This manual, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. The content of this manual is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in this book.

Except as permitted by such license, no part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe Systems Incorporated.

Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner.

Any references to company names in sample templates are for demonstration purposes only and are not intended to refer to any actual organization. Adobe, the Adobe logo, Acrobat, Acrobat Reader, Adobe Type Manager, ATM, Display PostScript, Distiller, Exchange, Frame, FrameMaker, FrameViewer, InstantView, and PostScript are trademarks of Adobe Systems Incorporated.

Apple, PowerBook, QuickTime, Macintosh and Power Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries. Focoltone is a registered trademark of Gordon Phillips Limited. ImageStream Graphics Filters and ImageStream are registered trademarks of Inso Corporation. Microsoft, MS-DOS, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Quadrалay and WebWorks are registered trademarks of Quadrалay Corporation. PANTONE®, PANTONE MATCHING SYSTEM®, PANTONE Process Color System®, and POCE™ are trademarks of Pantone, Inc. Proximity and Linguibase are registered trademarks of Proximity Technology Inc. A Merriam-Webster is a registered trademark of Merriam-Webster Inc. Sun is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries. TRUMATCH is a registered trademark of Trumatch Inc. Unix is a registered trademark and X Window System is a trademark of The Open Group. Verity and TOPIC are registered trademarks of Verity, Inc. All other trademarks are property of their respective owners.

The following are copyrights of their respective companies or organizations: Portions reproduced with the permission of Apple Computer, Inc. © 1996 Apple Computer, Inc. Milo © 1988 Ron Avitzur PANTONE® Computer Video simulations displayed may not match PANTONE-identified solid color standards. Use current PANTONE Color Reference Manuals for accurate color. “PANTONE Open Color Environment” (POCE®), © Pantone, Inc., 1994, 1996. Pantone, Inc. is the copyright owner of “PANTONE Open Color Environment” (POCE®), and Software which are licensed to Adobe to distribute for use only in combination with the Adobe Software. “PANTONE Open Color Environment” (POCE®) and Software shall not be copied onto another diskette or into memory unless as part of the execution of the Adobe Software. The Spelling and Thesaurus portions of this product are based on Proximity Linguistic Technology. The Proximity/Merriam-Webster Linguibase © 1983, 1990 Merriam-Webster, Inc. C.A. Stromberg AB; Espasa-Calpe; Hachette; Idee/As; Kruger; Lluis de Yzaguirre i Maura; Merriam-Webster Inc.; Munksgaard Int. Publishers Ltd.; Nathan; Text & Satz Datentechnik; Van Dale Lexicographie bv; William Collins Sons & Co. Ltd.; Zanichelli. All rights reserved. Color Database © Dainippon Ink & Chemicals, Inc., licensed to Adobe. Outside In® Viewer Technology, 1992-1996 Inso Corporation; ImageStream® Graphics and Presentation Filters, 1992-1996 Inso Corporation. All rights reserved. TRUMATCH 4-Color Selector © 1992 Trumatch, Inc. All rights reserved. Portions copyrighted for the FrameViewer Retrieval Tools © 1988-1995 Verity, Inc. All rights reserved.

APPLE COMPUTER, INC. ("APPLE") MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE APPLE SOFTWARE. APPLE DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE APPLE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS, OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE APPLE SOFTWARE IS ASSUMED BY YOU. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME STATES. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

IN NO EVENT WILL, APPLE, ITS DIRECTORS, OFFICERS, EMPLOYEES, OR AGENTS BE LIABLE TO YOU FOR ANY CONSEQUENTIAL, INCIDENTAL, OR INDIRECT DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, AND THE LIKE) ARISING OUT OF THE USE OR INABILITY TO USE THE APPLE SOFTWARE EVEN IF APPLE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU.

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

Notice to U.S. government end users. The software and documentation are “commercial items,” as that term is defined at 48 C.F.R. §2.101, consisting of “commercial computer software” and “commercial computer software documentation,” as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the commercial computer software and commercial computer software documentation are being licensed to U.S. government end users (A) only as commercial items and (B) with only those rights as are granted to all other end users pursuant to the terms and conditions set forth in the Adobe standard commercial agreement for this software. Unpublished rights reserved under the copyright laws of the United States.
Contents

Using Frame Developer Tools

- The Frame Developer's Kit
- Other FrameMaker features for developers
- Choosing the right Frame tools
- FDK documentation
- Naming conventions
- Style conventions

Part I: Frame Application Program Interface (API)

Chapter 1 Introduction to the Frame API

- How the API works
- Special types of clients
- Portability
- Running clients with different FrameMaker product interfaces
- Creating and running a client
- A simple example

Chapter 2 API Client Initialization

- Responding to the FrameMaker product's initialization call
- Initialization types
- Disabling the API

Chapter 3 Creating Your Client's User Interface

- Using API dialog boxes to prompt the user for input
- Using commands, menu items, and menus in your client
- Replacing FrameMaker product menus and commands
- Allowing users to configure your client's interface
- Using hypertext commands in your client's user interface
Chapter 4 Executing Commands with API Functions

- Responding to user-initiated events or FrameMaker product operations
- Implementing quick keys
- Freeing system resources by bailing out

Chapter 5 Getting and Setting Properties

- What you can do with object properties
- Getting the IDs of the objects you want to change
- Manipulating properties
- Getting and setting session properties
- Getting and setting document properties
- Getting and setting graphic object properties
- Getting and setting paragraph properties
- Getting and setting book properties
- Getting and setting FrameMaker properties

Chapter 6 Manipulating Text

- Getting text
- Getting and setting the insertion point or text selection
- Adding and deleting text
Chapter 7 Manipulating Asian Text

- Creating a rubi group
- Text encodings
- Using encoding data
- Inspecting and manipulating encoded text
- Parsing an encoded string
- Getting the encoding for a text item
- Special issues with double byte encodings

Chapter 8 Creating and Deleting API Objects

- Creating objects
- Deleting objects
- Implicit property changes

Chapter 9 Manipulating Commands and Menus with the API

- How the API represents commands and menus
- Getting the IDs of commands and menus
- Determining a session's menu configuration
- Arranging menus and menu items
- Getting and setting menu item labels
- Manipulating expandomatic menu items
- Using check marks
- Using context-sensitive commands and menu items
- Providing context-sensitive Help for client menu items

Chapter 10 Creating Custom Dialog Boxes for Your Client

- Overview
- How to create a dialog box
- Creating a DRE file
- Designing the layout of the dialog box
- Setting the properties of the dialog box
- Setting the properties of a dialog item
Saving a DRE file .................................................. 204
Testing a dialog box ............................................. 205
A simple example ................................................. 206
General tips for dialog editing ................................. 209
Summary of keyboard shortcuts ............................... 209

Chapter 11  Handling Custom Dialog Box Events ........ 211
How the API represents dialog boxes ......................... 211
Overview of using a custom dialog box in your client .... 214
Opening dialog resources ....................................... 217
Initializing items in a dialog box ............................. 218
Displaying a dialog box ......................................... 219
Updating items in a dialog box ................................. 220
Handling user actions in dialog boxes ....................... 221
Closing a dialog box ............................................ 227

Chapter 12  Using Imported Files and Insets ............. 229
Types of imported files and insets ............................ 229
Importing text and graphics ................................... 230
Updating text insets ............................................. 235
Client text insets ................................................ 236
Writing filter clients ............................................ 239
Specifying format IDs and filetype hint strings ............ 247
Associating a file format with signature bytes ............ 255
Writing graphic inset editors .................................. 256

Part II: Frame Product Architecture

Chapter 13  Frame Session Architecture .................... 269
Identifying objects ............................................... 269
Representing object characteristics with properties .... 270
FrameMaker product sessions .................................. 274
Chapter 14  Frame Document Architecture. ........................................... 279
  Documents ................................................................. 279
  Global document information ........................................ 284
  Pages ............................................................................ 288
  Graphic objects .............................................................. 293
  Flows ............................................................................ 297
  Paragraph Catalog formats ............................................. 300
  Paragraphs ................................................................. 302
  Character Catalog formats ............................................. 305
  Condition Formats .......................................................... 309
  Text ............................................................................. 311
  Markers ........................................................................ 319
  Cross-reference formats .................................................. 322
  Cross-references ............................................................. 323
  Variable formats ............................................................ 324
  Variables .................................................................... 326
  Footnotes ................................................................. 327
  Ruling Formats .............................................................. 328
  Table Catalog formats .................................................... 330
  Tables ........................................................................ 332
  Colors ........................................................................ 338
  Structural element definitions ........................................ 342
  Format rules and format rule clauses .................................. 344
  Format change lists ......................................................... 346
  Structural elements .......................................................... 347

Chapter 15  Frame Book Architecture ..................................................... 351
  What the user sees .......................................................... 351
  How the API represents books ......................................... 351
  Creating new books and components .................................. 355
  Updating a book ............................................................. 356
  Using the book error log ................................................... 360
Part III: Frame Development Environment (FDE)

Chapter 16  Introduction to FDE .............................................. 365
  How the FDE works ......................................................... 365
  How to make your client portable ..................................... 366
  A simple FDE filter ......................................................... 370

Chapter 17  Making I/O and Memory Calls Portable ................. 375
  Initializing the FDE ......................................................... 375
  Using platform-independent representations of pathnames ....... 375
  Making I/O portable with channels .................................... 377
  Assertion-handler functions ............................................. 378
  Making memory allocation portable ................................... 378
  Error and progress reporting .......................................... 379

Chapter 18  FDE Utility Libraries ........................................... 381
  String library ............................................................. 381
  The string list library .................................................... 382
  Character library ........................................................ 382
  The I/O library ............................................................ 382
  The hash library .......................................................... 383
  Metric library ............................................................. 384
  MIF data structures and macros ....................................... 384
  The MIF library ........................................................... 385
  Simple MIF library ........................................................ 387
The Frame Developer’s Kit

The Frame® Developer’s Kit™ (FDK) provides tools for developers to enhance the functionality of FrameMaker®. This chapter provides an overview of the FDK and other aspects of FrameMaker that are useful for developers. It also discusses the FDK documentation.

The principal parts of the FDK are:

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

The following sections describe these parts and discuss how you can use them.

Frame API

The Frame API allows you to write C language programs, called FDK clients, that can take control of a FrameMaker product session and communicate interactively with the user. With the API, a client can do nearly everything an interactive user can do and more. The API gives a client direct access to the text and graphic objects in documents. The API includes a set of header files, libraries, and makefiles for each supported platform. Here are some examples of the types of clients you can create with the API:

- Grammar checkers
- Bibliography utilities
- Voice control utilities
- Document reporting utilities
- Version control systems for documents
- Table utilities, such as sorting and totaling
- Database publishing packages
- Interfaces to document management systems
- Filters to exchange files between other desktop publishing applications and FrameMaker
Using Frame Developer Tools

FDE

The Frame Development Environment (FDE) helps you make your client portable to all platforms supported by FrameMaker. It provides platform-independent alternatives to platform-specific I/O, string, and memory allocation schemes. It also provides a variety of utility functions, such as Maker Interchange Format (MIF) writing functions. Frame encourages you to use the FDE to develop all your clients. The FDE includes header files and libraries for each supported platform.

Structure Import/Export API

The Structure Import/Export API allows you to write clients that control the import of markup documents into FrameMaker, and control the export of FrameMaker documents to markup (XML or SGML).

Other FrameMaker features for developers

FrameMaker provides other advanced features that are useful for developers. You do not need the FDK to use these features.

MIF

Maker Interchange Format (MIF) is an easily parsed ASCII format that describes a document’s text, graphics, formatting, and layout. FrameMaker can save a document or a book to a MIF file, and convert a MIF file back to a document or book, without losing any information.

You can write applications or scripts that convert a MIF file to the format of another desktop publishing package, or convert other formats to MIF.

Here are some examples of things you can use MIF for:

- Sharing files with earlier releases of FrameMaker products
- Converting database files into Frame documents
- Filtering word processor documents into Frame documents

You can find documentation for MIF in the online manuals folder for your FrameMaker installation.

fmbatch

The fmbatch utility is shipped with UNIX versions of FrameMaker. It allows you to open, print, reformat, and save FrameMaker product files without displaying them. You can use fmbatch directly at a UNIX graphics workstation or in a remote shell from a nongraphics terminal. The fmbatch utility provides interactive and batch modes. When you use it in batch
Using Frame Developer Tools

mode, you can execute operations on multiple files automatically. For example, you can use `fmbatch` for:

- Printing books or documents from a remote location
- Printing multiple books or documents automatically
- Updating multiple books or documents automatically
- Converting multiple books or documents to MIF

For more information on the `fmbatch` utility, see the FrameMaker product user documentation for UNIX platforms.

Choosing the right Frame tools

There are often several tools or combinations of tools that you can use to solve a given problem. In particular, you can use the API to perform many of the tasks that MIF and `fmbatch` perform. The tool or combination of tools you should use depends on your needs. Generally, MIF and `fmbatch` are more useful for one-time solutions to small problems, whereas the API is more useful for full-scale applications or applications where interaction with the user is required.

The following table summarizes the advantages and limitations of each Frame tool.

<table>
<thead>
<tr>
<th>Frame tool or feature</th>
<th>Advantages</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame API</td>
<td>Fast, interactive, and portable; easy to provide a user interface for your applications</td>
<td>Must be compiled</td>
</tr>
<tr>
<td>MIF</td>
<td>Can be used by text-processing utilities or UNIX shell script. It can also be used to provide “backwards” compatibility allowing files to be opened in earlier releases of the product. Third party MIF creators do not need to write complete MIF. FrameMaker will always write out complete MIF.</td>
<td>Files must be saved as MIF; not interactive</td>
</tr>
<tr>
<td><code>fmbatch</code></td>
<td>Doesn’t require C programming; can be used remotely from a nongraphics terminal</td>
<td>Only provides simple functions; only available on UNIX platforms</td>
</tr>
</tbody>
</table>

FDK documentation

FDK documentation assumes that you have a thorough knowledge of FrameMaker. For background information on FrameMaker, see your user documentation.

FDK documentation includes the following manuals.
FDK Platform Guide

The *FDK Platform Guide* is printed in several versions, each describing how to install the FDK on a specific platform. It provides the pathnames of sample programs and describes platform-specific differences in using the FDK.

FDK Programmer’s Guide

The *FDK Programmer’s Guide* describes how to use the FDK to create clients for FrameMaker. To get the most from this manual, you should be familiar with the C programming language and event-driven programming.

The *FDK Programmer’s Guide* is divided into three parts:

- Part I, *Frame Application Program Interface (API)*, provides instructions for creating API clients.
- Part II, *Frame Product Architecture*, provides a conceptual overview of how the API represents sessions, books, and documents.
- Part III, *Frame Development Environment (FDE)*, provides instructions for making filters and API clients platform-independent.

FDK Programmer’s Reference

The *FDK Programmer’s Guide*, which you are reading now, provides FDK reference information, such as error codes and data structure, function, and property descriptions.

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.

Naming conventions

To help you identify the structures, constants, and functions defined by the FDK, this manual and the FDK adhere to the following naming conventions:

<table>
<thead>
<tr>
<th>Type</th>
<th>Naming convention</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>API error codes</td>
<td>Begin with FE_</td>
<td>FE_NotPgf</td>
</tr>
<tr>
<td>API functions</td>
<td>Begin with F_Api</td>
<td>F_ApiGetInt()</td>
</tr>
<tr>
<td>API scriptable function</td>
<td>Begin with FS_</td>
<td>FS_NewDoc</td>
</tr>
<tr>
<td>property names</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FDE functions</td>
<td>Begin with F_</td>
<td>F_StrNew()</td>
</tr>
</tbody>
</table>
This manual uses the term *API graphic object* to refer to objects (such as `FO_Polygon` and `FO_TextFrame` objects) that the API uses to represent the graphic objects (such as polygons and text frames) that appear on a page.

<table>
<thead>
<tr>
<th>Type</th>
<th>Naming convention</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flags used by API functions</td>
<td>Begin with <code>FF_</code></td>
<td><code>FF_UFF_VAR</code></td>
</tr>
<tr>
<td></td>
<td>and all letters</td>
<td></td>
</tr>
<tr>
<td></td>
<td>are uppercase</td>
<td></td>
</tr>
<tr>
<td>Initialization constants</td>
<td>Begin with <code>FA_Init</code></td>
<td><code>FA_Init_First</code></td>
</tr>
<tr>
<td>Notification constants</td>
<td>Begin with <code>FA_Note</code></td>
<td><code>FA_Note_PreFileType</code></td>
</tr>
<tr>
<td>Object property names</td>
<td>Begin with <code>FP_</code></td>
<td><code>FP_Fill</code></td>
</tr>
<tr>
<td>Object types</td>
<td>Begin with <code>FO_</code></td>
<td><code>FO_Doc</code></td>
</tr>
<tr>
<td>Property value constants</td>
<td>Begin with <code>FV_</code></td>
<td><code>FV_Doc_Type_MIF</code></td>
</tr>
<tr>
<td>Typedefs</td>
<td>End with <code>T</code></td>
<td><code>MetricT</code></td>
</tr>
</tbody>
</table>

This manual uses the term *API graphic object* to refer to objects (such as `FO_Polygon` and `FO_TextFrame` objects) that the API uses to represent the graphic objects (such as polygons and text frames) that appear on a page.

**Style conventions**

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

FDK manuals may use the term *FrameMaker product* to refer to the FrameMaker software, as opposed to the software you write to work with the FrameMaker product.

**Structured program interface**

FrameMaker 7.0 and later ships with two program interfaces—Structured FrameMaker and FrameMaker. The structured program interface presents menus, icons, and commands for working with structured documents. The FDK includes some functions that only work on structured documents. For example, setting an element range makes no sense in a document that doesn’t contain any structure elements. Further, you can specify that an FDK client requires the Structured FrameMaker program interface. For example, assume you specify Structured FrameMaker when you register your client. If a user has your client installed, but is running the FrameMaker program interface (not structured), then his installation of FrameMaker will not initialize your client when it starts up. (In earlier versions you could specify which FrameMaker products would initialize your client; FrameMaker, FrameMaker+SGML, or FrameViewer.)

The *FDK Programmer’s Reference* indicates those FDK functions that apply only to structured FrameMaker documents, as follows:
Structured **F_ApiGetAttributeDefs()**

In this example the word *Structured* appears to the left of the function name, indicating that this function applies only to the content of a structured document. If you register a client to work with the FrameMaker program interface, you should be sure that your client doesn’t use any functions identified as *Structured*, otherwise your client may exhibit unpredictable behavior.

**Pathnames**

Code examples in this manual use UNIX pathnames. For example:

```c
docId = F_ApiSimpleOpen((StringT)"/tmp/mydoc.fm", False);
```

If you are using the FDK on the Macintosh® or on Windows®, change the code to use the filepath naming conventions for those platforms. For example, on Macintosh, change the code above to code like the following:

```c
docId = F_ApiSimpleOpen((StringT)"MyHardDrive:tmp:mydoc.fm", False);
```

On Windows, change it to code like the following:

```c
docId = F_ApiSimpleOpen((StringT)"c:\tmp\mydoc.fm", False);
```

The FDK provides functions for converting platform-specific filepaths. For more information, see “Making I/O portable with channels” on page 377 of the *FDK Programmer’s Guide*.

**Typographic conventions**

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

- What you type is shown in
  `text like this.`

- Function names, property names, structure names, returned values, constants, 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.`

  For example, this represents the name of your working directory:
  `/Mydir`

- Omitted code in source code examples is indicated with ellipses.

  For example, the ellipsis in the following code indicates that some of the code necessary to create a complete program is omitted:

  ```c
  . . .
  F_ApiAlert((StringT)"Hello world.", FF_ALERT_CONTINUE_NOTE);
  ```
Part I

Frame Application Program Interface (API)
1 Introduction to the Frame API

This chapter provides an overview of how the API works and how to create an FDK client. It also provides a simple example—a client that you can create and run right away.

The API enables you to create a client that takes control of a FrameMaker product session. With the API, a client can do almost anything an interactive user can do. It can create, save, and print documents; add and delete text and graphics; and perform many other formatting and document-management tasks. It can also interact with the user by responding to user actions, displaying dialog boxes, and creating menus.

How the API works

The API represents everything in a FrameMaker product session as an object. Each object has a type, a constant that indicates the type of thing it represents. For example, an object’s type can be FO_Doc (if it represents a document), FO_Rectangle (if it represents a graphic rectangle), or FO_Pgf (if it represents a paragraph).

FrameMaker products assign an identifier (ID) to each object in a session. You use this ID to identify an object when you call API functions.

An object’s characteristics are called properties. Each type of object has a particular set of properties or a property list. For example, an FO_Rectangle object’s property list includes properties named FP_Width and FP_Height, which represent its height and width. An FO_Pgf object’s property list includes properties named FP_LeftIndent and FP_Leading, which represent its left indent and its leading. Each property has a predetermined data type, such as IntT (integer, Boolean, or ordinal), StringT (string), or F_ObjHandleT (object ID).

Each of an individual object’s properties has a value. This value describes the property for that particular object. For example, suppose a document contains a smoothed rectangle that is 20 points wide and 10 points high. The Frame API represents the rectangle as an FO_Rectangle object with the following properties and values.

<table>
<thead>
<tr>
<th>Property</th>
<th>Data Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_Width</td>
<td>MetricT</td>
<td>20 * 65536</td>
</tr>
<tr>
<td>FP_Height</td>
<td>MetricT</td>
<td>10 * 65536</td>
</tr>
<tr>
<td>FP_RectangleIsSmoothed</td>
<td>IntT</td>
<td>True</td>
</tr>
</tbody>
</table>

1. Frame API objects should not be confused with the graphic objects that you create with the Tools palette, object elements in structured documents, or the objects of object-oriented programming languages.
Introduction to the Frame API

**TABLE 1.1**

<table>
<thead>
<tr>
<th>Property</th>
<th>Data Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_FrameParent</td>
<td>F_ObjHandleT</td>
<td>ID of the frame containing the rectangle</td>
</tr>
</tbody>
</table>

a. MetricT values are 32-bit integers that represent measurements in points. The 16 most significant bits represent the digits before the decimal. The 16 least significant bits represent the digits after the decimal. A point is 65536 (1 << 16) in MetricT units. For more information on MetricT, see “MetricT values” on page 675 of the *FDK Programmer’s Reference*.

FO_Rectangle objects actually have many more properties than are shown in the table above. For a complete list of Frame API objects and properties, see Chapter 3, “Object Reference” in the *FDK Programmer’s Reference*.

How clients can change FrameMaker documents, books, and sessions

A client can change FrameMaker documents, books, and sessions by:

- Creating and destroying objects
  
  The API provides functions to create and destroy objects.

- Changing object properties
  
  The API provides functions to get and set object properties.

- Programmatically executing FrameMaker product commands, such as Open, Print, Save, and Clear All Change Bars

How clients communicate with the user

A client can communicate with the user by:

- Creating menus and menu items

- Displaying dialog boxes
  
  The API allows a client to respond to user actions by:

- Notifying the client when the user initiates certain events, such as Open, Save, or Quit

- Passing a message to the client when the user clicks a hypertext marker that contains a message *apiclient hypertext command*

How clients work with FrameMaker

Clients use different mechanisms to work with FrameMaker on the different platforms the FDK supports. On the Macintosh, clients are dynamically loaded modules. On Windows, they are dynamic link libraries (DLLs), or they can be executable programs that use COM to communicate with a FrameMaker session. On UNIX platforms, they can be either dynamically...
linked code or separate processes that use the RPC mechanism to communicate with the FrameMaker product. A client does not need to be aware of the low-level details of integrating with FrameMaker, because the API provides high-level functions that are the same on all platforms.

When the user starts FrameMaker it sends an initialization call to each registered client. Clients can take control immediately, or they can request the FrameMaker product to notify them of specific events and wait for those events to occur.

**Special types of clients**

In addition to conventional clients that take control of a FrameMaker product session in response to user actions, the API allows you to create three special types of clients: document reports, filters, and take-control clients.

**Document reports**

A *document report* is a client that provides detailed information about a document. The user can start a document report by choosing Utilities>Document Reports from the File menu, and then choosing the report from the Document Reports dialog box. The FDK includes a sample document report, named `wordcnt`, which counts the number of words in a document. For more information on sample programs provided with the FDK, see the *FDK Platform Guide* for the platform you’re using.

**Filters**

A *filter* is a client that converts FrameMaker product files to or from other file formats.

An *import filter* is a filter that the FrameMaker product calls when the user attempts to open a non-Frame file and chooses a filter in the Unknown File Type dialog box. The import filter reads the file and converts it to a FrameMaker product document or book. The FDK includes a sample import filter, named `mmlimport`, that converts MML files to FrameMaker product documents.

An *export filter* is a filter that the FrameMaker product calls when the user attempts to save a FrameMaker product document or book in a particular format by choosing the format in the Save dialog box or by specifying a filename with a particular extension. The export filter writes information in the document or book to a file with a different format.

A *file-to-file* filter is a filter that the FrameMaker product can call to both import or export files of different formats. A single file-to-file filter client can actually consist of more than one filter. For example, the same client could filter CGM to FrameVector and FrameVector to CGM. The way you register the client’s different filters determines which filter to invoke for import and export. Another advantage of these filters is they can filter from an external file to an external file. For example, you could filter from CGM to TIFF, and the TIFF file can remain an external file that is imported into the document by reference.
Take-control clients

A take-control client is a client that takes control of a FrameMaker product session immediately after the FrameMaker product starts. The UNIX implementation of the FDK allows users to specify a take-control client in the FrameMaker product command line. Take-control clients are useful for conducting batch operations in which little or no user interaction is needed.

Portability

The API’s design makes it easy to create portable clients. In most cases, you only need to recompile your client to run it on a different platform. To ensure that your client is completely portable, use the FDE with the API. You should replace platform-specific I/O, string, and memory functions in your client with the alternatives the FDE provides. For more information on the FDE, see Part III, Frame Development Environment (FDE).

The FDE and the API provide alternatives to the C language’s fundamental data types. For example, the FDE and the API substitute IntT for a 32-bit int and UCharT for unsigned char. The API uses other types for specific purposes. For example, it uses F_ObjHandleT for object IDs. For a list of API data types, see Chapter 4, “Data Types and Structures Reference,” in the FDK Programmer’s Reference.

Due to platform limitations Asynchronous clients (clients that run outside of the FrameMaker process space) are not portable on all platforms. For specific information on Asynchronous clients see the FDK Platform guide for the platform in question.

Running clients with different FrameMaker product interfaces

FrameMaker ships with two product interfaces, Structured FrameMaker and FrameMaker. A client can only use functionality that is available in the product interface that is active for the currently running FrameMaker process. For example, if a client is running on the unstructured FrameMaker product interface, it can’t create or manipulate structural elements (FO_Element objects). On the other hand, all functions in the FDK are available to a client running on the Structured FrameMaker product interface.

To determine if a function is available in a particular product interface, see Chapter 2, “FDK Function Reference,” in the FDK Programmer’s Reference. For an example of how to programmatically determine which product interface is running, see “F_ApiGetString()” on page 158 of the FDK Programmer’s Reference.

Creating and running a client

To create and run a client, follow these general steps:
1. Write an initialization function.

Most clients need to define an `F_ApiInitialize()` callback function. When the FrameMaker product starts, it calls your client’s `F_ApiInitialize()` function. Normally, you will want to include code in `F_ApiInitialize()` to set up your client’s menus and request notification for particular events.

For more information on creating an `F_ApiInitialize()` callback, see Chapter 2, “API Client Initialization”.

2. Set up the client’s user interface.

Your client probably needs to interact with the user. To respond to user actions, you can define the following callback functions in your client:

- `F_ApiNotify()` to respond to the user initiating FrameMaker product operations, such as Open and Save
- `F_ApiCommand()` to respond to the user choosing menu items created by your client
- `F_ApiMessage()` to respond to the user clicking hypertext markers that contain the message `apiclient` command
- `F_ApiDialogEvent()` to respond to the user manipulating items in a dialog box created by your client

You can also display notices and prompt the user for input by using API dialog box functions, such as `F_ApiAlert()` and `F_ApiPromptString()`.

For more information on setting up your client’s user interface, see Chapter 3, “Creating Your Client’s User Interface”.

3. Add code to programmatically execute FrameMaker product commands.

Your client probably needs to execute some FrameMaker product commands, such as Open, Print, or Close. To execute these commands programmatically, use API functions, such as `F_ApiSimpleOpen()`, `F_ApiSilentPrintDoc()`, or `F_ApiClose()`.

For more information on using API functions to execute FrameMaker product commands, see Chapter 4, “Executing Commands with API Functions”.

4. Add code to get and set object properties.

To get or set an object property, use the `F_ApiGetPropertyType()` or `F_ApiSetPropertyType()` function that corresponds to the type of property you want to get or set. For example, to get or set an `IntT` property, use `F_ApiGetInt()` or `F_ApiSetInt()`. To get or set a `StringT` property, use `F_ApiGetString()` or `F_ApiSetString()`.

For more information on changing object properties, see Chapter 5, “Getting and Setting Properties”.

5. Add code to create objects.
To create objects, use the `F_ApiNewObject Type()` function that corresponds to the kind of object that you want to create. For example, to create a new anchored frame, use `F_ApiNewAnchoredObject()`.

For more information on creating objects, see Chapter 8, “Creating and Deleting API Objects”.

6. Compile your client.

The API function declarations are contained in the `fapi.h` header. Be sure to include this header in your client code. Include C library header files before the `fapi.h` header.

The FDK comes with sample makefiles or project files for each supported platform. To compile your client, use your platform’s make or build utility. For more information on using FDK makefiles or project files on a specific platform, see the FDK Platform Guide for that platform.

7. Register your client with the FrameMaker product.

The FrameMaker product needs to know about your client to initialize it. To let the FrameMaker product know about your client, you must make some changes to the environment under which the client runs.

- On UNIX platforms, when you start FrameMaker it looks for a text-only file named `fmInit/UILanguage/APIArea/apiclients`, where `UILanguage` is the user interface language (such as `usenglish` for US English). It looks in the directory from which it was started, in your home directory, and in `$FHOME`. To register your client, create an `fmInit/UILanguage/APIArea/apiclients` file in your home directory or the directory from which you intend to start the FrameMaker product. Add the following lines to the file:

  ```xml
  <APIClient
    <Name client_name>
    <Directory clientdir>
    <CommandLine $FAPIDIR/executable_name>
  >
  ```

  The path `clientdir` specifies the full path of the directory containing your client’s executable file and other resources, `client_name` is the name that FrameMaker and other clients use to reference your client, and `executable_name` is the name of your client’s executable file. The FDK sets the environment variable `$FAPIDIR` to `clientdir`.

- On Macintosh computers, copy or move your client to the FrameMaker product’s Modules folder.

- On Windows platforms, add the following lines to the `[APIClients]` section of the `maker.ini` file:

  ```ini
  ClientName=ClientType, description, path, mode
  ```

  `ClientName` is the name that the FrameMaker product and other clients use to reference your client. `ClientType` specifies your client type: for example, `Standard`, `DocReport`, or `TextImport`. `description` is a string describing your client. `path` is the pathname of your
client’s DLL. `mode` determines what product interfaces your client supports—can be one of all, maker, or structured.

You can also register a Windows client by setting values in the DLL’s VERSIONINFO resource, then copying or moving the DLL in the FrameMaker product’s Plugins folder.

For information on registering your client on a specific platform, see the *FDK Platform Guide* for that platform.

---

**A simple example**

The following client adds a menu with three items to the FrameMaker product menu bar when the FrameMaker product starts. The first menu item closes the active document; the second item sets the fill pattern of a selected graphic object; the third item adds a body page to the active document. Following the code is a line-by-line description of how it works.

1. `#include "fapi.h"
2. `#define CloseDoc 1`
3. `#define SetFill 2`
4. `#define AddPage 3`
5. `VoidT F_ApiInitialize(initialization)`
6. `IntT initialization; /* Code for initialization type */`
7. `{`
8. `F_ObjHandleT menuBarId, menuId;`
9. `/* Get ID of the FrameMaker product menu bar. */`
10. `menuBarId = F_ApiGetNamedObject(FV_SessionId, FO_Menu,`  
11. `"!MakerMainMenu");`
12. `/* Add menu named "API" to the FrameMaker product menu bar. */`
13. `menuId = F_ApiDefineAndAddMenu(menuBarId, "APIMenu", "API");`
14. `/* Add items to API menu. */`
15. `F_ApiDefineAndAddCommand(CloseDoc, menuId,"CloseDocCmd",`  
16. `"Close","\\CD");`
17. `F_ApiDefineAndAddCommand(SetFill, menuId,"SetFillCmd",`  
18. `"Set Fill","\\SF");`
19. `F_ApiDefineAndAddCommand(AddPage, menuId,"AddPageCmd",`  
20. `"Add Page","\\AP");`
21. `}
22. `VoidT F_ApiCommand(command)`
23. `IntT command;`
24. `{`
25. `F_ObjHandleT pgId, objId, docId;`
26. `/* Get the ID of the active document. */`
27. `docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);`
28. `if(!docId) return; /* Return if there is no active document. */`
switch (command) {
  case CloseDoc: /* Close document even if it’s changed. */
    F_ApiClose(docId, FF_CLOSE_MODIFIED);
    break;
  case SetFill: /* Set fill pattern to black. */
    /* Get ID of selected object. */
    objId = F_ApiGetId(FV_SessionId, docId, FP_FirstSelectedGraphicInDoc);
    /* If an object is selected, set its fill. */
    if (!objId) F_ApiAlert("Select an object first.", FF_ALERT_CONTINUE_WARN);
    else F_ApiSetInt(docId, objId, FP_Fill, FV_FILL_BLACK);
    break;
  case AddPage: /* Add a new page. */
    pgId = F_ApiNewSeriesObject(docId, FO_BodyPage, 0);
    break;
}

Lines 1.–5.
These lines include header files and define the constants for the client’s menu items. All clients must include the fapi.h header. To ensure your client is portable, include fapi.h before any other header files.

Lines 6.–25.
These lines define the F_ApiInitialize() callback function. When the user starts the FrameMaker product, it calls F_ApiInitialize().

The call to F_ApiGetNamedObject() gets the ID of the FrameMaker product menu bar (MakerMainMenu). The call to F_ApiDefineAndAddMenu() creates a menu named API on the menu bar. The calls to F_ApiDefineAndAddCommand() add menu items to the API menu and define keyboard shortcuts for the items.

Lines 26.–56.
These lines define the F_ApiCommand() callback function. When the user chooses a menu item, the FrameMaker product calls this callback with command set to the menu item number (in this example, CloseDoc, SetFill, or AddPage).

Lines 31.–34.
These lines get the ID of the document to change. To use most API functions, you need to specify an ID. This example gets the ID of the document that has input focus, that is, the active document.

In each FrameMaker product session there is only one active document at a time. The session object (FO_Session) has a property named FP_ActiveDoc that specifies its ID. To retrieve
the active document’s ID from the FO_Session object, you use F_ApiGetId() to query the FO_Session object’s FP_ActiveDoc property. The syntax for F_ApiGetId() is:

```
F_ObjHandleT F_ApiGetId(parentId, /* Object’s parent */
    objId, /* Object whose property you want to query */
    property); /* Constant specifying property to query */
```

The parentId parameter specifies the ID of the object’s parent—the session, book, or document that contains the object. No other object contains the FO_Session object, so parentId is set to 0. The ID of the FO_Session object (there can only be one) is always FV_SessionId, so objId is set to FV_SessionId.

