tools for FrameMaker®. The FDK includes:

- Application Program Interface (API)
- Frame Development Environment (FDE)
- Structure Import/Export Application Program Interface (Structure Import/Export API)

For a detailed description of FDK tools, see the *FDK Programmer's Guide* and the *FDK Programmer's Reference*.

This manual describes how to install and use the FDK on Windows®. It describes features and functions that are specific to the Windows implementation of the FDK, and how to compile, link, and register FDK clients.

FDK documentation

FDK documentation assumes that you have a thorough knowledge of FrameMaker for which you are developing clients. For background information on FrameMaker, see your user documentation. In addition, this manual assumes you have Windows application development experience.

FDK documentation includes the following manuals.

FDK Programmer's Guide

The *FDK Programmer's Guide* describes how to use the FDK to create clients for FrameMaker on all platforms. All FDK users should read the *FDK Programmer's Guide.*

FDK Programmer's Reference

The *FDK Programmer's Reference* provides detailed descriptions of all FDK functions and data structures. Use the *FDK Programmer's Reference* as your complete source of reference information for creating FDK clients.

FDK Platform Guide

The *FDK Platform Guide*, which you are reading now, is printed in several different versions. The following versions are currently available:

- Windows

- UNIX®

Read the version for each platform on which you intend to run your FDK clients.

FDK Sample Clients

FDK Sample Clients is a list of the sample clients that ship with the FDK, including brief descriptions of each client.

Structure Import/Export API Programmer's Guide

The *Structure Import/export API Programmer's Guide* provides instructions and reference information for using the Structure Import/Export API.

FDK Release Notes

The online manual *FDK Release Notes* contains last-minute information and tips on the FDK. In addition, it contains information on new features in FDK and changes from the previous release.

Conventions

FDK manuals distinguish between *you,* the programmer, and the *user,* the person for whom you write clients.

This manual uses different fonts to represent different types of information.

- What you type is shown in

  `text like this`.

- Filter names, program names, pathnames, and filenames are also shown in

  `text like this`.

- Placeholders (such as those representing names of files and directories) are shown in

  text like this.

This manual identifies commands on submenus by referring to both the submenu and the command name. For example, this manual refers to the Document Reports command on the Utilities submenu as Utilities>Document Reports.

This manual uses the term *FDK client* to refer to a program that you create with the FDK. Some other FrameMaker manuals refer to FDK clients as *API clients or plugins*.

This manual also uses the term FrameMaker, (as in FrameMaker product, FrameMaker file or FrameMaker session) to refer equally to FrameMaker running in structured or unstructured mode.

1 Introduction to the Windows Version of the FDK

This chapter provides an overview of how the FDK works on Windows and how to create and run an FDK client on Windows.

What you need

To compile and run FDK clients on Windows, you must have the following minimum system configuration:

- FDK for Windows
- Windows® 98, Windows NT® 4.0, Windows ME, Windows 2000, or Windows XP
- FrameMaker 7.2
- Microsoft® Development Environment 2003 (Version 7.1)

IMPORTANT: *This manual assumes you are familiar with Microsoft Development Environment 2003 (Version 7.1).*

How the FDK works on Windows

FDK clients on Windows are not implemented as true Windows clients. They are dynamic link libraries (DLLs) that provide entry points or callback functions, which FrameMaker can invoke.

There are several types of FDK clients:

- A *standard FDK client* is an FDK client that initializes when FrameMaker starts and then waits to respond to specific user actions, such as menu choices.
- A *take-control client* is an FDK client that responds to a special initialization and takes complete control of a FrameMaker session. Many of the effects you can get with this type of client can also be realized by an asynchronous client.
- A *filter* is an FDK client that converts FrameMaker files to or from other file formats. FrameMaker calls a filter when the user attempts to open, import, or save a file with a particular format.
- A *document report* is an FDK client that provides information about a document. The user can start a document report by choosing Utilities>Document Reports from the File menu and selecting the report from the Document Reports dialog box.

When FrameMaker starts, it reads the `maker.ini` file in the FrameMaker installation directory, and if applicable, the maker.`ini file` stored in the user's `Documents and Settings` directory. The `[APIClients]` section of the `maker.ini` file contains entries describing the FDK clients to be loaded.

FrameMaker then scans the `fminit/Plugins` directory and subdirectories and loads the FDK clients that have a `.dll` file extension and valid VERSIONINFO resource information. FrameMaker ignores all other files in the `fminit/Plugins` directory that do not have a `.dll` file extension and valid VERSIONINFO resource information. For more information on registering clients, see "Registering FDK clients" on page 29.

For information on how FrameMaker starts a client, see Chapter 2, "API Client Initialization," in the *FDK Programmer's Guide*.

Creating and running an FDK client on Windows

To create and run an FDK client on Windows, follow these general steps:

1. Install the FDK.

   For detailed instructions on installing the FDK and on the files shipped with the FDK, see Chapter 2, "Installing the FDK on Windows."

2. Write your client code and create any custom dialog boxes.

   Follow the instructions in the *FDK Programmer's Guide* to write your client code and to create your custom dialog boxes.

   To use some FDK features on Windows, you may need to modify your code. For example, to use FDK functions that return or accept platform-specific pathnames as arguments, you must use the FDK Windows pathname conventions. To specify common Windows keyboard shortcuts such as Control-h, you must use special characters when you call `F_ApiDefineAndAddCommand()`.

   For more information on writing FDK code for Windows, see Chapter 3, "Writing FDK Clients for Windows."

3. Configure your Development Environment 2003 for the FDK.

   To build FDK clients you must add the FDK `include` and `lib` directories to the paths that Development Environment 2003 searches for headers and library files.

   For a more complete set of steps to configure your Development Environment 2003 for the FDK, see "Compiling and registering your own FDK clients" on page 24.

4. Create a workspace for your client.

   Use Development Environment 2003 to create a project for a dynamic-link library (DLL). Make sure the linker links these FDK libraries: `fdk.lib`, `api.lib`, and `fmdbms32.lib`.

IMPORTANT: *Make sure the Use of MFC option in the General page of the Property Pages dialog box is set to Use Standard Windows Libraries.If it is set to another value, your client will not link correctly. Also make sure the Struct Member Alignment is set to 8 Bytes, and you use the single-threaded run-time libaray. Otherwise, your client may cause unexpected runtime errors.*

5. Compile and link your client.

   Choose Build from the Build menu. Development Environment 2003 compiles and links your client.

6. Register your client.

   To register your client, you can use either of the following methods:
   - Create and include in your client's project a VERSIONINFO resource that contains information about the client, and copy or move the compiled client into the `fminit/Plugins` directory.
   - Add an entry for your client in the `[APIClients]` section of the `maker.ini` file.

   For information on registering clients, see "Registering FDK clients" on page 29.

7. Start FrameMaker.

   After you register your client, the next time you start FrameMaker, it automatically loads your client into memory.

Working with Microsoft Visual C++ 6.0

Since the new FDK 7.2 uses Microsoft Development Environment 2003 (Version 7.1), it is recommended that FDK clients should be compiled in same environment too. If users wants to use Microsoft Visual C++ 6.0, then following lines can be added to source code of the FDK clients to make it compile and work in that case:

```
long _ftol( double );
long _ftol2( double dblSource ) { return _ftol( dblSource ); }
```

1 Introduction to the Windows Version of the FDK

2 Installing the FDK on Windows

This chapter describes how to install the FDK on Windows. It also lists the files shipped with the Windows version of the FDK.

Installing the FDK

The FDK is delivered via the Adobe Systems web site. Navigate to the Adobe Solutions Network page at http://partners.adobe.com then navigate to the FDK 7.2 Download page. After downloading the compressed file, double-click the installer and follow the instructions.

The FDK installation

The FDK installation contains the `FDK` folder, which contains FDK header files, libraries, and sample code.

The `FDK` folder contains the following folders:

Folder	What it contains
`include`	Public header files that you include in FDK clients
`lib`	FDK libraries that you link with your client
`doc`	*FDK Release Notes*, a list of sample clients, and online documentation for the FDK in Adobe® Acrobat® PDF
`samples`	Source code and project files for sample FDK clients and a dialog resource template

The following sections describe these folders and their contents.

include

The `include` folder contains FDK header files. The files are listed in the following table.

File	Purpose
`f_local.h`	Provides a location for declarations for your platform-specific functions.
`f_stdio.h`	Provides declarations for platform-independent C library functions.

File	Purpose
f_types.h	Defines Frame Development Environment (FDE) fundamental data types. It is included in fapi.h and fdetypes.h.
fapi.h	Provides definitions and function declarations for the FDK. You must include it in all FDK clients.
fapidefs.h	Defines constants you can use to specify objects, properties, and some function arguments. It is included by fapi.h.
fassert.h	Provides declarations for FDE assert functions.
fchannel.h	Provides declarations for FDE channel functions.
fcharmap.h	Provides declarations for FDE character functions.
fcodes.h	Provides declarations for function codes (f-codes).
fdetypes.h	Provides declarations for FDE data types. You must include it in all FDK clients that use the FDE.
fdk_env.h	Provides top-level header file for individual platforms.
fencode.h	Provides declarations for API and FDE font encoding functions.
fhash.h	Provides declarations for FDE hash functions.
fioutils.h	Provides declarations for FDE I/O utility functions.
fltstub.h	Provides declarations for filter functions.
fm_base.h	Defines types and data structures for the Structure Import/Export API. It is included by fm_struct.h.
fm_comma.h	Only present for backward compatibility. Use fcodes.h.
fm_psr.h	Defines types and data structures for the Structure Import/Export API. It is included by fm_struct.h.
fm_rdr.h	Defines types and data structures for the Structure Import/Export API. It is included by fm_struct.h.
fm_sgml.h	Retained for backward compatibility—use fm_struct.h instead.
fm_struct.h	Provides declarations for Structure Import/Export API functions. You must include it in all Structure Import/Export API clients.
fm_wtr.h	Defines types and data structures for the Structure Import/Export API. It is included by fm_struct.h.
fmemory.h	Provides declarations for FDE memory functions.
fmetrics.h	Provides declarations for FDE metric functions.
fmifdata.h	Provides declarations for FDE Maker Interchange Format (MIF) functions.
fmifmacr.h	Provides macros for writing MIF statements.

File	Purpose
fmifname.h	Provides definitions for MIF statements.
fmifstmt.h	Provides declarations for FDE MIF statement functions.
fmifstrt.h	Provides MIF data structures.
fmiftype.h	Provides basic data structures used by MIF data structures.
fpath.h	Provides definitions used by filepath functions.
fprogs.h	Provides declarations for FDE progress functions.
fstdio.h	Provides declarations for FDE I/O functions.
fstrings.h	Provides declarations for FDE string functions.
fstrlist.h	Provides declarations for FDE string list functions.
fstrres.h	Provides internally used string resource functions. Do not include with your FDK clients.
futils.h	Provides declarations for FDE utility functions.

lib

The lib folder contains the library files listed in the following table.

File	What it contains
api.lib	The API library. To use any API functions, you must link this library with your client.
fdk.lib	The FDE library. To use any FDE functions, you must link this library with your client.
fmdbms32.lib	FDK heap management library. Link all FDK clients with this library.
fmstruct.res	Provides SGML/XML resources. You must link all Structure Import/Export API clients with it.
struct.lib	Provides Structure Import/Export API functions. You must link all Structure Import/Export API clients with it.

doc

The doc folder contains the FDK documentation for all platforms in Adobe Acrobat PDF.