**Lines 37–40.**

These lines close the active document when the user chooses Close Doc from the API menu. The FF_CLOSE_MODIFIED flag instructs the API to close the document without warning the user, even if the document has unsaved changes.

**Lines 41–51.**

These lines set the fill pattern of a selected object to black when the user chooses Set Fill from the API menu. To set the selected object’s fill pattern, the client needs the object’s ID. To get the ID, the client uses F_ApiGetId() to query the document property, FP_FirstSelectedGraphicInDoc. If no object is selected, F_ApiGetId() returns 0.

The F_ApiAlert() call displays an alert that tells the user to select an object. The constant, FF_ALERT_CONTINUE_WARN, specifies the type of alert—an alert with a Continue button.

To set the object’s fill pattern, the client must set its FP_Fill property. FP_Fill is an IntT property, so the client must use F_ApiSetInt() to set it. The syntax for F_ApiSetInt() is:

```
VoidT F_ApiSetInt(parentId, /* Object’s parent */
    objId, /* Object whose property you want to set */
    property, /* Constant specifying property to set */
    value); /* Value to which to set the property */
```

FP_Fill can have any value between 0 and 15. The API-defined constant, FV_FILL_BLACK, specifies 0 (black).

**Lines 52–54.**

These lines add a body page to the document when the user chooses Add a Page from the API menu. A body page object is a series object. To create a series object, you use F_ApiNewSeriesObject(). The syntax for F_ApiNewSeriesObject() is:

```
F_ObjHandleT F_ApiNewSeriesObject(parentId, /* Object’s Parent */
    objectType, /* Constant specifying new object type */
    prevObjectId); /* Object for new object to follow */
```

The parentId parameter specifies the ID of the object that is to contain the new object. The new page should appear in the active document, so parentId is set to docId. The API uses FO_BodyPage objects to represent body pages, so objectType is set to FO_BodyPage. Specifying 0 for prevObjectId puts the new page at the beginning of the document. For more information on creating different types of objects, see “Creating objects” on page 143.
Compiling and running the example client

The source code for the example client and a makefile or project file are provided in the samples/myapi directory of your FDK installation. For the location of the files, see the FDK Platform Guide for your platform. To compile the client, use your platform’s make or build utility.

To run the example client, you must first register it as required by your platform’s implementation of the API:

- If you are working on a UNIX platform with US English as the user interface language, and the pathname of your compiled client is /usr/fdk/samples/ch1/fa.myapi, you can register it by creating a text-only file named
  ~/fminit/usenglish/APIArea/apiclients with the following lines in it:

    <APIclient
     <Name myapi>
     <Directory /usr/fdk/samples/ch1>
     <CommandLine $FAPIDIR/fa.myapi>
    >

    When you start the FrameMaker product, it reads this file and starts the fa.myapi client.

- If you are working on a Macintosh computer, copy or move the example client to the FrameMaker product’s Modules folder.

- If you are working on a Windows platform and you have compiled your client into a DLL named myapi.dll and copied or moved it to the FrameMaker fminit directory, add the following line to the maker.ini file:

```
myapi = Standard, Ch. 1 Sample, fminit\myapi.dll, all
```

For more information on how to register clients, see the FDK Platform Guide.

After you have registered the example client, start the FrameMaker product and open a document. The API menu should appear to the right of the FrameMaker product menus.
This chapter describes how to start interaction between your client and FrameMaker.

Responding to the FrameMaker product’s initialization call

When the FrameMaker product starts, it attempts to start all the clients registered with it, except document reports and filters. The FrameMaker product attempts to start each client by calling its F_ApiInitialize() callback function.

Your client should define F_ApiInitialize() as follows:

```c
VoidT F_ApiInitialize(initialization)
IntT initialization;
{
    /* Your client code goes here */
}
```

**TABLE 2.1**

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialization</td>
<td>A flag that indicates the type of initialization (see “Initialization types” on page 14)</td>
</tr>
</tbody>
</table>

Usually, you want your client to do something immediately after the user starts the FrameMaker product. For example, you may want to add menus to the menu bar or request notification for certain events. To do this, you call API functions from the F_ApiInitialize() function. For information on creating menus and requesting notification, see Chapter 3, “Creating Your Client’s User Interface.”

Suppose you want your client to display a dialog box after the FrameMaker product is started. To do this, you could use the following F_ApiInitialize() function:

```c
... VoidT F_ApiInitialize(initialization)
IntT initialization;
{
    F_ApiAlert("Client has started.", FF_ALERT_CONTINUE_NOTE);
}
...```

---

1. For information on registering your client with the FrameMaker product, see the *FDK Platform Guide* for your platform.
Initialization types

The following table summarizes the different types of initializations and the initialization constants FrameMaker can pass to your client’s `F_ApiInitialize()` callback.

**TABLE 2.2**

<table>
<thead>
<tr>
<th>Type of initialization</th>
<th>When F_ApiInitialize is called</th>
<th>Initialization constant</th>
<th>Clients that receive initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>FrameMaker starts with no special options</td>
<td>After starting</td>
<td><code>FA_Init_First</code></td>
<td>All except document reports and filters</td>
</tr>
<tr>
<td>FrameMaker starts with take-control client</td>
<td>After starting</td>
<td><code>FA_Init_First</code></td>
<td>All except document reports and filters</td>
</tr>
<tr>
<td></td>
<td>After all clients have finished processing the <code>FA_Init_First</code> initialization</td>
<td><code>FA_Init_TakeControl</code></td>
<td>All clients set up as take-control clients</td>
</tr>
<tr>
<td>Document report chosen from Document Reports dialog box</td>
<td>After report is chosen</td>
<td><code>FA_Init_DocReport</code></td>
<td>The chosen document report</td>
</tr>
<tr>
<td>Notification, menu choice, or hypertext command for a client that has bailed out</td>
<td>When the menu item is chosen, the hypertext command is clicked, or the notification should be issued</td>
<td><code>FA_Init_Subsequent</code></td>
<td>Clients that have bailed out and are waiting for an event, menu choice, or hypertext command to occur</td>
</tr>
</tbody>
</table>

**First initialization**

When the user starts the FrameMaker product, the product calls the `F_ApiInitialize()` function of each registered client (unless it’s a document report or filter) with initialization set to `FA_Init_First`.

**Take-control initialization**

The FDK allows you to set up clients to receive a special initialization called a `take-control` or `FA_Init_TakeControl` initialization. The FrameMaker product issues the `FA_Init_TakeControl` initialization after it has issued the `FA_Init_First` initialization and all clients have returned control. This initialization is useful if you want your client to conduct some batch processing after other clients have initialized, but before the interactive user has control.
On UNIX platforms, the FrameMaker product issues the FA_Init_TakeControl initialization to only one client. The user specifies this client by starting the FrameMaker product with the -apiclient client_name option (where client_name is the registered name of the client).

On Windows and Macintosh platforms, the FrameMaker product can issue the FA_Init_TakeControl initialization to several clients. To set up a client to receive FA_Init_TakeControl initializations on Windows, set the client’s type to TakeControl in the FrameMaker product .ini file. To set up a client to receive FA_Init_TakeControl initializations on the Macintosh, set the flags field of the client’s fdat resource to include ApiClientFlagCanTakeControl. For more information on setting up clients to receive take-control initializations, see the FDK Platform Guide for your platform.

Document report initialization

When the FrameMaker product is started, it does not attempt to initialize API clients that are registered as document reports. It initializes a document report only when the user chooses the document report from the Document Reports dialog box. When this occurs, the FrameMaker product calls the document report’s F_ApiInitialize() callback with initialization set to FA_Init_DocReport.

To appear in the Document Reports dialog box, a document report must be registered with the FrameMaker product as a document report. For information on registering document reports, see the FDK Platform Guide for your platform.

Filter initialization

If your client is registered as a filter, you should not define an F_ApiInitialize() function for it. When the user opens or saves a file and selects your filter, the FrameMaker product notifies your client by calling your client’s F_ApiNotify() callback. For more information on F_ApiNotify() and notification, see “Responding to user-initiated events or FrameMaker product operations” on page 34.

To receive notification, your filter must be registered as a filter. For information on registering filters, see the FDK Platform Guide for your platform.

Initialization after a client has bailed out

If your API client is waiting for an event and not performing any other processing, it can call F_ApiBailOut(). This exits your client’s process and frees all the system resources that it uses. If an event that your client is waiting for occurs, the FrameMaker product restarts your client by calling its F_ApiInitialize() function with initialization set to FA_Init_Subsequent.

A document report should always bail out after it completes processing, because the API initializes it each time the user chooses it from the Document Reports dialog box. A filter should always bail out after it filters a file, because the API initializes it each time a filterable file is opened, imported, or saved.
API Client Initialization

For more information on F_ApiBailOut(), see “F_ApiBailOut()” on page 48 of the FDK Programmer’s Reference.

Disabling the API

The user can disable all API clients before starting the FrameMaker product. On UNIX platforms, the user can disable all clients by specifying -noapi in a resource file or when starting the FrameMaker product from the command line. On Windows platforms, the user can disable all clients by changing the API=On setting in the maker.ini file to API=Off. On the Macintosh, the user can disable clients by dragging them from the FrameMaker product Modules folder. For more information on disabling individual clients, see the FDK Platform Guide for your platform.

IMPORTANT: Many FrameMaker features are implemented via API clients. If you disable the API then you also disable these features. Such features include XML and SGML import and export, Save As HTML, and Word Count.
This chapter describes how to use the Frame API to create a user interface for your FDK client.

Your client can interact with the user in the following ways:

- By displaying its own dialog boxes
- By implementing its own menus, menu items, and keyboard shortcuts
- By responding to the `message apiclient hypertext command`
- By responding to user-initiated events or FrameMaker product operations
- By implementing quick-key commands

The following sections discuss these ways of interacting with the user in greater detail.

### Using API dialog boxes to prompt the user for input

The Frame API provides a variety of premade dialog boxes. All of these dialog boxes are *modal*—the user must dismiss them before continuing. The following sections discuss how to use these dialog boxes.

The API also allows you to create and use custom modal and modeless dialog boxes. For more information, see Chapter 10, “Creating Custom Dialog Boxes for Your Client” and Chapter 11, “Handling Custom Dialog Box Events”

### Using alert boxes

To display a dialog box with a short message, use `F_ApiAlert()`.

The syntax for `F_ApiAlert()` is:

```c
IntT F_ApiAlert(StringT message,
    IntT type);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>The message that appears in the alert box</td>
</tr>
<tr>
<td>type</td>
<td>The type of alert box</td>
</tr>
</tbody>
</table>
Specify one of the following values for the type argument.

<table>
<thead>
<tr>
<th>type constant</th>
<th>Type of dialog box displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF_ALERT_OK_DEFAULT</td>
<td>Displays OK and Cancel buttons; OK is the default</td>
</tr>
<tr>
<td>FF_ALERT_CANCEL_DEFAULT</td>
<td>Displays OK and Cancel buttons; Cancel is the default</td>
</tr>
<tr>
<td>FF_ALERT_CONTINUE_NOTE</td>
<td>Displays Continue button</td>
</tr>
<tr>
<td>FF_ALERT_CONTINUE_WARN</td>
<td>Displays Continue button with a warning indication</td>
</tr>
<tr>
<td>FF_ALERT_YES_DEFAULT</td>
<td>Displays Yes and No buttons; Yes is the default</td>
</tr>
<tr>
<td>FF_ALERT_NO_DEFAULT</td>
<td>Displays Yes and No buttons; No is the default</td>
</tr>
</tbody>
</table>

F_ApiAlert() returns 0 if the user clicks OK, Continue, or Yes; otherwise, it returns a nonzero value.

**Example**

The following code displays the alert box shown in Figure 3.1:

```c
... 
IntT err;

err = F_ApiAlert((StringT)"This alert is an OK_DEFAULT.",
                   FF_ALERT_OK_DEFAULT);
... 
```

**Figure 3.1** FF_ALERT_OK_DEFAULT alert box

**Using string, integer, and metric input dialog boxes**

To prompt the user for a single string, integer, or metric value, use F_ApiPromptString(), F_ApiPromptInt(), or F_ApiPromptMetric(). These functions all allow you to provide a default value for the entry field.

Their syntax is:

```c
IntT F_ApiPromptString(StringT *stringp,
                        StringT message,
                        StringT stuffVal);
IntT F_ApiPromptInt(IntT *intp,
                    StringT message,
                    StringT stuffVal);
```
Creating Your Client’s User Interface

```c
IntT F_ApiPromptMetric(MetricT *metricp,
    StringT message,
    StringT stuffVal,
    MetricT defaultunit);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>stringp, intp, or metricp</td>
<td>A pointer to the user variable that gets the return value from the input field when the user clicks OK.</td>
</tr>
<tr>
<td>message</td>
<td>The message that appears in the dialog box.</td>
</tr>
<tr>
<td>stuffVal</td>
<td>The default value that appears in the input field when the dialog box is first displayed. It must be a string for integer and metric prompts, as well as string prompts.</td>
</tr>
<tr>
<td>defaultunit</td>
<td>The metric unit to use if the user doesn’t specify one. For example, to use inches as the default unit, specify 4718592. For more information on metric values, see “MetricT values” on page 675 of the FDK Programmer’s Reference.</td>
</tr>
</tbody>
</table>

These functions all return 0 if the user clicks OK. Otherwise, they return a nonzero error value. If the user clicks Cancel, the API does not assign a value to *stringp, *intp, or *metricp.

If the user types alphabetic text after a number in an F_ApiPromptInt() dialog box, the API ignores the text and just returns the number. For example, if the user types 10 cookies, the returned value is 10.

F_ApiPromptMetric() dialog boxes behave like metric dialog boxes in the user interface. If the user types a number followed by a string that represents a unit, the API converts the value into the equivalent number of metric units. For example, if the user types 5in or 5", the API returns 5*(4718592). If the user doesn’t specify a unit, the API uses the unit specified by defaultunit.

**IMPORTANT:** F_ApiPromptString() allocates memory for the string referenced by *stringp. Use the FDK function F_ApiDeallocateString() to free the string when you are done with it. For more information on F_ApiDeallocateString(), see “F_ApiDeallocateStructureType()” on page 74 of the FDK Programmer’s Reference.

**Examples**

The following code displays the dialog box shown in Figure 3.2:
...
#include "fmemory.h"
IntT err;
StringT sres;
err = F_ApiPromptString(&sres, (StringT)"String?",
                             (StringT)"Default text");
if (err) return;
/* Some code to use the string goes here. */
F_ApiDeallocateString(&sres);
...

**Figure 3.2** String prompt dialog box

The following code displays the dialog box shown in Figure 3.3:

...
#include "futils.h" /* Provides declaration for F_Sprintf(). */
IntT err, ires;
UCharT msg[256];
err = F_ApiPromptInt(&ires, (StringT)"Integer?", "1234");
if (err) F_Sprintf(msg,(StringT)"Cancelled, ires has no value");
else F_Sprintf(msg,(StringT)"The value of ires is %d.",ires);
F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);
...

**Figure 3.3** Integer prompt dialog box

The following code displays the dialog box shown in Figure 3.4:

...
#define IN (MetricT) 65536*72 /* Default unit (inches) */
IntT err;
MetricT mres;
err = F_ApiPromptMetric(&mres, (StringT)"Metric?",
                             "12.34in", IN);
...

Using file selection dialog boxes

To display dialog boxes similar to the FrameMaker product’s Open and Save dialog boxes, use \texttt{F_ApiChooseFile()}. \texttt{F_ApiChooseFile()} displays files and directories in a scrolling list and allows the user to choose a file or directory.

The syntax for \texttt{F_ApiChooseFile()} is:

\begin{verbatim}
IntT F_ApiChooseFile(StringT *choice, 
    StringT title, 
    StringT directory, 
    StringT stuffVal, 
    IntT mode, 
    StringT helpLink);
\end{verbatim}

\textbf{This argument} \hspace{2cm} \textbf{Means}

<table>
<thead>
<tr>
<th>choice</th>
<th>The selected pathname when the user clicks OK.</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>The message that appears in the dialog box.</td>
</tr>
<tr>
<td>directory</td>
<td>The default directory when the dialog box is first displayed. If you specify an empty string, the last directory used by your client is used. If your client hasn’t used any directories, the directory specified by the session property, \texttt{FP_OpenDir}, is used.</td>
</tr>
<tr>
<td>stuffVal</td>
<td>The default value that appears in the input field when the dialog box first appears. If the dialog box type specified by \texttt{mode} doesn’t have an input field, this string is ignored.</td>
</tr>
<tr>
<td>mode</td>
<td>A constant specifying the type of dialog box. For a list of dialog box types, see “\texttt{F_ApiChooseFile()}” on page 53 of the FDK Programmer’s Reference.</td>
</tr>
<tr>
<td>helpLink</td>
<td>Obsolete in versions 6.0 and later; pass an empty string. The name of a document containing help information for the dialog box and an optional hypertext link.</td>
</tr>
</tbody>
</table>

\textbf{IMPORTANT:} \texttt{F_ApiChooseFile()} allocates memory for the string referenced by \texttt{*choice}. Use \texttt{F_ApiDeallocateString()} to free the string when you are done with it.

\textbf{Example}

To create the dialog box shown in Figure 3.5, add the following code to your client:


```c
#include "futils.h"
#include "fmemory.h"

IntT err;
StringT sres;
UCharT msg[256];

err = F_ApiChooseFile(&sres, (StringT)"Choose a file",
    (StringT)="/tmp", (StringT)"",  //
    F_V_ChOOSE_SELECT, (StringT)"");  //
if (err)
    F_Sprintf(msg,(StringT)"Cancelled, sres is not defined.");
else
    F_Sprintf(msg,(StringT)"The value of sres is %s.",sres);

F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);
if (!err) F_ApiDeallocateString(&sres);
```

![File selection dialog box](image)

**Figure 3.5 File selection dialog box**

**Using scroll list dialog boxes**

To display a scroll list dialog box, use `F_ApiScrollBox()`. `F_ApiScrollBox()` displays an array of items that you provide and allows the user to choose one.

The syntax for `F_ApiScrollBox()` is:

```c
IntT F_ApiScrollBox(IntT *selected_item,
    StringT title,
    F_StringsT *stringslist,
    IntT default);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>selected_item</td>
<td>The index of the selected item when the user clicks OK (or double-clicks an item). The index of the first item is 0.</td>
</tr>
<tr>
<td>title</td>
<td>The title that appears on the dialog box.</td>
</tr>
</tbody>
</table>
Creating Your Client's User Interface

F_StringsT is defined as:

typedef struct {
    UIntT len; /* Number of strings */
    StringT *val; /* Array of strings */
} F_StringsT;

The F_ApiScrollBox() function returns a nonzero value if the user clicks Cancel or an error occurs, or 0 if the user clicks OK.

Example

To create the dialog box shown in Figure 3.6, add the following code to your client:

```c
#include "futils.h"

IntT err, choice, listLen = 3;
UCharT msg[256];
F_StringsT colors;

colors.val = (StringT *) F_Alloc(listLen*sizeof(StringT), NO_DSE);
if (colors.val) {
    colors.len = (UIntT)listLen;
    colors.val[0] = F_StrCopyString("red");
    colors.val[1] = F_StrCopyString("green");
    colors.val[2] = F_StrCopyString("blue");

    err = F_ApiScrollBox(&choice, "Choose a color.", &colors, 0);
    if (!err)
        F_Sprintf(msg, "The choice is %s.", colors.val[choice]);
    else
        F_Sprintf(msg, "Cancel was pressed");
    F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);
    F_ApiDeallocateStrings(&colors);
}
```

stringslist
The list of items that appears in the scroll list.
default
The index of the item that is selected when the dialog box first appears. For no default, specify -1.

This argument | Means
--- | ---
stringslist | The list of items that appears in the scroll list.
default | The index of the item that is selected when the dialog box first appears. For no default, specify -1.
Using commands, menu items, and menus in your client

The API allows you to use commands, menu items, and menus in your client’s user interface. A command is a part of FrameMaker product or FDK client functionality that a user can invoke by typing a shortcut. A menu item is an instance of a command that appears on a menu. There can be several menu items for each command.

A menu is a list of menu items or other menus. A menu bar is a list of menus that appears at the top of the screen on a Macintosh, at the top of a document or book window on UNIX platforms, and at the top of the FrameMaker product window on Windows platforms.

To use commands and menus in your client’s user interface, follow these general steps:

1. Get the IDs of the FrameMaker product menu bars and menus that you want to add your client’s menus and commands to.
2. Define your client’s commands and add them to menus.
3. Define your client’s menus and add them to FrameMaker product menus or menu bars.
4. Write an F_ApiCommand() callback to respond to the user invoking your client’s commands.

These steps are discussed in greater detail in the following sections.

Getting the IDs of FrameMaker product menus and menu bars

To get the IDs of commands, menus, or menu bars, use F_ApiGetNamedObject(). The syntax for F_ApiGetNamedObject() is:

F_ObjHandleT F_ApiGetNamedObject(F_ObjHandleT parentId, IntT objType, StringT objName);

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>parentId</td>
<td>The ID of the document, book, or session containing the object for which you want to get an ID. For commands and menus, it is always FV_SessionId.</td>
</tr>
</tbody>
</table>
The menu and command names you can specify for `objName` depend on the FrameMaker product and how the user has customized the menus. On UNIX platforms, the default command names are listed in `$FMHOME/fminit/UILanguage/configui/Commands` and the default menu names are listed in `$FMHOME/fminit/UILanguage/configui/ProductInterfaceMenus`, where `UILanguage` represents the user-interface language (such as `usenglish` for US English) and `ProductInterface` is the name of the product interface (either `maker` or `structured`). For information on where the menus are listed on other platforms, see the `FDK Platform Guide` for your platform.

The following table lists some FrameMaker product menus and the names you use to specify them:

<table>
<thead>
<tr>
<th>Menu title</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit</td>
<td>EditMenu</td>
</tr>
<tr>
<td>Element</td>
<td>ElementMenu</td>
</tr>
<tr>
<td>File</td>
<td>FileMenu</td>
</tr>
<tr>
<td>Format</td>
<td>FormatMenu</td>
</tr>
<tr>
<td>Graphics</td>
<td>GraphicsMenu</td>
</tr>
<tr>
<td>Special</td>
<td>SpecialMenu</td>
</tr>
<tr>
<td>Table</td>
<td>TableMenu</td>
</tr>
<tr>
<td>View</td>
<td>ViewMenu</td>
</tr>
<tr>
<td>Help</td>
<td>!HelpMenu</td>
</tr>
</tbody>
</table>

The following table lists the names of some FrameMaker product menu bars. Menu bar names starting with an exclamation point (!) can’t be removed by the user.

<table>
<thead>
<tr>
<th>FrameMaker product menu bar</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Menu bar for documents (complete menus)</td>
<td>!MakerMainMenu</td>
</tr>
<tr>
<td>Menu bar for documents (quick menus)</td>
<td>!QuickMakerMainMenu</td>
</tr>
<tr>
<td>Menu bar for books (complete menus)</td>
<td>!BookMainMenu</td>
</tr>
<tr>
<td>Menu bar for books (quick menus)</td>
<td>!QuickBookMainMenu</td>
</tr>
</tbody>
</table>
Creating Your Client’s User Interface

Example

The following code gets the ID of the Edit menu and the view-only menu bar:

```c
F_ObjHandleT editMenuId, viewOnlyMenuBarId;
editMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu, "EditMenu");
viewOnlyMenuBarId = F_ApiGetNamedObject(FV_SessionId, FO_Menu, "!ViewOnlyMainMenu");
```

Defining commands and adding them to menus

To define a command and add it to a menu, use `F_ApiDefineAndAddCommand()`. The syntax for `F_ApiDefineAndAddCommand()` is:

```c
F_ObjHandleT F_ApiDefineAndAddCommand(IntT cmd, F_ObjHandleT toMenuId, StringT name, StringT label, StringT shortcut);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cmd</code></td>
<td>The integer that the FrameMaker product passes to your client’s <code>F_ApiCommand()</code> function when the user chooses the menu item or types the keyboard shortcut for the command.</td>
</tr>
<tr>
<td><code>toMenuId</code></td>
<td>The ID of the menu to which to add the command.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>A unique name to identify the command.</td>
</tr>
<tr>
<td><code>label</code></td>
<td>The title of the command as it appears on the menu.</td>
</tr>
<tr>
<td><code>shortcut</code></td>
<td>The keyboard shortcut sequence. Many FrameMaker product commands use shortcuts beginning with Esc (!). To specify Esc when you create a command, use <code>\!</code> in the string you pass to <code>shortcut</code>.</td>
</tr>
</tbody>
</table>

`F_ApiDefineAndAddCommand()` returns the ID of the command it creates.

**IMPORTANT:** To ensure that your client is portable to the different platforms that FrameMaker supports, specify only shortcuts that begin with Esc. On UNIX, the Esc shortcut appears on the menu next to the menu item label. On the Macintosh or Windows, the shortcut does not appear. However, the user can still execute it. For example, if you specify `\!xy` for the keyboard shortcut, the user can execute the command from the keyboard by pressing Esc x y.
more information on specifying keyboard shortcuts, see the sections of your user documentation that discuss the FrameMaker product’s configurable user interface.

**Example**

The following code defines a command with the shortcut Esc N L and adds it to the Utilities menu:

```c
#define NUMBER_LINES 1
F_ObjHandleT utilsMenuId, cmdId;

utilsMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu, "UtilitiesMenu");
cmdId = F_ApiDefineAndAddCommand(NUMBER_LINES, utilsMenuId,
                                  "NumberLines","Number lines", "\\!NL");
```

**Defining and adding menus**

To define a menu and add it to a menu bar or another menu, use `F_ApiDefineAndAddMenu()`. The syntax for `F_ApiDefineAndAddMenu()` is:

```c
F_ObjHandleT F_ApiDefineAndAddMenu(F_ObjHandleT toMenuId, 
                                   StringT name, 
                                   StringT label);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>toMenuId</td>
<td>The ID of the menu or menu bar to which to add the new menu</td>
</tr>
<tr>
<td>name</td>
<td>A unique name that identifies the new menu</td>
</tr>
<tr>
<td>label</td>
<td>The title of the new menu as it appears on the menu or menu bar</td>
</tr>
</tbody>
</table>

`F_ApiDefineAndAddMenu()` returns the ID of the menu it creates.

If you specify a menu bar ID for `toMenuId`, the FrameMaker product implements the new menu as a pull-down menu. If you specify a pull-down or a pop-up menu ID for `toMenuId`, the FrameMaker product implements the new menu as a pull-right menu.

**IMPORTANT:** Your menu appears only on the menu bar you specify. For example, if you add a menu only to the !MakerMainMenu menu bar, the menu will not appear if the user switches to quick menus. For your menu to appear after the user has switched to quick menus, you must also add it to the !QuickMakerMainMenu menu bar.

**Adding commands to a menu that you have created**

To add a command to a menu that you have created, call `F_ApiDefineAndAddCommand()` with `toMenuId` set to the ID returned by the `F_ApiDefineAndAddMenu()` call that
created the menu. For example, the following code defines a menu and adds it to the FrameMaker document menu bar. Then it adds some commands to the menu.

```c
#define CHECK 1
#define PRINT 2
F_ObjHandleT menubarId, menuId, cmd1Id, cmd2Id;

/* Get the ID of the FrameMaker main menu bar. */
menubarId = F_ApiGetNamedObject(FV_SessionId, FO_Menu,
"!MakerMainMenu");

/* Define and add the menu to the main menu. */
menuId = F_ApiDefineAndAddMenu(menubarId, "GrammarMenu",
"Grammar");

/* Define some commands and add them to the menu. */
cmd1Id = F_ApiDefineAndAddCommand(CHECK, menuId,
"CheckGrammar","Check Grammar", "\!CG");
cmd2Id = F_ApiDefineAndAddCommand(PRINT, menuId,
"PrintErrors","Print Errors", "\!PE");
```

**Example**

The following code defines a menu and adds it to the Edit menu:

```c
#define CHECK 1
#define PRINT 2
F_ObjHandleT editMenuId, menuId, cmd1Id, cmd2Id;

/* Get the ID of the edit menu. */
editMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu,
"EditMenu");

/* Define the menu and add it to the Edit menu. */
menuId = F_ApiDefineAndAddMenu(editMenuId, "GrammarMenu",
"Grammar");

/* Define some commands and add them to the menu. */
cmd1Id = F_ApiDefineAndAddCommand(CHECK, menuId,
"CheckGrammar","Check Grammar", "\!CG");
cmd2Id = F_ApiDefineAndAddCommand(PRINT, menuId,
"PrintErrors","Print Errors", "\!PE");
```

**Figure 3.7** Edit menu with a client-defined pull-right menu
Responding to the user choosing a command

Whenever the user chooses a menu item or types a keyboard shortcut for a command created by your client, the FrameMaker product attempts to call your client’s F_ApiCommand() function. Your client should define this function as follows:

```c
VoidT F_ApiCommand(command)
    IntT command;
{
    /* Code to respond to command choices goes here. */
}
```

### Example

The following client defines some commands and adds them to the Special menu. It provides an F_ApiCommand() function to respond to the user choosing the commands.

```c
#include "fapi.h"
#define LOAD 1
#define QUERY 2

VoidT F_ApiInitialize(initialization)
    IntT initialization;
{
    F_ObjHandleT specialMenuId;

    /* Get the ID of the special menu. */
    specialMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu,
                                         "SpecialMenu");
    /* Define the commands and add them to the Special menu. */
    F_ApiDefineAndAddCommand(LOAD, specialMenuId,
                              "LoadDatabase", "Load Database", "");
    F_ApiDefineAndAddCommand(QUERY, specialMenuId,
                              "QueryDatabase", "Query Database", "");
}

VoidT F_ApiCommand(command)
    IntT command;
{
    switch(command) {
        case LOAD: /* Code to load database goes here. */
            break;
        case QUERY: /* Code to query database goes here. */
            break;
    }
}
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>The value of the cmd parameter in the F_ApiDefineAndAddCommand() call that created the command that the user chose</td>
</tr>
</tbody>
</table>

This argument Means

command The value of the cmd parameter in the F_ApiDefineAndAddCommand() call that created the command that the user chose
Replacing FrameMaker product menus and commands

You can replace FrameMaker product menus and commands with your own menus and commands by calling `F_ApiDefineAndAddCommand()` and `F_ApiDefineAndAddMenu()` with the `name` parameter set to the name of a FrameMaker product menu or command.

For example, the following code replaces the FrameMaker product Print command:

```c
#define PRINT_CMD 223
F_ObjHandleT fileMenuId, printCmdId;
fileMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu, "FileMenu");
printCmdId = F_ApiDefineAndAddCommand(PRINT_CMD, fileMenuId, "Print", "Print...", "!fp");
```

If you use this code, the Print command appears on the File menu exactly as it normally would. However, when the user chooses it, the FrameMaker product never executes the Print operation. Instead, it calls your client’s `F_ApiCommand()` callback with `command` set to `PRINT_CMD`. The `F_ApiCommand()` callback can execute your own version of the Print operation. For example, it can set the default number of copies to 1 and then call `F_ApiSilentPrintDoc()` to print the document. This prevents the user from printing more than one copy of a document at a time.

Allowing users to configure your client’s interface

When you call `F_ApiDefineAndAddCommand()` and specify the name of a command that is already defined in the user’s menu configuration files, the FrameMaker product gives precedence to the definition in the configuration files. If the configuration files assign a label or a shortcut to the command, the FrameMaker product uses it instead of the one you specify. If the command is already a menu item, the FrameMaker product ignores the menu that you specify and leaves the menu item where it is.

For example, if the Print command is already defined and appears on the File menu, the following code has the same effect as the sample code in the previous section:

```c
#define PRINT_CMD 223
F_ObjHandleT printCmdId, bogusMenuId = 12345;
printCmdId = F_ApiDefineAndAddCommand(PRINT_CMD, bogusMenuId, "Print", "This is ignored", "This too");
```

If you use this code, the Print command appears on the File menu exactly as it normally does. Because FrameMaker gives precedence to the labels, shortcuts, and menu item locations specified by the menu configuration files, users can configure your client’s interface. If users know the names of your client’s commands, they can assign labels and shortcuts to the
commands and specify where the commands appear on the menus by editing their menu configuration files.

For example, if your client defines a command with the following code:

```c
F_ObjHandleT editMenuId;
editMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu,
        "EditMenu");
F_ApiDefineAndAddCommand(1, editMenuId,
        "NumberLines","Number Lines", "\\!NL");
```

users can make the command appear on the Special menu instead of the Edit menu by adding the following line to their menu configuration files:

```xml
<Command NumberLines>
<Add NumberLines <Menu SpecialMenu>>
```

If users add this line to their menu configuration files and your client does not define the NumberLines command or is not running, the NumberLines menu item appears dimmed.

**IMPORTANT:** Adobe Systems recommends that you document the names of your client’s menus and commands so that users can customize them.

For more information on using menu configuration files, see the online manual, *Customizing FrameMaker*. For more information on changing commands, menu items, and menus, see Chapter 9, “Manipulating Commands and Menus with the API”

---

**Using hypertext commands in your client’s user interface**

You can embed hypertext commands in markers within FrameMaker product documents. The basic set of FrameMaker hypertext commands allows you to establish links within and between documents and to jump from link to link.

You can lock a FrameMaker document that contains hypertext commands so that it behaves like a command palette. For information on locking documents, see your FrameMaker product user documentation. Documents have a set of properties that specify their characteristics when they are locked. By setting these properties, you can change how a locked document window appears. For example, you can hide the scroll bars and the window control buttons. For a list of locked document properties, see “Document View Only properties” on page 578 of the *FDK Programmer’s Reference*.

FrameMaker provides a special hypertext command, `message apiclient`, that can send messages to your client. With this command, you can create an extremely flexible user interface. Your client only needs to define responses for the hypertext messages that are sent to it. Users and hypertext document designers can set up the interface that sends the messages. The `message apiclient` hypertext command is especially useful for setting up command palettes for your client.
To use the `message apiclient` hypertext command in your client’s interface, follow the general steps below:

1. Set up the hypertext commands.
2. Create a function named `F_ApiMessage()` in your client to respond to the user clicking a hypertext marker that contains a `message apiclient` command.

These steps are discussed in greater detail in the following sections.

### Setting up hypertext commands

The syntax for `message apiclient` is:

```
message apiclient yourmessage
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>apiclient</code></td>
<td>The name under which the client is registered with the FrameMaker product. On UNIX platforms, it is the name specified in the <code>&lt;Name&gt;</code> statement in the <code>apiclients</code> file. On the Macintosh, it is the name of the client file in the <code>Modules</code> folder. On Windows, it is the <code>ClientName</code> specified in the <code>[APIClients]</code> section of the <code>maker.ini</code> file. For more information on registering FDK clients, see the <em>FDK Platform Guide</em> for your platform.</td>
</tr>
<tr>
<td><code>yourmessage</code></td>
<td>The string that the FrameMaker product passes to the API client.</td>
</tr>
</tbody>
</table>

When the user clicks a hypertext command, the FrameMaker product calls the `F_ApiMessage()` function of the client specified by `apiclient` and passes the string specified by `yourmessage` to the client.

### Responding to message apiclient commands

To respond to the `message apiclient` hypertext command, your client must define `F_ApiMessage()` as follows:

```c
VoidT F_ApiMessage(message, docId, objId)
    StringT message;
    F_ObjHandleT docId;
    F_ObjHandleT objId;
{
    /* Code to respond to hypertext message goes here. */
}
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>message</code></td>
<td>The string from the hypertext command message</td>
</tr>
<tr>
<td><code>docId</code></td>
<td>The ID of the document containing the hypertext marker</td>
</tr>
<tr>
<td><code>objId</code></td>
<td>The ID of the hypertext marker the user clicked</td>
</tr>
</tbody>
</table>
Example

Suppose you want to create a command palette with two arrows in it. When the user clicks an arrow, it changes the fill pattern of a selected graphic object in the active document. To make this command palette, create a document with the graphics shown in Figure 3.8.

![Figure 3.8 Example hypertext markers](image)

Assuming your client is registered with the FrameMaker product as myapi, insert the following hypertext markers into the document:

- In the text column around the up arrow: message myapi 1
- In the text column around the down arrow: message myapi 2

Save the document in View Only format.

To respond to the user clicking one of the arrows, add the following code to your client:

```c
#define UPARROW   1
#define DOWNARROW 2

VoidT F_ApiMessage(message, docId, objId)
{
    StringT message;
    F_ObjHandleT docId;
    F_ObjHandleT objId;

    F_ObjHandleT targetDocId, targetGraphicId;
    IntT fillpatt;

    /* Get ID of active document. Note that View Only documents are not considered active. */
    targetDocId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
```
Creating Your Client’s User Interface

/* Get ID of selected object in active document. */
targetGraphicId = F_ApiGetId(targetDocId, targetDocId,
   FP_FirstSelectedGraphicInDoc);
if(!targetGraphicId) return;

/* Get selected object’s current fill pattern. */
fillpatt = F_ApiGetInt(targetDocId, targetGraphicId, FP_Fill);

switch(atoi(message))
{
  case UPARROW:
    /* Only 16 patterns available, so reset at 16. */
    if (++fillpatt == 16) fillpatt = 0; break;
  case DOWNARROW:
    if (--fillpatt == 0) fillpatt = 15; break;
}

/* Set the fill pattern of the selected graphic. */
F_ApiSetInt(targetDocId, targetGraphicId, FP_Fill, fillpatt);

For this example client to work, you should have the View Only document and one normal document open. Select a graphic in the normal document, then click one of the hypertext markers in the View Only document.

---

Responding to user-initiated events or FrameMaker product operations

Your client can respond when the user initiates certain events or FrameMaker product operations. For example, you may want your client to archive an extra copy of a document each time the user saves it, or to display a special dialog box when the user exits a document.

To make your client respond to events, follow these general steps:

1. Request notification for the events.
2. Create a callback function named \texttt{F_ApiNotify()} to respond to the events.

These steps are discussed in greater detail in the following sections.

---

Requesting notification for events

To receive notification for a particular event, your client must request it by calling \texttt{F_ApiNotification()}. The syntax for \texttt{F_ApiNotification()} is:
IntT F_ApiNotification(IntT notification, 
           IntT state);

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>notification</td>
<td>A constant that specifies the notification point. See the table below for a list of the constants.</td>
</tr>
<tr>
<td>state</td>
<td>Specifies whether to turn notification for the notification point on or off. Specify True to request notification or False to turn notification off.</td>
</tr>
</tbody>
</table>

For most events, there are several *notification points*, or stages in the event when the FrameMaker product can notify your client. For example, when the FrameMaker product saves a file, there are two notification points: one immediately before and one immediately after it saves the file. The following table lists the notification points and constants the FrameMaker product passes to `F_ApiNotify()` for some events.

<table>
<thead>
<tr>
<th>Event or operation</th>
<th>Notification points</th>
<th>Notification constants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame binary document opened</td>
<td>Before checking the type of the file to be opened</td>
<td>FA_Note_PreFileType</td>
</tr>
<tr>
<td></td>
<td>After checking the type of the file to be opened</td>
<td>FA_Note_PostFileType</td>
</tr>
<tr>
<td></td>
<td>Before opening the file</td>
<td>FA_Note_PreOpenDoc</td>
</tr>
<tr>
<td></td>
<td>After opening the file</td>
<td>FA_Note_PostOpenDoc</td>
</tr>
<tr>
<td>Filterable document opened</td>
<td>Before checking the type of the file to be opened</td>
<td>FA_Note_FilterIn</td>
</tr>
<tr>
<td>Document saved in Frame binary format</td>
<td>Before saving the document</td>
<td>FA_Note_PreSaveDoc</td>
</tr>
<tr>
<td></td>
<td>After saving the document</td>
<td>FA_Note_PostSaveDoc</td>
</tr>
<tr>
<td>Document saved as filterable type</td>
<td>Before saving the document</td>
<td>FA_Note_FilterOut</td>
</tr>
<tr>
<td>Document exited</td>
<td>Before exiting the document</td>
<td>FA_Note_PreQuitDoc</td>
</tr>
<tr>
<td></td>
<td>After exiting the document</td>
<td>FA_Note_PostQuitDoc</td>
</tr>
</tbody>
</table>

For a complete list of events and notification points, see “F_ApiNotification()” on page 226 of the *FDK Programmer’s Reference*.

You can request notification in your client’s `F_ApiInitialize()` callback or anywhere you want in your client.

**Example**

Suppose you want the FrameMaker product to notify your client whenever the user exits a document. To request this notification when your client is first started, write the `F_ApiInitialize()` callback as follows:
VoidT F_ApiInitialize(initialization)
    IntT initialization;
{
    /* Request notification for exit. */
    if (initialization == FA_Init_First)
        F_ApiNotification(FA_Note_PreQuitDoc, True);
}

Requesting notification for API filters

API client filters do not need to request notification. To receive filter notifications, filters only need to be registered with the FrameMaker product. If they are correctly registered, filters receive the following notifications:

<table>
<thead>
<tr>
<th>This type of filter</th>
<th>Receives this notification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import</td>
<td>FA_Note_FilterIn</td>
</tr>
<tr>
<td>Export</td>
<td>FA_Note_FilterOut</td>
</tr>
<tr>
<td>File-to-file</td>
<td>FA_Note_FilterFileToFile</td>
</tr>
</tbody>
</table>

For more information on writing client filters, see “Writing filter clients” on page 239. For more information on registering filters, see the FDK Platform Guide for your platform.

Adding the F_ApiNotify() callback

The FrameMaker product notifies your client of events for which it has requested notification by calling its F_ApiNotify() function. Your client should define F_ApiNotify() as follows:

VoidT F_ApiNotify(notification, docId, sparm, iparm)
    IntT notification;
    F_ObjHandleT docId;
    StringT sparm;
    IntT iparm;
{
    /* Code that responds to notifications goes here. */
}

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>notification</td>
<td>A constant that indicates the event and the notification point (see the table on page 35 for a list of some constants).</td>
</tr>
<tr>
<td>docId</td>
<td>The ID of the active document when the event occurs. For filters, the document into which the filter should import its data; if this is zero, the filter must create a new document.</td>
</tr>
</tbody>
</table>
Creating Your Client’s User Interface

F_ApiNotify() can call API functions to get or set object properties or to initiate FrameMaker product operations. The FrameMaker product calls F_ApiNotify() only at the notification points for which your client has requested notification.

For example, the following code prints the name of each document the user opens to the console:

```
VoidT F_ApiInitialize(initialization)
    IntT initialization;
{
    if (initialization == FA_InitFirst)
        F_ApiNotification(FA_Note_PostOpenDoc, True);
}
```

```
VoidT F_ApiNotify(notification, docId, sparm, iparm)
    IntT notification;
    F_ObjHandleT docId;
    StringType sparm;
    IntT iparm;
{
    if (notification == FA_Note_PostOpenDoc)
        F_Printf(NULL, "The user opened: \s\n", sparm);
}
```

**Canceling commands**

Your client can cancel any command or action for which it receives a FA_Note_PreNotificationPoint notification. For example, if it receives the FA_Note_PreQuitDoc notification, it can cancel the Close command and prevent the user from closing a document.

To abort a command, call F_ApiReturnValue(), with the parameter set to FR_CancelOperation, when your client receives notification for the command. For example, the following code cancels the Exit command, preventing the user from closing any documents:

---

**This argument** | **Means**
---|---

sparm | The string, if any, associated with the notification. For example, if the notification is for an Open or Save, sparm specifies the pathname of the affected file. If the notification is for text entry, sparm specifies the text the user typed. Depending on how fast the user is typing, sparm may specify one or several characters at a time.

iparm | The integer associated with the notification. For example, if notification is FA_NotePreFunction or FA_NotePostFunction, iparm specifies the f-code for the command.
Responding to text entry and actions that have no specific notifications

The API doesn’t provide specific notifications for most user actions. Instead, it provides the following general notifications, which it issues for nearly every user action.

<table>
<thead>
<tr>
<th>Event or operation</th>
<th>Notification points</th>
<th>Notification constants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any user action that the FrameMaker product processes</td>
<td>After the FrameMaker product finishes processing the action</td>
<td>FA_Note_BackToUser</td>
</tr>
<tr>
<td>FrameMaker product command invoked or text entered in a document</td>
<td>Before the FrameMaker product executes the command or adds text to the document</td>
<td>FA_Note_PreFunction</td>
</tr>
<tr>
<td></td>
<td>After the FrameMaker product executes the command or adds text to the document</td>
<td>FA_Note_PostFunction</td>
</tr>
</tbody>
</table>

The API issues the FA_Note_BackToUser notification after any user action the FrameMaker product processes, including insertion point changes, selection changes, and text entry. This notification is useful if you need to update a modeless dialog box containing settings that are dependent on the insertion point.

When the API issues the FA_Note_BackToUser notification, it indicates only that an action occurred; it does not specify which action. If you want to respond to specific actions, use the FA_Note_PreFunction or the FA_Note_PostFunction notification instead of FA_Note_BackToUser.

**IMPORTANT:** When the FrameMaker product performs a book-wide command (a command that process all documents in a book), it posts an FA_Note_PreFunction and FA_Note_PostFunction notification for the book file, and another for each document in the book. When trapping book-wide functions, you should check the value of docId to determine whether it indicates a document or the active book.

For example, if you search a book with two documents in it, the FrameMaker product posts the following function notifications:

FA_Note_PreFunction (start searching book)
When the API issues an `FA_Note_PreFunction` or `FA_Note_PostFunction` notification, it specifies the user action by setting `iparm` to a function code (f-code). An f-code is a hexadecimal code that specifies a command or other user action. The following table shows some common f-codes and the commands or user actions they specify.

<table>
<thead>
<tr>
<th>F-code</th>
<th>Command or user action</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGF_APPLY_TAG</td>
<td>The user applied a paragraph format</td>
</tr>
<tr>
<td>CHAR_APPLY_TAG</td>
<td>The user applied a character format</td>
</tr>
<tr>
<td>TXT_10</td>
<td>The user set the text size to 10 points</td>
</tr>
<tr>
<td>KBD_OPEN</td>
<td>The user chose Open</td>
</tr>
<tr>
<td>KBD_INPUT</td>
<td>The user typed some text</td>
</tr>
<tr>
<td>KBD_ALIGN</td>
<td>The user chose Align</td>
</tr>
</tbody>
</table>

For a complete list of f-codes, see the `fcodes.h` file shipped with the FDK.

If a user action is associated with a text string, the API passes the string to the `sparm` parameter of your client’s `F_ApiNotify()` function. For example, when the user types text, the API sets `sparm` to the text the user typed.

The following table lists some f-codes and the strings that are associated with them.

<table>
<thead>
<tr>
<th>F-code</th>
<th>Associated string that the API passes to <code>sparm</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>PGF_APPLY_TAG</td>
<td>The name of the paragraph format the user applied.</td>
</tr>
<tr>
<td>CHAR_APPLY_TAG</td>
<td>The name of the character format the user applied.</td>
</tr>
<tr>
<td>KBD_INPUT</td>
<td>The text the user typed. It can be one or more characters</td>
</tr>
<tr>
<td></td>
<td>depending on how fast the user types.</td>
</tr>
<tr>
<td>TXT_FAMILY_AND_VARIATION</td>
<td>The name of the font family the user chose.</td>
</tr>
</tbody>
</table>

Your client can cancel any action for which it receives the `FA_Note_PreFunction` notification by calling `F_ApiReturnValue()` with `retVal` set to `FR_CancelOperation`. Your client can even cancel text entry.

For example, the following code intercepts any text the user attempts to type in a document and prints it to the console:
Creating Your Client's User Interface

```c
#include "fcodes.h"
/* Turn on notification. */
F_ApiNotification(FA_Note_PreFunction, True);

VoidT F_ApiNotify(notification, docId, sparm, iparm)
    IntT notification;
    F_ObjHandleT docId;
    StringT sparm;
    IntT iparm;
{
    if (notification == FA_Note_PreFunction
        && iparm == KBD_INPUT)
    {
        F_Printf(NULL,"The user typed: %s
", sparm);
        /* Prevent text from being added to document. */
        F_ApiReturnValue(FR_CancelOperation);
    }
}

Responding to events initiated by API clients

FrameMaker notifies your client of any event that it has requested notification for. The event can be initiated directly by the user or by another client.

The Frame API provides a set of functions that allow API clients to programmatically execute Open, Save, and several other FrameMaker product operations. For more information on these functions, see Chapter 4, “Executing Commands with API Functions” When a client executes an operation with one of these functions, the FrameMaker product notifies all the other API clients that have requested notification for that event. It does not, however, notify the client that executed the operation. For example, to have your client automatically make an additional copy of a document when the user saves it, use the following code:

```c
/* Turn on notification. */
F_ApiNotification(FA_Note_PostSaveDoc, True);

VoidT F_ApiNotify(notification, docId, sparm, iparm)
    IntT notification;
    F_ObjHandleT docId;
    StringT sparm;
    IntT iparm;
{
```

---

1. An API client can explicitly instruct the FrameMaker product to suppress notifications to other API clients when it opens or saves a file by setting the `FS_DontNotifyAPIClients` property of the Open or Save script to `True`. For more information on properties in the Open and Save scripts, see “F_ApiGetOpenDefaultParams()” on page 138 and “F_ApiGetSaveDefaultParams()” on page 152 of the `FDK Programmer’s Reference`. 

```c
```
In the example above, \texttt{F_ApiNotify()}, which responds to a Save notification, uses \texttt{F_ApiSimpleSave()} to execute a Save operation itself. This does not result in infinite recursion because the FrameMaker product does not notify the client of the Save operation that it executes itself.

**Handling notification for Open operations**

The Open operation is more complex than most other operations. FrameMaker product does the following when it opens a file:

1. Determines whether the file is filterable.

   If the file is filterable, the FrameMaker product issues the \texttt{FA_Note_FilterIn} notification to the appropriate filter and abandons the Open operation. It is up to the filter to finish opening the file. No other client receives any notification.

   If the file is not filterable, the FrameMaker product continues with the Open operation.

2. Issues an \texttt{FA_Note_PreFileType} notification to all clients that have requested it.

   This allows clients to uncompress a file if it is compressed, check it out if it is under version control, or perform other operations that may change its type.

3. Checks the file’s type.

   If the file is a type that the FrameMaker product can’t open, the FrameMaker product displays a warning and cancels the Open operation. If the file is from a previous version of a FrameMaker product, it prompts the user to convert the file or cancel the Open operation.

4. Issues an \texttt{FA_Note_PostFileType} notification to all clients that have requested it.

5. Determines whether the file is a document or a book, and whether its format is Frame binary or MIF.

6. Issues an \texttt{FA_Note_PreOpenDoc, FA_Note_PreOpenBook, FA_Note_PreOpenMIF,} or \texttt{FA_Note_PreOpenBookMIF} notification.

7. Opens the document or book.

   If the document or book is MIF, the FrameMaker product converts it.

8. Issues an \texttt{FA_Note_PostOpenDoc, FA_Note_PostOpenMIF,} \texttt{FA_Note_PostOpenBook,} or \texttt{FA_Note_PostOpenBookMIF} notification.

   Normally, you don’t request the \texttt{FA_Note_PreFileType} and \texttt{FA_Note_PostFileType} notifications, because you don’t want to do anything with a file before the FrameMaker product has checked its type. However, if you want to change a file’s contents after the user has...
selected it but before the FrameMaker product has checked its type, you should request notification for the FA_Note_PreFileType notification point.

For example, suppose you want to uncompress a compressed document file when the user opens it. Normally, when a user attempts to open a compressed file, the FrameMaker product displays an “Unrecognized type” alert and cancels the Open operation when it checks the file’s type. You must uncompress the file after the user has chosen it, but before the FrameMaker product checks its type. To do this, you could use the following code:

```c
... 
F_ApiNotification(FA_Note_PreFileType, True);
... 
VoidT F_ApiNotify(notification, docId, sparm, iparm)
    IntT notification;
    F_ObjHandleT docId;
    StringT sparm;
    IntT iparm
{
    if (notification == FA_Note_PreFileType)
    {
    /* Code to test if file is compressed goes here. */
    F_ApiAlert("Uncompressing file.", FF_ALERT_CONTINUE_NOTE);
    /* Code to uncompress file goes here. */
    }
}
...```

### Implementing quick keys

FrameMaker provides a quick-key interface, which allows the user to choose commands in the document Tag area. In FrameMaker, for example, the user can apply a character format by pressing Esc q c: FrameMaker displays an f: prompt in the Tag area. The user can then choose a character format by typing the first few letters of the format’s name and pressing Return when the format appears in the Tag area.

Your client can implement its own quick-key interface by calling F_ApiQuickSelect(). The syntax for F_ApiQuickSelect() is:

```c
IntT F_ApiQuickSelect(F_ObjHandleT docId,
                      StringT prompt,
                      F_StringsT *stringlist);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document containing the Tag area in which to display the prompt</td>
</tr>
<tr>
<td>prompt</td>
<td>The prompt that appears in the Tag area</td>
</tr>
<tr>
<td>stringlist</td>
<td>The list of strings from which the user can choose</td>
</tr>
</tbody>
</table>
Creating Your Client’s User Interface

F_ApiQuickSelect() returns the index of the string the user chose or -1 if the user canceled the command.

For example, the following code implements the quick-key interface shown in Figure 3.9:

...  
F_StringsT fruits;
StringT strings[3];
IntT choice;
F_ObjHandleT docId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
strings[0] = (StringT)"Kumquat";
strings[1] = (StringT)"Durian";
strings[2] = (StringT)"Rambutan";
fruits.len = 3;
fruits.val = strings;
choice = F_ApiQuickSelect(docId, (StringT)"Fruit:", &fruits);

if (choice != -1)
    F_Printf(NULL, (StringT)"The user chose: %s.\n", strings[choice]);
...

Figure 3.9  Tag area with client-defined quick key

Freeing system resources by bailing out

Instead of leaving your client idle when it’s not executing, you may want to free the system resources it uses. The API provides a function named F_ApiBailOut(), which allows you to do this. When you call F_ApiBailOut(), the FrameMaker product waits until your client returns from the current callback, then exits your client, freeing the system resources it uses. If your client is a dynamically linked client on a UNIX platform, bailing out has no effect.

Important: Never call exit(), F_Exit(), or abort() from an API client. If you call one of these functions, the function exits the FrameMaker product and unpredictable results, including data loss, may occur.

After it exits your client, the FrameMaker product continues processing events that affect it. Your client’s menus remain on the menu bar and are still valid. If your client has requested notification for particular events, the FrameMaker product continues to monitor those events. The FrameMaker product also monitors message apiclient hypertext commands that specify your client.

If the user chooses one of your client’s menu items or hypertext commands, or initiates an event for which your client requested notification, the FrameMaker product restarts your client, calling its F_ApiInitialize() function with initialization set to FA_Init_Subsequent. After F_ApiInitialize() has returned control, the
FrameMaker product also calls the appropriate callback function for the menu choice, hypertext command, or event.

**IMPORTANT:** If your client bails out, it loses all its global variable settings.

The following code requests notification for the `FA_Note_PreQuitDoc` notification point and then bails out after the user starts the FrameMaker product. If the user subsequently closes a document, it displays dialog boxes indicating when the FrameMaker product reinitializes the client and when it issues the `FA_Note_PreQuitDoc` notification.

```c
VoidT F_ApiInitialize(initialization)
IntT initialization;
{
    switch (initialization){
    case FA_Init_First:
        /* Request notification. */
        F_ApiNotification(FA_Note_PreQuitDoc, True);
        /* Bail out and wait for the event. */
        F_ApiBailOut();
        break;

    case FA_Init_Subsequent:
        F_ApiAlert((StringT)"Client is reinitializing",
                    FF_ALERT_CONTINUE_NOTE);
        break;
    }
}

VoidT F_ApiNotify(notification, docId, filename)
IntT notification;
F_ObjHandleT docId;
StringT filename;
{
    if (notification == FA_Note_PreQuitDoc)
        F_ApiAlert((StringT)"Client has reinitialized.",
                   FF_ALERT_CONTINUE_NOTE);
}
```
This chapter discusses how to use Frame API functions to execute FrameMaker commands programmatically.

The API doesn’t provide a function to directly execute each of the commands available in the FrameMaker product user interface. This is because you can achieve the effect of some commands by setting object properties. For example, to set a graphic’s fill pattern, you set the object’s `FP_Fill` property. For more information on setting object properties, see Chapter 5, “Getting and Setting Properties”

---

**Handling errors**

When an API function fails, it stores an error code in the global variable, `FA_errno`. `FA_errno` retains the error code until another function fails and sets it or until your code explicitly sets it. To determine whether a set of API function calls has failed, initialize `FA_errno` to `FE_Success` once before all the calls and check it once after all the calls.

To find the error codes a function can generate, look up the function in the *FDK Programmer’s Reference*. For a list of all API error codes and their meanings, see the `fapidefs.h` header file included with FDK or Chapter 5, “Error Codes” in the *FDK Programmer’s Reference*. In the interest of brevity, the examples in this chapter do not include error-handling code. However, you should check `FA_errno` after calling functions that set it.

---

**Handling messages and warnings**

In the user interface, some commands such as Open and Save sometimes need to prompt the user with warnings or messages. The API provides two types of functions to execute these commands:

- **Simple functions** allow you to either suppress the messages and warnings entirely or to prompt the user with them.
- **Scriptable functions** allow you to specify a response for each possible message or warning.

**Using simple functions**

Simple functions enable you to execute commands such as Save and Open without specifying numerous parameters. They execute these commands in either an interactive or a noninteractive mode. If you call a simple function and specify the interactive mode, the FrameMaker product behaves exactly as it would if the user had initiated the command. If a message or warning condition occurs, the FrameMaker product prompts the user. For example,
if you call F_ApiSimpleOpen() in the interactive mode, the FrameMaker product displays the Open dialog box and prompts the user to choose a file to open. If the user chooses a text file, the FrameMaker product displays a Reading Text File dialog box.

If you are sure that executing a command won’t do something undesirable, and you don’t want the FrameMaker product to display error and warning messages, call a simple function in noninteractive mode. Be careful when you do this, because you may inadvertently destroy data. For example, suppose you attempt to save a file by calling F_ApiSimpleSave() in the noninteractive mode. If the file already exists, the FrameMaker product overwrites it without warning your client or the user. Noninteractive mode is useful for clients that need to carry out tasks without a user present.

Using scriptable functions

To specify a response for each possible message or warning that the FrameMaker product may issue while executing a command, use a scriptable function to execute the command. When you call a scriptable function, you pass it a script or property list that contains properties corresponding to possible messages or warnings. For most messages and warnings, you either specify a Yes, No, or Cancel response, or you can instruct the FrameMaker product to prompt the user for the response. Scriptable functions return detailed information on how the FrameMaker product executes a command. For example, the scriptable Open function F_ApiOpen() returns information, such as whether the file was filtered and whether an Autosave file was used.

The API provides a function named F_ApiAllocatePropVals(), which allocates a property list that you can use with scriptable functions. The API also provides functions that create default scripts for the different scriptable functions. You can use these functions to get a default script and then customize the script by changing individual properties.

Opening documents and books

The API provides two functions to open a document or book:

- F_ApiSimpleOpen() is an easy-to-use function for opening a document or book.
- F_ApiOpen() allows you to script the process of opening a document or book.

Opening a document or book with F_ApiSimpleOpen()

The syntax for F_ApiSimpleOpen() is:

F_ObjHandleT F_ApiSimpleOpen(StringT fileName, BoolT interactive);
Executing Commands with API Functions

If `F_ApiSimpleOpen()` is successful, it returns the ID of the FO_Doc or FO_Book object that represents the document or book that it opened. If a condition (such as a nonexistent file) makes it impossible to open a file, `F_ApiSimpleOpen()` aborts the operation and returns 0.

If you set `interactive` to True, the FrameMaker product displays the Open dialog box. It uses the path specified by the session property `FP_OpenDir` as the default path. The FrameMaker product also displays all the other messages and warnings it would normally display if the user chose the Open command. For example, if a document contains fonts that are not available in the current session, the FrameMaker product displays a “Fonts Unavailable. Open Anyway?” dialog box. If the user clicks Cancel, `F_ApiSimpleOpen()` aborts the operation and returns 0.

If you set `interactive` to False, the FrameMaker product does not display the Open dialog box or other messages and warnings. If it is necessary to modify a file to continue opening it, `F_ApiSimpleOpen()` aborts the operation without notifying the user, and returns 0. For example, if a document contains fonts that are not available, `F_ApiSimpleOpen()` aborts the Open operation instead of converting the fonts.

**Example**

The following code opens a document named `/tmp/my.doc` and displays its ID:

```c
#include "futils.h"
F_ObjHandleT docId;
UCharT msg[256];

docId = F_ApiSimpleOpen((StringT)"/tmp/my.doc", False);
if (!docId)
   F_ApiAlert((StringT)"Couldn’t open.", FF_ALERT_CONTINUE_NOTE);
else
   {
      F_Sprintf(msg, (StringT)"my.doc’s ID is 0x%x.", docId);
      F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);
   }
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileName</td>
<td>The absolute pathname of the file to open. For information on how filenames and paths on different platforms are expressed, see the FDK Platform Guide for your platform.</td>
</tr>
<tr>
<td>interactive</td>
<td>Specifies whether the FrameMaker product displays messages and warnings to the user.</td>
</tr>
</tbody>
</table>

If `F_ApiSimpleOpen()` is successful, it returns the ID of the FO_Doc or FO_Book object that represents the document or book that it opened. If a condition (such as a nonexistent file) makes it impossible to open a file, `F_ApiSimpleOpen()` aborts the operation and returns 0.

If you set `interactive` to True, the FrameMaker product displays the Open dialog box. It uses the path specified by the session property `FP_OpenDir` as the default path. The FrameMaker product also displays all the other messages and warnings it would normally display if the user chose the Open command. For example, if a document contains fonts that are not available in the current session, the FrameMaker product displays a “Fonts Unavailable. Open Anyway?” dialog box. If the user clicks Cancel, `F_ApiSimpleOpen()` aborts the operation and returns 0.

If you set `interactive` to False, the FrameMaker product does not display the Open dialog box or other messages and warnings. If it is necessary to modify a file to continue opening it, `F_ApiSimpleOpen()` aborts the operation without notifying the user, and returns 0. For example, if a document contains fonts that are not available, `F_ApiSimpleOpen()` aborts the Open operation instead of converting the fonts.

**Example**

The following code opens a document named `/tmp/my.doc` and displays its ID:

```c
#include "futils.h"
F_ObjHandleT docId;
UCharT msg[256];

docId = F_ApiSimpleOpen((StringT)"/tmp/my.doc", False);
if (!docId)
   F_ApiAlert((StringT)"Couldn’t open.", FF_ALERT_CONTINUE_NOTE);
else
   {
      F_Sprintf(msg, (StringT)"my.doc’s ID is 0x%x.", docId);
      F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);
   }
```

1. Examples in this manual use UNIX pathnames. For information on specifying pathnames on other platforms, see the FDK Platform Guide for that platform.
Opening a document or book with F_ApiOpen()

To open a document or book and programmatically specify responses to warnings and messages that the FrameMaker product issues, use F_ApiOpen(). With F_ApiOpen(), you can specify aspects of the Open operation, such as whether to make a document visible and whether to use an Autosave file. You can specify all aspects of the operation, or you can specify some aspects and allow the user to decide others. For example, you can instruct the FrameMaker product to only open a MIF file but allow the user to choose the file.

To use F_ApiOpen(), you should first understand property lists and how to manipulate them directly. For more information on this subject, see “Representing object characteristics with properties” on page 270 and “Manipulating property lists directly” on page 93.

The syntax for F_ApiOpen() is:

```c
F_ObjHandleT F_ApiOpen(StringT fileName,
    F_PropValsT *openParamsp,
    F_PropValsT **openReturnParamspp);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileName</td>
<td>The absolute pathname of the file to open. If you are using F_ApiOpen() to create a document, specify the template name.</td>
</tr>
<tr>
<td>openParamsp</td>
<td>A property list (script) that tells the FrameMaker product how to open the file and how to respond to errors and other conditions that arise. Use F_ApiGetOpenDefaultParams() or F_ApiAllocatePropVals() to create and allocate memory for this property list. To use the default list, specify NULL.</td>
</tr>
<tr>
<td>openReturnParampp</td>
<td>A property list that returns the pathname and provides information on how the FrameMaker product opened the file.</td>
</tr>
</tbody>
</table>

**IMPORTANT:** Always initialize the pointer to the property list that you specify for openReturnParampp to NULL before you call F_ApiOpen().

If F_ApiOpen() is successful, it returns the ID of the opened document or book. Otherwise, it returns 0.

To call F_ApiOpen(), do the following:

1. Initialize the pointer to the openReturnParampp property list to NULL.

2. Create an openParamsp property list.

   You can get a default list by calling F_ApiGetOpenDefaultParams(), or you can create a list from scratch.

3. Call F_ApiOpen().

4. Check the Open status.

   Check the returned values in the openReturnParampp list for the name of the opened file and other information about how the FrameMaker product opened the file.
5. Deallocate memory for the openParamsp and openReturnParamsp property lists. Use F_ApiDeallocatePropVals() to deallocate memory for the lists.

Steps 2, 4, and 5 are discussed in the following sections.

Creating an openParamsp script with F_ApiGetOpenDefaultParams()

If you need to specify a number of properties in the openParamsp property list, it is easiest to get a default list with F_ApiGetOpenDefaultParams() and then modify individual properties in the list.

The syntax for F_ApiGetOpenDefaultParams() is:

```
F_PropValsT F_ApiGetOpenDefaultParams();
```

The following table lists some of the properties in the property list returned by F_ApiGetOpenDefaultParams(). The first value listed for each property is the default value used in the list. You can change the list to use the other listed values. For the complete list of properties in the property list, see “F_ApiGetOpenDefaultParams()” on page 138 of the FDK Programmer’s Reference.

<table>
<thead>
<tr>
<th>Property</th>
<th>Instruction or situation and possible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>FS_ShowBrowser</td>
<td>Display Open dialog box.</td>
</tr>
<tr>
<td></td>
<td>False: don’t display it.</td>
</tr>
<tr>
<td></td>
<td>True: display it.</td>
</tr>
<tr>
<td>FS_OpenDocViewOnly</td>
<td>Open document as View Only.</td>
</tr>
<tr>
<td></td>
<td>False: don’t open as View Only.</td>
</tr>
<tr>
<td></td>
<td>True: open as View Only.</td>
</tr>
<tr>
<td>FS_NameStripe</td>
<td>String specifying the name that appears on the document title bar. Setting this property on Macintosh platforms has no effect.</td>
</tr>
<tr>
<td></td>
<td>NULL.</td>
</tr>
<tr>
<td>FS_NewDoc</td>
<td>Create a new document.</td>
</tr>
<tr>
<td></td>
<td>False: open an existing document.</td>
</tr>
<tr>
<td></td>
<td>True: create a new document.</td>
</tr>
</tbody>
</table>

For example, to get a default openParamsp property list and modify it so that it instructs F_ApiOpen() to show the Open dialog box, use the following code:

```
. . .
F_ObjHandleT docId;
F_PropValsT params, *returnParamsp = NULL;
IntT i;

/* Get a default property list. */
params = F_ApiGetOpenDefaultParams();
```
Executing Commands with API Functions

/* If F_ApiGetOpenDefaultParams() fails, len will be 0. */
if(params.len == 0) return;

/* Get index of FS_ShowBrowser property, then set it to True. */
i = F_ApiGetPropIndex(&params, FS_ShowBrowser);
params.val[i].propVal.u.ival = True;

/* Change default to /tmp when Open dialog box appears. */
F_ApiSetString(0, FV_SessionId, FP_OpenDir, "/tmp");

docId = F_ApiOpen("", &params, &returnParamsp);
F_ApiDeallocatePropVals(&params);
F_ApiDeallocatePropVals(returnParamsp);

The API allocates memory for the property list created by
F_ApiGetOpenDefaultParams(). Use F_ApiDeallocatePropVals() to free the
property list when you are done with it.

For another example of how to call F_ApiOpen() using a default
property list created by F_ApiGetOpenDefaultParams(), see

Creating an openParamsp script from scratch

If you only need to specify a few properties when you call F_ApiOpen(), it is most efficient
to create a property list from scratch. To create the property list, you must allocate memory for
it and then set up the individual properties.

To allocate memory for the property list, use the API convenience function,
F_ApiAllocatePropVals(). The syntax for F_ApiAllocatePropVals() is:
F_PropValsT F_ApiAllocatePropVals(IntT numProps);

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>numProps</td>
<td>The number of properties for which to allocate memory</td>
</tr>
</tbody>
</table>

For example, the following code creates an openParamsp property list that instructs
F_ApiOpen() to show the Open dialog box:

F_ObjHandleT docId;
F_PropValsT params, *returnParamsp = NULL;

/* Allocate memory for the list. */
params = F_ApiAllocatePropVals(1);

/* Set up the FS_ShowBrowser property and set it to True. */
params.val[0].propIdent.num = FS_ShowBrowser;
params.val[0].propVal.valType = FT_Integer;
params.val[0].propVal.u.ival = True;
docId = F_ApiOpen("", &params, &returnParamsp);
F_ApiDeallocatePropVals(&params);
F_ApiDeallocatePropVals(returnParamsp);

The API allocates memory for the property list created by F_ApiAllocatePropVals(). Use F_ApiDeallocatePropVals() to free the property list when you are done with it.

Checking the Open status

F_ApiOpen() stores a pointer to a property list (F_PropValsT structure) in openReturnParamspp. To get the name of the file that F_ApiOpen() opened and other information about how F_ApiOpen() opened the file, check this property list. It includes the properties shown in the following table.