PDF file	Description
fdkguide.pdf	The *FDK Programmer's Guide*

PDF file	Description
fdkref.pdf	The *FDK Programmer's Reference*
relnotes.pdf	The release notes for the FDK and information on new features and changes to the FDK since the last release
samplelist.pdf	A list of the code samples shipped with the FDK, including brief descriptions of each one.
structapi.pdf	The *Structure Import/Export API Programmer's Guide*
unxguide.pdf	The *FDK Platform Guide* for UNIX
winguide.pdf	The *FDK Platform Guide* for Windows

samples

The samples folder contains the code for sample clients and a sample dialog resource file. The samples\winsamp folder contains the code for a sample client that is specific to the Windows platform.

For a list of the samples that are included with the FDK, and a brief description of each one, see the online document samplelist.pdf. This file is included with the FDK in the doc folder.

The other folders in the samples folder (with the exception of dre) include one or more source (.c) files , appropriate header (.h) files and appropriate workspace, solution, and project file. All the sample code in those folders is platform independent. With an appropriate makefile, you can compile it on any of the platforms the FDK supports. For information about a client, see the comments at the beginning of the client's source (.c) file.

IMPORTANT: *Permission to use, reproduce, modify, and distribute the Sample Clients is for the sole purpose of integrating your software applications with Adobe Systems Incorporated ("Adobe") software ("Sample Clients" are defined as those files located in the fdk\samples folder). Such permission is hereby granted without fee, provided that*
(i) you distribute the Sample Clients only as part of your software application;
(ii) the following copyright notice appears in and on all copies of your software application:

```
ADOBE CONFIDENTIAL
_____________________________________________________________

Copyright 1999 - 2005 Adobe Systems Incorporated All
Rights Reserved.

NOTICE:  All information contained herein is, and remains
the property of Adobe Systems Incorporated and its
suppliers, if any.  The intellectual and technical
concepts contained herein are proprietary to Adobe
```

3 Writing FDK Clients for Windows

This chapter describes how to write FDK clients so they will run on Windows. It also discusses important differences between the Windows implementation and other implementations of the FDK.

IMPORTANT: *You cannot compile clients on Windows versions earlier than Windows 95 or Windows NT 4.0.*

How to write an FDK client for Windows

When you write an FDK client, you should do the following for it to compile and run correctly on Windows:

- Include the correct FDK header files in the correct order
- Replace platform-specific functions and data types with FDE equivalents
- Include calls to initialize the FDE if your client calls FDE functions

The following sections discuss these tasks in greater detail.

Including FDK header files

The following table lists the header files you must include in your client in the order in which you must include them.

If you are using	Include
Any FDK function or constant	`fapi.h`.
Any FDE type	`fdetypes.h`
A specific FDE function	Header file for the function's group (for example, `fhash.h` for a hash function). For more information, see the function's description in the *FDK Programmer's Reference*.
Any Structure Import/Export API functions	`fm_struct.h`.
Constants for Frame f-codes	`fcodes.h`.

IMPORTANT: *You must include the* `fapi.h` *header file before any other FDK header files.*

For example, if your client uses API functions and FDE metric utility functions, it must include header files as follows:

```
#include "fapi.h"
#include "fdetypes.h"
#include "fmetrics.h"
```

If you need to include any C library header files or your own platform-specific header files, include them before the FDK header files.

Replacing platform-specific functions and data types

To help you make your clients portable, the FDK provides platform-independent substitutes for C data types and string and memory functions. For example, it provides a data type named `IntT`, which you can use instead of `int`, and a function named `F_Alloc()`, which you can use instead of `malloc()`.

To ensure that your client does not use platform-specific data types or functions, the FDK redefines them when you include the file `fdetypes.h`. If your client uses a platform-specific data type or function, Development Environment 2003 (Version 7.1) issues an error message when you attempt to compile it. For example, if your client declares the following variable:

```
char ch;
```

Development Environment 2003 (Version 7.1) issues one or more compilation error messages similar to the following:

```
error C2065: 'error' : undeclared identifier
error C2143: syntax error : missing ';' before '!'
error C2065: 'Non_FDE_token' : undeclared identifier
error C2143: syntax error : missing ';' before 'string'
error C2143: syntax error : missing ';' before '!'
error C2065: 'ch' : undeclared identifier
```

To avoid these error messages, you can do one of the following:

- Use the FDK substitute for the platform-dependent data type or function.

  For example, use `CharT` instead of `char`. This is the recommended solution for portable FDK clients.

- Add the following code above the `#include "fdetypes.h"` statement:

  ```
  #define DONT_REDEFINE
  ```

  This prevents the FDK from redefining any data types or functions.

- Use `#undef` to undefine the specific types or functions that you want to use.

  For example, add the following line after the `#include "fdetypes.h"` statement:
  ```
  #undef char
  ```

  The result of this prevents Development Environment 2003 (Version 7.1) from generating an error message for the `char` type, but allows it to generate errors if your client uses any other platform-specific types.

Adding calls to initialize the FDE

If your client calls FDE functions, it must call `F_FdeInit()` once before it calls the functions. The syntax for `F_FdeInit()` is:

```
ErrorT F_FdeInit(VoidT);
```

To call `F_FdeInit()`, your client must include the `fdetypes.h` header file.

Writing filter clients

You can use filter clients to translate documents from one format to another. FrameMaker invokes an import filter client when it recognizes a file of a particular format or when the file has a registered suffix. It invokes an export filter when you choose a particular format from the Format pop-up menu of the Save As dialog box or save a file using a registered suffix. For example, if you register a suffix for a text import filter and then open a file with that suffix, the Unknown File Type dialog box appears with the appropriate filter preselected.

You must register your filter client before use. For information on registering clients, see Chapter 4, "Compiling, Registering, and Running FDK Clients."

You can also use your filter to import text or graphic files into a document. If you import a file by reference, FrameMaker stores in the document the registered format and vendor ID of the filter used in the import operation. If you import the file by copy, FrameMaker stores the facet name in the document. The information in both these cases ensures that FrameMaker invokes the correct filter for updating the next time you open the document.

IMPORTANT: *If you are writing a filter client, FrameMaker will not fully recognize it unless you include function calls that actually cause the API library to link with your client. To make sure the client links properly, you can include the following as minimal code in your F_ApiNotification() function:*

```
. . .
F_ObjHandleT docId;
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
. . .
```

Identifying your filter

To identify your filter to FrameMaker, you need to supply information in the line that registers the filter. This information identifies the filter on all platforms and identifies the original import filter when reimporting the file. FrameMaker uses several pieces of information that you specify for this purpose:

- The vendor ID is a four-character string describing the provider of the filter.
- The format ID is a four-character string describing the file format of files on which the filter operates.
- The facet name is an arbitrary-length string describing the filter.

For example, assume you create a filter for Windows machines that translates Himyaritic documents to English. You give it the format ID "HIMF" and the vendor ID "FAPI". If you create a document under Windows and create a text inset using that filter, FrameMaker stores this information with the inset. The next time you open that document, FrameMaker knows to update the inset with your Himyaritic filter.

In addition, if you open the document on another platform, such as UNIX, that version of FrameMaker looks for a filter with the same vendor and format ID to create the text inset. If you have written a Himyaritic filter on the UNIX platform and registered it with the same information, FrameMaker can open your file successfully. If you register different information for the Himyaritic filter on UNIX, FrameMaker cannot find it. In this case, FrameMaker tries to automatically identify the file format and invoke the appropriate filter.

FrameMaker reserves the following vendor IDs:

- "FRAM"
- "FFLT"
- "IMAG"
- "XTND"
- "AW4W"
- "ADBE"
- "ADBI"

Your client cannot use these vendor IDs. FrameMaker recognizes FAPI as a valid ID for any FDK filter client. However, you do not have to use this ID. You can use any other four-character string as your vendor ID.

FrameMaker reserves the following format IDs for the indicated file formats. FrameMaker does not supply filters for all of these formats on all platforms. However, to aid in portability of your clients, you should not use one of these format IDs unless it is for the specified file format:

Format ID	Description
`"CDR "`	CorelDRAW
`"CGM "`	Computer Graphics Metafile
`"CVBN"`	Corel Ventura compound document (Windows)
`"DCA "`	DCA to MIF (UNIX)
`"DIB "`	Device-independent bitmap (Windows)
`"DRW "`	Micrografx CAD
`"DXF "`	Autodesk Drawing eXchange file (CAD files)
`"EMF "`	Enhanced Metafile (Windows)
`"EPS "`	Encapsulated PostScript®

Format ID	Description
"EPSB"	Encapsulated PostScript Binary (Windows)
"EPSD"	Encapsulated PostScript with Desktop Control Separations (DCS)
"EPSF"	Encapsulated PostScript
"EPSI"	Encapsulated PostScript Interchange
"FRMI"	FrameImage
"FRMV"	FrameVector
"G4IM"	CCITT Group 4 to Image
"GEM "	GEM file
"GIF "	Graphics Interchange Format (CompuServe)
"HPGL"	Hewlett-Packard Graphics Language
"IAF "	Interleaf compound document
"IGES"	Initial Graphics Exchange Specification (CAD files)
"JPEG"	Joint Photographic Experts Group
"MIAF"	MIF to IAF export
"MIF "	Maker Interchange Format
"MML "	Maker Markup Language
"MooV"	QuickTime Movie
"MRTF"	MIF to RTF export
"MWPB"	MIF to WordPerfect export
"OLE "	Object Linking and Embedding Client (Microsoft)
"OLE2"	Object Linking and Embedding Client (Microsoft)
"PCX "	PC Paintbrush
"PICT"	QuickDraw PICT
"PNG "	Portable Network Graphics
"PNTG"	MacPaint
"RTF "	Microsoft's RTF compound document
"SNRF"	Sun™ Raster File
"SRGB"	SGI RGB
"TANS"	Text ANSI (Windows)

Format ID	Description
"TASC"	Text ASCII
"TBG5"	Traditional Chinese (BIG-5)
"TEUH"	Traditional Chinese (EUC-CNS)
"TEUJ"	Japanese (EUC)
"TEXT"	Plain text
"TIFF"	Tag Image File Format
"TJIS"	Japanese (JIS)
"TKOR"	Korean
"TMAC"	Text (Macintosh)
"TRFA"	troff -man to MIF (UNIX only)
"TRFE"	troff -me to MIF (UNIX only)
"TRFF"	troff to MIF (UNIX only)
"TRFS"	troff -ms to MIF (UNIX only)
"TSJS"	Japanese (Shift-JIS)
"TXGB"	Simplified Chinese (GB)
"TXHZ"	Simplified Chinese (HZ)
"TXIS"	Text ISO Latin 1
"TXRM"	Text Roman 8
"WDBN"	Microsoft Word compound document
"WMF "	Windows Metafile
"WPBN"	WordPerfect compound document
"WPG "	WordPerfect Graphics
"XBM "	X BitMap
"XWD "	X Windows System™ Window Dump file
"0050"	Microsoft Word DOS 3.0, 3.1
"0051"	Microsoft Word DOS 4.0
"0052"	Microsoft Word DOS 5.0/6.0
"0070"	WordPerfect DOS/Windows 5.0
"0071"	WordPerfect DOS/Windows 5.1

Format ID	Description
"0150"	DCA0
"0151"	DCA1
"0190"	Microsoft Rich Text Format (RTF) (import)
"0192"	Microsoft Rich Text Format (RTF) (export)
"0203"	Lotus 123 4.0
"0204"	Lotus 123 5.0
"0214"	Microsoft Excel 5.0
"0330"	Ami Professional 1.0
"0331"	Ami Professional 2.0-3.1
"0440"	Microsoft Word 1.0
"0441"	Microsoft Word 2.0
"0460"	Interleaf compound document (IAF)
"0480"	WordPerfect DOS/Windows 6.0
"0481"	WordPerfect DOS/Windows 6.1
"0482"	WordPerfect DOS/Windows 7.0
"0490"	Microsoft Word 6.0/7.0
"049m"	Microsoft Word Mac 6.0
"0540"	Microsoft Word Mac 3.x
"0541"	Microsoft Word Mac 4.x
"0542"	Microsoft Word Mac 5.x
"0590"	WordPerfect Mac 1.0
"0600"	WordPerfect Mac 2.0-2.1
"0601"	WordPerfect Mac 3.0 to 3.5 (import)
"0602"	WordPerfect Mac 3.5 (export)

Automatic recognition of a file format

Some graphic file formats have signature bytes. Signature bytes are a set of bytes that have a unique value and location in a particular file format. FrameMaker can use signature bytes to identify a graphic file's format.

The documentation for the file format that your graphics filter converts may contain information on the signature bytes for that format. If it does, you can register the signature bytes in the [FormatList] section of the `maker.ini` file. Each graphic file format description must be on a separate line and must have the following format:

```
n=facet_name start_offset signature_size signature
```

where `n` is any number, `facet_name` is the file format's description (also used in the client registration), `start_offset` is how many bytes from the start of the file the signature begins, `signature_size` is the size in bytes of the signature, and signature is the hexadecimal value of the signature. You can enclose any of the arguments in double quotation marks. For example, you can register the file format for MIF with the following:

```
[FormatList]
100="MIF" 0 8 0x3c4d494646696c65
```

where `0x3c4d494646696c65` is the hexadecimal encoding of the characters `<MIFFile`.

Using Windows pathnames

The FDK for Windows delimits pathnames with backslashes (\). When you specify a pathname in an FDK function call, follow these rules:

- Follow the drive letter with a colon.
- Don't terminate a pathname that specifies a file with a backslash.

The following table lists examples of files and directories and the pathname strings that specify them.

File or directory	Absolute pathname	Relative pathname
File named `myfile.doc` on the `c:` drive	`c:\myfile.doc`	`myfile.doc`
Directory named `mydir` on the `c:` drive	`c:\mydir`	`mydir`

Because the backslash is a special character, you must precede it with another backslash when you specify it in a string. For example, to open a file named c:\myfile.doc with F_ApiSimpleOpen(), use the following code:

```
F_ApiSimpleOpen("c:\\myfile.doc", False);
```

Using pathnames returned by FDK functions

Pathnames returned by FDK functions don't end with a backslash, unless they specify root directories, such as `c:\`.

Using F_PathNameToFilePath()

To specify an absolute pathname when you call `F_PathNameToFilePath()`, you must specify a pathname that includes the drive and begins with the root directory of the drive. If the pathname does not include the drive and begin with the root directory of the drive, `F_PathNameToFilePath()` assumes the pathname is relative.

If you call `F_PathNameToFilePath()` with anchor set to `NULL` and you do not specify an absolute pathname, `F_PathNameToFilePath()` adds the currently open directory or the currently open directory of the specified drive to the pathname. For example, if you specify c:myfile.c for pathname, `F_PathNameToFilePath()` generates: `c:\`*`cwd`*`\myfile.c`, where *cwd* is the currently open directory on drive `c:`. If you specify `\\myfile.c` for pathname, `F_PathNameToFilePath()` generates: *current_drive*`:\myfile.c`, where *current_drive* is the current drive.

If you do not set anchor to `NULL`, `F_PathNameToFilePath()` constructs the filepath relative to the path specified by anchor. If the pathname you specify for pathname and the filepath you specify for anchor are inconsistent, `F_PathNameToFilePath()` ignores anchor and constructs the filepath with the currently open directory.

Using F_FilePathGetNext()

The function `F_FilePathGetNext()` returns the next file in a specified directory. To do so, this function uses DOS system calls. As a result, since DOS is case-insensitive, the returned `FilePathT` structure uses only uppercase letters. This may not match a `FilePathT` structure you have created.

For example, assume you want to create a filepath and then at some later time process all files in the same directory other than the one you created. You might be tempted to use this code:

```
/* Bad code! */
. . .
/* Create the new filepath */
newpath = F_PathNameToFilePath ("vpg.doc", NULL, FDosPath);
. . .
DirHandleT handle;
FilePathT *path, *file;
IntT statusp;
pathname = StringT;
handle = F_FilePathOpenDir(newpath, &statusp);
if (handle) {
  pathname = F_FilePathToPathName (newpath);
  while ((file = F_FilePathGetNext (handle, &Statusp)) != NULL) {
/* WRONG! This attempts to compare current file to the one you created. */
    if ! (F_StrEqual (pathname, F_FilePathToPathName (file)))
       ProcessFile (file);
    F_FilePathFree (file);
  }
}
/* Bad code! */
. . .
```

The string returned by `F_FilePathToPathName(newpath)` contains the lowercase letters as specified in the earlier call to the function `F_PathNameToFilePath()`. On the other hand, the string returned by each call to `F_FilePathToPathName()` always contains only uppercase letters. Therefore, the call to `F_StrEqual()` never succeeds.

Instead of calling `F_StrEqual()`, you should call `F_StrIEqual()`.

Using menus and commands

The following sections describe how to use menus and commands in your FDK client.

Finding FrameMaker menu and command names

The [Files] section of the `maker.ini` file specifies the location of the menu and command configuration files that list FrameMaker's menus and commands. The following are the default entries in the `maker.ini` file:

```
MathCharacterFile = fminit\mathchar.cfg
ConfigCommandsFile = fminit\cmds.cfg
MSWinConfigCommandsFile = fminit\wincmds.cfg
ConfigMathFile = fminit\mathcmds.cfg
ConfigMenuFile = fminit\maker\menus.cfg
ConfigCustomUIFile = fminit\customui.cfg
```

The following table lists the menus and commands each file contains.

Menu or command file	Contents
`MathCharacterFile`	Special math characters
`ConfigCommandsFile`	Basic commands
`MSWinConfigCommandsFile`	Windows-specific commands
`ConfigMathFile`	Math commands
`ConfigMenuFile`	Standard menus
`ConfigCustomUIFile`	Custom menus

Defining keyboard shortcuts

Keyboard shortcuts beginning with Esc (Escape) do not appear on menus in Windows versions of FrameMaker. However, the user can still use these shortcuts to execute commands. For example, if you use the following code to define and add a command to the Edit menu:

```
F_ApiDefineAndAddCommand (TRANSLATE, EditMenuId, "I1",
     "Translate", "\\!tt");
```

the command label Translate appears on the menu, but the shortcut (! t t) does not. The user can execute the command by pressing Esc t t.

The Windows version of the FDK allows you to specify keyboard shortcuts that include the modifier key symbols on a menu. The following table lists the modifier keys and strings that specify them.

To display	Specify
`Alt`	`~`
`Ctrl`	`^`
`Shift`	`+`

For example, the following code defines and adds a command with a Control-t shortcut:

```
F_ApiDefineAndAddCommand (TRANSLATE, EditMenuId, "I1",
        "Translate", "^t");
```

Using FDK functions that write to the FrameMaker console

The following functions write output to the FrameMaker console on Windows:

- `F_ApiPrintFAErrno()`
- `F_ApiPrintOpenStatus()`
- `F_ApiPrintPropVals()`
- `F_ApiPrintSaveStatus()`
- `F_Printf()` with `Channel` set to `NULL`
- `F_Warning()`

For descriptions of these functions, see the *FDK Programmer's Reference*.

As with `printf()`, the `F_Printf()` function does not automatically print a line feed (`"\n"`) after the output. If you don't end the output with `"\n"`, the next call to one of the functions listed above begins printing on the last line printed by the `F_Printf()` call.

Using platform-dependent session properties

Session (`FO_Session`) objects have the following platform-dependent properties:

Property	Value
`FP_FM_BinDir`	Pathname of the `bin` directory in the FrameMaker installation directory
`FP_FM_CurrentDir`	Pathname of the FrameMaker installation directory

Property	Value
FP_FM_HomeDir	Pathname of the FrameMaker installation directory
FP_FM_InitDir	Pathname of the fminit directory in the FrameMaker installation directory
FP_HostName	Host name specified for PCName in the maker.ini file
FP_OpenDir	Pathname of the FrameMaker installation directory
FP_Path	Path specified by the $PATH environment variable
FP_TmpDir	Directory specified by the $TEMP environment variable
FP_UserHomeDir	Pathname of the FrameMaker installation directory
FP_UserLogin	The user name under which FrameMaker is registered
FP_UserName	The user name under which FrameMaker is registered

Although the values of some of these properties specify directory pathnames, they are not terminated with a backslash.

Unsupported FDK functions

The Windows version of the FDK does not support the following functions, which are documented in the *FDK Programmer's Reference*:

- F_ApiDoneCommand()
- F_ApiTakeControl()

4 Compiling, Registering, and Running FDK Clients

This chapter describes how to compile, register, and run FDK clients on Windows. It also briefly explains how to debug your FDK clients.

Compiling FDK clients

The following sections describe how to compile FDK sample clients and your own clients.

Supported compilers

To compile FDK clients for Windows, you must use Microsoft Development Environment 2003 (Version 7.1)

Compiling, registering, and running the sample clients

The following sections describe how to compile, register, and run the sample clients provided with the FDK.

Compiling and registering sample clients in Development Environment 2003 (Version 7.1)

To compile a sample FDK client in Development Environment 2003 (Version 7.1), follow these steps:

1. Start Development Environment 2003 (Version 7.1).
2. Choose Open->Project from the File menu and then choose the solution file for one of the sample clients.

   For example, to compile the `aframes` sample client, choose *fdk_install_dir*`\samples\aframes\aframes.sln`, where *`fdk_install_dir`* is the pathname of the directory in which the FDK is installed.

   NOTE: The project settings for the sample clients have relative paths to the FDK lib and include files already specified. If you open a sample project from its location in the FDK installation, these paths will be valid. If you move the sample client to a different location, you may need to specify new paths for the include and lib files. For more information, see steps 7. and 8. of "Compiling and registering your own FDK clients" on page 24.

3. Use the Development Environment 2003 (Version 7.1) build utility to build the client.

   Choose Rebuild Solution from the Build menu. Development Environment 2003 (Version 7.1) compiles your code into a DLL file named *project*`.dll` in the `debug` subdirectory of your client directory, where *`project`* is the name of the sample project. For example, the `aframes` sample client compiles into `debug\aframes.dll`.

4. Register the sample client.

   Each of the following sample clients includes a VERSIONINFO resource, and you register each by placing the DLL file in the Plugins folder:
   - pickfmts
   - elemutils
   - dialog

   Because the remaining sample clients do not include a VERSIONINFO resource, you must register them in the `maker.ini` file. For more information see "Registering clients in the FrameMaker maker.ini file" on page 36.