<table>
<thead>
<tr>
<th>Property</th>
<th>Meaning and possible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>FS_OpenedFileName</td>
<td>A string that specifies the opened file’s pathname. If you scripted FS_ShowBrowser, or the file was filtered, or you didn’t specify the pathname, this pathname can be different from the one you specified in the Open script.</td>
</tr>
<tr>
<td>FS_OpenNativeError</td>
<td>The error condition. If the file is opened successfully, it is set to FE_Success. For a complete list of the other values it can be set to, see “F_ApiOpen()” on page 238 of the FDK Programmer’s Reference.</td>
</tr>
<tr>
<td>FS_OpenStatus</td>
<td>A bit field indicating what happened when the file was opened. For a complete list of the possible status flags, see “F_ApiOpen()” on page 238 of the FDK Programmer’s Reference.</td>
</tr>
</tbody>
</table>

The FS_OpenNativeError property and the FA_errno global variable indicate the result of a call to F_ApiOpen(). The FS_OpenStatus flags indicate how or why this result occurred. For example, if you attempt to open a file with F_ApiOpen() and the Open operation is canceled, FS_OpenNativeError and FA_errno are set to FE_Canceled. If the operation was canceled because the user canceled it, the FV_UserCanceled bit of the FS_OpenStatus property list is set.

The API provides a function named F_ApiCheckStatus(), which allows you to determine if a particular FS_OpenStatus bit is set. The syntax for F_ApiCheckStatus() is:

```
IntT F_ApiCheckStatus(F_PropValsT *p,
                      IntT statusBit);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>The openReturnParamsp property list returned by F_ApiOpen()</td>
</tr>
<tr>
<td>statusBit</td>
<td>The status bit you want to test</td>
</tr>
</tbody>
</table>
If the specified bit is set, `F_ApiCheckStatus()` returns `True`. For example, the following code determines if an Open operation was canceled because a document used unavailable fonts:

```c
F_ObjHandleT docId;
F_PropValsT params, *returnParamsp = NULL;
/* Get default property list. */
params = F_ApiGetOpenDefaultParams();
docId = F_ApiOpen("/tmp/my.doc", &params, &returnParamsp);
if (F_ApiCheckStatus(returnParamsp, FV_CancelFontsMapped))
    F_ApiAlert("CANCELED because my.doc has unavailable fonts.",
               FF_ALERT_CONTINUE_NOTE);
/* Deallocate property lists. */
F_ApiDeallocatePropVals(&params);
F_ApiDeallocatePropVals(returnParamsp);
```

The API also provides a convenience function named `F_ApiPrintOpenStatus()`, which prints the Open status values to the console on UNIX platforms, the error log window on the Macintosh, and the Frame console on Windows. `F_ApiPrintOpenStatus()` is useful for debugging clients that use `F_ApiOpen()`. For more information, see “`F_ApiPrintOpenStatus()`” on page 250 of the `FDK Programmer’s Reference`.

### Deallocating Open script property lists

After you are done with the Open script property lists, call the API convenience function, `F_ApiDeallocatePropVals()`, to free the memory they use.

The syntax for `F_ApiDeallocatePropVals()` is:

```
VoidT F_ApiDeallocatePropVals(F_PropValsT *pvp);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pvp</code></td>
<td>The property list</td>
</tr>
</tbody>
</table>

### Example

The following code opens a document named `/tmp/my.doc`. It creates a property list that instructs `F_ApiOpen()` to open the document as View Only and to display the title, Doc, in the title bar.

```c
#include "fstrings.h"
F_PropValsT params, *returnParamsp = NULL;
F_ObjHandleT docId;
/* Allocate memory for Open script
```
Executing Commands with API Functions

with two properties. */
params = F_ApiAllocatePropVals(2);
if(params.len == 0) return;

/* Force title displayed on title bar to be "Doc". */
params.val[0].propIdent.num = FS_NameStripe;
params.val[0].propVal.valType = FT_String;
params.val[0].propVal.u.sval = (StringT)F_StrCopyString("Doc");

/* Open the file as View Only. */
params.val[1].propIdent.num = FS_OpenDocViewOnly;
params.val[1].propVal.valType = FT_Integer;
params.val[1].propVal.u.ival = True;

docId = F_ApiOpen("/tmp/my.doc", &params, &returnParamsp);

/* Free memory used by the Open scripts. */
F_ApiDeallocatePropVals(&params);
F_ApiDeallocatePropVals(returnParamsp);

Creating documents

To create a new document, you can use the following functions:

- **F_ApiSimpleNewDoc()** is an easy-to-use function that allows you to specify a template and interactive or noninteractive modes.
- **F_ApiCustomDoc()** uses the FrameMaker product’s default new document template and some parameters that you specify to create the new document.
- **F_ApiOpen()** allows you to script the New operation.

For information on creating books, see “Creating a book” on page 146. The following sections describe how to create a new document in greater detail.

Creating a document with **F_ApiSimpleNewDoc()**

To create a new document from a specific template, use **F_ApiSimpleNewDoc()**.

The syntax for **F_ApiSimpleNewDoc()** is:

```
F_ObjHandleT F_ApiSimpleNewDoc(StringT templateName, IntT interactive);
```
Executing Commands with API Functions

If you set `interactive` to `True`, the FrameMaker product creates a document from the specified template and displays messages and warnings to the user. If you set `interactive` to `False`, the FrameMaker product does not display messages and warnings; if the FrameMaker product encounters a condition for which it normally displays a dialog box, `F_ApiSimpleNewDoc()` attempts to do what’s necessary to continue creating the file. If `F_ApiSimpleNewDoc()` is successful, it returns the ID of the document it created; otherwise, it returns `0`. You don’t provide the name for the new document until you save it.

**IMPORTANT:** If you call `F_ApiSimpleNewDoc()` with `interactive` set to `True` and the user clicks Portrait, Custom, or Landscape in the New dialog box, `F_ApiSimpleNewDoc()` does not create a document. It returns `0` and sets `FA_errno` to `FE_WantsPortrait`, `FE_WantsCustom`, or `FE_WantsLandscape`. It is up to your client to create a portrait, custom, or landscape document by calling `F_ApiCustomDoc()`. For more information on creating custom documents, see “Creating a custom document” on page 55.

**Example**

The following code creates a document from the `/templates/Reports/Report1` template and saves it as `/tmp/mynew.doc`. It then uses `F_ApiSimpleSave()` to save the document. For more information on `F_ApiSimpleSave()`, see “Saving documents and books” on page 59.

```c
F_ObjHandleT docId;

docId = F_ApiSimpleNewDoc("/templates/Reports/Report1", False);

if (!docId)
    F_ApiAlert("Can’t create document.", FF_ALERT_CONTINUE_NOTE);
else
    F_ApiSimpleSave(docId, "/tmp/mynew.doc", False);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>templateName</code></td>
<td>The absolute pathname of the template to use. For information on how filenames and paths on different platforms are expressed, see the <em>FDK Platform Guide</em> for that platform.</td>
</tr>
<tr>
<td><code>interactive</code></td>
<td>Specifies whether the FrameMaker product displays messages and warnings to the user.</td>
</tr>
</tbody>
</table>

This argument Means

- `templateName`: The absolute pathname of the template to use. For information on how filenames and paths on different platforms are expressed, see the *FDK Platform Guide* for that platform.
- `interactive`: Specifies whether the FrameMaker product displays messages and warnings to the user.
Creating a custom document

To create a custom new document, use `F_ApiCustomDoc()`. This function uses the FrameMaker product’s default new-document template to create the custom document. For more information on the default new-document template, see “Documents” on page 279.

The syntax for `F_ApiCustomDoc()` is:

```c
F_ObjHandleT F_ApiCustomDoc(MetricT width,
    MetricT height,
    IntT numCols,
    MetricT columnGap,
    MetricT topMargin,
    MetricT botMargin,
    MetricT leftinsideMargin,
    MetricT rightoutsideMargin,
    IntT sidedness,
    BoolT makeVisible);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>Page width. The Frame API expresses linear measurements with MetricT values. For more information on MetricT values, see Chapter 4, “Data Types and Structures Reference,” in the FDK Programmer’s Reference.</td>
</tr>
<tr>
<td>height</td>
<td>Page height.</td>
</tr>
<tr>
<td>numCols</td>
<td>Default number of columns.</td>
</tr>
<tr>
<td>columnGap</td>
<td>Default column spacing.</td>
</tr>
<tr>
<td>topMargin</td>
<td>Page top margin.</td>
</tr>
<tr>
<td>botMargin</td>
<td>Page bottom margin.</td>
</tr>
<tr>
<td>leftinsideMargin</td>
<td>Left margin (for single-sided documents) or the inside margin (for double-sided documents).</td>
</tr>
<tr>
<td>rightoutsideMargin</td>
<td>Right margin (for single-sided documents) or the outside margin (for double-sided documents).</td>
</tr>
<tr>
<td>sidedness</td>
<td>Constant that specifies whether the document is single-sided or double-sided and on which side the document starts. See the following table for the list of constants.</td>
</tr>
<tr>
<td>makeVisible</td>
<td>Specifies whether to make the document visible. True makes it visible.</td>
</tr>
</tbody>
</table>

The `sidedness` argument can have any of the values shown in the following table.
If successful, `F_ApiCustomDoc()` returns the ID of the document it created. Otherwise, it returns 0.

**Example**

The following code creates a custom document with the characteristics specified in the dialog box in Figure 4.1:

```c
#include "fmetrics.h"
#define in (MetricT)(65536*72) /* A Frame metric inch */

F_ObjHandleT docId;

docId = F_ApiCustomDoc(F_MetricFractMul(in,17,2), 11*in, 1,
                       F_MetricFractMul(in,1,4), in, in, in,
                       FF_Custom_SingleSided, True);
```

![Figure 4.1 Specifications for custom document](image)

**Creating a document with F_ApiOpen()**

To create a document with `F_ApiOpen()`, set the `FS_NewDoc` property in the `openParams` script to True. For the syntax of `F_ApiOpen()`, see “Opening a document or book with F_ApiOpen()” on page 48.

When you use `F_ApiOpen()` to create a document, set `fileName` to the name of the template that you want to use. You don’t provide the name for the new document until you save it.
For example, the following code creates a document from a template named /tmp/template and saves it as /tmp/mynew.doc.

```c
F_ObjHandleT docId;
F_PropValsT params, *returnParamsp = NULL;

params = F_ApiAllocatePropVals(1);
if(params.len == 0) return;

/* Set up the FS_NewDoc property and set it to True. */
params.val[0].propIdent.num = FS_NewDoc;
params.val[0].propVal.valType = FT_Integer;
params.val[0].propVal.u.ival = True;

docId = F_ApiOpen("/tmp/template", &params, &returnParamsp);

/* See “Saving documents and books” on page 59 for syntax. */
F_ApiSimpleSave(docId, "/tmp/mynew.doc", False);

/* Deallocate property lists. */
F_ApiDeallocatePropVals(&params);
F_ApiDeallocatePropVals(returnParamsp);
```

**IMPORTANT:** If you are creating a document with `F_ApiOpen()` and you display the New dialog box (by setting `FS_ShowBrowser` to True), the user may click Portrait, Custom, or Landscape. If this occurs, `F_ApiOpen()` does not create a new document. It returns 0 and sets `FA_errno` to `FE_WantsPortrait`, `FE_WantsCustom`, or `FE_WantsLandscape`. It is up to your client to create a portrait, custom, or landscape document by calling `F_ApiCustomDoc()`.

### Printing documents and books

To print a document or book, use `F_ApiSilentPrintDoc()`. `F_ApiSilentPrintDoc()` uses the default print settings for a document. The default print settings are the settings that appear in the Print dialog box when the user attempts to print the document in the user interface.

The syntax for `F_ApiSilentPrintDoc()` is:

```c
IntT F_ApiSilentPrintDoc(F_ObjHandleT docId);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document or book to print</td>
</tr>
</tbody>
</table>
When you call `F_ApiSilentPrintDoc()`, the FrameMaker product doesn’t notify the user about error or warning conditions that occur when it attempts to print. To determine whether an error occurred, check `FA_errno`.

**Changing the print settings for a document**

When you print a document in the user interface, you can change the print settings in the Print dialog box. FrameMaker saves most print settings with a document. For example, if you set the scale to 90 percent and print the document in the same session or save the document, the default setting for the scale will be 90 percent. Similarly, if an API client calls `F_ApiSilentPrintDoc()` to print the document, the scale will be 90 percent, if the client doesn’t change it.

The API represents a document’s print settings with a set of document properties. For example, a document’s `FP_PrintNumCopies` property specifies the number of copies of the document to print. To change a print setting programmatically, you change the property that represents it. For more information on changing properties, see Chapter 5, “Getting and Setting Properties”. For a list of document print properties, see “Document print properties” on page 561 of the *FDK Programmer’s Reference*.

**Examples**

The following code opens a document named `/tmp/my.doc` and prints it using the default print settings:

```c
F_ObjHandleT docId;
docId = F_ApiSimpleOpen("/tmp/my.doc", False);
F_ApiSilentPrintDoc(docId);
```

The following code opens `/tmp/my.doc` and modifies its default print settings so that the FrameMaker product will print two copies of the document to a printer named `ps2`. It does this by setting the document properties that specify the number of copies (`FP_PrintNumCopies`) and the printer (`FP_PrinterName`) to 2 and `ps2`, respectively:

```c
F_ObjHandleT docId;
/* Open the document. */
docId = F_ApiSimpleOpen("/tmp/my.doc", False);

/* Change my.doc’s print properties. */
F_ApiSetInt(FV_SessionId, docId, FP_PrintNumCopies, 2);
F_ApiSetString(FV_SessionId, docId, FP_PrinterName, "ps2");

F_ApiSilentPrintDoc(docId);
```
If you save `/tmp/my.doc` or attempt to print it within the same session, the default printer will be ps2 and the default number of copies will be 2 unless your client or the user changes the values of `FP_PrinterName` and `FP_PrintNumCopies`.

### Saving documents and books

To save a document or book, use one of the following functions:

- `F_ApiSimpleSave()` is an easy-to-use function for saving a document or book.
- `F_ApiSave()` allows you to script the process for saving a document or book.

### Saving a document or book with `F_ApiSimpleSave()`

The syntax for `F_ApiSimpleSave()` is:

```c
F_ObjHandleT F_ApiSimpleSave(F_ObjHandleT docId,
    StringT saveAsName,
    IntT interactive);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>docId</code></td>
<td>ID of the document or book to save.</td>
</tr>
<tr>
<td><code>saveAsName</code></td>
<td>Name of the pathname to save the document or book to. For information on how filenames and paths on different platforms are represented, see the FDK Platform Guide for that platform.</td>
</tr>
<tr>
<td><code>interactive</code></td>
<td>Specifies whether the FrameMaker product displays messages and warnings to the user (True to display messages and warnings).</td>
</tr>
</tbody>
</table>

If you set `interactive` to `False` and you specify the document or book’s current name, the FrameMaker product saves the document or book under its current name. If you specify another filename for `saveAsName`, the FrameMaker product saves the document or book to that filename.

If you set `interactive` to `True`, the FrameMaker product displays the Save dialog box and allows the user to choose a filename. The document or book’s current name appears as the default name.

If `F_ApiSimpleSave()` is successful, it returns the ID of the document it saved. If you save the document under its current name, the returned ID is the same ID you specified in the `docId` parameter. If you specify another filename for `saveAsName`, the returned ID is the ID of the new document. If `F_ApiSimpleSave()` can’t save the file, it returns `0`.

**Example**

The following code opens and then saves a document named `/tmp/my.doc`. After it has saved the document as `/tmp/my.doc`, it saves a copy of it as `mynew.doc`:


```c
#include "futils.h"
F_ObjHandleT mydocId, mynewdocId;
UCharT msg[256];

mydocId = F_ApiSimpleOpen("/tmp/my.doc", False);

/* Save my.doc as itself. */
F_ApiSimpleSave(mydocId, "/tmp/my.doc", False);

/* Save my.doc as mynew.doc. */
mynewdocId = F_ApiSimpleSave(mydocId, "/tmp/mynew.doc", False);

/* If the Save As was successful, display ID of mynew.doc. */
if (!mynewdocId)
   F_ApiAlert("Couldn’t save as mynew.doc.",
              FF_ALERT_CONTINUE_NOTE);
else {
   F_Sprintf(msg, "The ID of mynew.doc is 0x%x.", mynewdocId);
   F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);
}
```

Saving a document or book with F_ApiSave()

To save a document or book and specify responses to warnings and messages that the FrameMaker product issues, use the scriptable save function, F_ApiSave(). With F_ApiSave(), you can specify aspects of the Save operation, such as the file format (for example, MIF or Text Only) and the UNIX file permissions. You can specify all aspects, or you can specify some and allow the user to decide others. For example, you can specify that the FrameMaker product should save a document as Text Only, but allow the user to decide how to convert the document’s tables to text.

The syntax for F_ApiSave() is:

```c
F_ObjHandleT F_ApiSave(F_ObjHandleT docId,
                       StringT saveAsName,
                       F_PropValsT *saveParamsp,
                       F_PropValsT **saveReturnParamspp);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document or book to save.</td>
</tr>
<tr>
<td>saveAsName</td>
<td>The pathname to save the document or book to.</td>
</tr>
<tr>
<td>saveParamsp</td>
<td>A property list that tells the FrameMaker product how to save the file and how to respond to errors and other conditions. Use F_ApiGetSaveDefaultParams() or F_ApiAllocatePropVals() to create and allocate memory for this property list. To use the default list, specify NULL.</td>
</tr>
</tbody>
</table>
IMPORTANT: Always initialize the pointer to the property list that you specify for 
saveReturnParamsp to NULL before you call F_ApiSave().

If F_ApiSave() is successful, it returns the ID of the document or book it saved. If 
F_ApiSave() performs a Save operation, it returns the ID that you specified in the docId 
parameter. If F_ApiSave() performs a Save As operation, it returns the ID of the new 
document or book. If F_ApiSave() can’t save a file, it returns 0.

To call F_ApiSave(), do the following:

1. Initialize the pointer to the saveReturnParamsp property list to NULL.
2. Create a saveParamsp property list.
   You can get a default list by calling F_ApiGetSaveDefaultParams(), or you can 
   create a list from scratch.
3. Call F_ApiSave().
4. Check the Save status.
   Check the returned values in the saveReturnParamsp list for the name of the saved 
   file and other information about how the FrameMaker product saved the file.
5. Deallocate the saveParamsp and saveReturnParamsp property lists.

Steps 2, 4, and 5 are discussed in the following sections.

Creating a saveParamsp script with F_ApiGetSaveDefaultParams() 
The API provides a function named F_ApiGetSaveDefaultParams() that creates a 
default saveParamsp property list. If you are setting a number of properties, it is easiest to 
use F_ApiGetSaveDefaultParams() to get a default property list and then change 
individual properties as needed.

The syntax for F_ApiGetSaveDefaultParams() is:
F_PropValsT F_ApiGetSaveDefaultParams();

The following table lists some of the properties in the property list returned by 
F_ApiGetSaveDefaultParams(). The first value listed for each property is the default 
value returned by F_ApiGetSaveDefaultParams(). You can change the list to use the 
other listed values.

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>saveReturnParamsp</td>
<td>A property list that returns information about how the FrameMaker</td>
</tr>
<tr>
<td></td>
<td>product saved the file.</td>
</tr>
</tbody>
</table>

This argument Means
saveReturnParamsp A property list that returns information about how the FrameMaker product saved the file.
### Executing Commands with API Functions

For the complete property list returned by `F_ApiGetSaveDefaultParams()`, see "F_ApiGetSaveDefaultParams()" on page 152 of the *FDK Programmer’s Reference*.

For example, to get a default `saveParamsp` property list and modify it so that it instructs `F_ApiSave()` to save the active document as Text Only, use the following code:

```c
F_PropValsT params, *returnParamsp = NULL;
F_ObjHandleT mydocId;
IntT i;

/* Get the ID of the active document. */
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Meaning and possible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>FS_FileType</td>
<td>Specifies the type of file to save to</td>
</tr>
<tr>
<td>FV_SaveFmtBinary</td>
<td>save in Frame binary format for this version of FrameMaker.</td>
</tr>
<tr>
<td>FV_SaveFmtBinary60</td>
<td>save in binary format for FrameMaker 6.0</td>
</tr>
<tr>
<td>FV_SaveFmtInterchange</td>
<td>save as MIF.</td>
</tr>
<tr>
<td>FV_SaveFmtStationery</td>
<td>save in Stationery format (Macintosh only).</td>
</tr>
<tr>
<td>FV_SaveFmtViewOnly</td>
<td>save in View Only format.</td>
</tr>
<tr>
<td>FV_SaveFmtSgml</td>
<td>save in SGML format.</td>
</tr>
<tr>
<td>FV_SaveFmtText</td>
<td>save in Text Only format.</td>
</tr>
<tr>
<td>FV_SaveFmtXml</td>
<td>save in XML format.</td>
</tr>
<tr>
<td>FV_SaveFmtPdf</td>
<td>save as PostScript, and then invoke Acrobat Distiller a to create a PDF version of the document. This is the same as choosing PDF from the Format popup menu in the Save As dialog box.</td>
</tr>
</tbody>
</table>

*a. Note: This will only work if the Acrobat Distiller software is installed on the system.*

<table>
<thead>
<tr>
<th>FS_AlertUserAboutFailure</th>
<th>Specifies whether to notify the user if something unusual occurs while the file is being saved</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>don’t notify user</td>
</tr>
<tr>
<td>True</td>
<td>notify user</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FS_SaveMode</th>
<th>Specifies whether to use Save or Save As mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>FV_ModeSaveAs</td>
<td>use Save As mode</td>
</tr>
<tr>
<td>FV_ModeSave</td>
<td>use Save mode</td>
</tr>
</tbody>
</table>
mydocId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

/* Get default property list. */
params = F_ApiGetSaveDefaultParams();
/* Get index of FS_FileType property and set it to Text Only. */
i = F_ApiGetPropIndex(&params, FS_FileType);
params.val[i].propVal.u.ival = FV_SaveFmtText;

/* Save to text only file named my.txt. */
F_ApiSave(mydocId, "/tmp/my.txt", &params, &returnParamsp);

/* Deallocate property lists. */
F_ApiDeallocatePropVals(&params);
F_ApiDeallocatePropVals(returnParamsp);

The API allocates memory for the property list created by
F_ApiGetSaveDefaultParams(). Use F_ApiDeallocatePropVals() to free the
property list when you are done with it.

Creating a saveParamsp script from scratch

If you want to specify only a few properties when you call F_ApiSave(), it is most efficient
to create a property list from scratch. To create the property list, you must allocate memory for
it, and then set up the individual properties.

Use the API convenience function, F_ApiAllocatePropVals(), to allocate memory for
the property list. For example, the following code creates a saveParamsp property list that
instructs F_ApiSave() to save a file as text only:

F_PropValsT params, *returnParamsp = NULL;
F_ObjHandleT mydocId;

/* Get the ID of the active document. */
mydocId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

/* Allocate memory for the list. */
params = F_ApiAllocatePropVals(1);

/* Set up the FS_FileType property and set it to True. */
params.val[0].propIdent.num = FS_FileType;
params.val[0].propVal.valType = FT_Integer;
params.val[0].propVal.u.ival = FV_SaveFmtText;

F_ApiSave(mydocId, "/tmp/my.txt", &params, &returnParamsp);

Checking Save status

F_ApiSave() stores a pointer to a property list in returnParamsp. This property list
provides information on how the FrameMaker product saved the file. It includes the properties
shown in the following table.
The `FS_SaveNativeError` property and the `FA_errno` value indicate the result of the call to `F_ApiSave()`. The `FS_SaveStatus` flags indicate how or why this result occurred.

To determine if a particular `FS_SaveStatus` bit is set, use `F_ApiCheckStatus()`. For example, the following code determines if a Save operation was canceled because the current FrameMaker document was already open:

```c
F_PropValsT params, *returnParamsp = NULL;
F_ObjHandleT mydocId;

/* Get the ID of the active document. */
mydocId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

params = F_ApiGetSaveDefaultParams();

F_ApiSave(mydocId, "/tmp/my.doc", &params, &returnParamsp);

if (F_ApiCheckStatus(returnParamsp, FV_FileWasInUse))
    F_ApiAlert("Save cancelled - file is in use", FF_ALERT_CONTINUE_NOTE);

/* Deallocate property lists. */
F_ApiDeallocatePropVals(&params);
F_ApiDeallocatePropVals(returnParamsp);
```

The API provides a utility function named `F_ApiPrintSaveStatus()`, which prints the save error values to the console on UNIX platforms, the FrameMaker console on Windows, and the error log window on the Macintosh. For more information, see “F_ApiPrintSaveStatus()” on page 253 of the FDK Programmer’s Reference.

**Example**

The following code opens `/tmp/my.doc` and saves it as a View Only document named `/tmp/viewonly.doc`. It gets the name of the saved file from the returned property list and displays it.

```
Property | Meaning and Possible Values
---|---
FS_SavedFileName | A string that specifies the saved file’s pathname.
FS_SaveNativeError | The error condition. If the file is saved successfully, it is set to `FE_Success`. For a complete list of the other values it can be set to, see “F_ApiSave()” on page 272 of the FDK Programmer’s Reference.
FS_SaveStatus | A bit field indicating what happened when the file was saved. For a complete list of the possible status flags, see “F_ApiSave()” on page 272 of the FDK Programmer’s Reference.
```
Executing Commands with API Functions

```c
#include "futils.h"

IntT i;
UCharT msg[1024];
F_PropValsT params, *returnParamsp = NULL;
F_ObjHandleT mydocId, viewonlydocId;

params = F_ApiAllocatePropVals(1);

mydocId = F_ApiSimpleOpen("/tmp/my.doc", False);
if(!mydocId) return;

/* Set file type to View Only. */
params.val[0].propIdent.num = FS_FileType;
params.val[0].propVal.valType = FT_Integer;
params.val[0].propVal.u.ival = FV_SaveFmtViewOnly;

/* Save document as viewonly.doc. */
viewonlydocId = F_ApiSave(mydocId,="/tmp/viewonly.doc",
&params, &returnParamsp);

/* Get index of property specifying filename and display it. */
i = F_ApiGetPropIndex(returnParamsp, FS_SavedFileName);
F_Sprintf(msg, "Saved: %s",
returnParamsp->val[i].propVal.u.sval);
F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);

/* Deallocate Save scripts. */
F_ApiDeallocatePropVals(&params);
F_ApiDeallocatePropVals(returnParamsp);
```

Closing documents and books

To close a document or book, use `F_ApiClose()`.

The syntax for `F_ApiClose()` is:

```c
IntT F_ApiClose(F_ObjHandleT Id,
    IntT flags);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>The ID of the document, book, or session to close. To close the session, specify <code>FV_SessionId</code>.</td>
</tr>
<tr>
<td>flags</td>
<td>Specifies whether to abort or to close open documents or books if they have unsaved changes. Set the <code>FF_CLOSE_MODIFIED</code> flag to close open documents and books regardless of their state.</td>
</tr>
</tbody>
</table>
F_ApiClose() behaves somewhat differently than the Close command in the user interface. If there are unsaved changes in a file and you set FF_CLOSE_MODIFIED for the flags argument, F_ApiClose() abandons the changes and closes the file anyway. If you set flags to 0, F_ApiClose() aborts the Close operation and returns FE_DocModified.

**IMPORTANT:** If you are closing an individual document, make sure Id specifies a valid document ID and not 0. If Id is set to 0, F_ApiClose() quits the Frame session (because FV_SessionId is defined as 0).

### Examples

The following code closes the active document. If the document has unsaved changes, the client prompts the user.

```c
F_ObjHandleT docId;
IntT resp = 0;

/* Get the ID of active document. Return if there isn’t one. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
if(!docId) return;

/* See if document has been modified. */
if (F_ApiGetInt(FV_SessionId, docId, FP_DocIsModified))
    resp = F_ApiAlert("Document was changed, close it anyway?", FF_ALERT_OK_DEFAULT);
if (!resp) F_ApiClose(docId, FF_CLOSE_MODIFIED);
```

The following code closes the active document unless it has unsaved changes:

```c
F_ObjHandleT docId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
if(!docId) return;
if (F_ApiClose(docId, 0))
    F_ApiAlert("Unsaved changes. Can’t close.", FF_ALERT_CONTINUE_WARN);
```

### Quitting a Frame session

To quit a Frame session, call F_ApiClose(), with Id set to FV_SessionId. For the syntax of F_ApiClose(), see “Closing documents and books” on page 66.

For example, to close all the open documents and books in the current Frame session, and quit the session, use the following code:

```c
F_ApiClose(FV_SessionId, 0);
```
Because of the flag set to FF_CLOSE_MODIFIED, if any books or documents have been changed, the FrameMaker product abandons the changes.

Comparing documents and books

To compare two versions of a document or book using the FrameMaker product’s built-in comparison feature, use F_ApiCompare().

The syntax for F_ApiCompare() is:

```c
F_CompareRetT F_ApiCompare(F_ObjHandleT olderId,
                          F_ObjHandleT newerId,
                          IntT flags,
                          StringT insertCondTag,
                          StringT deleteCondTag,
                          StringT replaceText,
                          IntT compareThreshold);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>olderId</td>
<td>The ID of the older version of the document or book.</td>
</tr>
<tr>
<td>newerId</td>
<td>The ID of the newer version of the document or book.</td>
</tr>
<tr>
<td>flags</td>
<td>Bit flags that specify how to generate the summary and composite documents.</td>
</tr>
<tr>
<td>insertCondTag</td>
<td>The condition tag to apply to insertions shown in the composite document. For no insert condition tag, specify NULL.</td>
</tr>
<tr>
<td>deleteCondTag</td>
<td>The condition tag to apply to deletions shown in the composite document. For no delete condition tag, specify NULL.</td>
</tr>
<tr>
<td>replaceText</td>
<td>Text to appear in place of the deleted text. For no replacement text, specify NULL.</td>
</tr>
<tr>
<td>compareThreshold</td>
<td>If zero, the compare function uses the default value of 75. If not zero, the function uses the setting in the FrameMaker configuration files. Threshold expresses the percentage of words that can change before paragraphs are considered not equal. If two paragraphs are equal, word differences between them are shown within a paragraph in the composite document.</td>
</tr>
</tbody>
</table>

The F_CompareRetT structure is defined as:
typedef struct {
    F_ObjHandleT sumId; /* ID of the summary document */
    F_ObjHandleT compId; /* ID of the composite document */
} F_CompareRetT;

The following values can be ORed into the flags argument.

<table>
<thead>
<tr>
<th>This value</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF_CMP_SUMMARY_ONLY</td>
<td>Generate summary document, but not composite document</td>
</tr>
<tr>
<td>FF_CMP_CHANGE_BARS</td>
<td>Turn on change bars in the composite document</td>
</tr>
<tr>
<td>FF_CMP_HYPERLINKS</td>
<td>Put hypertext links in the summary document</td>
</tr>
<tr>
<td>FF_CMP_SUMKIT</td>
<td>Open the summary document</td>
</tr>
<tr>
<td>FF_CMP_COMPKIT</td>
<td>Open the composite document</td>
</tr>
</tbody>
</table>

If you specify the FF_CMP_SUMKIT or FF_CMP_COMPKIT flags, F_ApiCompare() opens the summary and comparison documents and returns their IDs in the F_CompareRetT structure. It does not make these documents visible to the user. If you want them to be visible, you must set each of the document’s FP_DocIsOnScreen properties to True.

Example

The following code opens two documents and compares them as specified in the dialog boxes shown in Figure 4.2. It makes the summary document visible.

```c
F_ObjHandleT oldId, newId;
IntT flags;
F_CompareRetT cmp;

oldId = F_ApiSimpleOpen("/tmp/1Chapter", False);
newId = F_ApiSimpleOpen("/tmp/1Chapter.new", False);

flags = FF_CMP_CHANGE_BARS | FF_CMP_COMPKIT | FF_CMP_SUMKIT;

cmp = F_ApiCompare(oldId, newId, flags, "Comment",
                    ", " , "Replaced Text", 75);

if (FA_errno != FE_Success)
    F_ApiAlert("Couldn’t compare", FF_ALERT_CONTINUE_NOTE);
```

...
Updating and generating documents and books

The API provides a variety of functions that allow you to generate, update, and reformat documents and books. The following sections discuss these functions.

Updating the files in a book

To update the numbering, text insets, cross-references, etc. in all the files in a book, and to programmatically specify responses to warnings and messages that the FrameMaker product issues, use `F_ApiUpdateBook()`. You can specify all aspects of the operation, or you can specify some aspects and allow the user to decide others. For example, you can instruct the FrameMaker product to update view-only files, or to abort the update when it encounters a view-only file.

To use `F_ApiUpdateBook()`, you should first understand property lists and how to manipulate them directly. For more information on this subject, see “Representing object characteristics with properties” on page 270 and “Manipulating property lists directly” on page 93.

The syntax for `F_ApiUpdateBook()` is:

```c
ErrorT F_ApiOpen(F_ObjHandleT bookId,
    F_PropValsT *updateParamsp,
    F_PropValsT **updateReturnParamspp);
```
Executing Commands with API Functions

**IMPORTANT:** Always initialize the pointer to the property list that you specify for `openReturnParamspp` to NULL before you call `F_ApiUpdateBook()`.

If `F_ApiUpdateBook()` is successful, it returns `FE_Success`. Otherwise, it returns an error which has the same value as `FA_errno`.

To call `F_ApiUpdateBook()`, do the following:

1. Initialize the pointer to the `updateReturnParamspp` property list to NULL.
2. Create an `updateParamsp` property list.
   - You can get a default list by calling `F_ApiGetUpdateBookDefaultParams()`, or you can create a list from scratch.
3. Call `F_ApiUpdateBook()`.
4. Check the Update status.
   - Check the returned values in the `updateReturnParamspp` list for the name of the opened file and other information about how the FrameMaker product opened the file.
5. Deallocate memory for the `updateParamsp` and `updateReturnParamspp` property lists.
   - Use `F_ApiDeallocatePropVals()` to deallocate memory for the lists.

### Generating files for a book

To generate and update files for a book, use `F_ApiSimpleGenerate()`. The book and its generated files must be set up before you call `F_ApiSimpleGenerate()`.