Running the sample FDK clients

It is best to store client DLL files in the FrameMaker Plugins folder (*install_dir*`\FrameMaker7.2\fminit\Plugins`), or in a folder below it. If you register your clients via the VersionInfo resource, you must store them in this way. When you register a client in the `.ini` file, you can specify any location for the DLL file.

After you have compiled and registered a sample FDK client, start FrameMaker to test the client. Some of the sample clients add menus and commands to the FrameMaker menus. For example, if you have compiled and registered the sample client described in Chapter 1, "Introduction to the Frame API," of the *FDK Programmer's Guide*, a menu named API appears on the FrameMaker menu after you start FrameMaker. To test the commands on this menu, open or create a document, and choose each of the commands.

Compiling and registering your own FDK clients

To compile and register one of your own FDK clients, follow the instructions in this section.

Compiling and registering the client

To compile and register the FDK client, follow these general steps:

1. Create a project directory for your FDK client project.

2. Start Development Environment 2003 (Version 7.1) and create a new Win32 Dynamic-Link Library project.

   Choose New and then project from the File menu. The New dialog box appears. Select Visual C++ Projects and then Win32 from Project Types. Select Win32 Project, type your client's name in the Name field and then click OK. Win32 application wizard appears.

Click on Application Settings, select DLL from Application Type and Empty project from Additional Options.

3. Create or place your souce files in the project directory, then add those files to your project.
4. (Optional) Create a resource for any custom dialog boxes.

   If your client contains custom dialog boxes, you need to create a resource for them. For instructions, see "Using custom dialog boxes" on page 26.
5. (Optional) Create a VERSIONINFO resource.

   Including a VERSIONINFO resource is one method for registering a client. For more information on registering clients, see "Registering FDK clients" on page 29.
6. Choose Properties from the Project menu to display the Properties Pages dialog box.

   In the Properties Pages dialog box, choose General . Set Use of MFC field to Use Standard Windows Libraries.

   IMPORTANT: *If you don't set the* Use of MFC field to"Use Standard Windows Libraries, , *your client will not link correctly.*
7. Set your project's C/C++ Language options.

   In the Property Pages dialog box, choose C/C++.
   - Choose Code Generation and choose 8 Byte or default from the Struct Member Alignment pull down menu. 8 bytes is also the default value for this field.

     IMPORTANT: *If you don't set the Struct Member Alignment to 8 Bytes, your client may cause unexpected runtime errors.*
   - With Code Generation still selected, choose Single-Threaded from the Runtime Library popup list.

     The FDK ships in a single-threaded version. By default, the project sets this option to Multi-threaded. Compiling the FDK with a multi-threaded runtime library produces the following warning:

     ```
     defaultlib "LIBC" conflicts with use of other libs;
     ```

     IMPORTANT: *For Version 7.0 and later of the FDK, it is important that you make this setting. Earlier versions of the FDK did not use symbols that conflicted with the multi-threaded runtime library. However, for version 7.0 and later the FDK and the Structure Import/Export API use conflicting symbols.*
   - In the General page under C/C++ language options, add path to the FDK include files in Additional Include Directories field.

     You can specify an absolute path or a relative path. For example, the Property Pages for the sample clients all use the following relative path:

     ```
     ..\..\include
     ```

8. Set your project's Linker options

   In the Property Pages dialog box, choose the Linker page.

   – Choose General and then Input.

     Add the FDK libraries `fdk.lib`, `api.lib`, and `fmdbms32.lib` to the additional dependencies field.

     If you are compiling a structure import/export client, be sure to also link the Structure Import/Export API library. For more information, see "Linking the Structure Import/Export API library" on page 28.

     IMPORTANT: *If your client includes custom dialog boxes, you must add* `/section:.rsrc,w` *to the Project Options. For more information, see "Compiling clients with custom dialog boxes" on page 28.*

   – In the Category field, choose Input, then add the path to the FDK lib files in the Additional library path field.

     You can specify an absolute path or a relative path. For example, the project settings for the sample clients all use the following relative path:
     `..\..\lib`

     NOTE: As an alternative, you can specify access to the FDK include and lib directories for the Development Environment 2003. To do this, choose Tools > Options to display the Options dialog box. Select Project and then VC++ Directories, and enter the paths to the FDK include and lib directories for Include files and Library files.

9. Use the Development Environment 2003 (Version 7.1) build utility to build your client.

   Choose Rebuild All from the Build menu. Development Environment 2003 (Version 7.1) compiles your code into a dynamic link library file with the name you typed in the New dialog box. It puts this library file into the `debug` subdirectory of your client directory.

10. Register the client.

    You can register the client by using either of these two methods:

    – As mentioned in step 5, create and include in your client's project a VERSIONINFO resource that contains information about the client, and copy or move the compiled client into the `fminit/Plugins` directory.

    – Add an entry for your client in the `[APIClients]` section of the `maker.ini` file in the FrameMaker directory.

    For more information on registering clients, see "Registering FDK clients" on page 29.

Using custom dialog boxes

The FDK samples include a template document for designing custom dialog boxes. You open this document in FrameMaker and edit it with the FrameMaker graphic tools and commands.

When you save a custom dialog box in a Windows version of FrameMaker, it generates two Windows resource definition files, a `.dlg` file and a `.xdi` file.

- The `.dlg` file is a text file containing resource statements. These statements are standard Windows descriptions of the dialog box and its controls.
- The `.xdi` file is a text file containing a user-defined resource statement. This statement contains data used by FrameMaker to manipulate the dialog boxes.

When creating the `.dlg` and `.xdi` files, FrameMaker uses the name of the `.dre` file (without the extension) to name the files and the actual dialog resource. For example, when saving the file named `mydlg.dre`, FrameMaker creates the resource description files `mydlg.dlg` and `mydlg.xdi`. Both files describe the dialog resource named `mydlg`.

To compile the `.dlg` and `.xdi` files in your dll you must create a resource for the project, and provide directives to include these files in the resource. In the process of compiling the client, these resource definition files are compiled into a single resource file (`.rc`). This resource file is linked to your client.

To set up the resource definition files to be compiled, follow these general steps:

1. Start Development Environment 2003 (Version 7.1).
2. If one doesn't already exist for the project, create a resource script.

   Choose Add New Item from the File menu. The Add New Item dialog box appears. Choose Resource File.
3. Include the resource description files generated by FrameMaker.

   Choose Resource Includes from the Edit menu. The Resource Includes dialog box appears. In the Compile-Time Directives field, type `#include` statements to include the resource description files.

   For example, suppose you create two custom dialog boxes named `pgftag.dre` and `chartag.dre`. When FrameMaker saves these files it also creates the files `pgftag.dlg`, `pgftag.xdi`, `chartag.dlg`, and `chartag.xdi`.

   To include these files in the resource script, type the following in the Compile-Time Directives field:

   ```
   #include "pgftag.dlg"
   #include "pgftag.xdi"
   #include "chartag.dlg"
   #include "chartag.xdi"
   ```

   Make sure you use the correct syntax; Development Environment 2003 (Version 7.1) does not check the syntax or warn you of this before you dismiss the dialog box.
4. Save the resource script.

   For information on adding your resource script to your client, see "Compiling and registering the client" on page 24.

Compiling clients with custom dialog boxes

If your FDK client uses custom dialog boxes, you need to specify a special link option before compiling it:

1. In Development Environment 2003 (Version 7.1), choose Project->Properties.

   This displays the Project Properties dialog box.

2. Choose Linker and then Command Line.

3. Add the following option to the Additional Options field:

   ```
   /section:.rsrc,w
   ```

   This link option makes the dialog resources writable. If you do not specify it before compiling, your FDK client may exit unexpectedly when it attempts to display a custom dialog box.

4. Repeat steps 3 for each target in your project.

Making adjustments to custom dialog boxes

Since the `.dlg` files produced by FrameMaker are text files containing resource statements, you can open these files in Development Environment 2003 (Version 7.1) as resources. You can use the built-in tools for dialog editing to view, adjust, and test the dialog box.

Because you are modifying the `.dlg` file but not the `.xdi` file, you should not make major changes to the dialog box (for example, do not add new items to the dialog box). If you do, the description in the `.dlg` file will not match the description in the `.xdi` file.

Linking the Structure Import/Export API library

To link the Structure Import/Export API library on Windows follow these steps:

1. In Development Environment 2003 (Version 7.1), open your client's project.

2. Choose Properties from the Project menu to display the Properties Pages dialog box.

3. In the Property Pages dialog box, click on Linker and then Input.

4. Add the Structure Import/Export API library `struct.lib` and the resource `fmstruct.res` to the Additional Dependencies field.

   Add `struct.lib` before `fdk.lib`, and add `fmstruct.res` to the end of the Object/Library Modules field.

5. Add the following link option to the 'Additional Options' field in 'Command Line' property page:

   ```
   /section:.rsrc,w
   ```

IMPORTANT: *This link option is required for some of the dialog boxes that are internal to the structure import/export functionality in FrameMaker. Without this link option, your client may crash when it interacts with these dialog boxes.*

Registering FDK clients

For FrameMaker to recognize your client, you must register it on the system on which you intend to run it. When registering your client, you can name it anything you like, although the name cannot contain spaces. Also, you should not use a name that is already used by one of the clients that ships with FrameMaker.

To register your client, you can use either of the following methods:

- Create and include in your client's project a VERSIONINFO resource that contains information about the client, and copy or move the compiled client into the `fminit\Plugins` directory.

  When FrameMaker starts, it recursively scans the `fminit\Plugins` directory and automatically registers any clients found. For more information on registering your client using the VERSIONINFO resource, see Registering a client by using the VERSIONINFO resource.

- Add an entry for your client in the `[APIClients]` section of the `maker.ini` file in the FrameMaker directory.

  The `[APIClients]` section of the `maker.ini` file lists the FDK clients to load when FrameMaker starts.

  For more information on registering your client using the `maker.ini` file, see "Registering clients in the FrameMaker maker.ini file" on page 36.

IMPORTANT: *You can use either method of registering a client, but do not use both methods for the same client.*

Registering a client by using the VERSIONINFO resource

To register a client using the VERSIONINFO resource, you create a VERSIONINFO resourcescript, or add a VERSIONINFO resource to an existing resource script for the client. Then you insert the `.rc` file into the client's project, compile the client, and then copy or move the compiled client into the `fminit/Plugins` directory. You can also place the client in a subdirectory of the `fminit/Plugins` directory.