The syntax for `F_ApiSimpleGenerate()` is:

```c
IntT F_ApiSimpleGenerate(F_ObjHandleT bookId,
                         IntT interactive,
                         IntT makeVisible);
```
Executing Commands with API Functions

Importing formats

To import formats from a document to all the documents in a book or from one document to another document, use `F_ApiSimpleImportFormats()`. The syntax for `F_ApiSimpleImportFormats()` is:

```c
IntT F_ApiSimpleImportFormats(F_ObjHandleT bookId, 
   F_ObjHandleT fromDocId, 
   IntT formatFlags);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>bookId</td>
<td>The ID of the book for which to generate files</td>
</tr>
<tr>
<td>interactive</td>
<td>Specifies whether to display warnings and messages to the user (True displays messages and warnings)</td>
</tr>
<tr>
<td>makeVisible</td>
<td>Specifies whether to display generated files (True displays the files)</td>
</tr>
</tbody>
</table>

You can OR the values in the following table into the `formatFlags` parameter to specify which formats to import.

<table>
<thead>
<tr>
<th>This value</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF_UFF_COLOR</td>
<td>Import colors</td>
</tr>
<tr>
<td>FF_UFF_COMBINED_FONTS</td>
<td>Import combined font definitions</td>
</tr>
<tr>
<td>FF_UFF_COND</td>
<td>Import conditional text settings</td>
</tr>
<tr>
<td>FF_UFF_DOCUMENT_PROPS</td>
<td>Import document properties</td>
</tr>
<tr>
<td>FF_UFF_FONT</td>
<td>Import Character Catalog formats</td>
</tr>
<tr>
<td>FF_UFF_MATH</td>
<td>Import equation settings</td>
</tr>
<tr>
<td>FF_UFF_PAGE</td>
<td>Import page layouts</td>
</tr>
<tr>
<td>FF_UFF_PGF</td>
<td>Import Paragraph Catalog formats</td>
</tr>
<tr>
<td>FF_UFF_REFPAGE</td>
<td>Import reference pages</td>
</tr>
<tr>
<td>FF_UFF_TABLE</td>
<td>Import Table Catalog formats</td>
</tr>
</tbody>
</table>
Executing other updating and formatting commands

The API provides several functions that allow you to execute FrameMaker product commands that update and reformat entire documents.

The syntax for the functions is:

- `IntT F_ApiClearAllChangebars(F_ObjHandleT docId);`
- `IntT F_ApiRehyphenate(F_ObjHandleT docId);`
- `IntT F_ApiResetReferenceFrames(F_ObjHandleT docId);`
- `IntT F_ApiResetEqnSettings(F_ObjHandleT docId);`
- `IntT F_ApiRestartPgfNumbering(F_ObjHandleT docId);`
- `IntT F_ApiUpdateVariables(F_ObjHandleT docId);`
- `IntT F_ApiUpdateXRefs(F_ObjHandleT docId, IntT updateXRefFlags);`

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>ID of the document to update or reformat</td>
</tr>
</tbody>
</table>

These functions behave like the corresponding commands in the user interface. They are useful for clients that need to update and reformat multiple files. For more information on a particular function, look it up in *FDK Programmer's Reference*

### Example

The following code opens a book and resets the change bars in each of its component documents:

```c
...  
#include "fmemory.h"

F_ObjHandleT bookId, compId, docId;
StringT compName;

bookId = F_ApiSimpleOpen("/tmp/my.book", False);
compId = F_ApiGetId(FV_SessionId, bookId,  
                      FF_FirstComponentInBook);

/* Traverse book’s components, opening each one  
 * and clearing its change bars. */
```
while(compId)
{
    compName = F_ApiGetString(bookId, compId, FP_Name);
    docId = F_ApiSimpleOpen(compName, False);
    F_Free(compName);
    F_ApiClearAllChangebars(docId);
    compId = F_ApiGetId(bookId, compId, FP_NextComponentInBook);
}

Simulating user input

To simulate user input, call the API function F_ApiFcodes().

F_ApiFcodes() sends an array of function codes (f-codes) to the FrameMaker product. F-codes are hexadecimal codes that specify individual user actions, such as cursor movement and text entry. They are especially useful for manipulating windows. For example, the f-code KBD_EXPOSEWIN brings the active document or book window to the front. When you use F_ApiFcodes() to send an array of f-codes to FrameMaker, it executes each f-code as if the user performed the action.

IMPORTANT: F_ApiFcodes() does not work with dialog boxes on Windows.

The syntax for F_ApiFcodes() is:

```c
IntT F_ApiFcodes(IntT len,
                 IntT *vec);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>len</td>
<td>The length of the array of f-codes in bytes</td>
</tr>
<tr>
<td>vec</td>
<td>The array of f-codes to send to the FrameMaker product</td>
</tr>
</tbody>
</table>

The following table lists some user actions and the f-codes that emulate them.

<table>
<thead>
<tr>
<th>User action</th>
<th>F-code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move insertion point to first flow on the current page</td>
<td>CSR_TOP</td>
</tr>
<tr>
<td>Move insertion point to the start of the next word</td>
<td>CSR_NEXT_BOW</td>
</tr>
<tr>
<td>Move insertion point to the beginning of the next paragraph</td>
<td>CSR_NEXT_BOP</td>
</tr>
<tr>
<td>Center the current paragraph</td>
<td>PGF_CENTER</td>
</tr>
<tr>
<td>Left justify the current paragraph</td>
<td>PGF_LEFT</td>
</tr>
<tr>
<td>Make selected text bold</td>
<td>TXT_BOLD</td>
</tr>
</tbody>
</table>

For a complete list of f-codes, see the `fcodes.h` file shipped with the FDK.
F_ApiFcodes() uses the current focus in a dialog box or a visible document. If you want to execute a set of f-codes in a particular dialog box or document, make sure that the dialog box or document is active. To make a dialog box active, use f-codes such as FOCUS_INPUT_SEARCH and FOCUS_INPUT_PGFFMT. To make a document active, set the session property FP_ActiveDoc to the document’s ID.

Many f-codes perform tasks that API functions also perform. Whenever possible, try to use the other API functions instead of F_ApiFcodes() to perform these tasks. F_ApiFcodes() does not provide error or status feedback for individual f-codes, whereas each API function stores an error code to FA_errno when it fails. It is also difficult to debug lengthy f-code sequences.

The following code uses f-codes to enter the string HI!, select the text, and then make it bold:

```c
static IntT fcodes[] = {CSR_TOP,'H','I','!',HIGH_WORD_PREV, TXT_BOLD};
F_ApiFcodes(sizeof(fcodes)/sizeof(IntT), fcodes);
```

### Straddling table cells

To straddle and unstraddle table cells, use F_ApiStraddleCells() and F_ApiUnStraddleCells().

The syntax for these functions is:

```c
IntT F_ApiStraddleCells(F_ObjHandleT docId,
                        F_ObjHandleT cellId,
                        IntT heightInRows,
                        IntT widthInCols);
IntT F_ApiUnStraddleCells(F_ObjHandleT docId,
                           F_ObjHandleT cellId,
                           IntT heightInRows,
                           IntT widthInCols);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document containing the table</td>
</tr>
<tr>
<td>cellId</td>
<td>The ID of the first (leftmost and uppermost) cell to straddle or unstraddle</td>
</tr>
<tr>
<td>heightInRows</td>
<td>The number of cells to straddle or unstraddle vertically</td>
</tr>
<tr>
<td>widthInCols</td>
<td>The number of cells to straddle or unstraddle horizontally</td>
</tr>
</tbody>
</table>

Both heightInRows and widthInCols must be greater than 0. At least one of them must be greater than 1. The cells you straddle must all be from the same type of row. You can’t, for example, straddle a set of cells that are in both heading and body rows. You also can’t straddle cells that are already straddled. If the cells you specify include cells that are already straddled, F_ApiStraddleCells() returns FE_BadOperation.
When you or the user straddle table cells, the FrameMaker product does not delete any of the FO_Cell objects that represent the cells. It links the paragraphs from the straddled cells into a single list. The FP_FirstPgf and FP_LastPgf properties of each cell in the straddle specify the first and last paragraphs in this list.

Example

The following code straddles the first two cells in the first column of a table:

```c
F_ObjHandleT docId, tableId, firstrowId, cellId;

/* Get IDs of document, table, first row, and first cell. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tableId = F_ApiGetId(FV_SessionId, docId, FP_FirstTblInDoc);
firstrowId = F_ApiGetId(docId, tableId, FP_FirstRowInTbl);
cellId = F_ApiGetId(docId, firstrowId, FP_FirstCellInRow);

/* If there are two cells in the row, straddle them. */
if (F_ApiGetInt(docId, tableId, FP_TblNumCols) < 2)
    F_ApiAlert("Not enough columns!", FF_ALERT_CONTINUE_NOTE);
else F_ApiStraddleCells(docId, cellId, 1, 2);
```

Executing Structured FrameMaker commands

The following sections describe how to programmatically execute FrameMaker commands.

Manipulating elements

The API provides several functions that allow you to execute basic FrameMaker commands that manipulate elements.

The syntax for these functions is:

- VoidT F_ApiDemoteElement(F_ObjHandleT docId);
- VoidT F_ApiMergeIntoFirst(F_ObjHandleT docId);
- VoidT F_ApiMergeIntoLast(F_ObjHandleT docId);
- VoidT F_ApiPromoteElement(F_ObjHandleT docId);
- VoidT F_ApiSplitElement(F_ObjHandleT docId);
- VoidT F_ApiUnWrapElement(F_ObjHandleT docId);
- VoidT F_ApiWrapElement(F_ObjHandleT docId, F_ObjHandleT edefId);
Executing Commands with API Functions

These functions behave like the corresponding commands in the user interface. They all use the current text or element selection in the specified document to determine which text and elements to manipulate. You can allow the user to set the text or element selection, or you can do it programmatically. For information on setting the text selection programmatically, see “Getting and setting the insertion point or text selection” on page 114. For more information on setting the element selection programmatically, see “Getting and setting the structural element selection” on page 120.

Importing element definitions into FrameMaker documents and books

To import element definitions from a FrameMaker document or book to a FrameMaker document or book, use F_ApiSimpleImportElementDefs().

The syntax for F_ApiSimpleImportElementDefs() is:

```c
IntT F_ApiSimpleImportElementDefs(
    F_ObjHandleT docOrBookId,
    F_ObjHandleT fromDocOrBookId,
    IntT importFlags);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docOrBookId</td>
<td>The ID of the document or book to import element definitions to.</td>
</tr>
<tr>
<td>fromDocOrBookId</td>
<td>The ID of the document or book from which to import element definitions.</td>
</tr>
<tr>
<td>importFlags</td>
<td>See the following table for the flags that you can OR into this parameter.</td>
</tr>
</tbody>
</table>

The following table lists flags that you can OR into the importFlags parameter:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF_IED_REMOVE_OVERRIDES</td>
<td>Clear format overrides.</td>
</tr>
<tr>
<td>FF_IED_REMOVE_BOOK_INFO</td>
<td>If docOrBookId specifies a document, clear formatting inherited from the parent book.</td>
</tr>
<tr>
<td>FF_IED_DO_NOT_IMPORT_EDD</td>
<td>If the document specified by fromDocOrBookId is an EDD, don’t treat it as an EDD; just import its element catalog.</td>
</tr>
</tbody>
</table>
Executing Commands with API Functions

If you import element definitions to a book, `F_ApiSimpleImportElementDefs()` imports element definitions to each book component for which the `FP_ImportFmtInclude` property is set to `True`.

Calling FrameMaker clients programmatically

Much of the structured document functionality FrameMaker provides is implemented in FDK clients. To call this functionality programmatically, you must use `F_ApiCallClient()`.

`F_ApiCallClient()` requires you to specify a client’s registered name and a string, which it passes to the client. The following table lists FrameMaker functionality and the registered names of the clients you can call to invoke it programmatically.

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Registered client name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element catalog manager</td>
<td>Element Catalog Manager</td>
</tr>
<tr>
<td>Structure generator</td>
<td>Structure Generator</td>
</tr>
<tr>
<td>Reading and writing SGML documents and reading, writing, and updating DTD and EDD documents</td>
<td>FmDispatcher</td>
</tr>
</tbody>
</table>

The following table lists the strings you pass to the structure generator client to programmatically generate structure in a document or book.

<table>
<thead>
<tr>
<th>String</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>INPUTDOCID</code> <code>objectID</code></td>
<td>The ID of the input document or book.</td>
</tr>
<tr>
<td><code>RULEDOCID</code> <code>objectID</code></td>
<td>The ID of the rule table document.</td>
</tr>
<tr>
<td><code>OUTPUTDOCNAME</code> <code>filename</code></td>
<td>The full pathname of the output document or book. This string is optional. If you do not specify a pathname, the structure generator leaves the document unsaved and open.</td>
</tr>
<tr>
<td><code>LOGNAME</code> <code>filename</code></td>
<td>The full pathname of a log file. This string is optional. If you do not specify a pathname, the structure generator leaves the log file unsaved and open.</td>
</tr>
<tr>
<td>GenerateDoc</td>
<td>Instructs the structure generator to generate structure, using the strings listed above.</td>
</tr>
</tbody>
</table>

To programmatically generate structure for a document or a book, you call `F_ApiCallClient()` multiple times, each time passing it one of the strings listed in the table above. For example, the following code generates structure for a document:
...  
F_ObjHandleT inputDocId, ruleTblDocId;  
UCharT buf[64];  
...  
F_Sprintf(buf, "INPUTDOCID %d", inputDocId);  
F_ApiCallClient("StructGen", buf);  

F_Sprintf(buf, "RULEDOCID %d", ruleTblDocId);  
F_ApiCallClient("StructGen", buf);  

F_ApiCallClient("StructGen", "OUTPUTDOCNAME /tmp/mystruct.doc");  
F_ApiCallClient("StructGen", "LOGNAME /tmp/logfile.doc");  
F_ApiCallClient("StructGen", "GenerateDoc");  
...

Note that all of the documents you specify must be open before you call the structure generator. If you are generating structure for a large number of documents, you can greatly speed processing by opening the documents invisibly. To open a document invisibly, set the FS_MakeVisible property of the Open script to False.

For a complete list of the strings you can pass to the structure generator and other FrameMaker clients, see “Calling Clients Shipped With FrameMaker” on page 699 of the FDK Programmer’s Reference.
This chapter describes how to make changes in a FrameMaker product session, book, or document by getting and setting property values. It discusses how to get and set individual properties and entire property lists. It also provides some tips for getting and setting the properties of specific types of objects.

**What you can do with object properties**

In the FrameMaker product user interface, the user can change an object in a variety of ways. For example, the user can change the size and fill pattern of a graphic object or the starting page number of a book component.

Each API object has a *property list*, a set of properties describing its attributes. Your API client can do anything a user can do to an object by getting and setting the properties in the object’s property list. For example, your client can set properties to:

- Change a graphic object’s size, fill pattern, or position in the back-to-front order
- Make a document or book active
- Change a book component’s position in a book
- Change a paragraph’s format

Your client can also change properties that the user doesn’t have access to. For example, your client can set properties to:

- Make a document or book visible or invisible
- Keep the FrameMaker product from reformatting a document every time a change is made

The API ensures that your client doesn’t corrupt a document by setting properties to illegal values. When you change a property, the API also automatically changes other properties as needed to preserve the integrity of the document or book.

There are a number of read-only properties that you can get but not set. For a complete list of object properties and their possible values, see Chapter 3, “Object Reference,” in the *FDK Programmer’s Reference*.

To change a session, document, or book by setting object properties, follow these general steps:

1. Find out which objects represent the things you want to change.

   To change something in a session, book, or document, you need to know which objects the API uses to represent it. For a description of how the API uses objects to represent things in FrameMaker, see Part II, *Frame Product Architecture*. 
2. Get the IDs of the objects you want to change.

   To set an object’s properties, you must specify its ID. The API provides functions for retrieving object IDs.

3. Manipulate the objects’ properties.

The API provides functions for getting and setting individual properties and entire property lists.

For example, the API represents a FrameMaker product session with an `FO_Session` object. You don’t need to get a session’s ID, because there is only one session and its ID is always `FV_SessionId`. To find all the session characteristics you can change, look up “Session” on page 631 in Chapter 3, “Object Reference,” in the FDK Programmer’s Reference.

You can, for example, change the session’s automatic save time. The API represents the automatic save time with an integer (`IntT`) property named `FP_AutoSaveSeconds`. To set it to 60 seconds, use the following code:

```c
F_ApiSetInt(0, /* Sessions have no parent */
    FV_SessionId, /* The session’s ID */
    FP_AutoSaveSeconds, /* The property to set */
    60); /* The value to set it to */
```

The following sections describe steps 2 and 3 in greater detail.

---

### Getting the IDs of the objects you want to change

Every object in a session has an ID. To get or set the properties of a particular object, you must specify its ID. In Frame book and document architecture, objects are organized in linked lists: an object has properties that specify the IDs of other objects, which have properties that specify the IDs of other objects, and so on. To get the IDs of specific objects, you traverse the linked lists by querying these properties. For diagrams and descriptions of the linked lists in Frame architecture, see Part II, “Frame Product Architecture.”

To query a property that specifies an object ID, use `F_ApiGetId()`, which is defined as:

```c
F_ObjHandleT F_ApiGetId(F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>docId</code></td>
<td>The ID of the document, book, or session containing the object whose property you want to query.</td>
</tr>
<tr>
<td><code>objId</code></td>
<td>The ID of the object whose property you want to query.</td>
</tr>
<tr>
<td><code>propNum</code></td>
<td>The property to query. Specify one of the API-defined constants, such as <code>FP_ActiveDoc</code>.</td>
</tr>
</tbody>
</table>
F_ApiGetId() returns the ID specified by the property. If the property doesn’t specify an ID or an error occurs, F_ApiGetId() returns 0.

To get an object’s ID, you start traversing at the object that represents the session (the FO_Session object), because it is the only object whose ID (FV_SessionId) you know from the start.

From the FO_Session object, you can get the IDs of the active and open documents and books in the session. FO_Session objects have properties, named FP_ActiveDoc and FP_ActiveBook, that specify the IDs of the active document or book. A document or a book is active if it has input focus.

FO_Session objects also have properties, named FP_FirstOpenDoc and FP_FirstOpenBook, that specify the first document and the first book in the linked lists of open documents and books in a session. FO_Doc objects have a property named FP_NextOpenDocInSession that specifies the ID of the next FO_Doc object in the list of open documents. FO_Book objects have a property named FP_NextOpenBookInSession that specifies the ID of the next FO_Book object in the list of open books. If an FO_Doc or an FO_Book object is the last object in the list, its FP_NextOpenDocInSession or FP_NextOpenBookInSession property is set to 0. For a diagram of how the API represents the documents and books in a session, see Figure 13.2 on page 276.

Suppose you want to display the IDs of the active document and all the open documents in a session. You can use the following code to do this:

```c
#include "futils.h"
F_ObjHandleT docId;
UCharT msg[256];

/* Get the ID of the active document and display it. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
if (docId)
{
    F_Sprintf(msg, "The active document’s ID is 0x%x.", docId);
    F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);
}

/* Get ID of the first document in list of open documents. */
docId = F_ApiGetId(0, FV_SessionId, FP_FirstOpenDoc);

/* Traverse list of open documents and display their IDs. */
while (docId)
{
    F_Sprintf(msg, "The document’s ID is 0x%x.", docId);
    F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);
    docId = F_ApiGetId(FV_SessionId, docId,
                       FP_NextOpenDocInSession);
}
...
This code displays the ID of the active document twice, because the active document is included in the list of open documents.

The linked list of open documents in a session isn’t in any particular order. The first document in the list is not necessarily the active document or the first document that was opened.

Another way to get a document ID is to use `F_ApiSimpleOpen()`, `F_ApiOpen()`, or `F_ApiSimpleNewDoc()` to open or create the document. These functions all return the IDs of the document they open or create.

**Traversing lists of objects in a document**

Once you have the ID of a document, you can query its properties to get to the lists of objects that it contains. The document has a number of properties that point to these lists. For example, the document’s `FP_FirstGraphicInDoc` property specifies the ID of the first graphic object in the list of its graphic objects and its `FP_FirstBodyPageInDoc` property specifies the first body page in the list of its body pages. Except for the lists of pages, the lists are completely unordered. For example, the first graphic object in the list of graphic objects is not necessarily the first graphic that appears in the document.

Suppose you want to traverse the list of all the paragraphs in the active document. To do this, you can use the following code:

```c
#include "futils.h"
F_ObjHandleT docId, pgfId;
UCharT msg[256];

/* Get the ID of the active document. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

/* Get ID of the first paragraph in the list of paragraphs. */
pgfId = F_ApiGetId(FV_SessionId, docId, FP_FirstPgfInDoc);

/* Traverse the list of paragraphs and display their IDs. */
** Even empty documents have several paragraphs, because text columns on master pages contain paragraphs. */
while (pgfId)
{
    F_Sprintf(msg, "The paragraph’s ID is 0x%x.", pgfId);
    F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);
    pgfId = F_ApiGetId(docId, pgfId, FP_NextPgfInDoc);
}
```

The paragraphs in the list are not ordered.

**Traversing lists of graphic objects**

The API does not maintain separate lists of the different types of graphic objects in a document. For example, a document’s text columns (FO_TextFrame objects), rectangles
(FO_Rectangle objects), and anchored frames (FO_AFrame objects) are all in the same list. To determine objects’ types as you traverse them, use F_ApiGetObjectType().

The syntax for F_ApiGetObjectType() is:

```
UIntT F_ApiGetObjectType(F_ObjHandleT docId,
                         F_ObjHandleT objId);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document, book, or session containing the object</td>
</tr>
<tr>
<td>objId</td>
<td>The ID of the object whose type you want to get</td>
</tr>
</tbody>
</table>

For example, the following code counts the number of anchored frames in the active document:

```
...  
#include "futils.h"
IntT numFrames = 0;
F_ObjHandleT docId, objId;
UCharT msg[256];

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
/* Get ID of first graphic in the list of graphics. */
objId = F_ApiGetId(FV_SessionId, docId, FP_FirstGraphicInDoc);
/* Traverse list of graphics, counting anchored frames. */
while (objId)
{
    if (F_ApiGetObjectType(docId,objId) == FO_AFrame) numFrames++;
    objId = F_ApiGetId(docId, objId, FP_NextGraphicInDoc);
}
F_Sprintf(msg, "The document has %d anchored frames.",
          numFrames);
F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);
...  
```

**Traversing ordered lists of objects**

Traversing the list of all the objects of a certain type in a document is useful if you want to get every object of that type and the order doesn’t matter to you. However, it isn’t very useful if you want the objects in some kind of order, such as the order in which they appear on a document’s pages. To get objects in order, you must traverse the ordered lists that the API maintains. There are ordered lists of the graphic objects in a frame, the text columns within a flow, and many other objects. These lists can be deeply nested, for example, when a frame contains a frame that contains some graphic objects.

There are a variety of object properties you can query to get to ordered lists. For example, to get to the list of graphic objects in a frame, you can query the frame’s FP_FirstGraphicInFrame or FP_LastGraphicInFrame properties. If you already
have one of the graphic object’s IDs, you can query its FP_PrevGraphicInFrame and FP_NextGraphicInFrame properties to get to the objects behind it and in front of it in the list. The order of the list corresponds to the back-to-front order of the graphics in the frame. For information on the linked lists that a particular object is included in, see the section that discusses that object in Chapter 14, “Frame Document Architecture”.

Although there are ordered lists of the paragraphs within each of a document’s flows, there is no ordered list of flows. You can get the paragraphs only in the order in which they occur within an individual flow.

To get the paragraphs within an individual flow in order, you navigate from the flow to the first text frame in the flow, to the first paragraph in that text frame. For example, to get the paragraphs in a document’s main flow in order, you can use the following code:

```c
#include "futils.h"
F_ObjHandleT docId, pgfId, flowId, textFrameId;
UCharT msg[256];
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
/* Get ID of main flow, then the first text column in the
 * flow, then the first paragraph in that text column.
 */
flowId = F_ApiGetId(FV_SessionId, docId, FP_MainFlowInDoc);
textFrameId = F_ApiGetId(docId, flowId, FP_FirstTextFrameInFlow);
pgfId = F_ApiGetId(docId, textFrameId, FP_FirstPgf);
/* Traverse ordered list of paragraphs in the flow. */
while (pgfId)
{
    F_Sprintf(msg, "The paragraph’s ID is 0x%x.", pgfId);
    F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);
    pgfId = F_ApiGetId(docId, pgfId, FP_NextPgfInFlow);
}
```

For a diagram of the links between flows, text frames, and paragraphs, see “The list of paragraphs in a flow” on page 304.

### Getting the IDs of selected objects

Document objects have properties that allow you to get the IDs of the following types of selected objects:

- Graphic objects
- Tables and table rows

To get the IDs of selected structural elements in FrameMaker documents, you must call a special function, F_ApiGetElementRange().
For background information on selection in Frame documents, see “How the API represents the selection in a document” on page 285. For information on getting selected text, see “Getting and setting the insertion point or text selection” on page 114.

**Getting the IDs of selected graphic objects**

The API maintains an unordered list of all the selected graphic objects in a document. To manipulate graphic objects the user has selected, you traverse this list. For example, the following code sets the fill pattern of all the selected graphic objects in the active document to black:

```c
... 
F_ObjHandleT docId, objId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
objId = F_ApiGetId(FV_SessionId, docId, FP_FirstSelectedGraphicInDoc);
while (objId)
{
    F_ApiSetInt(docId, objId, FP_Fill, FV_FILL_BLACK);
    objId = F_ApiGetId(docId, objId, FP_NextSelectedGraphicInDoc);
}
... 
```

**Getting the IDs of selected tables and table rows**

If a table is selected, you can get its ID by querying the document property, FP_SelectedTbl. A table is considered selected if any of its cells are selected or the insertion point is in any of its cells.

To get the IDs of the rows selected within a table, query the FO_Tbl object’s FP_TopRowSelection and FP_BottomRowSelection properties. FP_TopRowSelection specifies the ID of the row at the top of the selection; FP_BottomRowSelection specifies the ID of the row at the bottom of the selection.

To determine which cells in a row are selected, query a table’s FP_LeftColNum and FP_RightColNum properties. FP_LeftColNum specifies the number (starting from 0) of the leftmost selected column; FP_RightColNum specifies the number of the rightmost selected column.

If a range of text that includes several tables is selected, and you want to get the tables’ IDs, you must get the text selection and traverse all the table anchor text items in it. For more information on getting the text selection, see “Getting and setting the insertion point or text selection” on page 114.

**Getting the IDs of selected structural elements**

For information on getting the IDs of selected structural elements, see “Getting and setting the structural element selection” on page 120.
Getting the IDs of formats and other named objects

The following are some of the types of objects that are named (identified by a unique name).

- FO_CharFmt
- FO_Color
- FO_CombinedFontDefn
- FO_Command
- FO_CondFmt
- FO_ElementDef
- FO_Flow
- FO_FmtChangeList
- FO_MarkerType
- FO_MasterPage
- FO_Menu
- FO_PgfFmt
- FO_TblFmt
- FO_UnanchoredFrame (named frames on reference pages)
- FO_VarFmt
- FO_XRefFmt

The API maintains all the named objects of a particular type in a linked list. To get the objects, you can query FO_Doc properties and traverse the list. For example, to get all the variable formats in a document, query the FO_Doc object’s FP_FirstVarFmtInDoc property, and then traverse the FP_NextVarFmtInDoc properties from one FO_VarFmt object to the next.

If you only want the ID for a single named object, it is usually easier to use F_ApiGetNamedObject(). The syntax for F_ApiGetNamedObject() is:

```c
F_ObjHandleT F_ApiGetNamedObject(F_ObjHandleT docId, 
   IntT objType, 
   StringT name);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document or book containing the object</td>
</tr>
<tr>
<td>objType</td>
<td>The type of object (for example, FO_VarFmt)</td>
</tr>
<tr>
<td>name</td>
<td>The name of the object for which to get the ID</td>
</tr>
</tbody>
</table>

For example, the following code gets the ID of the Paragraph Catalog format named Body in the active document:
Getting and Setting Properties

... F_ObjHandleT docId, pgfFmtId; docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc); pgfFmtId = F_ApiGetNamedObject(docId, FO_PgfFmt, "Body"); ...

Getting IDs for objects that have persistent identifiers

FrameMaker assigns a persistent unique identifier (UID) to each object in a document that isn’t identified by a name. The UID, unlike the object’s ID, does not change from one session to another. No matter how many times you open and exit a document, an object’s UID remains the same.

It is important to note that cut and paste functions will assign new UID’s to the text. For example conditionalizing text will change a paragraph’s UID.

To get an object’s UID, you query its FP_Unique property with F_ApiGetInt(). If you already know an object’s UID, you can find its ID in the current session by calling F_ApiGetUniqueObject().

The syntax for F_ApiGetUniqueObject() is:

F_ObjHandleT F_ApiGetUniqueObject(F_ObjHandleT docId, IntT objType, IntT unique);

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document containing the object</td>
</tr>
<tr>
<td>objType</td>
<td>The object type (for example FO_Pgf)</td>
</tr>
<tr>
<td>unique</td>
<td>The object’s UID</td>
</tr>
</tbody>
</table>

For an example of how you can use UIDs, see “F_ApiGetUniqueObject()” on page 185 of the FDK Programmer’s Reference.

Manipulating properties

The API allows you to get and set either an individual property or a property list for an object. It is generally easier to get and set properties individually. However, some tasks, such as applying a Paragraph Catalog format to a paragraph, are easier to perform by getting and setting property lists.

Getting and setting individual properties

To get or set an individual property, use the F_ApiGetPropertyType() or F_ApiSetPropertyType() function that corresponds to the property’s data type. For
example, to get an integer, enum, or boolean (IntT data type) property, use 
F_ApiGetInt(). To set a property that represents a set of strings (F_StringsT data type),
use F_ApiSetStrings(). The data types of API properties are listed in Chapter 3, “Object
Reference” of the FDK Programmer’s Reference. The following table lists the functions you
use to set different types of properties.

The API also provides special functions to get and set properties that are identified by names.
These functions are used for getting and setting inset properties only. They are discussed in
Chapter 12, “Using Imported Files and Insets”

<table>
<thead>
<tr>
<th>Property’s data type</th>
<th>Functions to get and set property</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_AttributesT</td>
<td>F_ApiGetAttributes()</td>
</tr>
<tr>
<td></td>
<td>F_ApiSetAttributes()</td>
</tr>
<tr>
<td>F_AttributeDefsT</td>
<td>F_ApiGetAttributeDefs()</td>
</tr>
<tr>
<td></td>
<td>F_ApiSetAttributeDefs()</td>
</tr>
<tr>
<td>F_ElementCatalogEntriesT</td>
<td>F_ApiGetElementCatalog()</td>
</tr>
<tr>
<td>F_ElementFmtsT</td>
<td>F_ApiGetElementFormats()</td>
</tr>
<tr>
<td></td>
<td>F_ApiSetElementFormats()</td>
</tr>
<tr>
<td>F_ElementRangeT</td>
<td>F_ApiGetElementRange()</td>
</tr>
<tr>
<td></td>
<td>F_ApiSetElementRange()</td>
</tr>
<tr>
<td>F_ObjHandleT</td>
<td>F_ApiGetId()</td>
</tr>
<tr>
<td></td>
<td>F_ApiSetId()</td>
</tr>
<tr>
<td>IntT (including boolean, enum, and ordinal)</td>
<td>F_ApiGetInt()</td>
</tr>
<tr>
<td></td>
<td>F_ApiSetInt()</td>
</tr>
<tr>
<td>F_IntsT</td>
<td>F_ApiGetInts()</td>
</tr>
<tr>
<td></td>
<td>F_ApiSetInts()</td>
</tr>
<tr>
<td>MetricT</td>
<td>F_ApiGetMetric()</td>
</tr>
<tr>
<td></td>
<td>F_ApiSetMetric()</td>
</tr>
<tr>
<td>F_MetricsT</td>
<td>F_ApiGetMetrics()</td>
</tr>
<tr>
<td></td>
<td>F_ApiSetMetrics()</td>
</tr>
<tr>
<td>F_PointsT</td>
<td>F_ApiGetPoints()</td>
</tr>
<tr>
<td></td>
<td>F_ApiSetPoints()</td>
</tr>
<tr>
<td>StringT</td>
<td>F_ApiGetString()</td>
</tr>
<tr>
<td></td>
<td>F_ApiSetString()</td>
</tr>
<tr>
<td>F_StringsT</td>
<td>F_ApiGetStrings()</td>
</tr>
<tr>
<td></td>
<td>F_ApiSetStrings()</td>
</tr>
<tr>
<td>F_TabsT</td>
<td>F_ApiGetTabs()</td>
</tr>
<tr>
<td></td>
<td>F_ApiSetTabs()</td>
</tr>
<tr>
<td>F_TextLocT</td>
<td>F_ApiGetTextLoc()</td>
</tr>
<tr>
<td></td>
<td>F_ApiSetTextLoc()</td>
</tr>
<tr>
<td>F_TextRangeT</td>
<td>F_ApiGetTextRange()</td>
</tr>
<tr>
<td></td>
<td>F_ApiSetTextRange()</td>
</tr>
</tbody>
</table>
The syntax for most _F_ApiGetPropertyType()_ and _F_ApiSetPropertyType()_ functions is similar. For example, the syntax for _F_ApiGetInt()_ is:

```c
IntT F_ApiGetInt(F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum);
```

### This argument | Means
--- | ---
`docId` | The ID of the document, book, or session containing the object. If the object is a session, specify 0.
`objId` | The ID of the object whose property you want to query.
`propNum` | The property to query (for example, `FP_FnNum`).

The syntax for _F_ApiSetString()_ is:

```c
VoidT F_ApiSetString(F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum,
    StringT setVal);
```

### This argument | Means
--- | ---
`docId` | The ID of the document, book, or session containing the object
`objId` | The ID of the object whose property you want to set
`propNum` | The property to set, for example, `FP_PrintFileName`
`setVal` | The string to which to set the property

You can look up the exact syntax of an _F_ApiGetPropertyType()_ or _F_ApiSetPropertyType()_ function in Chapter 2, “FDK Function Reference,” in the _FDK Programmer’s Reference_.

Suppose you want your client to change some characteristics of the Heading1 paragraph format. To find out how the API represents paragraph formats, look up paragraph formats in Part II, “Frame Product Architecture.” For a complete list of paragraph format properties, see “Object Reference” on page 495 of the _FDK Programmer’s Reference_.

The following code demonstrates how to change different types of paragraph format properties:

```c
#define in (MetricT) (72 * 65536) /* A Frame metric inch */
F_ObjHandleT docId, pgfFmtId;

/* Get the ID of Heading1 format in active document. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
pgfFmtId = F_ApiGetNamedObject(docId, FO_PgfFmt, "Heading1");
```
/* Set Next Pgf Tag to Heading2. */
F_ApiSetString(docId, pgfFmtId, FP_NextTag, "Heading2");

/* Turn on Keep With Next. */
F_ApiSetInt(docId, pgfFmtId, FP_KeepWithNext, True);

/* Set the left indent to 1 inch. */
F_ApiSetMetric(docId, pgfFmtId, FP_LeftIndent, in);

This code changes only the Heading1 Paragraph Catalog format. It does not change the formats of paragraphs that have already been tagged with Heading1.

## Getting and setting property lists

Because most objects have relatively long property lists, it is often easier to get and set individual properties. However, to perform the following types of tasks, you may need to get and set entire property lists:

- Getting and setting text properties
- Applying table, paragraph, and character formats
- Copying graphic object properties

To get and set property lists, you need to understand how the API represents them. For more information, see “Property lists” on page 272.

F_ApiGetProps() and F_ApiSetProps() make it easy to get and set property lists.