When FrameMaker starts, it recursively scans the `fminit/Plugins` directory and subdirectories, and automatically registers any clients found. The client must be named *client_name*`.dll` and it must contain valid VERSIONINFO resource information. (The `.dll` extension is the default value set in the "PluginExtensions" entry in the [Preferences] section of the `maker.ini` file.) FrameMaker ignores any filenames in the `fminit/Plugins` directory and subdirectories that do not contain the `.dll` extension (or other extensions defined in the `maker.ini` file). FrameMaker determines a client's type and other properties

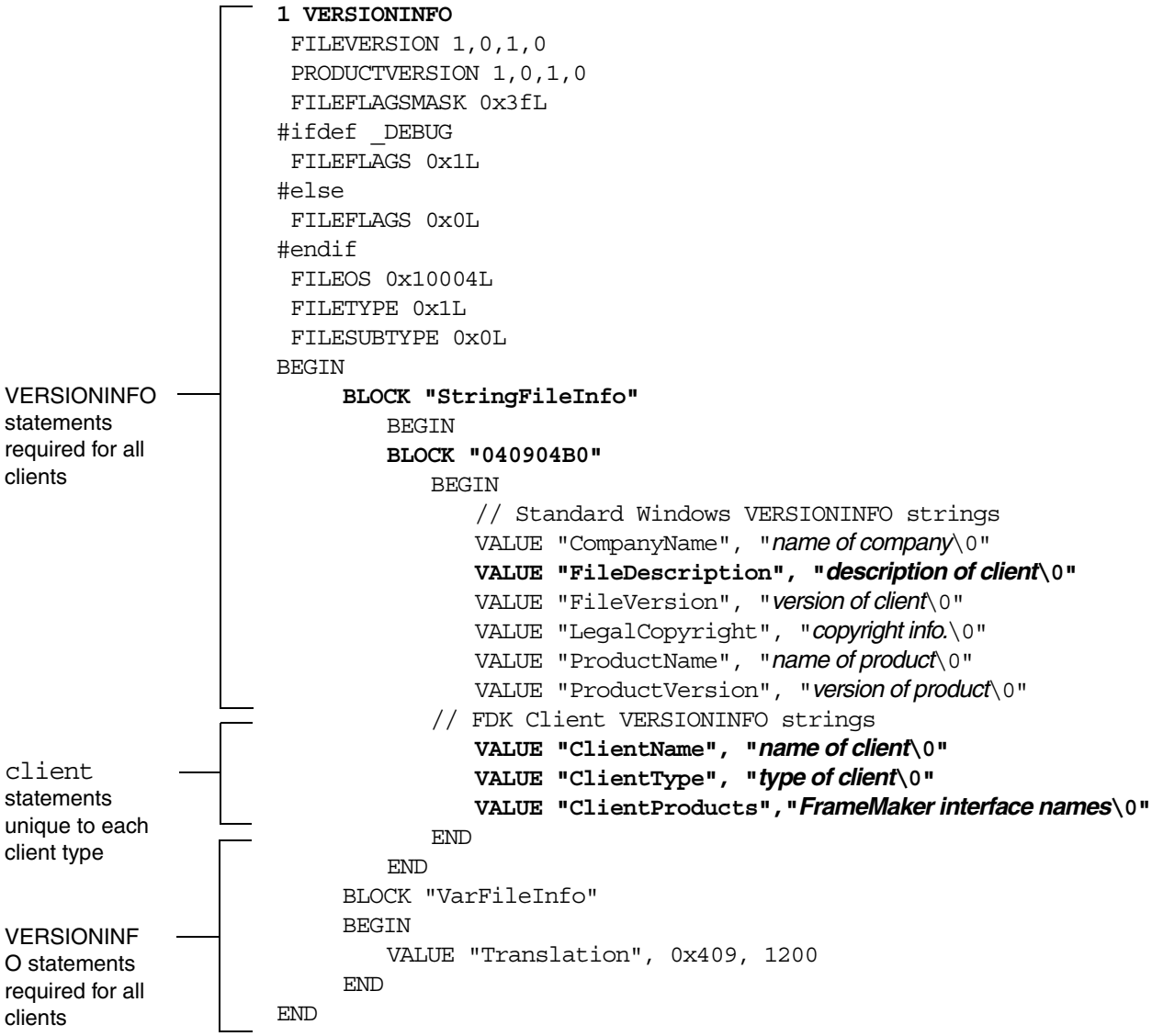
by examining the client's VERSIONINFO resource. FrameMaker ignores any files that do not have the required version information resource information.

The VERSIONINFO resource information required for all clients

The types of information at the beginning and end of the VERSIONINFO resource is the same and required for all clients. The resource must also contain `client` statements which are unique to each type of client. This section describes the resource information that is required for all clients. The following sections describe the `client` statements for each type of client. For detailed information on creating and using the VERSIONINFO resource, see the documentation for Microsoft Development Environment 2003 (Version 7.1).

The following code illustrates the syntax of a VERSIONINFO resource for n FDK client. The statements in bold are used by FrameMaker and must be specified exactly as shown. The statements not shown in bold are required by Microsoft Development Environment 2003 (Version 7.1), but they are not used by FrameMaker. The nesting of the statements must also be set exactly as shown.

The table that follows the code describes the statements shown in bold.




```
1 VERSIONINFO
 FILEVERSION 1,0,1,0
 PRODUCTVERSION 1,0,1,0
 FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
 FILEFLAGS 0x1L
#else
 FILEFLAGS 0x0L
#endif
 FILEOS 0x10004L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
BEGIN
     BLOCK "StringFileInfo"
        BEGIN
        BLOCK "040904B0"
           BEGIN
              // Standard Windows VERSIONINFO strings
              VALUE "CompanyName", "name of company\0"
              VALUE "FileDescription", "description of client\0"
              VALUE "FileVersion", "version of client\0"
              VALUE "LegalCopyright", "copyright info.\0"
              VALUE "ProductName", "name of product\0"
              VALUE "ProductVersion", "version of product\0"
           // FDK Client VERSIONINFO strings
              VALUE "ClientName", "name of client\0"
              VALUE "ClientType", "type of client\0"
              VALUE "ClientProducts","FrameMaker interface names\0"
           END
        END
     BLOCK "VarFileInfo"
     BEGIN
        VALUE "Translation", 0x409, 1200
     END
END
```

For this statement	Specify
`1 VERSIONINFO`	A version ID of `1` at the beginning of the VERSIONINFO resource file.
`BLOCK "StringFileInfo"`	Specify `"StringFileInfo"`.
`BLOCK "040904B0"`	Specify `"040904B0"`.
`VALUE "FileDescription"`	Specify a description for the client. FrameMaker displays the description of a client when the user clicks About. The description can contain spaces.

IMPORTANT: *The version ID in the VERSIONINFO resource file must be* `1` *or else the VERSIONINFO resource file will be invisible to FrameMaker. Although identifying the version ID in the VERSIONINFO resource file is the easiest method, you could also choose to use a symbolic constant such as Development Environment 2003 (Version 7.1)'s "VS_VERSION_INFO:" string. However, if you choose to use "VS_VERSION_INFO:", it is critical that you define this string to be a number, and that you define it to be* `1`.

The FILEVERSION statement is often used by installation programs to avoid overwriting new versions of clients with older versions.

The following sections describe the `client` statements for each type of client.

The VERSIONINFO resource information for standard FDK clients

To register an FDK client that is not a filter, take control client, or document report, add the following `client` statements to the VERSIONINFO resource in this syntax:

```
1 VERSIONINFO
     ...
// FDK Client VERSIONINFO strings
     VALUE "ClientName", "name of client\0"
     VALUE "ClientType", "type of client\0"
     VALUE "ClientProducts", "names of FrameMaker interfaces\0"
     ...
```

For this statement	Specify
`ClientName`	The registered name for your client. It does not need to be the same as your client's executable filename. Be sure your client name does not conflict with the names of other clients, including those such as ClickPrint that are supplied with FrameMaker. The `ClientName` string is required for all clients.
`ClientType`	`Standard` for a standard client. The `ClientType` string is required for all clients.
`ClientProducts`	The name of FrameMaker product interfaces that you want your client to run with. These names are expressed in terms of the old FrameMaker products. Separate interface names with a space. You can specify `Maker`, `MakerSGML`, or both values. This string is optional for all clients; if you exclude ityour client runs with `Maker` and `MakerSGML`.

Example

For example, you can use the following `client` statements in a VERSIONINFO resource for a standard client:

```
1 VERSIONINFO
     ...
// FDK Client VERSIONINFO strings
     VALUE "ClientName", "api1\0"
     VALUE "ClientType", "Standard\0"
...
```

In the preceding example, FrameMaker can initialize the client under the structured and unstructured program interfaces. If you want only structured FrameMaker to initialize your client, add the `ClientProducts` statement as follows:

```
1 VERSIONINFO
     ...
// FDK Client VERSIONINFO strings
     VALUE "ClientName", "api1\0"
     VALUE "ClientType", "Standard\0"
     VALUE "ClientProducts", "MakerSGML\0"
...
```

The VERSIONINFO resource information for filters

A *filter* is an FDK client that converts FrameMaker files to or from other file formats. To register a filter, add the following `client` statements to the VERSIONINFO resource in this syntax:

```
1 VERSIONINFO
     ...
// FDK Client VERSIONINFO strings
     VALUE "ClientName", "name of client\0"
     VALUE "ClientType", "type of filter\0"
     VALUE "ClientProducts", "names of FrameMaker product interfaces\0"
     VALUE "ClientFacet", "name of file format\0"
     VALUE "ClientFormatID", "format id of filter\0"
     VALUE "ClientVendor", "vendor of filter\0"
     VALUE "ClientSuffix", "filename suffix of filterable files\0"
     ...
```

For this statement	Specify
`ClientName`	For import filters, this name appears in the Unknown File Type dialog box. For export filters, this name appears in the Format menu of the Save As dialog box. The `ClientName` string is required for all filter clients.
`ClientType`	The `ClientType` string is required for all filter clients. ● `TextImport` ● `GFXImport` ● `ExportFilter` ● `FileToFileTextImport` ● `FileToFileTextExport` ● `FileToFileGFXImport` ● `FileToFileGFXExport`

For this statement	Specify
ClientProducts	The name of FrameMaker product interfaces that you want your client to run with. These names are expressed in terms of the old FrameMaker products. Separate interface names with a space. You can specify Maker, MakerSGML, or both values. This string is optional for all clients; if you exclude it, your client runs with Maker and MakerSGML.
ClientFacet	The name of the file format. This name is used in the [FormatList] section of the maker.ini file if you want FrameMaker to automatically recognize the file format. For more information on automatic recognition of file formats, see "Automatic recognition of a file format" on page 17. The ClientFacet string is required for all filter clients.
ClientFormatID	A four-character string that identifies the file format. For more information on format IDs, see "Identifying your filter" on page 13. The ClientFormatID string is required for all filter clients.
ClientVendor	A four-character string that identifies the vendor of the filter. For more information on vendor IDs, see "Identifying your filter" on page 13. The ClientVendor string is required for all filter clients.
ClientSuffix	The filename extension of filterable files. The ClientSuffix string is required for all filter clients.

Example

For example, you can use the following client statements in a VERSIONINFO resource for a filter client:

```
1 VERSIONINFO
     ...
// FDK Client VERSIONINFO strings
     VALUE "ClientName", "KurtWrite Files\0"
     VALUE "ClientType", "TextImport\0"
     VALUE "ClientProducts", "Maker MakerSGML\0"
     VALUE "ClientFacet", "kurt\0"
     VALUE "ClientFormatID", "KRT\0"
     VALUE "ClientVendor", "FAPI\0"
     VALUE "ClientSuffix", "krt\0"
        ...
```

If you start FrameMaker and open a file with a .krt extension, FrameMaker uses the KurtWrite Files filter.

The VERSIONINFO resource information for take-control clients

To register a take control client, add the following client statements to the VERSIONINFO resource in this syntax:

```
1 VERSIONINFO
     ...
// FDK Client VERSIONINFO strings
     VALUE "ClientName", "name of client\0"
     VALUE "ClientType", "type of client\0"
     VALUE "ClientProducts", "names of FrameMaker interfaces\0"
     ...
```

For this statement	Specify
`ClientName`	The registered name for your client. It does not need to be the same as your client's executable filename. Be sure your client name does not conflict with the names of other clients, including those such as ClickPrint that are supplied with FrameMaker. The `ClientName` string is required for all clients.
`ClientType`	Use `TakeControl` for a take control client. The `ClientType` string is required for all clients.
`ClientProducts`	The name of FrameMaker product interfaces that you want your client to run with. These names are expressed in terms of the old FrameMaker products. Separate interface names with a space. You can specify `Maker`, `MakerSGML`, or both values. This string is optional for all clients; if you exclude ityour client runs with `Maker` and `MakerSGML`.

Example

For example, you can use the following `client` statements in a VERSIONINFO resource for a take control client:

```
1 VERSIONINFO
     ...
// FDK Client VERSIONINFO strings
     VALUE "ClientName", "api1\0"
     VALUE "ClientType", "TakeControl\0"
     ...
```

The VERSIONINFO resource information for document reports

A *document report* is an FDK client that provides information about a document. To register a document report, add the following `client` statements to the VERSIONINFO resource in this syntax:

```
1 VERSIONINFO
     ...
// FDK Client VERSIONINFO strings
     VALUE "ClientName", "name of document report\0"
     VALUE "ClientType", "type of client\0"
     VALUE "ClientProducts", "names of FrameMaker interfaces\0"
     ...
```

For this statement	Specify
`ClientName`	The name that appears in the Document Reports dialog box. The `ClientName` string is required for all clients.
`ClientType`	Use `DocReport` for a document report client, The `ClientType` string is required for all clients.
`ClientProducts`	The name of FrameMaker product interfaces that you want your client to run with. These names are expressed in terms of the old FrameMaker products. Separate interface names with a space. You can specify `Maker`, `MakerSGML`, or both values. This string is optional for all clients; if you exclude ityour client runs with `Maker` and `MakerSGML`.

Example

For example, you can use the following `client` statements in a VERSIONINFO resource for a document report client:

```
1 VERSIONINFO
     ...
// FDK Client VERSIONINFO strings
     VALUE "ClientName", "Aframes Document Report\0"
     VALUE "ClientType", "DocReport\0"
     VALUE "ClientProducts", "All\0"
     ...
```

Inspecting a client's version resource information

After compiling a client, you can check its version resource information to make sure all of the values are correct. To display the version resource information, right-click the client's icon on the desktop or in Explorer and choose Properties from the pop-up menu that appears. In the Properties dialog box, choose the Version tab to display the resource information for the client.

Registering clients in the FrameMaker maker.ini file

The other method of registering a client is by adding an entry for the client in the FrameMaker `maker.ini` file. The `[APIClients]` section of the `maker.ini` file lists the FDK clients to load when FrameMaker starts. Each client description must be on a separate line and cannot contain line breaks. Clients that are not filters use the following format:

client = type, description, DLL_file, mode

where

For this statement	Specify
client	the client's name
type	the type of client—valid types for clients other than filters are Standard, TakeControl, and DocReport
DLL_file	the pathname of the client's DLL file—can specify a full pathname or a relative pathname based on the FrameMaker installation directory.
mode	whether the client can run with FrameMaker in unstructured or structured mode. This fields can be one of maker, structured, or all. The mode field is required.

The fields in this line are separated by a comma and zero or more spaces.

For example, if you have compiled the aframes sample client into c:\fdk\samples\aframes\debug\aframes.dll, and you want to register it with FrameMaker, add the following to the maker.ini file in the FrameMaker installation directory (without any line breaks):

```
AFrames=DocReport,Anchored Frames Report,c:\fdk\samples\aframes\
debug\aframes.dll, all
```

If the client is a filter, register it with the following line:

client = type, facet_name, format_id, vendor_id, display_name, description, DLL_file, mode, suffix

where the variables are:

For this statement	Specify
type	One of: • TextImport • GFXImport • Export • FileToFileTextImport • FileToFileTextExport • FileToFileGFXImport • FileToFileGFXExport
facet_name	the name of the file format supported by the client.
format_id	a four-character string that identifies the file format
vendor_id	a four-character string that identifies the client's provider.
display_name	the filter name to display in in dialog boxes when opening or saving a file of the given format. This name must match the client name.
description	a description of the client that appears when you choose About
DLL_file	the pathname of the client's DLL file

For this statement	Specify
mode	whether the client can run with FrameMaker in unstructured or structured mode. This fields can be one of maker, structured, or all. The mode field is required.
suffix	the filename extension of the file type that the client filters

For information on format and vendor IDs, see "Writing filter clients" on page 13.

For example, assume you have a graphics import filter for the CGM format that uses ACGM as its facet name, has its executable stored in acgmflt.dll, and should be invoked on files with the suffix cgm. You can register this filter with this line:

```
ACGMFILTER=GFXImport,ACGM,CGM,FAPI,ACGMFILTER,acgmflt.dll,all, cgm
```

Specifying no description for a client

When you register your client by using the FrameMaker maker.ini file, and you don't want to specify a description, enter a space in the description field. For example:

```
client= Standard, ,c:\clients\myclient\debug\myclient.dll, all
```

The description field must contain at least one character. If no characters appear between the commas delimiting the description field, your client will not be registered.

Running FDK clients

When FrameMaker starst, it reads the maker.ini The [APIClients] section of the maker.ini file contains entries describing the FDK clients to be loaded.

FrameMaker then scans the fminit/Plugins directory and subdirectories and loads the FDK clients that have a .dll file extension and valid VERSIONINFO resource information. FrameMaker ignores any files in the fminit/Plugins directory and subdirectories that do not have a name with the .dll extension, or do not contain valid VERSIONINFO resource information.

For information on how FrameMaker starts a client, see Chapter 2, "API Client Initialization," in the *FDK Programmer's Guide*.

Compatibility between FDK and FrameMaker product releases

To ensure your existing Windows clients are compatible with release 7.2 of FrameMaker, you should recompile them. It is possible to run a a client compiled in an earlier version of the FDK with FrameMaker 7.2, as long as the client does not use any functions or properties that have changed. However, it is recommended that you recompile your clients with the newer version of the FDK as soon as possible.

Disabling FDK clients

To disable all FDK clients, edit the following line in the `maker.ini` file in the FrameMaker installation directory, or in the version of the `.ini` file that is stored in the user's `Documents and Settings` directory:

```
API=On
```

Replace `On` with `Off`. The next time you start FrameMaker, no FDK clients will be started.

IMPORTANT: *Some FrameMaker features, such as the Word Count document report, Save As HTML, or import and export of SGML and XML are implemented as FDK clients. If you disable all FDK clients, these features will not be available.*

Debugging FDK clients

You debug your client as part of the FrameMaker executable. The FrameMaker executable is not compiled with debugging information, so you don't have access to any symbols within FrameMaker.

To use Development Environment 2003 (Version 7.1) to debug your client as part of the FrameMaker executable, follow these general steps:

1. Start Development Environment 2003 (Version 7.1).
2. Open your client's project and add breakpoints.
3. Select Project->Properties and then Debugging page. Go to Command Field and add the path to FrameMaker executable.pen the FrameMaker executable.

   For example, if FrameMaker is installed in `c:\Program Files\Adobe\FrameMaker7.2`, then to open its executable, open `c:\Program Files\Adobe\FrameMaker7.2\FrameMaker.exe`.
4. From the Build menu select Configuration Manager. Highlight the Debug Project Configuration.
5. From the Debug menu, choose Start.

   Alternately, if you have already started the debugger for your program, from the Debug menu choose Restart. If FrameMaker isn't able to load your client, it displays the following error message in an alert box:
   `File Error: Cannot find` *client_name*`.dll`

   FrameMaker may not be able to load your client for the following reasons:
   - The client is not located in the fminit/Plugins directory or subdirectories, or does not have a name with the `.dll` extension.
   - The client's VERSIONINFO resource information is missing or invalid.
   - The `maker.ini` file doesn't specify the correct full pathname for your client's DLL.

– The FrameMaker release is incompatible with the FDK release that you used to compile the client.

To check that your FDK client has control, you can have it display a string in the status bar of the document or book window. For more information, see the descriptions of `FO_Book` and `FO_Doc` in the *FDK Programmer's Reference*.

5 Writing an Asynchronous FDK Client

This chapter describes how to create asynchronous clients on Windows, and provides instructions for compiling and running a sample asynchronous client. Before writing an asynchronous API client you should be familiar with both the FrameMaker FDK and Windows API programming. Also read the UNIX platform guide for more information on Asynchronous clients.

The purpose of many FDK clients is to modify FrameMaker in some way, such as by changing or adding functionality. In these applications the main goal of the resultant application is still for the end user to use FrameMaker.

A different kind of application is one that uses FrameMaker to support some aspect of the application's functionality, but in which use of FrameMaker is not the goal. For example, you might create a data base and want to use FrameMaker to print catalogs from it. In this case, your application runs primarily independently of FrameMaker, but calls FrameMaker (possibly as a child process) during some part of its operation.

The UNIX and Windows versions of the FDK allow you to create asynchronous applications that control a FrameMaker process. Even though the main purpose of the application may not be to run FrameMaker, this chapter refers to such an application as an FDK client, since it calls FDK functions.

An asynchronous client does not run as part of the FrameMaker process nor as a child process. Instead, it is its own application in a separate process, communicating with a FrameMaker process via Microsoft RPC (Remote Procedure Calls). You should be aware of some consequences of this difference:

- An asynchronous client can be started independently of any FrameMaker product. It can be an EXE, or a DLL of some EXE other than FrameMaker.
- It must have its own `main()` function.
- You can use MFC or any other application framework to develop an asynchronous client.
- An asynchronous client can run on a machine other than that running the associated FrameMaker process.

End user installations

To run asynchronous clients, the executable applications or the DLL files must be installed correctly. An EXE can be installed wherever the user wants. A DLL that is a plugin for another application must be installed correctly for that application. A DLL that is a plugin for FrameMaker must be installed in the appropriate Plugins directory, or its path must be specified in the `maker.ini` file.

The user also must have the following files installed in his or her FrameMaker installation directory, at the same level as the FrameMaker application:

- afmfdk.dll
- fmrnclnt.exe

In addition, the user must have the following entries in the `maker.ini` file:

```
[Files]
MarshallingDLL = afmfdk.dll
RunWrappedPlugin = fmrnclnt.exe
. . .
[Preferences]
ExecutablePlugins = EXE
WrappedPlugins = DLX
PluginExtensions = DLL, DLX, EXE
```

The `[Preferences]` entries tell FrameMaker which filename extensions are valid for different types of clients.

- `PluginExtensions` must list extensions for all the files you want to be loaded as clients of any type.
- `ExecutablePlugins` lists extensions for clients that are built as executables which run outside of the FrameMaker process.
- `WrappedPlugins` lists extensions for clients that are built as DLLs, but will run in an address space that is external to the FrameMaker process. Such a client uses `fmrnclnt.exe` to wrap its DLL and runs in the `fmrnclnt.exe` address space.

Note that you can substitute other extensions for the ones shown in the example above. For more information, see "Types of asynchronous clients" on page 43.

Registering asychronous clients

You can register asynchronous clients just as you register other clients; you can store the registration data in the client's VersionInfo resource, or you can make an entry in the `maker.ini` file for FrameMaker. Additionally, your client can pass an `F_PropValsT` structure to `F_ApiWinConnectSession()` that is a list of registration data.