The syntax for these functions is:

```c
F_PropValsT F_ApiGetProps(F_ObjHandleT docId,
                         F_ObjHandleT objId);

VoidT F_ApiSetProps(F_ObjHandleT docId,
                    F_ObjHandleT objId,
                    F_PropValsT *setVal);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the session, book, or document containing the object</td>
</tr>
<tr>
<td>objId</td>
<td>The ID of the object to get or set the property list for</td>
</tr>
<tr>
<td>setVal</td>
<td>The property list to apply to the object</td>
</tr>
</tbody>
</table>

The F_PropValsT structure returned by F_ApiGetProps() references memory that is allocated by the API. Use F_ApiDeallocatePropVals() to free this memory when you are done with it. If F_ApiGetProps() fails, the API sets the len field of the returned structure to 0.
Example
The following code copies the properties from one selected graphic object to another:

```c
F_PropValsT props;
F_ObjHandleT obj1Id, obj2Id, docId;

/* Get ID of active document and the two selected objects. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
obj1Id = F_ApiGetId(FV_SessionId, docId, FP_FirstSelectedGraphicInDoc);
obj2Id = F_ApiGetId(docId, obj1Id, FP_NextSelectedGraphicInDoc);
/* Make sure two objects are selected, then copy properties. */
if (!(obj1Id && obj2Id)) F_ApiAlert("First select two objects.", FF_ALERT_CONTINUE_WARN);
else{
    props = F_ApiGetProps(docId, obj1Id);
    if(props.len == 0) return; /* Get props failed. */
    F_ApiSetProps(docId, obj2Id, &props);
}
```

Because a graphic object’s x and y coordinates are included in its property list, this code moves the two graphic objects to the same location, with one object overlaying the other.

After you have copied a property list to an object, you can customize the list by changing individual properties.

**Manipulating property lists directly**

If you are setting individual text properties or using scriptable functions, such as F_ApiOpen(), you need to manipulate property lists directly.

The order of the properties in property lists is not guaranteed to remain the same in different versions of FrameMaker products and the Frame API. So, to get a particular property in a list, you must traverse the entire property list and check each property’s identifier until you find it. The API provides a convenience routine named F_ApiGetPropIndex() that does this for you.

The syntax for F_ApiGetPropIndex() is:

```c
IntT F_ApiGetPropIndex(F_PropValsT *pvp, IntT propNum);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>pvp</td>
<td>The property list</td>
</tr>
<tr>
<td>propNum</td>
<td>The property whose index you want to get</td>
</tr>
</tbody>
</table>
F_ApiGetPropIndex() returns the index of the F_PropValT structure that represents the property’s property-value pair. If you specify an invalid property for propNum, F_ApiGetPropIndex() returns FE_BadPropNum.

Suppose you want to display the session property that provides the name of the current FrameMaker product interface. The easy way to do this would be to use the following code:

```c
StringT productName;

productName = F_ApiGetString(0, FV_SessionId, FP_ProductName);
F_ApiAlert(productName, FF_ALERT_CONTINUE_NOTE);
```

To do the same thing by getting the property list for the session and accessing the property directly, use the following code:

```c
IntT i;
F_PropValsT props;

props = F_ApiGetProps(0, FV_SessionId);

i = F_ApiGetPropIndex(&props, FP_ProductName);
F_ApiAlert(props.val[i].propVal.u.sval, 
          FF_ALERT_CONTINUE_NOTE);
```

### Allocating and deallocating memory for properties

The F_ApiGetPropertyType() functions that return pointers to arrays make copies of the arrays, allocating memory for them. For example, F_ApiGetString() does not return a pointer to the actual string used by the FrameMaker product. Instead, it creates a copy of the string and returns a pointer to the copy. The API does not deallocate memory used by the copy of the string. When you are done with it, you must deallocate it.

Similarly, when you call a F_ApiSetPropertyType() function such as F_ApiSetString(), the function does not set a pointer to the string you pass to it. Instead it copies the string. The API does not deallocate the string you pass. When you are done with it, you must deallocate it.

For example, the following code queries and displays the FP_OpenDir property. It uses the FDE function, F_Free(), to free the returned string.

```c
```
Getting and Setting Properties

Getting and setting session properties

The following sections describe useful tasks you can perform by getting and setting session properties.

Making a document or book active

In addition to finding out which document is active by getting the session’s `FP_ActiveDoc` property, you can make a document active by setting this property. For example, the following code makes the document specified by `docId` active:

```
F_ObjHandleT docId;
F_ApiSetId(0, FV_SessionId, FP_ActiveDoc, docId);
```

When you make a visible document active, its window gets input focus. On some platforms, the windowing system highlights a window’s title bar or brings it to the front.
Getting the user home, $FMHOME, and current directories

Some platforms (such as UNIX) support the concepts of user home, $FMHOME, and current directories. The following code displays the names of these directories for the current session:

```c
#include "futils.h"
StringT homeDir, fmhomeDir, currentDir;
UCharT msg[512];

/* Get directory names. */
homeDir = F_ApiGetString(0, FV_SessionId, FP_UserHomeDir);
fmhomeDir = F_ApiGetString(0, FV_SessionId, FP_FM_HomeDir);
currentDir = F_ApiGetString(0, FV_SessionId, FP_FM_CurrentDir);

F_Sprintf(msg, "Home dir: %s. FMHOME: %s. Current dir: %s",
    homeDir, fmhomeDir, currentDir);

/* Display the directory names in an alert box. */
F_ApiAlert(msg, FF_ALERT_CONTINUE_NOTE);

/* Free the strings. */
F_Free(homeDir);
F_Free(fmhomeDir);
F_Free(currentDir);

Disabling redisplaying to avoid screen flicker

If you change numerous properties at once, it may cause screen flicker, an effect that occurs when the FrameMaker product executes a long series of changes that aren’t user-initiated. By default, FrameMaker reformatrs the document after each change.

You can avoid screen flicker by batching changes. To batch changes, set the FO_Session property FP_Displaying to False. As long as FP_Displaying is set to False, the FrameMaker product does not refresh the documents in the current session when you or the user changes them. To refresh the documents, you must call F_ApiRedisplay() for each changed document.

The syntax for F_ApiRedisplay() is:

```c
IntT F_ApiRedisplay(F_ObjHandleT docId);
```

For example, to change a number of properties at once, use code similar to the following:
Getting and Setting Properties

The following sections describe useful tasks you can perform by getting and setting document properties.

Getting a document’s pathname

A document’s absolute pathname is specified by its FP_Name property. The following code displays the active document’s absolute pathname:

```c
F_ObjHandleT docId;
StringT docName;

/* Get the document ID and name. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
docName = F_ApiGetString(FV_SessionId, docId, FP_Name);
F_ApiAlert(docName, FF_ALERT_CONTINUE_NOTE);
F_Free(docName);
```

Manipulating document windows

The API provides several properties that allow you to manipulate document and book windows. To change a document window’s size and screen location, set the document’s FP_ScreenX, FP_ScreenY, FP_ScreenWidth, and FP_ScreenHeight properties. To bring the window to the front, set the document’s FP_IsInFront property.
Setting a document or book title bar

The API allows you to set the title bars of both documents and books. By default, a document or book’s title bar displays its name. However, you can make it display another string by setting the document or book’s `FP_Label` property to the string. For example, the following code displays the string `MyTitle` in the title bar of the active document:

```c
F_ObjHandleT docId;
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
F_ApiSetString(FV_SessionId, docId, FP_Label, "MyTitle");
```

After you set a document’s title bar, it doesn’t change until you reset it or the user closes the document.

Setting the title bar of a document or book doesn’t change the name of the document or book file itself. If you set the document or book title bar and the user closes and reopens the document or book, the document or book name appears in the title bar again.

Setting a document or book status bar

The API allows you to set the status bars of both documents and books. If your client conducts extensive processing, it can display status messages in the status bar to inform users of its progress. To set the status bar of a document or a book, set its `FP_StatusLine` property.

The string you set `FP_StatusLine` to remains in the status bar only until a client or the FrameMaker product overwrites it. FrameMaker overwrites the status bar frequently. For example, every time the user moves the insertion point to a different paragraph in a document, the FrameMaker product redisplays the paragraph format in the status bar.

Enhancing performance by making documents invisible

The API allows you to make a document invisible. Your client can still make changes to an invisible document. If your client needs to batch process multiple documents, using invisible documents can increase its performance considerably.

To make a document invisible, you can use the following code:

```c
F_ObjHandleT docId;
F_ApiSetInt(FV_SessionId, docId, FP_IsOnScreen, False);
```

You can also open documents invisibly by setting the `FP_MakeVisible` property of the Open script to `False`.

**IMPORTANT:** Because an invisible document can’t get input focus, it can’t be the active document specified by the session property `FP_ActiveDoc`. You can’t send f-codes to an invisible document.
Displaying a particular page

Document objects (FO_Doc) have a property named FP_CurrentPage that specifies the ID of the current page. The current page is the page that appears on the screen. If more than one page appears on the screen, it is the page that appears with a dark border around it. You can make a page current by making the document that contains it the active document and then setting the document’s FP_CurrentPage property to the page’s ID.

For example, the following code displays the second body page and then the first reference page of the active document:

```c
... F_ObjHandleT docId, bPg1Id, bPg2Id, rPg1Id;
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
/* Get second body page ID. */
bPg1Id = F_ApiGetId(FV_SessionId, docId, FP_FirstBodyPageInDoc);
bPg2Id = F_ApiGetId(docId, bPg1Id, FP_PageNext);
if (bPg2Id)
    { F_ApiSetId(FV_SessionId, docId, FP_CurrentPage, bPg2Id);
      F_ApiAlert("Now at 2nd body page.", FF_ALERT_CONTINUE_NOTE);
    }
/* Go to first reference page. */
rPg1Id = F_ApiGetId(FV_SessionId, docId, FP_FirstRefPageInDoc);
if (rPg1Id)
    F_ApiSetId(FV_SessionId, docId, FP_CurrentPage, rPg1Id);
...```

Getting and setting graphic object properties

The following sections describe useful tasks you can perform by getting and setting graphic object properties.

Changing an object’s size and location within a frame

Each graphic object has FP_Height and FP_Width properties, which specify its height (the distance between its highest and lowest points) and its width (the distance between its leftmost and rightmost points). To change an object’s size, use F_ApiSetMetric() to set these properties. For example, the following code increases a selected object’s width by 10 points:
#define pts (MetricT) 65536 /* Frame metric point */
F_ObjHandleT docId, objId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
/* Get ID of selected object. */
objId = F_ApiGetId(FV_SessionId, docId, FP_FirstSelectedGraphicInDoc);
/* Get object’s current width and add 10 points to it. */
if (objId)
    F_ApiSetMetric(docId, objId, FP_Width, F_ApiGetMetric(docId, objId, FP_Width) + 10*pts);

If you set the FP_Height and FP_Width properties of a polyline or polygon, the API
changes all the object’s vertices proportionally. If you want to change a polygon or polyline’s
vertices independently, use F_ApiSetPoints() to set its FP_Points property. For an
example of how to set the FP_Points property, see “F_ApiSetPoints()” on page 295 of the
FDK Programmer’s Reference.

All graphic objects have an FP_LocX property, which specifies the distance of the object’s
leftmost point from the left side of the parent frame, and an FP_LocY property, which
specifies the distance of the object’s uppermost point from the top of its parent frame. To
change an object’s location within a frame, use F_ApiSetMetric() to set these properties.

**Moving graphics forward or back in the draw order**

FrameMaker maintains the graphic objects in each frame in a linked list. Each graphic object
has FP_PrevGraphicInFrame and FP_NextGraphicInFrame properties that specify the
graphic objects before and after it in the list. The order of this list corresponds to the back-to-
front draw order. The first object in the list is the first object the FrameMaker product draws,
and therefore appears in back of objects later in the list. To move a graphic object forward or
back in the draw order, you change its FP_PrevGraphicInFrame or
FP_NextGraphicInFrame property so that it specifies a different object. You need to
change only one of these properties. The FrameMaker product automatically changes the other
one for you. It also automatically changes the FP_PrevGraphicInFrame or
FP_NextGraphicInFrame properties of the object’s siblings.

To move an object all the way to the back of the objects in a frame, set its
FP_PrevGraphicInFrame property to 0. To move an object all the way to the front, set its
FP_NextGraphicInFrame property to 0.

For example, the following code moves a selected graphic object forward one level:
Getting and Setting Properties

F_ObjHandleT docId, objId, sibId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

/* Get ID of one (and only one) selected object. */
objId = F_ApiGetId(FV_SessionId, docId,
                  FP_FirstSelectedGraphicInDoc);
if (!objId || F_ApiGetId(docId, objId,
             FP_NextSelectedGraphicInDoc))
    F_ApiAlert("Select a single object", FF_ALERT_CONTINUE_NOTE);
else
{
    /* Try to get ID of object in front of selected object. */
    sibId = F_ApiGetId(docId, objId, FP_NextGraphicInFrame);

    /* If there is an object in front, put it behind. */
    if(sibId)
        F_ApiSetId(docId, objId, FP_PrevGraphicInFrame, sibId);
}

Moving graphic objects to different frames or pages

To move a graphic object to a different frame, set its \texttt{FP\_FrameParent} property to the ID of that frame. The API automatically changes all the properties that need to be changed to maintain the lists of objects in the object’s old and new parent frames.

To move a graphic object to a different page, set its \texttt{FP\_FrameParent} property to the ID of a frame on that page. All pages have an invisible frame, called a page frame. To put a graphic object directly on a page, set its \texttt{FP\_FrameParent} property to the ID of the page’s page frame. For more information on page frames, see “How the API represents pages” on page 289.

For an example of how to move objects from a frame to a page frame, see “F\_ApiSetId()” on page 285 of the \textit{FDK Programmer’s Reference}.

Grouping objects

To group a set of objects, you first use \texttt{F\_ApiNewGraphicObject()} to create a group (\texttt{FO\_Group}) object. Then you add the objects to the group object by setting their \texttt{FP\_GroupParent} properties to the ID of the group object. The objects must be in the same frame as the group object. For information on how to use \texttt{F\_ApiNewGraphicObject()}, see “Creating graphic objects” on page 146.

To remove an object from a group, set the object’s \texttt{FP\_GroupParent} property to \texttt{0}. 
Getting and Setting Properties

Copying properties from one graphic object to another

Each type of graphic object has a number of properties, such as \texttt{FP\_Fill} and \texttt{FP\_BorderWidth}, which are common to all graphic objects. Some of these properties don’t manifest themselves for all graphic objects. For example, rectangles have an \texttt{FP\_ArrowType} property, although they don’t have arrowheads. For a list of properties common to all graphic objects, see “Common graphics properties” on page 592 of the \textit{FDK Programmer’s Reference}.

You can use \texttt{F\_ApiGetProps()} and \texttt{F\_ApiSetProps()} to copy common properties from one graphic object to another, as shown in the example in “Getting and setting property lists” on page 92. When you copy properties from one graphic object to another, the objects do not have to be the same type. For example, you can copy the properties from a line to a rectangle. The API copies only the common properties, leaving properties that are specific to the rectangle, such as \texttt{FP\_RectangleIsSmoothed}, intact.

Getting and setting paragraph properties

The following sections describe useful tasks you can perform by getting and setting paragraph properties.

Applying paragraph and Paragraph Catalog formats

Paragraph (\texttt{FO\_Pgf}) objects and Paragraph Catalog format (\texttt{FO\_PgfFmt}) objects have the same formatting properties. To apply the properties from a paragraph to a Paragraph Catalog format or from a Paragraph Catalog format to a paragraph, you can use \texttt{F\_ApiGetProps()} and \texttt{F\_ApiSetProps()}.

For example, the following code applies the Paragraph Catalog format named Body to the paragraph containing the insertion point:

```c
F\_PropValsT props;
F\_TextRangeT tr;
F\_ObjHandleT docId, pgfId, bodyFmtId;
StringT pgfName;
docId = F\_ApiGetId(0, FV\_SessionId, FP\_ActiveDoc);
/* Get ID of the Body Paragraph Catalog format - then get selection. */
bodyFmtId = F\_ApiGetNamedObject(docId, FO\_PgfFmt, "Body");
if (!bodyFmtId) return;
tr = F\_ApiGetTextRange(FV\_SessionId,docId, FP\_TextSelection);
if (tr.beg.objId == 0) return;
/*
 * Get properties from the Body Paragraph Catalog format.
 * Then apply Body properties to paragraph containing insertion point (or the beginning of the text selection).
 */
```
Getting and Setting Properties

```c
props = F_ApiGetProps(docId, bodyFmtId);
if(props.len == 0) return;
F_ApiSetProps(docId, tr.beg.objId, &props);
```

If you have changed the Body format, you may want to reapply it to all paragraphs that are tagged Body. To change these paragraphs, you must traverse every paragraph in the document, determine if it’s tagged Body, and set its properties if it is. You can do this by adding the following code to the code shown above:

```c
pgfId = F_ApiGetId(FV_SessionId, docId, FP_FirstPgfInDoc);
while (pgfId)
{
    /* Get each paragraph’s tag and see if it’s Body. */
    pgfName = F_ApiGetString(docId, pgfId, FP_Name);
    if (F_StrEqual((StringT)"Body", pgfName))
        F_ApiSetProps(docId, pgfId, &props);
    F_Free(pgfName);
    pgfId = F_ApiGetId(docId, pgfId, FP_NextPgfInDoc);
}
```

Adding tabs

To get and set the tabs for a paragraph or Paragraph Catalog format, use `F_ApiGetTabs()` and `F_ApiSetTabs()`.

The syntax for `F_ApiGetTabs()` and `F_ApiSetTabs()` is:

```c
F_TabsT F_ApiGetTabs(F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum);
 voidT F_ApiSetTabs(F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum,
    F_TabsT *setVal);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document containing the paragraph or paragraph format whose tabs you want to query or set.</td>
</tr>
<tr>
<td>objId</td>
<td>The ID of the paragraph or paragraph format whose tabs you want to query or set.</td>
</tr>
</tbody>
</table>

1. Some examples in this chapter use FDE functions, such as `F_StrEqual()`, `F_Alloc()`, and `F_Realloc()`. For more information on using the FDE and these functions, see Part III, “Frame Development Environment (FDE).”
5

Getting and Setting Properties

This argument

Means

propNum

The property to query. Specify FP_Tabs.

setVal

The F_TabsT structure to which to set the property.

The F_TabsT structure is defined as:
typedef struct {
UIntT len; /* The number of tabs in val */
F_TabT *val; /* Structures that describe the tabs */
} F_TabsT;

The F_TabT structure is defined as:
typedef struct {
MetricT x; /* Offset from paragraph’s left margin */
UCharT type; /* Constant for tab type, e.g. FV_TAB_RIGHT */
StringT leader; /* Characters before tab, e.g. "." */
UCharT decimal; /* Character for decimal tab, e.g. "." */
} F_TabT;

When you get the tabs for a paragraph or paragraph format, the API returns them in left-toright order in the val array. However, when you insert a tab, you don’t have to insert it in this
order. You just add it to the end of the val array. When you call F_ApiSetTabs(), the API
sorts the tabs for you. For example, the following code adds a 4-inch decimal tab to the Body
paragraph format:
#include "fmemory.h"
#define in (MetricT) (65536 * 72)
F_ObjHandleT docId, pgfFmtId;
F_TabsT tabs;
/* Get the ID for the Body paragraph format. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
pgfFmtId = F_ApiGetNamedObject(docId, FO_PgfFmt, "Body");
if (!pgfFmtId) return;
/* Get the tabs and allocate space for new tab. */
tabs = F_ApiGetTabs(docId, pgfFmtId, FP_Tabs);
if (tabs.len++)
tabs.val=(F_TabT*)F_Realloc(tabs.val,
tabs.len*sizeof(F_TabT), NO_DSE);
else
tabs.val = (F_TabT*) F_Alloc(sizeof(F_TabT),NO_DSE);
/* Add the tab. */
tabs.val[tabs.len-1].type = FV_TAB_DECIMAL;
tabs.val[tabs.len-1].x = 4*in;
tabs.val[tabs.len-1].decimal = ’,’;
tabs.val[tabs.len-1].leader = F_StrCopyString(" ");
/* Set paragraph format’s tabs property to the array of tabs. */
F_ApiSetTabs(docId, pgfFmtId, FP_Tabs, &tabs);

104

FDK Programmer’s Guide


Getting and Setting Properties

Getting and setting book properties

To rearrange book components, you change their `FP_PrevComponentInBook` and `FP_NextComponentInBook` properties. For example, to move the first component in a book down one position, you can use the following code:

```c
F_ObjHandleT bookId, firstC, nextC;

bookId = F_ApiGetId(0, FV_SessionId, FP_ActiveBook);
firstC = F_ApiGetId(FV_SessionId, bookId, FP_FirstComponentInBook);
nextC = F_ApiGetId(bookId, firstC, FP_NextComponentInBook);

if (nextC)
    F_ApiSetId(bookId, firstC, FP_PrevComponentInBook, nextC);
else
    F_ApiAlert("Only one component.", FF_ALERT_CONTINUE_NOTE);
```

Getting and setting FrameMaker properties

There are some special issues involved in getting and setting properties in structured FrameMaker documents. The following sections discuss some of these issues.

Traversing elements

To traverse the elements in a structured FrameMaker document, you use slightly different code than you would use to traverse other objects, such as paragraphs. If you want to traverse all the elements in a document, you can’t query only `FP_NextSiblingElement` properties. You must also recursively traverse each element’s child elements. For example, the following function prints the IDs of all the direct children of a specified element:

```c
VoidT traverseElement(F_ObjHandleT docId, F_ObjHandleT elementId)
{
    StringT name;

    if (elementId)
        elementId = F_ApiGetId(docId, elementId, FP_FirstChildElement);
    while(elementId)
    {
        F_Printf(NULL, (StringT) "Element ID is 0x%x.\n",elementId);
        traverseElement(docId, elementId);
        elementId = F_ApiGetId(docId, elementId,
```
Manipulating format change list properties

Most object types in the FDK have a single list of properties that applies to all objects of that type. For example, if you call `F_ApiGetProps()` for any FO_Pgf object in a document, it will always return the same list of properties. The values of the properties may be different for each paragraph, but the list of properties will always be the same. This is not the case with FO_FmtChangeList objects.

All FO_FmtChangeList objects have the following common properties:

- FP_Name
- FP_NextFmtChangeListInDoc
- FP_PgfCatalogReference

However, individual FO_FmtChangeList objects can have different sets of additional properties, depending on what formatting characteristics they set. An FO_FmtChangeList object can have all the properties listed under “Format change lists” on page 582 of the FDK Programmer’s Reference, or it may have just a small subset of these properties.

For example, the format change list in Figure 5.1 has only the common properties listed above and the FP_FontFamily property. If you call `F_ApiGetProps()` for this format change list, the function returns only four properties: the three common properties listed above and the FP_FontFamily property.

If you attempt to use an `F_ApiGetPropertyType()` function to get a property that a format change list doesn’t have, the function fails, setting FA_errno to FE_PropNotSet.

Adding properties to a format change list

To add a property to a format change list, you just set the property on the FO_FmtChangeList object. You can do this by calling an `F_ApiSetPropertyType()` function or by creating a property list containing the property and calling `F_ApiSetProps()` to set the list on the object. For example, the following code uses these two methods to add properties to the Code format change list:

```c
#define pts (MetricT) 65536
F_PropValsT props;
F_ObjHandleT docId, changeListId;
```
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

/* Create the Code change list. */
changeListId = F_ApiNewNamedObject(docId, FO_FmtChangeList, "Code");

/* Add the FP_PairKern property to turn pair kerning off. */
F_ApiSetInt(docId, changeListId, FP_PairKern, False);

/* Set up list with FP_FontSize property to set size to 10. */
props = F_ApiAllocatePropVals(1);
props.val[0].propIdent.num = FP_FontSize;
props.val[0].propVal.valType = FT_Metric;
props.val[0].propVal.u.ival = 10*pts;

F_ApiSetProps(docId, changeListId, &props);

. . .
Note that the  F_ApiSetProps()  call in the code above only adds the  FP_FontSize property. It does not affect the other properties of the format change list.

Removing properties from a format change list
To remove a property from a format change list, call  F_ApiDeletePropByName() . For example, the following code removes the  FP_PairKern  property from the Code format change list:

. . .
F_ObjHandleT docId, changeListId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
changeListId = F_ApiGetNamedObject(docId, FO_FmtChangeList, "Code");
if (!changeListId) return;
F_ApiDeletePropByName(docId, changeListId, "FP_PairKern");

. . .

Setting format rule clause properties
Format rule clauses (FO_FmtRuleClause objects) have several properties that are only indirectly settable. For example, you can’t set the  FP_RuleClauseType  property directly. You can only set it indirectly by setting one of the following properties:

● FP_FormatName
● FP_FmtChangeListTag
● FP_FmtChangeList
● FP_SubFmtRule

For example, if you set FP_FmtChangeListTag to Code, FrameMaker automatically sets FP_RuleClauseType to FV_RC_CHANGELIST_TAG.
You can also set FP_FmtChangeList and FP_SubFmtRule only indirectly—by calling F_ApiSubObject() to add a change list or subformat rule to the format rule clause or by calling F_ApiDelete() to delete a change list or subformat rule. For an example of how to use F_ApiSubObject(), see “Creating format rules, format rule clauses, and format change lists” on page 152.

Setting element definition properties

Element definitions have the following properties that are only indirectly settable:

- FP_TextFmtRules
- FP_ObjectFmtRules
- FP_PrefixRules
- FP_SuffixRules
- FP_FirstPgfRules
- FP_LastPgfRules

Each of these properties specifies a list of format rules. You can’t directly add a format rule to one of these lists. Instead, you must call F_ApiSubObject(). F_ApiSubObject() creates an FO_FmtRule object and adds it to the end of the specified list.

You also can’t directly remove a format rule from a list. Instead, you must call F_ApiDelete() to delete the format rule. For example, the following code adds and deletes a text format rule:

```c
F_ObjHandleT docId, paraEdefId, fmtRuleId;
paraEdefId = F_ApiGetNamedObject(docId, FO_ElementDef, "Para");
/* Add a text format rule to the element definition. */
fmtRuleId = F_ApiSubObject(docId, paraEdefId, FP_TextFmtRules);
/* Delete the text format rule. */
F_ApiDelete(docId, fmtRuleId);
```

Determining the formatting that applies to an element

To determine the formatting that applies to an element, you first get the following properties:

- FP_MatchingTextClauses
- FP_MatchingObjectClauses
- FP_MatchingPrefixClauses
- FP_MatchingSuffixClauses
● **FP_MatchingFirstPgfClauses**

● **FP_MatchingLastPgfClauses**

Each of these properties specifies a list of format rule clauses that applies to the element. Getting these properties only for the element itself is not sufficient to determine the element’s formatting. These properties specify only the format rule clauses that are in the element definition’s format rules (that is, the format rules specified by the element definition’s **FP_TextFmtRules**, **FP_ObjectFmtRules**, **FP_PrefixRules**, **FP_SuffixRules**, **FP_FirstPgfRules**, and **FP_LastPgfRules** properties). In order to fully determine the element’s formatting, you must find any applicable format rule clauses that the element inherits from its ancestor elements. To determine whether an element inherits format rule clauses from ancestor elements, you must traverse up the structure tree and check the **FP_MatchingClauses** properties for each ancestor element.

---

**Determining which element contains an object**

Frequently, it is useful to determine which element contains an object, such as a cross-reference, a marker, or a table. The following table lists the properties you query to get the ID of an object’s containing element.

<table>
<thead>
<tr>
<th>Object</th>
<th>Property that returns ID of containing element</th>
</tr>
</thead>
<tbody>
<tr>
<td>FO_Marker</td>
<td><strong>FP_Element</strong></td>
</tr>
<tr>
<td>FO_Fn</td>
<td></td>
</tr>
<tr>
<td>FO_XRef</td>
<td></td>
</tr>
<tr>
<td>FO_Var</td>
<td></td>
</tr>
<tr>
<td>FO_AFrame</td>
<td></td>
</tr>
<tr>
<td>FO_Row</td>
<td></td>
</tr>
<tr>
<td>FO_Cell</td>
<td></td>
</tr>
<tr>
<td>FO_Tbl(^a)</td>
<td><strong>FP_TblElement</strong></td>
</tr>
<tr>
<td></td>
<td><strong>FP_TblTitleElement</strong></td>
</tr>
<tr>
<td></td>
<td><strong>FP_TblHeaderElement</strong></td>
</tr>
<tr>
<td></td>
<td><strong>FP_TblBodyElement</strong></td>
</tr>
<tr>
<td></td>
<td><strong>FP_TblFooterElement</strong></td>
</tr>
</tbody>
</table>

\(^a\). If a table has a title or different types of rows, it can comprise several elements.

To determine the ID of the object an element contains, you query the element’s **FP_Object** property.
Getting and Setting Properties

Specifying client data for an element

The FO_Element property FP_UserString allows your client to store its own data with individual structural elements. The FP_UserString property is persistent between sessions; after a client sets it, it remains the same until a client resets it. If an element is cut and pasted, it retains its FP_UserString property. If an element is copied and pasted, both the original element and the pasted element retain the FP_UserString property.

Improving performance in FrameMaker clients

If you are using the API to create FrameMaker documents, you may need to add a large number of elements or element definitions at a time. By default, FrameMaker validates elements and applies format rules each time you add an element or element definition. This can decrease performance considerably. To keep FrameMaker from validating elements and applying format rules, set the FO_Session properties FP_Validating and FP_ApplyFmtRules to False.
Manipulating Text

This chapter describes how to use the API to manipulate text in Frame documents. Specifically, it discusses how to:

- Retrieve text from a document
- Get and set the location of the insertion point or current text selection
- Add and delete text
- Get and set text formatting
- Programmatically execute Clipboard operations

To better understand the material in this chapter, you may want to learn more about how the API represents text. For information on this subject, see “Text” on page 311.

Getting text

Text in Frame documents is contained in objects, such as FO_Cell, FO_Element, FO_Fn, FO_Pgf, FO_TextLine, FO_Var, FO_SubCol, FO_TextFrame, and FO_Flow objects. To get text, you must get the ID of the object that contains it. For information on getting object IDs, see “Getting the IDs of the objects you want to change” on page 82.

Once you have the ID of an object that contains text, you use F_ApiGetText() to retrieve the text.

The syntax for F_ApiGetText() is:

F_TextItemsT F_ApiGetText(F_ObjHandleT docId,
                         F_ObjHandleT objId,
                         IntT flags);

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document containing the object for which you want to get text.</td>
</tr>
<tr>
<td>objId</td>
<td>The ID of the object (FO_Flow, FO_Element, FO_Fn, FO_Pgf, FO_Cell, FO_SubCol, FO_TextFrame, FO_TextLine, or FO_Var) containing the text.</td>
</tr>
<tr>
<td>flags</td>
<td>Bit flags that specify the type of text items to retrieve. To get specific types of text items, OR the constants that represent them (for example, FTI_FlowBegin and FTI_String) into flags. To get all types of text items, specify -1. For a complete list of the constants that represent text item types, see “F_ApiGetText()” on page 164 of the FDK Programmer’s Reference.</td>
</tr>
</tbody>
</table>
The F_TextItemsT structure contains an array of text items. Each string of characters with common character and condition properties, each anchor, and each line or column break in the text constitutes a separate text item.

F_TextItemsT is defined as:

typedef struct {
    UIntT len;  /* The number of text items */
    F_TextItemT *val;  /* Array of text items */
} F_TextItemsT;

The API represents each text item with an F_TextItemT structure. F_TextItemT is defined as:

typedef struct {
    IntT offset;  /* Characters from beginning */
    IntT dataType;  /* Text item type, e.g. FTI_String */
    union {
        StringT sdata;  /* String if the type is FTI_String */
        F_ObjHandleT idata;  /* ID if item is an anchor */
    } u;
} F_TextItemT;

If a text item represents a string of characters, F_TextItemT.dataType is set to FTI_String and F_TextItemT.u.sdata contains the string. If the text item represents an anchor, F_TextItemT.dataType is set to a constant indicating the anchor type (for example, FTI_TblAnchor) and F_TextItemT.u.idata contains the ID of the anchored object (for example, an FO_Tbl object).

For more information on the F_TextItemsT structure, see “How the API represents text” on page 312.

After you are finished with an F_TextItemsT structure, free the memory that it uses with F_ApiDeallocateTextItems(). The syntax for F_ApiDeallocateTextItems() is:

VoidT F_ApiDeallocateTextItems(F_TextItemsT *itemsp);

where itemsp is the F_TextItemsT structure that you want to free.

If you call F_ApiGetText() for a structural element (FO_Element object), the returned information depends on the type of element, as shown in the following table:

<table>
<thead>
<tr>
<th>Element’s FP_ElementType value</th>
<th>Information returned by F_ApiGetText()</th>
</tr>
</thead>
<tbody>
<tr>
<td>FV_FO_CONTAINER</td>
<td>All the text items from the beginning to the end of the element.</td>
</tr>
<tr>
<td>FV_FO_SYS_VAR</td>
<td>All the text items from the beginning to the end of the variable.</td>
</tr>
<tr>
<td>FV_FO_XREF</td>
<td>All the text items from the beginning to the end of the cross-reference.</td>
</tr>
<tr>
<td>FV_FO_FOOTNOTE</td>
<td>All the text items from the beginning to the end of the footnote.</td>
</tr>
</tbody>
</table>
Example

The following code retrieves and prints the text in the active document’s main flow to the console. It retrieves and prints only strings and line ends.