`F_ApiWinConnectSession()` is defined as:

```
F_ApiWinConnectSession(const F_PropValsT *connectProps,
     ConStringT hostname, const struct _GUID *service);
```

You can include the following properties in `connectProps`:

This property	corresponds to this statement in a client's VERSIONINFO resource
`FI_PLUGIN_NAME`	the name of the client.
`FI_PLUGIN_TYPE`	the type of client.

This property	corresponds to this statement in a client's VERSIONINFO resource
FI_PLUGIN_PRODUCTS	specifies structured or unstructured FrameMaker, using the names of FrameMaker products this client supports—use a space-delimited string with one or both of Maker and MakerSGML
FI_PLUGIN_FACET	the name of the file format supported by the client (filters, only)
FI_PLUGIN_FORMATID	a four-character string that identifies a file format (filters, only).
FI_PLUGIN_VENDOR	a four-character string that identifies the client's provider.
FI_PLUGIN_SUFFIX	the filename extension of the file type that the client filters (filters, only).
FI_PLUGIN_INFORMAT	the file format for the file to filter (filters, only)
FI_PLUGIN_OUTFORMAT	the file format for the resulting file (filters, only)
FI_PLUGIN_DESCRIPTION	a description of the client that appears when you choose About.
FI_PLUGIN_PRODUCTNAME	the name by which customers know your client.

If connectProps is NULL, the FrameMaker process uses the client's VersionInfo resource or the entries in the maker.ini file. If the client has no registration information in any of these sources, the FrameMaker process registers it as a standard client.

For more information about registering clients, see "Compiling, Registering, and Running FDK Clients" on page 23.

Types of asynchronous clients

Asynchronous clients can be executable applications (EXE), dynamically linked libraries (DLLs) that are a part of another application, or DLLs that are plugins for FrameMaker (wrapped plugins).

Asynchronous EXE applications

An EXE can be either a console application or a Windows application. After connecting with the FrameMaker process, the EXE application passes calls to FrameMaker through afmfdk.dll.

IMPORTANT: *Because they don't have a Windows message processing loop, console applications cannot handle notifications from the FDK. For example, this means a console application cannot process commands from menus it adds to FrameMaker. Nor can it process notifications such as* FA_Note_PreOpenDoc *or* FA_Note_PreSaveDoc.

Asynchronous DLLs

A DLL that is part of another application can call `F_ApiStartUp()` to make a connection with a FrameMaker process. For example, you could write a plugin for Acrobat Exchange that writes the data from Acrobat Forms to a FrameMaker document. In that case, the DLL communicates with the FrameMaker process, as a part of its parent EXE, via `afmfdk.dll`.

A DLL that runs as a wrapped plugin for FrameMaker runs in its own memory space. After connecting with the FrameMaker process, the DLL invokes `fmrnclnt.exe` to run as a wrapper for the DLL. The wrapped DLL then communicates with FrameMaker via `afmfdk.dll`, as though it is an EXE.

Registering multiple FrameMaker processes as servers

When you first run FrameMaker, it registers istelf in the system registry as the default instance of the FrameMaker instance on that machine. By default, asynchronous clients connect to this instance.

You can register multiple instances of the FrameMaker process, each with a unique entry in the system registry. Then you can use these processes as a bank of servers, and your client can choose among them when making a connection.

You identify a FrameMaker process as a server by its entry in the system registry. The entry can specify:

- A name to identify the GUID for that specific process.
- Whether the process starts up when called by a client, or whether it must already be running before the client can connect to it.

To register a process, you start FrameMaker with specific commandline options. This creates an entry in the system registry for the machine on which you start FrameMaker.

To start FrameMaker with commandline options:

1. Choose Run from the Start menu.

   The Run Application dialog box appears.

2. In the text box, type the full pathname of the `FrameMaker.exe` file, followed by the commandline options.

   Alternately, you can start FrameMaker from a DOS Command Prompt window. For example, type *FrameMaker_path*`\FrameMaker7.2 /`*option*, where *FrameMaker_path* is the install path for the version of FrameMaker you want to run, and `/`*option* is one or more of:

- progid:process_name

  where `process_name` is a name you provide. This option registers a name for the FrameMaker process.

- auto

  This option allows the FrameMaker process to automatically start up if it isn't running when another process calls it.

- noauto

  This option disallows automatic start-up.

This creates an entry in the system registry for the machine on which you started FrameMaker.

Registering a name for a FrameMaker process

To specify a name for the process, use the `/progid` option. For example, type *FrameMaker_path*`\FrameMaker7.2 /progid:MyProcess.Api1`, where *`FrameMaker_path`* is the install path for the version of FrameMaker you want to run. This establishes a name, `MyProcess.Api1`, for the process.

When you start FrameMaker with no `/progid` option, you create system registry entry with the default name of `FrameMaker.API.1.`

Asynchronous clients running locally on the host can refer to processes by their names. In this way, your client can choose which process to run for a given task.

IMPORTANT: *Clients connecting to a remote host cannot use the process name to connect to a FrameMaker process. Instead, they must use the GUID for that process, as it is specified in the system registry.*

Registering automatic start-up for a process

If the FrameMaker process is not running, an asynchronous client can still call it. If the process is so registered, it will start up when the client calls it. Alternatively, you can register the process in a way that does not allow automatic start-up.

To register the process for automatic start-up, use the `/auto` option.

To disallow automatic start-up, use the `/noauto` option.

For example, type *FrameMaker_path*`\FrameMaker7.2 /progid:MyProcess.Api1 /auto`, where *`FrameMaker_path`* is the install path for the version of FrameMaker you want to run. This establishes a process named MyProcess.Api1, which will start automatically when an asynchronous client calls it.

Running asynchronous clients on remote hosts

With systems that support DCOM, you can run a client on one machine (the client machine), connected to a FrameMaker process on another machine (the host machine). To accomplish this, you make use of the DCOM services provided with your operating system. Also, both machines must be in the same domain, and the same user must have the accounts on both machines.

For an asynchronous client to connect to a FrameMaker process on a remote host:

1. Register the FrameMaker process as a server process on the host machine.

   This establishes entries on the host machine's system registry for the FrameMaker processes you want to run as servers. For more information see "Registering multiple FrameMaker processes as servers" on page 44.

2. Run dcomcnfg on the host machine to configure DCOM accessibility for each process you want to run as a server.

   This enables DCOM connections to the FrameMaker server processes that are registered on the host machine.

3. Run dcomcnfg on the client machine to configure its DCOM accessibility.

   This enables the client machine to connect to the host machine via DCOM.

Enabling DCOM for the server processes on the host

To enable DCOM for a FrameMaker process on the host machine:

1. Choose Run from the Start menu.

   The Run dialog box appears.

2. In the Run dialog box, type dcomcnfg

   The DCOM Configuration Properties service application appears.

3. Select the Default Properties tab and click Enable Distributed COM on this Computer.

4. In the Applications list box, double-click the FrameMaker process you want to enable, then set the appropriate security options.

5. Click the Security tab and make sure *Use default configuration permissions* is turned on.

6. Apply any other settings to the FrameMaker process or your computer that are appropriate for your network configuration.

   You should check with the system administrator to ensure the options you set are compatible with his administration procedures.

7. Click Ok.

Enabling DCOM for client machine

To enable DCOM on the client machine:

1. Choose Run from the Start menu.

   The Run dialog box appears.

2. In the Run dialog box, type dcomcnfg

   The DCOM services application appears.

3. Select the Default Properties tab and click Enable Distributed COM on this Computer.

4. Apply any other settings to your computer that are appropriate for your network configuration.

   You should check with the system administrator to ensure the options you set are compatible with his administration procedures.

5. Click Ok.

To find more information on DCOM see the Windows On-line Help.

Connecting with a FrameMaker process

Asynchronous clients connect with a FrameMaker process by calling `F_ApiStartUp()` or `F_ApiWinConnectSession()`. When connecting to a process on a local host, FrameMaker does not have to be registered as a server. For a process on remote host, your client must know the GUID for that process.

A machine may have more than one FrameMaker process running at a time. In that case, the processes must be registered as servers, and they should be registered with a name for each process. For information about registering FrameMaker processes as servers, see "Registering multiple FrameMaker processes as servers" on page 44.

IMPORTANT: *Asynchronous clients use COM to communicate with FrameMaker processes. If any FDK call returns* `FE_Busy`*, then you probably need to register a message filter.*

When using COM, an application should always register a message filter. If your code calls `F_ApiStartUp()` *or* `F_ApiWinConnectSession()` *before initializing COM, these routines automatically initialize COM and register a message filter.*

However, if you initialized COM before calling these routines, they assume your application already registered a message filter. If your application initializes COM but does not register a message filter, be sure to call `F_ApiWinInstallDefaultMessageFilter()`.

Connecting to the default process on a local host

You use `F_ApiStartUp()` when the desired FrameMaker process is running on the local machine. For example, a DLL that is a FrameMaker plugin calls `F_ApiStartUp()`. In that case, the FrameMaker process that invokes the DLL identifies itself by passing a globally unique identifier (GUID) via the FMGUID environment variable. Likewise, if you want an EXE to connect locally to the currently active FrameMaker process, use `F_ApiStartUp()`.

The following call makes this connection:

```
F_ApiStartUp(NULL);
```

For more information, see `F_ApiStartUp()` in the *FDK Programmer's Reference*.

Connecting to a named process on a local host

To connect to a named process on a local machine, you need to convert the process name to a GUID. Then you can pass that GUID to `F_ApiWinConnectSession()` to initiate communication between your client and the FrameMaker process.

Note that `F_ApiStartUp()` makes a reliable connection only when the desired FrameMaker process is the *only* FrameMaker process running on the local host. If no FrameMaker process is running, `F_ApiStartUp()` will not work. Also, if more than one process is running, `F_ApiStartUp()` cannot determine which process will finally connect with your client.

To choose one of many FrameMaker processes on a local host, you should have all of the processes registered as servers on that host. For more information, see "Registering multiple FrameMaker processes as servers" on page 44.

If you have registered the process as a named server, and your client is connecting to it on a local host, you can use the Win32 API to get the GUID associated with that name. Then you pass the GUID to `F_ApiWinConnectSession()`.

The following example uses the Win32 API function `CLSIDFromProgID()` to get the GUID for a process named `MyProcess.Api1`. It then calls `F_ApiWinConnectSession()` to connect to the process. Note that you need a Unicode string for the process name. The example uses the Win32 API call, `MultiByteToWideChar()` to convert a string to Unicode.

```
#define WBUFLEN 512
OLECHAR progStr;
CLSID serviceId;
StringT myProcess = F_StrCopyString("MyProcess.API.1");
. . .
progStr = (OLECHAR*)malloc( WBUFLEN*sizeof(wchar_t) );
MultiByteToWideChar(CP_ACP, 0, (char *)opt_progid, -1, progStr, WBUFLEN );
if(CLSIDFromProgID(progStr, &serviceId))
     F_ApiConnectWinSession(0, 0, &serviceId);
. . .
```

Note that `F_ApiWinConnectSession()` takes three parameters. In the first parameter you can pass a list of properties that correspond to the entries you provide when registering a FrameMaker client.

The second parameter is for the address of a remote host, when making a connection to a remote host. If this parameter is NULL or 0, `F_ApiWinConnectSession()` connects to the local host.

The third parameter specifies the desired FrameMaker process on the host machine. If this parameter is `NULL` or `0`, `F_ApiWinConnectSession()` uses the value of the FMGUID environment variable on the specified host.

For more information, see `F_ApiWinConnectSession()` in the *FDK Programmer's Reference*.

Connecting to a remote host

To connect to a remote machine, you need the address of that machine. Once you have the address, you can call `F_ApiWinConnectSession()` to initiate communication between your client process and the FrameMaker process on the host machine. The following call makes this connection to the currently running FrameMaker process on the remote host:

```
F_ApiWinConnectSession(0, remote, 0);
```

where `remote` is the address of the remote host.

The above call only works when the desired FrameMaker process is the *only* FrameMaker process running on the remote host. If no FrameMaker process is running, this will not work. Also, if more than one process is running, you cannot predict which process will finally connect with your client.

To choose one of many FrameMaker processes on a remote host, you should have all of the processes registered as servers on that host. For more information, see "Registering multiple FrameMaker processes as servers" on page 44.

To choose a registered process, you must know the GUID for that process ahead of time; you pass that GUID to `F_ApiWinConnectSession()`. Assuming you have specified a GUID in `serviceId`, the following call connects to a specific process on the remote host:

```
stringT remote;‘
CLSID serviceId;
. . .
F_ApiWinConnectSession(0, remote, &serviceId);
```

where `remote` is the address string of the machine that is running the FrameMaker process.

How to write an asynchronous FDK client

To write an asynchronous client that communicates with FrameMaker, you proceed as you would for any C application, providing a `main()` function and adding whatever functionality you need.

Unlike the UNIX FDK it is not necessary for a remote client to "take control" by calling F_ApiTakeControl(). In fact this function will not work on the Windows platform. Lacking this a Windows client can get control of a FrameMaker process by invoking F_ApiCallClient() to call itself. For the duration of the notification, that is while the client is processing the resulting callback, the client has exclusive control of FrameMaker.

At some point in its processing, your client needs to communicate with a FrameMaker process. To do so, it follows these general steps:

1. Connect to the FrameMaker process.

   To connect to a local host, use `F_ApiStartUp()` or `F_ApiWinConnectSession()`. To connect to a remote host, use `F_ApiWinConnectSession()`. For information about connecting to FrameMaker processes, see "Connecting with a FrameMaker process" on page 47. For information about the functions to connect to FrameMaker processes, see `F_ApiWinConnectSession()` and `F_ApiStartUp()` in the *FDK Programmer's Reference*.

2. Depending on your client, wait for requests from FrameMaker or perform some operations using FrameMaker.

   Once connected to a running FrameMaker process, your client can use the FDK to control the FrameMaker process, or receive notifications from it. However, bear in mind that console programs cannot handle notifications from the FDK. (This is because console programs do not have a Windows message processing loop; applications running in console programs must not request notifications.)

   Note that a client can take exclusive control of the FrameMaker process by requesting notification for `FA_Note_ClientCall` and then calling itself via `F_ApiCallClient()`. While handling the notification, no other clients can take control of the FrameMaker process.

3. When done, disconnect from the FrameMaker process.

   How your client disconnects depends on the situation.

   With a client that is a plugin for FrameMaker, you can call `F_ApiBailOut()` to terminate the client. After calling `F_ApiBailOut()`, the client's notification points are still registered with the FrameMaker process. If a notification event occurs, the FrameMaker process restarts the client by calling `F_ApiInitialize()` with initialization set to `FA_Init_Subsequent`. When it starts up subsequently, the client's global variable settings are lost.

   If the FrameMaker process still exists when your client is completely done communicating with it, your client should call the function `F_ApiDisconnectFromSession()` to break the RPC connection.

   Alternatively, the FrameMaker process may have shut down when your client wants to break the connection (for example, due to a user request or due to a command from your client). If so, your client should call the function `F_ApiShutDown()` to close its side of the RPC connection.

Writing a Main routine in Windows.

Unlike UNIX, Windows does not provide a default main routine for remote plugins. You must provide your own main routine. Simply include the following lines in your client:

```
#define DONT_REDEFINE /* We need to use native types. */
```

```
#include 'fapi.h'
#include <windows.h>
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow)
{
     return F_ApiRun();
}
```

The routine F_ApiRun() is documented in the FDK manuals and is implemented as follows:

```
IntT
F_ApiRun(VoidT)
{
            ConStringT s = F_ApiStartUp(NULL);
            if (s)
               F_ApiErr(s);
            else
while (!FA_bailout)
            F_ApiService(NULL);
            F_ApiShutDown();
        return s !=NULL;
}
```

All of this is documented in the FDK Platform Guide for UNIX except that in Windows `F_ApiStartup()` and `F_ApiService()` ignore their parameters and should be passed `NULL`.

It is not necessary to call `F_ApiRun()`. You may choose to implement your main routine using these primitives directly. If your program has a windows message loop you need only call `F_ApiStartup(NULL)`.

However if your remote plugin does not call F_ApiRun(), it must either periodically check the FA_bailout flag or arrange to terminate based on the FA_NotePostQuitSession notification. You must make these checks, otherwise FrameMaker can terminate leaving your client running.

Compiling and running a sample client

The following code sample is a console application that connects to the default FrameMaker session and gets the name of the active FrameMaker document. Following the code is a line-by-line description of how it works.

```
1. #define DONT_REDEFINE // Console app needs native types
2. #define WBUFLEN 512
3.
4. #include "fdetypes.h"
5. #include "futils.h"
6. #include "fapi.h"
7. #include "fstrings.h"
8. #include <windows.h>
```

```
#include <ddeml.h> //not required
#include <stdarg.h> //not required

int main(int argc, char **argv)
{
      StringT opt_progid;
      CLSID pclsid;
      LPOLESTR progStr;
      HRESULT res;
      F_ObjHandleT docId;

      // Get the process name.
      if(argc == 2)
         opt_progid = F_StrCopyString((StringT)argv[1]);
      else {
         fprintf(stderr, "You must provide a process name.");
         return(1);
      }

      // Convert the process name into a GUID
      progStr = (OLECHAR*)malloc( WBUFLEN*sizeof(wchar_t) );
      if(0 == MultiByteToWideChar(CP_ACP, 0, (char *)opt_progid, -1,
                  progStr, WBUFLEN )) {
         fprintf(stderr, "failed to allocate\n");
         return(1);
      }
      if (progStr[0] == '{') // hex-codes within brackets
         res = CLSIDFromString(progStr, &pclsid);
      else
         res = CLSIDFromProgID(progStr, &pclsid);

      if(res == S_OK)
         F_ApiWinConnectSession(NULL, NULL, &pclsid);
      if (!F_ApiAlive()) {
         fprintf(stderr, "No connection: %s\n", opt_progid);
         return 1;
      }
      // Print the name of the current document.
      docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
      if (docId) {
         StringT docname = F_ApiGetString(FO_Session, docId, FP_Name);
         fprintf(stderr, "Current document: %s\n", docname);
         F_ApiDeallocateString(&docname);
      } else
         fprintf(stderr, "No active document\n");

      return 0;
}
```

Line 1

To compile this client as a console application, you need to use types that are native to the C language. This statement keeps the FDE from redefining those types.

Lines 20–26

These lines parse the commandline options you pass to the client when you invoke it. You invoke the exe with the name of a FrameMaker process as an argument. To run the default process, use the name `FrameMaker.API.1`. For example, assuming the exe is named fmRemote.exe, type the following to invoke it with the default FrameMaker process:

```
fmRemote.exe FrameMaker.API.1
```

For more information, see "Registering a name for a FrameMaker process" on page 45.

Lines 28–38

These lines convert the process name into a valid GUID. Note that you need a Unicode string for the process name. The code uses the Win32 API call, `MultiByteToWideChar()` to convert the process name to Unicode. It then uses the Win32 API functions `CLSIDFromProgID()` or `CLSIDFromString()` to get the GUID for the specified process.

Lines 40–45

If you successfully retrieve a GUID for the process, these lines make the connection to a FrameMaker session.

Lines 46–56

Now that the client has connected with a session, it can use the FDK to interact with that session. These lines are standard FDK code to get the name of the active document for the current session. You can add code to perform other actions such as adding menus to the application window, manipulating the active document, or anything else you can do via the FDK.

IMPORTANT: *Because they don't have a Windows message processing loop, console applications cannot handle notifications from the FDK, such as menu commands or notifications such as* `FA_Note_PreSaveDoc`.

Compiling and registering the sample client

To compile the sample client in Microsoft Development Environment 2003 (Version 7.1), follow these steps:

1. Create a project for a console application.

   Use the Project Wizard to create a new project for a console application.

2. Set up the project options and settings as described in Chapter , "Compiling, Registering, and Running FDK Clients."

IMPORTANT: *Your link settings must include* `fdk.lib` *and* `api.lib`*. but neither* `fmdbms32.lib` *nor* `fmdebug.lib`*. In previous versions of the FDK,* `fmdbms32.lib` *and* `fmdebug.lib` *were required to compile. These libraries are now obsolete, but we include them so you don't have to change the link settings to compile existing FDK projects. If a remote client fails to start up and you see these libraries mentioned in the error text, then you must remove them from your link settings and recompile.*

3. Compile the client.

4. Register the client

   There are three ways to register an asynchronous client. See "Registering asychronous clients" on page 42.

   You must also be sure the end user has a correct installation to run asynchronous clients. See "End user installations" on page 41.

5. Connect the client with a named FrameMaker process

   To connect with a named FrameMaker process:

   – On your machine, register the FrameMaker process as a server

     See "Registering multiple FrameMaker processes as servers" on page 44. Be sure to register it with a name. See "Registering a name for a FrameMaker process" on page 45.

   – In a command window, type the filename for the client, followed with the name of the FrameMaker process the argument.

   – To connect to the default FrameMaker process, use the process name, `FrameMaker.API.1`.

   For example, type `remote.exe` *`process_name`*, where *`process_name`* is the name you assigned to a FrameMaker process. Note that unlss you registered the process to start up automatically, that process must be running when you invoke the sample client. See "Registering automatic start-up for a process" on page 45.

Summary of supporting functionality

Until such time as a common RPC protocol is supported on all platforms, Windows FDK clients can only communicate with Windows FrameMaker and UNIX FDK clients can only communicate with UNIX FrameMaker.

To support communication with a FrameMaker process, the Windows version of the FDK provides the following functions:

Function	Purpose
`F_ApiWinConnectSession()`	Initiates communication between the calling process and an identified FrameMaker process
`F_ApiDisconnectFromSession()`	Severs communication with a FrameMaker process
`F_ApiSetClientDir()`	Identifies a directory the FrameMaker process associates with an unregistered client
`F_ApiShutDown()`	Closes a client's connection with the API
`F_ApiWinInstallDefaultMessageFilter()`	Registers the default FDK message filter for a COM session.
`F_ApiService()`	useful if you are providing a replacement for `F_ApiRun()`. Used the same way as on UNIX but parameters are ignored.
`F_ApiStartup()`	Works as in Unix, parameter is ignored. See below.
`F_ApiAlive()`	
`F_ApiErr(message)`	Prints client name and message to console.
`F_ApiRun`	provides the minimum functionality required in an FDK client's `main()` function

Using `F_ApiStartup(F_FdFuncT)` the `F_FdFuncT` argument is ignored because Windows RPC is not based on sockets. `F_ApiStartup` queries the application's version information for client configuration data, if present, and connects to FrameMaker.

For information on these functions and properties, see the *FDK Programmer's Reference*.

Index

Numerics

A

C

D

E

F

G

H

I

J

K

L

M

O

P

Q

R

S

T

U

V

W

X