```c
#include "futils.h"
F_ObjHandleT docId, flowId;
IntT i;
F_TextItemsT tis;
F_TextItemT *ip;

/* Get IDs for active document and main flow. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
flowId = F_ApiGetId(FV_SessionId, docId, FP_MainFlowInDoc);

/* Traverse text items and print strings and line ends. */
for (i=0; i<tis.len; i++)
{
    ip = &tis.val[i];
    if (ip->dataType == FTI_String)
        F_Printf(NULL, "%s", ip->u.sdata);
    else F_Printf(NULL, "\n");
}
F_ApiDeallocateTextItems(&tis);
```

<table>
<thead>
<tr>
<th>Element’s FPElementType value</th>
<th>Information returned by F_ApiGetText()</th>
</tr>
</thead>
<tbody>
<tr>
<td>FV_FO_TBL_TITLE</td>
<td>All the text items from the beginning to the end of the table title.</td>
</tr>
<tr>
<td>FV_FO_TBL_CELL</td>
<td>All the text items from the beginning to the end of the cell.</td>
</tr>
<tr>
<td>FV_FO_TBL_HEADING</td>
<td>Nothing. F_ApiGetText() fails.</td>
</tr>
<tr>
<td>FV_FO_TBL_BODY</td>
<td></td>
</tr>
<tr>
<td>FV_FO_TBL_FOOTING</td>
<td></td>
</tr>
<tr>
<td>FV_FO_MARKER</td>
<td></td>
</tr>
<tr>
<td>FV_FO_TBL</td>
<td></td>
</tr>
<tr>
<td>FV_FO_GRAPHIC</td>
<td></td>
</tr>
<tr>
<td>FV_FO_EQN</td>
<td></td>
</tr>
<tr>
<td>FV_FO_TBL_ROW</td>
<td></td>
</tr>
</tbody>
</table>
Manipulating Text

Getting and setting the insertion point or text selection

The Frame API uses the document property `FP_TextSelection` to specify the insertion point or text selection in a document. This property specifies a text range, or `F_TextRangeT` structure, which is defined as:

```c
typedef struct {
    F_TextLocT beg; /* Beginning of the text range */
    F_TextLocT end; /* End of the text range */
} F_TextRangeT;
```

The `F_TextLocT` structure, which specifies a text location (a particular point in text), is defined as:

```c
typedef struct {
    F_ObjHandleT objId; /* Object that contains the text */
    IntT offset; /* Characters from beginning */
} F_TextLocT;
```

If a range of text is selected, `FP_TextSelection` specifies a selection; `F_TextRangeT.beg` and `F_TextRangeT.end` specify the beginning and end of the selection. If there is an insertion point, `FP_TextSelection` specifies an insertion point; `F_TextRangeT.beg` and `F_TextRangeT.end` are the same—both specify the location of the insertion point. If there is no text selection or insertion point, the `objId` and `offset` fields of both `F_TextRangeT.beg` and `F_TextRangeT.end` are set to 0.

For example, suppose the first five characters of the first paragraph on the page shown in Figure 6.1 are selected.

![Figure 6.1 Page with text frame containing two subcolumns](image)

The fields of the `F_TextRangeT` structure specified by the document’s `FP_TextSelection` property have the following values:
Manipulating Text

If no text is selected and the insertion point is at the very beginning of the paragraph, the fields of the `F_TextRangeT` structure have the following values:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>beg.objId</td>
<td>ID of the first paragraph</td>
</tr>
<tr>
<td>beg.offset</td>
<td>0</td>
</tr>
<tr>
<td>end.objId</td>
<td>ID of the first paragraph</td>
</tr>
<tr>
<td>end.offset</td>
<td>5</td>
</tr>
</tbody>
</table>

**IMPORTANT:** A valid text range can span multiple paragraphs, subcolumns, or text frames. It can’t span multiple flows, footnotes, table cells, or text lines.

It is possible for a document to have no text selection or insertion point at all. This can occur in the following circumstances:

- One or more graphic objects in the document are selected
- One or more entire table cells in the document are selected
- There is no selection of any type in the document

If a document has no text selection or insertion point at all, the fields of the `F_TextRangeT` structure have the following values:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>beg.objId</td>
<td>0</td>
</tr>
<tr>
<td>beg.offset</td>
<td>0</td>
</tr>
<tr>
<td>end.objId</td>
<td>0</td>
</tr>
<tr>
<td>end.offset</td>
<td>0</td>
</tr>
</tbody>
</table>

For more information on the different types of selection and the properties that represent it, see “How the API represents the selection in a document” on page 285.

To get and set a document’s insertion point (or text selection), use `F_ApiGetTextRange()` and `F_ApiSetTextRange()` to get and set its `FP_TextSelection` property.
The syntax for these functions is:

```c
F_TextRangeT F_ApiGetTextRange(F_ObjHandleT parentId,
                               F_ObjHandleT objId,
                               IntT propNum);

VoidT F_ApiSetTextRange(F_ObjHandleT parentId,
                        F_ObjHandleT objId,
                        IntT propNum,
                        F_TextRangeT *setVal);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>parentId</code></td>
<td>The ID of the object containing <code>objId</code>. If <code>objId</code> specifies a document ID, <code>parentId</code> should specify <code>FV_SessionId</code>. If <code>objId</code> specifies a flow, text frame, or table cell ID, <code>parentId</code> should specify the ID of the document that contains it.</td>
</tr>
<tr>
<td><code>objId</code></td>
<td>The ID of the object whose property you want to get or set. To get or set the insertion point or text selection in a document, specify the document’s ID.</td>
</tr>
<tr>
<td><code>propNum</code></td>
<td>The property to get or set. To get or set the insertion point or text selection in a document, set <code>propNum</code> to <code>FP_TextSelection</code>.</td>
</tr>
<tr>
<td><code>setVal</code></td>
<td>The text range to which to set the property.</td>
</tr>
</tbody>
</table>

The `beg.objId` and `end.objId` fields of the `F_TextRangeT` structure returned by `F_ApiGetTextRange()` always specify paragraph or text line IDs. The `beg.objId` and `end.objId` fields of the `F_TextRangeT` structure that you pass to `F_ApiSetTextRange()` can specify paragraph or text line IDs, but they can also specify flow, footnote, subcolumn, table cell, or text frame IDs.

For example, to set the insertion point at the beginning of the first paragraph on the page shown in Figure 6.1, you can use the following code:

```c
. . .
F_ObjHandleT docId, pgfId;
F_TextRangeT tr;
. . .
/* Get document and paragraph IDs here. */
. . .
/* Create text range that specifies an insertion point. */
tr.beg.objId = tr.end.objId = pgfId;
tr.beg.offset = tr.end.offset = 0;

/* Set document’s insertion point. */
F_ApiSetTextRange(FV_SessionId, docId, FP_TextSelection, &tr);
. . .
```

Instead of setting `tr.beg.objId` and `tr.end.objId` to the ID of the first paragraph, you can set them to the ID of the A flow, the text frame, or the left subcolumn. For example, the following code also sets the insertion point at the beginning of the first paragraph on the page shown in Figure 6.1:
The `beg.offset` and `end.offset` fields of the `F_TextRangeT` structure returned by `F_ApiGetTextRange()` always specify offsets relative to the beginning of a paragraph or text line object. The `beg.offset` and `end.offset` fields of the `F_TextRangeT` structure that you pass to `F_ApiSetTextRange()` can specify offsets relative to the beginning of an object, but they can also use the special value `FV_OBJ_END_OFFSET`. `FV_OBJ_END_OFFSET` specifies the offset of the last character in the object containing the text range. To specify offsets near the end of an object, you can add or subtract integers from `FV_OBJ_END_OFFSET`. For example, the following code selects the last five characters in a paragraph and the end of paragraph symbol:

```c
F_ObjHandleT docId, pgfId;
F_TextRangeT tr;
/* Get document and paragraph IDs here. */
tr.beg.objId = tr.end.objId = pgfId;
tr.beg.offset = FV_OBJ_END_OFFSET - 6;
tr.end.offset = FV_OBJ_END_OFFSET;
F_ApiSetTextRange(FV_SessionId, docId, FP_TextSelection, &tr);
```

The following code selects all the text in a cell:

```c
F_ObjHandleT docId, cellId;
F_TextRangeT tr;
/* Get document and cell IDs here. */
tr.beg.objId = tr.end.objId = cellId;
tr.beg.offset = 0;
tr.end.offset = FV_OBJ_END_OFFSET;
F_ApiSetTextRange(FV_SessionId, docId, FP_TextSelection, &tr);
```
Getting the text in a text range

To get the text in a specific text range, use `F_ApiGetTextForRange()`. The syntax for `F_ApiGetTextForRange()` is:

```c
F_TextItemsT F_ApiGetTextForRange(F_ObjHandleT docId,
                                 F_TextRangeT *tr,
                                 IntT flags);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document containing the text range.</td>
</tr>
<tr>
<td>tr</td>
<td>The text range containing the text you want to get.</td>
</tr>
<tr>
<td>flags</td>
<td>Bit flags that specify the type of text items to retrieve. For a complete list of the constants that represent text item types, see “F_ApiGetText()” on page 164 of the FDK Programmer’s Reference.</td>
</tr>
</tbody>
</table>

For example, the following code gets the selected text in the active document:

```c
...  
F_ObjHandleT docId;  
F_TextRangeT tr;  
F_TextItemsT tis;  

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);  
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);  
/* If there’s just an insertion point, no text is selected. */  
if(tr.beg.objId == tr.end.objId  
   && tr.beg.offset == tr.end.offset) return;  

tis = F_ApiGetTextForRange(docId, &tr, FTI_String);  
...  
```

Getting and setting table selections

If a table contains cells that are selected, you can get the table’s ID by querying the document property, `FP_SelectedTbl`. For more information, see “Getting the IDs of selected tables and table rows” on page 87.

If a range of text that includes several tables is selected, you can get the tables’ IDs by calling `F_ApiGetText()` and retrieving the `FTI_TblAnchor` text items for the selection. Each `FTI_TblAnchor` text item specifies the ID of a table.

To make the selection in a document include several tables, set the text selection so that it includes the text that contains the tables’ anchors. To make the selection include specific rows and columns within a single table, call `F_ApiMakeTblSelection()`. For more information, see “F_ApiMakeTblSelection()” on page 201 of the FDK Programmer’s Reference.
Element ranges in structured tables

If the current element range is within a cell, or if it indicates a selected table part, you can get the table's ID by querying the document property, FP_SelectedTbl. However, it's possible for a client to set the current element range to a point between table part elements. In this case, the document property, FP_SelectedTbl is NULL. For this reason, you cannot always use FP_SelectedTbl to determine whether the current element range is in a table.

If FP_SelectedTbl returns NULL, the following code determines whether the current element location is within a table, as well as the type of the parent element:

```c
F_ObjHandleT docId;
F_PropValT propVal;
F_ElementRangeT er;

/* Get the ID of the active document. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
if(!docId) return;
er = F_ApiGetElementRange(FV_SessionId, docId, FP_ElementSelection);

/* First check to see if there is a selected table. */
propVal = F_ApiGetPropVal(FV_SessionId, docId, FP_SelectedTbl);
if (!propVal.propVal.u.ival) {
    /* Now determine whether the current element range is within a table. */
    propVal = F_ApiGetPropVal(docId, er.beg.parentId, FPElementType);
    if(propVal.propVal.u.ival == FV_FO_TBL) {
        F_Printf(NULL,(StringT)"You are in a table\n");
    } else if(propVal.propVal.u.ival == FV_FO_TBL_TITLE) {
        F_Printf(NULL,(StringT)"You are in a table title\n");
    } else if(propVal.propVal.u.ival == FV_FO_TBL_HEADING) {
        F_Printf(NULL,(StringT)"You are in a table heading\n");
    } else if(propVal.propVal.u.ival == FV_FO_TBL_BODY) {
        F_Printf(NULL,(StringT)"You are in a table body\n");
    } else if(propVal.propVal.u.ival == FV_FO_TBL_FOOTING) {
        F_Printf(NULL,(StringT)"You are in a table footing\n");
    } else if(propVal.propVal.u.ival == FV_FO_TBL_ROW) {
        F_Printf(NULL,(StringT)"You are in a table row\n");
    } else {
        F_Printf(NULL,(StringT)"You are not in a table at all\n");
    }
}

/* Be sure to deallocate memory for the property value. */
F_ApiDeallocatePropVal(&propVal);
```
Getting and setting the structural element selection

Although you can get and set selected structural elements in a FrameMaker document by getting and setting the text selection, it is usually easier to use the following functions:

- F_ApiGetElementRange() gets the structural element selection in a document or book.
- F_ApiSetElementRange() sets the structural element selection in a document or book.

The syntax for these functions is:

```c
F_ElementRangeT F_ApiGetElementRange(
    F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum);

VoidT F_ApiSetElementRange(
    F_ObjHandleT docId,
    F_ObjHandleT objId,
    IntT propNum,
    F_ElementRangeT *setVal);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>docId</code></td>
<td>The object containing <code>objId</code>. To get or set the element selection in a document, specify <code>FV_SessionId</code>.</td>
</tr>
<tr>
<td><code>objId</code></td>
<td>The ID of the document or book in which you want to get or set the element selection.</td>
</tr>
<tr>
<td><code>propNum</code></td>
<td>The property to get or set. To get or set the element selection, specify <code>FP_ElementSelection</code>.</td>
</tr>
<tr>
<td><code>setVal</code></td>
<td>The element range to set the property to.</td>
</tr>
</tbody>
</table>

The `F_ElementRangeT` structure is defined as:

```c
typedef struct {
    F_ElementLocT beg; /* Beginning of the element range. */
    F_ElementLocT end; /* End of the element range. */
} F_ElementRangeT;
```

The `F_ElementLocT` structure specifies a location within an element. It is defined as:

```c
typedef struct {
    F_ObjHandleT parentId; /* Parent element ID. */
    F_ObjHandleT childId; /* Child element ID. */
    IntT offset; /* Offset within child/parent element. */
} F_ElementLocT;
```

For information on how FrameMaker sets the fields of the `F_ElementRangeT` structure specified by `FP_ElementSelection` to represent different types of selection, see “How the API represents the element selection in a structured FrameMaker document” on page 286. For examples of how to get and set element selections, see “F_ApiGetElementRange()” on page 113 and “F_ApiSetElementRange()” on page 282 of the FDK Programmer’s Reference.
To traverse the selection returned by `F_ApiGetElementRange()`, traverse the child elements of the element specified by `beg.childId`. Then traverse its sibling elements and all of their child elements until you reach the element specified by `end.childId`. To traverse an element’s child elements, you query its `FP_FirstChildElement` property and then query each child element’s `FP_NextSiblingElement` property. To traverse an element’s siblings, you query its `FP_NextSiblingElement` property and then query each sibling element’s `FP_NextSiblingElement` property.

### Adding and deleting text

To add and delete text, use `F_ApiAddText()` and `F_ApiDeleteText()`.

The syntax for these functions is:

```c
F_TextLocT F_ApiAddText(F_ObjHandleT docId,
    F_TextLocT *textLocp,
    StringT text);

IntT F_ApiDeleteText(F_ObjHandleT docId,
    F_TextRangeT *textRangep);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document to which you’ll add text</td>
</tr>
<tr>
<td>textLocp</td>
<td>The point in text (text location) at which you’ll add text</td>
</tr>
<tr>
<td>text</td>
<td>The text to add</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document to delete text from</td>
</tr>
<tr>
<td>textRangep</td>
<td>The text range to delete</td>
</tr>
</tbody>
</table>

`F_ApiAddText()` returns the text location at the end of the text that was added. `F_ApiDeleteText()` deletes any objects, such as tables and markers, anchored in the text it deletes.

To add text to, or delete text from, a text inset, you must first unlock it by setting its `FP_TiLocked` property to `False`. After you are done adding or deleting text, relock the inset by setting its `FP_TiLocked` property to `True`.

To specify special characters, line breaks, or paragraph breaks when you add text, use octal codes within the text string. For example, to specify an em dash, use `\321`. For more information on special characters, see “How the API represents special characters” on page 316. For a list of the characters in the FrameMaker product character set and the corresponding codes, see “Character Sets” in your FrameMaker product user documentation.
Example

The following code adds some text at the insertion point (or the beginning of the current text selection) and then deletes it. The text has a dagger (†) at the end of it.

```c
F_TextLocT trm;
F_TextRangeT tr;
F_ObjHandleT docId;

/* Get current text selection. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId,
                        FP_TextSelection);

/* Return if there is no selection or IP. */
if(!tr.beg.objId) return;

/* Insert text at insertion point or beginning of selection.
 * Use the octal code 240 to display the dagger.
 */
trm = F_ApiAddText(docId, &tr.beg, "Here's some text.\240");
F_ApiAlert("Now we'll delete it.", FF_ALERT_CONTINUE_NOTE);

/* Set tr to end at end of the added text. Then delete it. */
tr.end.offset = trm.offset;
F_ApiDeleteText(docId, &tr);
```

Adding text to table cells

To add text to a table cell, you must first get the ID of the cell. To do this, you traverse from the table to the row containing the cell, and then to the cell. Once you have the ID of the cell that you want to add text to, you add text to it by calling `F_ApiAddText()`.

The following code adds some text to the first cell in the first row of the selected table in the active document:

```c
F_TextLocT ip;
F_ObjHandleT docId, tblId, rowId, cellId;

/* Get the document and selected table IDs. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tblId = F_ApiGetId(FV_SessionId, docId, FP_SelectedTbl);

/* Get the ID of the first row and cell. */
rowId = F_ApiGetId(docId, tblId, FP_FirstRowInTbl);
cellId = F_ApiGetId(docId, rowId, FP_FirstCellInRow);
```
/* Set up text location at beginning of cell. */
ip.objId = cellId;
ip.offset = 0;

F_ApiAddText(docId, &ip, "This text appears in the cell.");

For an example of how to create a table and add text to its title, see “Creating tables” on page 154.

---

**Getting and setting text formatting**

Although the API doesn’t represent text as objects, the characters in text have properties. Each character has a property list describing its font, color, condition tags, and other character formatting attributes. The API provides special functions to get and set the properties in this list.

You can also get and set text formatting by getting and setting paragraph, paragraph format, and character format properties. For more information, see “Getting and setting paragraph properties” on page 102.

**Getting text properties**

To get an individual property for a character, use `F_ApiGetTextPropVal()`. To get the entire list of text properties for a character, use `F_ApiGetTextProps()`.

The syntax for these functions is:

```c
F_PropValT F_ApiGetTextPropVal(F_ObjHandleT docId,
                              F_TextLocT *textLocp,
                              IntT propNum);

F_PropValsT F_ApiGetTextProps(F_ObjHandleT docId,
                               F_TextLocT *textLocp);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document containing the character.</td>
</tr>
<tr>
<td>textLocp</td>
<td>The text location of the character that you want to get text properties for. The returned properties are the properties that apply to the character to the right of the specified location.</td>
</tr>
<tr>
<td>propNum</td>
<td>The text property, such as <code>FP_FontFamily</code> or <code>FP_FontSize</code>, that you want to get.</td>
</tr>
</tbody>
</table>

The API allocates the returned properties. Use `F_ApiDeallocatePropVal()` or `F_ApiDeallocatePropVals()` to free the properties when you’re done with them.
You can get the text properties for only one character at a time, because they can be different for each character. For more information on how the API represents characters and text properties, see “How the API represents text” on page 312.

Example

The following code gets the name of the character tag for the character to the right of the insertion point:

```c
F_TextRangeT tr;
F_PropValT prop;
F_ObjHandleT docId;

/* Get the current insertion point. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if(!tr.beg.objId) return;

prop = F_ApiGetTextPropVal(docId, &tr.end, FP_CharTag);

F_Printf(NULL,"The character tag is %s.\n",
        prop.propVal.u.sval);
```

Setting text properties

To set the text properties for a text range, use `F_ApiSetTextPropVal()` or `F_ApiSetTextProps()`.

The syntax for these functions is:

```c
VoidT F_ApiSetTextPropVal(F_ObjHandleT docId,
            F_TextRangeT *textRangep,
            F_PropValT *setVal);

VoidT F_ApiSetTextProps(F_ObjHandleT docId,
            F_TextRangeT *textRangep,
            F_PropValsT *setVal);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document containing the text</td>
</tr>
<tr>
<td>textRangep</td>
<td>The text range</td>
</tr>
<tr>
<td>rval</td>
<td>The property or property list to apply to the text range</td>
</tr>
</tbody>
</table>

Applying a character format to text

To apply a character format to a text range, copy the property list of the `FO_CharFmt` object that represents the character format to the text range. For example, to apply the character format named Emphasis to the current text selection, use the following code:
Manipulating Text

...  
F_TextRangeT tr;
F_PropValsT props;
F_ObjHandleT docId, charFmtId;
IntT i;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
if (!docId) return;

/* Get current text selection. Return if there isn’t one.*/
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if (!tr.beg.objId) return;

/* Get Emphasis properties. */
charFmtId = F_ApiGetNamedObject(docId, FO_CharFmt, "Emphasis");

props = F_ApiGetProps(docId, charFmtId);

/* Apply properties to selection. */
F_ApiSetTextProps(docId, &tr, &props);

This code has the same effect as choosing Emphasis in the Character Catalog. If no text is selected, the code has no effect.

Changing individual text properties

If you need to apply only an individual property to a text range, use F_ApiSetTextProp(). For example, the following code changes the font family of the selected text to AvantGarde:

...  
#include "fstrings.h"
F_TextRangeT tr;
F_PropValT prop;
F_ObjHandleT docId;
UIntT i = 0;
F_StringsT strings;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if (!tr.beg.objId) return;

/* Get list of font families available in current session. */
strings = F_ApiGetStrings(0, FV_SessionId, FP_FontFamilyNames);

/* Find index of AvantGarde in list of families in session. */
for (i=0; i<strings.len && !F_StrEqual("AvantGarde",strings.val[i]); i++);
    if (i == strings.len) return; /* Font not found. */

/* Free the returned strings. */
F_ApiDeallocateStrings(&strings);
/* Set up property. Set it to the index for AvantGarde. */
prop.propIdent.num = FP_FontFamily;
prop.propVal.valType = FT_Integer;
prop.propVal.u.ival = i;

/* Apply the property to the text selection. */
F_ApiSetTextPropVal(docId, &tr, &prop);

Applying conditions to text

The API uses the text property FP_InCond to specify the conditions applied to a text location. FP_InCond specifies an array that includes the IDs of conditions that apply to the text location.

To apply conditions to a text range, set the FP_InCond property for the text range. For example, the following code applies the Comment condition to the selected text:

```c
F_TextRangeT tr;
F_PropValT prop;
F_ObjHandleT docId, commentId;
F_IntsT condIds;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if (!tr.beg.objId) return;

/* Get the ID of the Comment condition. */
commentId = F_ApiGetNamedObject(docId, FO_CondFmt, "Comment");

/* Set up array of conditions (just one, in this case). */
condIds.val = (IntT*) &commentId;
condIds.len = 1;

/* Set up property. */
prop.propIdent.num = FP_InCond;
prop.propVal.valType = FT_Ints;
prop.propVal.u.isval = condIds;

/* Apply the property to the text selection. */
F_ApiSetTextPropVal(docId, &tr, &prop);
```

Setting type-in properties

A Frame document has a set of properties called type-in properties, which specify the text characteristics of the insertion point in the document. Type-in properties do not apply to text that is already in a document; they apply only to text as the user types it in the document.

Whenever the user changes the insertion point in a document, the FrameMaker product sets the document’s type-in properties to match the text properties of the character to the left of the
insertion point. However, a document’s type-in properties do not have to match the text properties of the character to the left of the insertion point. You can change them with the F_ApiSetPropertyType() and F_ApiSetProps() functions. For example, the following code sets the active document’s type-in properties so that text the user types appears in uppercase and is underlined:

```c
F_ObjHandleT docId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
F_ApiSetInt(FV_SessionId, docId,
    FP_Capitalization, FV_CAPITAL_CASE_UPPER);
F_ApiSetInt(FV_SessionId, docId,
    FP_Underlining, FV_CB_SINGLE_UNDERLINE);
```

---

**Executing Clipboard functions**

The API provides functions that programmatically execute Frame Clipboard functions. It also provides a Clipboard stack, which allows you to save the Clipboard contents.

**Executing Frame Clipboard functions**

The API provides the following functions to programmatically execute Frame Clipboard functions:

- `F_ApiCut()` cuts the current selection to the Clipboard.
- `F_ApiCopy()` copies the current selection to the Clipboard.
- `F_ApiPaste()` pastes Clipboard contents to the insertion point or current selection.
- `F_ApiClear()` clears the current selection.

All these functions work only on the active document. They use the Frame Clipboard and the current selection (or insertion point) in a document. They work with text, table cells, and graphic objects. You can allow the user to set the selection or insertion point, or you can do it programmatically.

The syntax for the functions is:

```c
IntT F_ApiCut(F_ObjHandleT docId, IntT flags);
IntT F_ApiCopy(F_ObjHandleT docId, IntT flags);
IntT F_ApiPaste(F_ObjHandleT docId, IntT flags);
IntT F_ApiClear(F_ObjHandleT docId, IntT flags);
```
Manipulating Text

Specifying 0 for flags instructs these functions to behave in the following ways:

- F_ApiCut() and F_ApiClear() leave selected table cells empty and delete hidden text.
- F_ApiPaste() inserts table columns to the left of the current columns and rows above the current row.
- All functions suppress any Frame dialog boxes or alert boxes that arise.

To specify the behavior of the functions, you can OR the following values into the flags argument.

<table>
<thead>
<tr>
<th>This value</th>
<th>Means</th>
<th>Applies to</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF_INTERACTIVE</td>
<td>Prompt user with dialog boxes or alert boxes that arise.</td>
<td>All Clipboard functions</td>
</tr>
<tr>
<td>FF_CUT_TBL_CELLS</td>
<td>Remove cut or cleared table cells.</td>
<td>F_ApiClear() and F_ApiCut()</td>
</tr>
<tr>
<td>FF_DONT_DELETE_HIDDEN_TEXT</td>
<td>Don’t delete hidden text.</td>
<td>F_ApiClear(), F_ApiCut(), and F_ApiPaste()</td>
</tr>
<tr>
<td>FF_INSERT_BELOW_RIGHT</td>
<td>Add columns to the right of the current column or below the current row.</td>
<td>F_ApiPaste()</td>
</tr>
<tr>
<td>FF_VISIBLE_ONLY</td>
<td>Cut, copy, paste, or clear only the visible portion of the selection.</td>
<td>All Clipboard functions</td>
</tr>
<tr>
<td>FF_REPLACE_CELLS</td>
<td>Replace selected cells with cells on the Clipboard.</td>
<td>F_ApiPaste()</td>
</tr>
<tr>
<td>FF_DONT_APPLY_ALL_ROWS</td>
<td>Don’t apply condition setting on the Clipboard to all rows. If whole table is selected and Clipboard contains condition setting, cancel the paste.</td>
<td>F_ApiPaste()</td>
</tr>
</tbody>
</table>

The FF_INTERACTIVE flag takes precedence over other flags. So, if you specify FF_INTERACTIVE | FF_DONT_DELETE_HIDDEN_TEXT and the selection contains hidden text, F_ApiClear() and F_ApiCut() will leave the selected cells empty and delete hidden text, while F_ApiPaste() will insert columns to the left of the current columns and rows above the current row.
text, the FrameMaker product prompts the user and allows the user to choose whether to delete the hidden text.

**Saving the Clipboard contents**

In some cases, you may want to use the Clipboard and then restore its original contents when you are done. The API provides a Clipboard stack, which allows you to do this. To manipulate the Clipboard stack, use F_ApiPushClipboard() and F_ApiPopClipboard(). The syntax for these functions is:

```c
IntT F_ApiPushClipboard(VoidT);
IntT F_ApiPopClipboard(VoidT);
```

F_ApiPushClipboard() pushes the current Clipboard contents onto the Clipboard stack. F_ApiPopClipboard() pops the set of Clipboard contents on the top of the Clipboard stack to the Clipboard.

For example, the following code executes Copy and Paste operations and then restores the original Clipboard contents:

```c
F_ObjHandleT docId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
F_ApiPushClipboard();
F_ApiCopy(docId, 0);
F_ApiPaste(docId, 0);
F_ApiPopClipboard();

...
This chapter describes how to use the API to manipulate Asian text in Frame documents. Specifically, it discusses:

- Creating a rubi group
- The text encodings supported by FrameMaker
- How to use encoding data in an API client
- Inspecting and manipulating encoded text
- Parsing encoded strings
- Getting the encoding for a text item
- Special issues such as decimal tabs, character offsets, and internal strings

To better understand the material in this chapter, you may want to learn more about how the API represents text. For information on this subject, see “Text” on page 311.

Creating a rubi group

Users create rubi groups by selecting the oyamoji text and choosing Rubi from the Special menu. Internally, a rubi group is stored as an anchored object. To create a rubi group via the API, use `F_ApiNewAnchoredObject()`.

Note that `F_ApiNewAnchoredObject()` accepts a text location, but not a selection as an argument. This changes the process for creating a group slightly:

- Identify the location for the rubi group
- Get the oyamoji location
- Insert the oyamoji text
- Get the rubi location
- Insert the rubi text

Once you have a rubi group, you can get the text range for the oyamoji and rubi text. In this way, you can edit the rubigroup contents. The following example shows how to create a new rubi group:

```c
F_TextRangeT tr, oyamojiTr, rubiTr;
F_ObjHandle docId, rubiGrpId;
StringT s1, s2;

/* Assuming known contents in s1 and Hiragana chars in s2... */
```
Manipulating Asian Text

```c
    docId = F_ApiGetID(0, FV_SessionId, FP_ActiveDoc);
    tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
    if (tr.beg.objId == 0) return;

    /* Insert the Rubi Group */
    rubiGrpId = F_ApiNewAnchoredObject(docId, FO_Rubi, &tr.beg);
    /* Get the location of the oyamoji and add the text. */
    oyamojiTr = F_ApiGetTextRange(docId, rubiGrpId, FP_OyamojiTextRange);
    F_ApiAddText(docId, &oyamojiTr.beg, s1);
    /* Get the location of the rubi and add the text. */
    rubiTr = F_ApiGetTextRange(docId, rubiGrpId, FP_RubiTextRange);
    F_ApiAddText(docId, &rubiTr.beg, s2);
    ...
```

Text encodings

Asian character sets include thousands of characters, and so the single byte range used for
Roman character sets is insufficient. Asian character sets use single and double byte codes to
identify each character. They also reserve the same single byte encodings for nearly every
character in the 7-bit ASCII range. Any differences among characters in the ASCII range are
relatively insignificant.

For example, Shift-JIS (an encoding for Japanese characters) replaces the ASCII characters
"|", "~", and "\" with a solid vertical bar, and overbar, and the Yen symbol, respectively.
Pathnames using the "\" character in a Roman encoding would appear with the Yen symbol in
Shift-JIS. However, the pathname would still parse correctly because the character codes are
the same.

For text that must be shared across platforms and encodings, it is best to stay within the 7-bit
ASCII range. Filenames are a good example of text that should follow this rule. Also, keeping
within 7-bit ASCII for tag names (paragraph format tags, character format tags, etc.) is a good
way to ensure a document will be usable on systems that support different languages.

Encoding schemes

There are a number of encoding schemes that map the codes to individual characters. Not only
are there different encodings for each language, but within a language there might be a number
of encodings. FrameMaker can save and import text in a number of encodings:
Of these encodings, FrameMaker uses the following to represent characters internally. The following strings are the names Framemaker uses to identify these internal encodings:

<table>
<thead>
<tr>
<th>Language</th>
<th>FrameMaker internal encoding names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roman</td>
<td>FrameRoman</td>
</tr>
<tr>
<td>Japanese</td>
<td>JISX0208,ShiftJIS</td>
</tr>
<tr>
<td>Traditional Chinese</td>
<td>Big5</td>
</tr>
<tr>
<td>Simplified Chinese</td>
<td>GB2312-80,EUC</td>
</tr>
<tr>
<td>Korean</td>
<td>KSC5601-1992</td>
</tr>
</tbody>
</table>

**FrameMaker Roman encoding**

8-bit Roman character sets all share the same 7-bit ASCII characters. FrameMaker also uses the characters in the x80 - xFF range for special characters such as non-breaking hyphens or em spaces. Asian fonts cannot be expected to support the same special characters. As a result, if the `FP_DialogEncodingName` is set to an Asian encoding, the user cannot type these special characters in text boxes or other parts of the user interface that are controlled by this setting. However, the user can type these characters in document text whenever the current font is a Roman font that supports them.
Using encoding data

The FDE provides functions to operate on strings and characters of a specified internal encoding. To use font encoding information, you must first initialize the font encoding data. Then you can get the internal encoding for a given character or string, set the encoding, or convert from the FrameMaker internal encoding to a different one; from Shift-JIS to EUC, for example.

Initializing encoding data and setting the U/I encoding

The API uses \texttt{F\_FdeInitFontEncs()} to:

- initialize the font encoding data
- set the encoding for your client’s user interface.

Initializing the encoding data sets up structures to represent each internal encoding supported by the current release of the FrameMaker product. This is true even if the current session doesn’t support the languages those encodings represent. For example, you could process a string of Korean text with the FDE, even if the current session of FrameMaker would not be able to display it correctly.

The typical way to use this function is to initialize the FDE and then get the encoding name used for the current FrameMaker session. Then you pass that encoding name to \texttt{F\_FdeInitFontEncs()} so your client will use the same encoding for its dialog boxes.

The syntax for \texttt{F\_FdeInitFontEncs()} is:

\begin{verbatim}
FontEncIdT F\_FdeInitFontEncs(ConStringT fontEncName);
\end{verbatim}

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{fontEncName}</td>
<td>The name of the font encoding to use for your client’s dialog boxes.</td>
</tr>
</tbody>
</table>

Possible values for \texttt{fontEncName} are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>FrameRoman</td>
<td>Roman text</td>
</tr>
<tr>
<td>JISX0208.ShiftJIS</td>
<td>Japanese text</td>
</tr>
<tr>
<td>BIG5</td>
<td>Traditional Chinese text</td>
</tr>
<tr>
<td>GB2312-80.EUC</td>
<td>Simplified Chinese text</td>
</tr>
<tr>
<td>KSC5601-1992</td>
<td>Korean text</td>
</tr>
</tbody>
</table>

The returned \texttt{FontEncIdT} is the ID of the font encoding you specified for your dialog boxes.
Example
The following code initializes the FDE and ensures the dialog box encoding is one the client can support. If the dialog box encoding for the current session is Japanese or Simplified Chinese, it passes that encoding to F_FdeInitFontEncs(). Otherwise, it passes Roman to F_FdeInitFontEncs():

```c
FontEncIdT feId;
StringT encName;

F_FdeInit();
encName = F_ApiGetString(0, FV_SessionId, FP_DialogEncodingName);
if (F_StrIEqual(encName, "JISX0208.ShiftJIS") ||
    F_StrIEqual(encName, "GB2312-80.EUC")
    feId = F_FdeInitFontEncs((ConStringT) encName);
else
    feId = F_FdeInitFontEncs((ConStringT) "FrameRoman");
```

Getting the encoding for fonts
Font families and individual fonts have associated encodings. The possible encodings are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>FrameRoman</td>
<td>Roman text</td>
</tr>
<tr>
<td>JISX0208.ShiftJIS</td>
<td>Japanese text</td>
</tr>
<tr>
<td>BIG5</td>
<td>Traditional Chinese text</td>
</tr>
<tr>
<td>GB2312-80.EUC</td>
<td>Simplified Chinese text</td>
</tr>
<tr>
<td>KSC5601-1992</td>
<td>Korean text</td>
</tr>
<tr>
<td>Multiple</td>
<td>More than one encoding for the font family</td>
</tr>
</tbody>
</table>

If the returned encoding is Multiple, the font family has different encodings for its different variations. In that case, you must get the encoding for each variation. Non-text fonts may return FrameRoman, or they may return the family name of the font. For example, on some platforms the encoding for the Symbol font family is indicated by the string Symbol.

Getting the encoding for a font family
To get the encoding for a font family, first get a list of font families, then loop through that list to get the index of the family you want. Then you pass the index to F_ApiGetEncodingForFamily().

The syntax for F_ApiGetEncodingForFamily() is:
StringT F_ApiGetEncodingForFamily(IntT family);

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>family</td>
<td>The font family for which you want to know the encoding.</td>
</tr>
</tbody>
</table>

**Example**

The following code gets the index for the Minchu font family from the session list of font families. It then gets the encoding for that font family:

```c
#include "futils.h"
#include "fstrings.h"
#include "fencode.h"
...
F_StringsT families;
StringT encoding;
UIntT i;
/* First get the list of font families for the session */
families = F_ApiGetStrings(0, FV_SessionId, FP_FontFamilyNames);
/* Now get the index of the Minchu family */
for (i=0; i < families.len; i++)
  if (F_StrIEqual(families.val, (StringT) "minchu")) break;
if (i == families.len) return; /* Minchu not found */
/* Now use the index to get the encoding for Minchu */
encoding = F_ApiGetEncodingForFamily(i);
...
/* Free the strings */
F_ApiDeallocateStrings(&families);
F_ApiDeallocateString(&encoding);
```

**Getting the encoding for a font variation**

If the font family returns an encoding of Multiple, you should use F_ApiFamilyFonts() to get a list of the variations for the family. Then you can use F_ApiGetEncodingForFont() to get the encoding for a specific variation.

The syntax for F_ApiGetEncodingForFont() is:

```c
StringT F_ApiGetEncodingForFont(F_FontT *font);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>font</td>
<td>Pointer to a structure listing the font’s name, weight, angle, and variation</td>
</tr>
</tbody>
</table>

**Example**

The following code loops through the session fonts, then loops through the permutations of each and prints the encoding for each permutation to the console:

```c
... F_FontsT fam;
F_StringsT families, weights, variations, angles;
```
StringT encoding;
UIntT i, j;

/* Get lists of families, variations, weights, and angles. */
families = F_ApiGetStrings(0, FV_SessionId, FP_FontFamilyNames);
weights = F_ApiGetStrings(0, FV_SessionId, FP_FontWeightNames);
variations = F_ApiGetStrings(0, FV_SessionId,
    FP_FontVariationNames);
angles = F_ApiGetStrings(0, FV_SessionId,FP_FontAngleNames);

/* Loop through each session font */
for (i=0; i < families.len; i++) {
    /* Now print the encoding for each variation to the console */
    fam = F_ApiFamilyFonts(i);
    for (j = 0; j < fam.len; j++) {
        encoding = F_ApiGetEncodingForFont(fam.val[j]);
        F_Printf(NULL, "The encoding for %s-%s-%s-%s is %s\n"
            families.val[fam.val[j].family],
            weights.val[fam.val[j].weight],
            variations.val[fam.val[j].variation],
            angles.val[fam.val[j].angle],
            encoding);
        F_ApiDeallocateString(&encoding);
    }
}
/* Be sure to free the structures and strings */
...

Determining which encodings are currently supported

The API has two functions to determine which encodings are supported for the current session.
F_ApiIsEncodingSupported() returns True if the passed encoding is currently supported. F_ApiGetSupportedEncodings() returns a F_StringsT list of all the encodings supported for the current session.

The syntax for F_ApiIsEncodingSupported() is:
BoolT F_ApiIsEncodingSupported(ConStringT encodingName);

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>encodingName</td>
<td>The encoding of interest. Possible values are: FrameRoman JISX0208.ShiftJIS BIG5 GB2312-80.EUC KSC5601-1992</td>
</tr>
</tbody>
</table>

The syntax for F_ApiGetSupportedEncodings() is:
F_StringsT F_ApiGetSupportedEncodings();
Inspecting and manipulating encoded text

For text of a given encoding, you can perform actions such as string comparison, search for the occurrence of a character in a string, character count, truncation, concatenation, and others. The functions to perform these actions are much like the corresponding string functions for Roman text. However, they must be passed an encoding ID so they can check a character code against the encoding.

For example, a single byte code might be a single byte character in one encoding, while in another encoding it might be the first or last byte of a double byte character. The API provides functions to determine exactly that.

Getting encoding IDs

When you initialize the encoding data, the FDE assigns an ID to each encoding data structure. String functions that use this encoding data generally require the ID to identify the encoding.

The FDE includes functions to get the encoding ID assigned to an encoding name, and to get the encoding name that is associated with a given encoding ID.

F_FontEncId() returns the FontEncIdT for the encoding data associated with the specified encoding name. If the encoding name is not supported for the current session, this function returns the ID for the FrameRoman encoding.

The syntax for F_FontEncId() is:

FontEncIdT F_FontEncId(ConStringT fontEncName);

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>fontEncName</td>
<td>The encoding of interest. Possible values are: FrameRoman, JISX0208.ShiftJIS, BIG5, GB2312-80.EUC, KSC5601-1992, Multiple</td>
</tr>
</tbody>
</table>

F_FontEncName() returns the encoding name associated with the specified FontEncIdT. If the specified FontEncIdT is not valid, this function returns a NULL string.

The syntax for F_FontEncName() is:

ConStringT F_FontEncName(FontEncIdT fontEncId);

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>fontEncId</td>
<td>The encoding ID of interest</td>
</tr>
</tbody>
</table>
Functions for encoded strings

The FDE includes the following functions for handling strings of a given encoding. For more information, see these functions in the *FDK Programmers Reference*.

- `F_StrChrEnc()`
- `F_StrRChrEnc()`
- `F_StrStrEnc()`
- `F_StrIEqualEnc()`
- `F_StrIEqualNEnc()`
- `F_StrICmpEnc()`
- `F_StrCmpNEnc()`
- `F_StrTruncEnc()`
- `F_StrLenEnc()`
- `F_StrCatDb1CharNEnc()`
- `F_StrIPrefixEnc()`
- `F_StrISuffixEnc()`
- `F_StrCatNEnc()`
- `F_StrNCatNEnc()`
- `F_StrCpyNEnc()`

Parsing an encoded string

For the characters in a given string, you might need to know whether a character is single byte, whether a single byte is the first or last byte of a double byte character, or whether two consecutive bytes comprise a valid double byte character. The API has the following functions that map the given byte or bytes to the specified encoding:

- `BoolT F_CharIsDoubleByteFirst(UCharT char, FoneEncIdT feId);`
- `BoolT F_CharIsDoubleByteSecond(UCharT char, FoneEncIdT feId);`
- `BoolT F_CharIsDoubleByte(UCharT firstChar, UCharT secondChar, FoneEncIdT feId);`

Example

The following code checks each character in a string to see whether it is one byte or two and increments by the correct amount:

```c
... StringT dbEncString;
UCharT currChar;
IntT i = 0;
FontEncIdT feId;
feId = F_FontEncId((ConStringT) "JISX0208.ShiftJIS");
...
/* Assume there is a Japanese string in dbEncString... * We do not need to also check if the second byte is null, * as all of the supported encodings do not have '\0' in their
```
Getting the encoding for a text item

The function F_ApiGetText() returns a F_TextItemsT structure, which is a list of text items that makes up a range of document text. Note that among other things, a single text item can represent a string of characters with common text properties. For a change in character encoding to occur, there must be a corresponding change in some text property such as a new font family or font variation. This means that any text item that is a string must be a string of a given encoding. For more information about text items, see “Text” on page 311.

For a given text item, you can use the offset to determine a text location corresponding to that text item. You can then use F_ApiGetTextPropVal() to get the font at that location. Given the font, you can use F_ApiGetEncodingForFont() to determine the encoding at that location. You now know the encoding for a given text item.

As you scan the items in a F_TextItemsT structure, you can flag changes to the text encoding. Until you see a change in text encoding, you can assume any string text items are of the current encoding.

Keep in mind that a change of text encoding necessarily occurs at a change in character properties. This is indicated in the F_TextItemT as a data type of FTI_CharPropsChange. The data for an FTI_CharPropsChange is a flag to indicate the type of change. If the flag indicates FTF_ENCODING, you know the encoding has changed, and you must get the encoding for the next string text item. For more information, see “FTI_CharPropsChange” on page 314 and “FTF_ENCODING” on page 315.

Special issues with double byte encodings

Following are some special issues to keep in mind when working with double byte text.

* valid range.
*/
while (dbEncString[i] != '\0') {
    if (F_CharIsDoubleByte(dbEncString[i], dbEncString[i+1], feId))
    {
        /* This is a double byte character... */
        i = i + 2;
    }
    else {
        /* This is either a true single byte char,
        * or the second byte was not a double-byte second,
        * so treat as a single char for proper scanning.
        */
        i++;
    }
}
Decimal tabs

The F_TabT data structure describes an individual tab stop. It includes a field for a character (such as a period or a comma) for decimal-aligned tab stops. FrameMaker only supports single byte characters to align tab stops.

Offsets into strings

Unless otherwise noted in the FDK Programmers Reference, string functions that return an offset into the string express the offset in terms of characters, and not bytes. This should keep your existing code viable. For example, code to set text locations and text ranges should still work, even for text that contains double byte text.

To get the count of characters in an encoded string, use F_StrLenEnc(). This function returns the number of characters in the string, even though some characters might be single byte and others might be double byte. (Remember that double byte encodings reserve single byte space for certain characters.) The syntax for F_StrLenEnc() is:

\[
\text{IntT F_StrLenEnc(ConStringT s, FontEncIdT feid)};
\]

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>The string whose characters you want to count</td>
</tr>
<tr>
<td>feId</td>
<td>The ID of the encoding for s</td>
</tr>
</tbody>
</table>

Internal strings in FrameMaker

Internal strings such as encoding names, marker text, or FrameMaker product client names all use text in the 7-bit ASCII range. Tags (paragraph format tags, for example) can use double byte text. However, for FrameMaker documents the tag names of elements cannot use double byte text.

Text in an unsupported encoding

A document can include text in an encoding that is not supported by the current system configuration. In this case, FrameMaker retains the encoding identification with the text even though it can’t display the text correctly. This is referred to as a ghost encoding.

For example, suppose a document contains Japanese text, but the system can only display Western text. The Japanese text appears as a series of arbitrary characters, each character corresponding to a single byte of what might be a double-byte or single-byte Japanese character. If you get the encoding for that text, the FDE will return JISX0208.ShiftJIS.

Note that ghost encoded text is displayed as though it is FrameRoman text. This is important because it is likely that line breaks will split up double-byte characters. Furthermore, functions like F_CharIsDoubleByteFirst() and F_CharIsDoubleByteSecond() return
unreliable results. For this reason, you have no way to ensure ghost encoded text items are valid, and we suggest you specifically do not process any text that uses ghost encodings.
Creating and Deleting API Objects

To create or destroy anything in a FrameMaker document, you must create or destroy the object the API uses to represent it. This chapter discusses how to create and destroy objects.

Before you use API functions to create and delete objects, you need an understanding of how the Frame API organizes objects. For background information on this subject, see Part II, “Frame Product Architecture.”

Creating objects

The API provides different functions for creating different types of objects. For example, you use `F_ApiNewTable()` to create tables and `F_ApiNewSeriesObject()` to create objects that occur in ordered series. The following table lists the API object types and the functions you use to create them.

<table>
<thead>
<tr>
<th>To create objects of this type</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>FO_Book</td>
<td>F_ApiNewNamedObject()</td>
</tr>
<tr>
<td>FO_CharFmt</td>
<td></td>
</tr>
<tr>
<td>FO_Color</td>
<td></td>
</tr>
<tr>
<td>FO_CondFmt</td>
<td></td>
</tr>
<tr>
<td>FO_ElementDef</td>
<td></td>
</tr>
<tr>
<td>FO_FmtChangeList (named)</td>
<td></td>
</tr>
<tr>
<td>FO_MasterPage</td>
<td></td>
</tr>
<tr>
<td>FO_MenuItemSeparator</td>
<td></td>
</tr>
<tr>
<td>FO_PgfFmt</td>
<td></td>
</tr>
<tr>
<td>FO_RefPage</td>
<td></td>
</tr>
<tr>
<td>FO_RulingFmt</td>
<td></td>
</tr>
<tr>
<td>FO_TblFmt</td>
<td></td>
</tr>
<tr>
<td>FO_VarFmt</td>
<td></td>
</tr>
<tr>
<td>FO_XRefFmt</td>
<td></td>
</tr>
</tbody>
</table>
## Creating and Deleting API Objects

<table>
<thead>
<tr>
<th>To create objects of this type</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>FO_Arc</td>
<td>F_ApiNewGraphicObject()</td>
</tr>
<tr>
<td>FO_Ellipse</td>
<td>F_ApiNewGraphicObject()</td>
</tr>
<tr>
<td>FO_Flow</td>
<td>F_ApiNewGraphicObject()</td>
</tr>
<tr>
<td>FO_Group</td>
<td>F_ApiNewGraphicObject()</td>
</tr>
<tr>
<td>FO_Inset</td>
<td>F_ApiNewGraphicObject()</td>
</tr>
<tr>
<td>FO_Line</td>
<td>F_ApiNewGraphicObject()</td>
</tr>
<tr>
<td>FO_Math</td>
<td>F_ApiNewGraphicObject()</td>
</tr>
<tr>
<td>FO_Polyline</td>
<td>F_ApiNewGraphicObject()</td>
</tr>
<tr>
<td>FO_Polygon</td>
<td>F_ApiNewGraphicObject()</td>
</tr>
<tr>
<td>FO_Rectangle</td>
<td>F_ApiNewGraphicObject()</td>
</tr>
<tr>
<td>FO_RoundRect</td>
<td>F_ApiNewGraphicObject()</td>
</tr>
<tr>
<td>FO_TextFrame</td>
<td>F_ApiNewGraphicObject()</td>
</tr>
<tr>
<td>FO_TextLine</td>
<td>F_ApiNewGraphicObject()</td>
</tr>
<tr>
<td>FO_UnanchoredFrame</td>
<td>F_ApiNewGraphicObject()</td>
</tr>
<tr>
<td>FO_AFrame</td>
<td>F_ApiNewAnchoredObject()</td>
</tr>
<tr>
<td>FO_Fn</td>
<td>F_ApiNewAnchoredObject()</td>
</tr>
<tr>
<td>FO_Marker</td>
<td>F_ApiNewAnchoredObject()</td>
</tr>
<tr>
<td>FO_Rubi</td>
<td>F_ApiNewAnchoredObject()</td>
</tr>
<tr>
<td>FO_Tbl</td>
<td>F_ApiNewAnchoredObject()</td>
</tr>
<tr>
<td>FO_TiApiClient</td>
<td>F_ApiNewAnchoredObject()</td>
</tr>
<tr>
<td>FO_BodyPage</td>
<td>F_ApiNewSeriesObject()</td>
</tr>
<tr>
<td>FO_BookComponent</td>
<td>F_ApiNewSeriesObject()</td>
</tr>
<tr>
<td>FO_Pgf</td>
<td>F_ApiNewSeriesObject()</td>
</tr>
<tr>
<td>FO_BookComponent</td>
<td>F_ApiNewBookComponentInHierarchy()</td>
</tr>
<tr>
<td>FO_Element</td>
<td>F_ApiNewElement()</td>
</tr>
<tr>
<td></td>
<td>F_ApiNewElementInHierarchy()</td>
</tr>
<tr>
<td>FO_FMTRule</td>
<td>F_ApiNewSubObject()</td>
</tr>
<tr>
<td>FO_FMTRuleClause</td>
<td>F_ApiNewSubObject()</td>
</tr>
<tr>
<td>FO_FMTChangeList (unnamed)</td>
<td>F_ApiNewSubObject()</td>
</tr>
<tr>
<td>FO_Tbl</td>
<td>F_ApiNewTable()</td>
</tr>
<tr>
<td>FO_Tbl</td>
<td>F_ApiNewTable()</td>
</tr>
<tr>
<td>FO_Var</td>
<td>F_ApiNewAnchoredFormattedObject()</td>
</tr>
<tr>
<td>FO_XRef</td>
<td>F_ApiNewAnchoredFormattedObject()</td>
</tr>
<tr>
<td>FO_Cell</td>
<td>F_ApiAddCols()</td>
</tr>
<tr>
<td></td>
<td>F_ApiAddRows()</td>
</tr>
<tr>
<td>FO_Row</td>
<td>F_ApiAddCols()</td>
</tr>
<tr>
<td></td>
<td>F_ApiAddRows()</td>
</tr>
<tr>
<td>FO_Inset</td>
<td>F_ApiImport()</td>
</tr>
<tr>
<td>FO_TiFlow</td>
<td>F_ApiImport()</td>
</tr>
<tr>
<td>FO_TiText</td>
<td>F_ApiImport()</td>
</tr>
<tr>
<td>FO_TiTextTable</td>
<td>F_ApiImport()</td>
</tr>
</tbody>
</table>
Creating and Deleting API Objects

Creating named objects

A named object is an object, such as a master page or a Paragraph Catalog format, that is identified by a unique name. To create named objects, use \texttt{F_ApiNewNamedObject()}. The syntax for \texttt{F_ApiNewNamedObject()} is:

\begin{verbatim}
F_ObjHandleT F_ApiNewNamedObject(F_ObjHandleT docId,
    IntT objType,
    StringT name);
\end{verbatim}

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document to which to add the object. To create a book, specify \texttt{FV_SessionId}.</td>
</tr>
<tr>
<td>objType</td>
<td>The type of object to create (for example, \texttt{FO_MasterPage}, \texttt{FO_PgfFmt}, or \texttt{FO_Book}).</td>
</tr>
<tr>
<td>name</td>
<td>The name to give to the object. If the object is an \texttt{FO_Book} object, specify the pathname of the book file to create.</td>
</tr>
</tbody>
</table>

\texttt{F_ApiNewNamedObject()} uses a set of default properties when it creates a new named object. Because the property lists for most named objects are quite long, it is often easier to copy the properties from a similar object and then change individual properties.

Creating a paragraph format

The following code creates a paragraph format named MyHead, which looks like Heading 1 except that it’s indented two inches:
#define in (MetricT)(65536*72)
F_PropValsT proplist;
F_ObjHandleT docId, Heading1Id, myHeadId;

docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

/* Create MyHead paragraph format. */
myHeadId = F_ApiNewNamedObject(docId, FO_PgfFmt, "MyHead");

/* Get ID for Heading1. */
Heading1Id = F_ApiGetNamedObject(docId, FO_PgfFmt, "Heading1");
if (!Heading1Id) return;

/* Get properties for Heading1 and copy them to MyHead. */
proplist = F_ApiGetProps(docId, Heading1Id);
F_ApiSetProps(docId, myHeadId, &proplist);
F_ApiSetMetric(docId, myHeadId, FP_LeftIndent, 2*in);
F_ApiDeallocatePropVals(&proplist);

... 

Creating a book

The following code creates a book named mybook in the tmp directory. It uses F_ApiNewSeriesObject() to add book components. For more information on F_ApiNewSeriesObject(), see “Creating series objects” on page 149. For more information on creating books and book components, see “Creating new books and components” on page 355.

... 

bkId = F_ApiNewNamedObject(FV_SessionId, FO_Book, 
"/tmp/mybook");

/* Create component and then change its name. */
componentId = F_ApiNewSeriesObject(bkId, 
FO_BookComponent, 0);
F_ApiSetString(bkId, componentId, FP_Name, "doc1");

/* Add another component after the first one. */
componentId = F_ApiNewSeriesObject(bkId, 
FO_BookComponent, componentId);
F_ApiSetString(bkId, componentId, FP_Name, "doc2");

... 

Creating graphic objects

To create any graphic object except an anchored frame, use F_ApiNewGraphicObject().
The syntax for `F_ApiNewGraphicObject()` is:

```c
F_ObjHandleT F_ApiNewGraphicObject(F_ObjHandleT docId,
                                   IntT objType,
                                   F_ObjHandleT parentFrameId);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document in which to create the new object</td>
</tr>
<tr>
<td>objType</td>
<td>The type of graphic object to create (for example, FO_Rectangle or FO_Line)</td>
</tr>
<tr>
<td>parentFrameId</td>
<td>The ID of the parent frame in which to create the object</td>
</tr>
</tbody>
</table>

You can create a graphic object only in a frame. To create a graphic object directly on a page (not in an anchored or unanchored frame), you create it in the *page frame*. A page frame is an invisible frame that every page has. For more information on page frames and how FrameMaker organizes graphics, see “How the API represents graphic objects” on page 294.

The API maintains a frame’s *child objects* in a linked list. The order of this list corresponds to the back-to-front draw order. If the frame you specify for `parentFrameId` already has child objects, `F_ApiNewGraphicObject()` adds the new object to the end of the linked list; that is, it puts it in front of the other objects in the frame. The API automatically updates the properties of the parent frame and the last object in the list to reflect the addition of the new object. For instructions on moving objects forward or back in the draw order, see “Moving graphics forward or back in the draw order” on page 100.

`F_ApiNewGraphicObject()` uses a set of arbitrary default values for the properties of the graphic objects that it creates. Usually, you will need to change most of these default properties.

**Example**

To draw a circle with a one-inch diameter directly on the current page of a document, use the following code:

```c
...  
#define in (MetricT)(65536*72)
F_ObjHandleT docId, pageId, pFrameId, circleId;
/* Get the document, current page, and page frame IDs. 
** The F_ApiGet[property_type]() and F_ApiSet[property_type]() 
** functions are explained in Chapter 5. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
pageId = F_ApiGetId(FV_SessionId, docId, FP_CurrentPage);
pFrameId = F_ApiGetId(docId, pageId, FP_PageFrame);
/* Create the circle on the page frame. */
circleId = F_ApiNewGraphicObject(docId, FO_Ellipse, pFrameId);
/* Change the size of the circle to 1 inch. */
```
Creating anchored objects

An anchored object is an object, such as a cross-reference, table, or anchored frame, that the user can insert in text. The API provides the following functions to create anchored objects:

- To create tables, use `F_ApiNewTable()`.
- To create variables and cross-references, use `F_ApiNewAnchoredFormattedObject()`.
- To create other anchored objects, use `F_ApiNewAnchoredObject()`.

`F_ApiNewTable()` is discussed in “Creating tables” on page 154.

The syntax for `F_ApiNewAnchoredFormattedObject()` and `F_ApiNewAnchoredObject()` is:

```
F_ObjHandleT F_ApiNewAnchoredFormattedObject(F_ObjHandleT docId,
   IntT objType,
   StringT format,
   F_TextLocT *textLocp);

F_ObjHandleT F_ApiNewAnchoredObject(F_ObjHandleT docId,
   IntT objType,
   F_TextLocT *textLocp);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>docId</code></td>
<td>The ID of the document to which to add the object</td>
</tr>
<tr>
<td><code>objType</code></td>
<td>The type of object to create (for example, FO_Marker or FO_XRef)</td>
</tr>
<tr>
<td><code>format</code></td>
<td>The string that specifies the object’s format (for example, Heading &amp; Page or Current Date (Long))</td>
</tr>
<tr>
<td><code>textLocp</code></td>
<td>The text location at which to insert the anchored object</td>
</tr>
</tbody>
</table>

`F_ApiNewAnchoredObject()` and `F_ApiNewAnchoredFormattedObject()` use a set of arbitrary default properties to create new objects. For example, the default width and height of a new anchored frame is 0.25 inches.

Example

The following code adds a Filename (Long) variable at the insertion point (or the beginning of the text selection) of the active document:
Creating and Deleting API Objects

Creating series objects

A *series object* is any object, other than a graphic object, that occurs in an ordered series. Paragraphs and body pages are examples of series objects. To create a series object, use `F_ApiNewSeriesObject()`. The syntax for `F_ApiNewSeriesObject()` is:

```c
F_ObjHandleT F_ApiNewSeriesObject(F_ObjHandleT docId,
                                   IntT objType,
                                   F_ObjHandleT prevId);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document or book to which to add the object.</td>
</tr>
<tr>
<td>objType</td>
<td>The type of object to create (for example, FO_BodyPage or FO_Pgf).</td>
</tr>
<tr>
<td>prevId</td>
<td>The ID of the object that you want to add the new object after.</td>
</tr>
</tbody>
</table>

To add a paragraph at the beginning of a flow, specify the flow’s ID. To add other objects at the beginning of a series, specify 0.

Example

The following code inserts a paragraph after the paragraph containing the insertion point:

```c
F_ObjHandleT docId, pgfId;
F_TextRangeT tr;
F_TextLocT textLoc;

/* Get the insertion point. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if (!tr.beg.objId) return;

/* Insert the variable. */
variableId = F_ApiNewAnchoredFormattedObject(docId, FO_Var,
                                             "Filename (Long)", &tr.beg);
...
```
/* Add the paragraph. */
pgfId = F_ApiNewSeriesObject(docId, FO_Pgf, tr.beg.objId);

/* Put some text in the paragraph. */
textLoc.objId = pgfId;
textLoc.offset = 0;
F_ApiAddText(docId, &textLoc, "Here’s some text");

Creating flows

You can’t create a flow directly with API functions. However, you can create one indirectly by creating a text frame with F_ApiNewGraphicObject(). Each time you create a text frame, the API automatically creates a flow to contain it. For example, to create a flow on the first body page of the active document, you can use the following code:

... F_ObjHandleT docId, tFrameId, flowId, pageId, pFrameId;

/* Get ID of first body page’s page frame. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
pageId = F_ApiGetId(FV_SessionId, docId, FP_FirstBodyPageInDoc);
tFrameId = F_ApiGetId(docId, pageId, FP_PageFrame);

/* Create the text frame and get the flow’s ID from it. */
tFrameId = F_ApiNewGraphicObject(docId, FO_TextFrame, pFrameId);
flowId = F_ApiGetId(docId, tFrameId, FP_Flow);

To connect a text frame in a flow, set its FP_NextTextFrameInFlow or FP_PrevTextFrameInFlow property to the ID of a text frame that is already in the flow. To disconnect a text frame from a flow, set its FP_NextTextFrameInFlow or FP_PrevTextFrameInFlow property to 0.

Creating structural elements

The API provides the following functions for creating structural elements in FrameMaker documents and books:

- F_ApiNewElement() inserts a new element at a specified text location in a document.
- F_ApiNewElementInHierarchy() inserts a new element at a specified position in the element hierarchy of a document or book.

The syntax for F_ApiNewElement() is:
Creating and Deleting API Objects

F_ObjHandleT F_ApiNewElement(F_ObjHandleT docId,
    F_ObjHandleT elemDefId,
    F_TextLocT *textLocp);

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document to which to add the element</td>
</tr>
<tr>
<td>elemDefId</td>
<td>The ID of the element definition for the new element</td>
</tr>
<tr>
<td>textLocp</td>
<td>The text location at which to insert the new element</td>
</tr>
</tbody>
</table>

The syntax for `F_ApiNewElementInHierarchy()` is:

F_ObjHandleT F_ApiNewElementInHierarchy(F_ObjHandleT docId,
    F_ObjHandleT elemDefId,
    F_ElementLocT *elemLocp);

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document or book to which to add the element</td>
</tr>
<tr>
<td>elemDefId</td>
<td>The ID of the element definition for the new element</td>
</tr>
<tr>
<td>elemLocp</td>
<td>The location at which the element is inserted</td>
</tr>
</tbody>
</table>

You can’t use `F_ApiNewElementInHierarchy()` to add elements to an unstructured document. You must structure the document first by adding a root element with `F_ApiNewElement()`.

**Examples**

The following code adds a Para element at the insertion point:

```c
F_ElementRangeT elemSelect;
F_ObjHandleT docId, elemId, paraEdefId;

/* Get ID of active document and the Para element definition. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
paraEdefId = F_ApiGetNamedObject(docId, FO_ElementDef, "Para");

/* Get current element selection in active document. */
elemSelect = F_ApiGetElementRange(FV_SessionId, docId,
    FP_ElementSelection);

if (elemSelect.beg.parentId == 0 || paraEdefId == 0) return;

/* Insert the new element. */
elemId = F_ApiNewElementInHierarchy(docId, paraEdefId,
    &elemSelect.beg);
```

. . .
The following code adds a highest-level element, named Appendix, to the main flow of the active document:

```
F_ObjHandleT docId, elemId, chapEdefId;
F_TextLocT tl;

/* Get IDs of document, main flow, and element definition. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
chapEdefId = F_ApiGetNamedObject(docId, FO_ElementDef, "Appendix");

/* Set up text location for beginning of main flow. */
tl.objId = F_ApiGetId(FV_SessionId, docId, FP_MainFlowInDoc);
tl.offset = 0;

/* Insert the new element. */
elemId = F_ApiNewElement(docId, chapEdefId, &tl);
```

Creating format rules, format rule clauses, and format change lists

To create format rules, format rule clauses, and unnamed format change lists in FrameMaker documents and books, use `F_ApiNewSubObject()`.

The syntax for `F_ApiNewSubObject()` is:

```
F_ObjHandleT F_ApiNewSubObject(F_ObjHandleT docOrBookId,
    F_ObjHandleT parentId,
    IntT property);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docOrBookId</td>
<td>The ID of the document in which to create the new object</td>
</tr>
<tr>
<td>parentId</td>
<td>The ID of the object’s parent object</td>
</tr>
<tr>
<td>property</td>
<td>The property of the parent object to associate with the new object</td>
</tr>
</tbody>
</table>

`F_ApiNewSubObject()` allows you to associate the new object with a specified property of its parent object. For example, you can create an FO_FmtRule object as the suffix format rule of an element definition or as a subformat rule of a format rule clause. For a complete list of the properties with which you can associate new format rule objects, see “F_ApiNewSubObject()” on page 221 of the FDK Programmer’s Reference.

The following code creates a prefix rule and adds it to the Quotation element definition so that the element definition appears as shown in Figure 8.1:

```
This argument | Means |
--- | --- |
docOrBookId | The ID of the document in which to create the new object |
parentId | The ID of the object’s parent object |
property | The property of the parent object to associate with the new object |
```
Creating and Deleting API Objects

F_ObjHandleT docId, quoteEdefId, prefixRuleId, clauseId;

/* Get ID of Para element definition. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
quoteEdefId = F_ApiGetNamedObject(docId, FO_ElementDef, "Quotation");

/* Add the prefix format rule. */
prefixRuleId = F_ApiNewSubObject(docId, quoteEdefId, FP_PrefixRules);

/* Set the rule type. */
F_ApiSetInt(docId, prefixRuleId, FP_FmtRuleType, FV_CONTEXT_RULE);

/* Add rule clause to suffix rule. 322 is left quote. */
clauseId = F_ApiNewSubObject(docId, prefixRuleId, FP_FmtRuleClauses);
F_ApiSetInt(docId, clauseId, FP_IsTextRange, True);
F_ApiSetString(docId, clauseId, FP_ElemPrefixSuffix, "\322");

Element (Container): Quotation
   General rule: <Text>.
   Text format rules
      1. In all contexts.
         Text range.
         No additional formatting.
   Prefix rules
      1. In all contexts
         Prefix: “
         Text range.

**Figure 8.1 Quotation element definition**

To create a named format change list, use `F_ApiNewNamedObject()` . To add the format change list to a format rule clause, set the format rule clause object’s`FP_FMT_CHANGE_LIST_TAG` property to the name of the change list. For example, the following code creates the Code format change list shown in **Figure 8.2** and adds it to the first format rule clause of the Para element definition’s first text format rule:

F_ObjHandleT docId, changeListId;
F_ObjHandleT edefId;
UIntT i;
F_StringsT fonts;
F_IntsT rules, clauses;
Creating and Deleting API Objects

```c
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
changeListId = F_ApiNewNamedObject(docId, FO_FmtChangeList,
   "Code");

/*
** Get the index of the Courier font family in the list of
** fonts available in the current session.
*/
fonts = F_ApiGetStrings(0, FV_SessionId, FP_FontFamilyNames);
for (i=0; i<fonts.len &&
   !F_StrEqual("Courier",fonts.val[i]); i++);
if (i == fonts.len) return; /* Courier font not found. */

/* Add the FP_FontFamily property; set it to Courier. */
F_ApiSetInt(docId, changeListId, FP_FontFamily, i);

/* Add the FP_PairKern property. */
F_ApiSetInt(docId, changeListId, FP_PairKern, False);

/* Get ID of the first clause of the first text format rule. */
edefId = F_ApiGetNamedObject(docId, FO_ElementDef,
   "Para");
rules = F_ApiGetInts(docId, edefId, FP_TextFmtRules);
clauses = F_ApiGetInts(docId, rules.val[0], FP_FmtRuleClauses);

/* Add the Code format change list to the format rule clause. */
F_ApiSetString(docId, clauses.val[0],
   FP_FmtChangeListTag, "Code");
```

![Format change list: Code
Default font properties
Family: Courier
Pair kerning: No](image)

**Figure 8.2** Code format change list

Creating tables

The API provides the following functions for creating tables:

* F_ApiNewTable() is usually easier to use because it allows you to specify a format and the number of rows and columns.

* F_ApiNewAnchoredObject() creates a table with a single column and a single body row.

For instructions on using F_ApiNewAnchoredObject(), see “Creating anchored objects” on page 148.

The syntax for F_ApiNewTable() is:
Creating and Deleting API Objects

F_ObjHandleT F_ApiNewTable(F_ObjHandleT docId,
   StringT format,
   IntT numCols,
   IntT numBodyRows,
   IntT numHeaderRows,
   IntT numFooterRows,
   TextLocT *textLocp);

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document.</td>
</tr>
<tr>
<td>format</td>
<td>The table format tag (for example, FormatA or Wide Table). To use the default format, specify NULL. The default format is the format of the last table the user inserted.</td>
</tr>
<tr>
<td>numCols</td>
<td>The number of columns in the table. To use the default number of columns from the Table Catalog format, specify -1.</td>
</tr>
<tr>
<td>numBodyRows</td>
<td>The number of rows in the table. To use the default number of body rows from the Table Catalog format, specify -1.</td>
</tr>
<tr>
<td>numHeaderRows</td>
<td>The number of header rows in the table. To use the default number of header rows from the Table Catalog format, specify -1.</td>
</tr>
<tr>
<td>numFooterRows</td>
<td>The number of footer rows in the table. To use the default number of footer rows from the Table Catalog format, specify -1.</td>
</tr>
<tr>
<td>textLocp</td>
<td>The location at which to insert the new table. The location can’t be within a footnote or a table.</td>
</tr>
</tbody>
</table>

If successful, F_ApiNewTable() returns the ID of the new FO_Tbl object. Otherwise, it returns 0 and sets FA_errno to an error code.

Example

The following code inserts the table shown in Figure 8.3:

```c
F_ObjHandleT docId, pgfId, tblId, titlePgId;
F_TextRangeT tr;
F_TextLocT textLoc;

/* Get the insertion point. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if (!tr.beg.objId) return;

/* Insert the table at the insertion point. */
tblId = F_ApiNewTable(docId, NULL, 3, 3, 0, 0, &tr.beg);

/* Return if IP in FO_Tbl or FO_Fn, and table wasn’t created. */
if (!tblId) return;
```
/* Get ID of paragraph in table’s title. */
titlePgfId = F_ApiGetId(docId, tblId, FP_FirstPgf);

/* Put some text in the table’s title. */
textLoc.objId = titlePgfId;
textLoc.offset = 0;
F_ApiAddText(docId, &textLoc, "My Table");
.
.
My Table

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 8.3** Table created with F_ApiNewTable()

For an example of how to add text to table cells, see “Adding text to table cells” on page 122.

**Adding table rows and columns**

To add table rows or columns to an existing table, use these functions:

- F_ApiAddCols() to add table columns
- F_ApiAddRows() to add table rows

You can’t add rows by changing the FQ_Tbl object’s FP_TblNumCols and FP_TblNumRows properties. These properties are read-only.

The syntax for F_ApiAddCols() is:

```c
IntT F_ApiAddCols(F_ObjHandleT docId, F_ObjHandleT tblId,
                    IntT refColNum,
                    IntT direction,
                    IntT numNewCols);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document containing the table.</td>
</tr>
<tr>
<td>tableId</td>
<td>The ID of the table to which to add columns.</td>
</tr>
<tr>
<td>refColNum</td>
<td>The column at which to start adding columns. The columns are numbered from left to right starting with column 0.</td>
</tr>
<tr>
<td>direction</td>
<td>The direction in which to add columns. To add columns to the left of the reference column specify FV_Left. To add columns to the right, specify FV_Right.</td>
</tr>
<tr>
<td>numNewCols</td>
<td>The number of columns to add.</td>
</tr>
</tbody>
</table>

If successful, F_ApiAddCols() returns FE_Success. Otherwise, it returns an error code. The syntax for F_ApiAddRows() is:
Creating and Deleting API Objects

**Creating and Deleting API Objects**

```
IntT F_ApiAddRows(F_ObjHandleT docId, 
                 F_ObjHandleT refRowId, 
                 IntT direction, 
                 IntT numNewRows);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document containing the table.</td>
</tr>
<tr>
<td>refRowId</td>
<td>The ID of the row at which to start adding rows. The added rows will be the same type as this row. For example, if refRowId specifies a heading row, the added rows will also be heading rows.</td>
</tr>
<tr>
<td>direction</td>
<td>The direction in which to add rows. To add rows above the reference row, specify FV_Above. To add them below, specify FV_Below. For a list of the other constants you can specify for this parameter, see “F_ApiAddRows()” on page 39 of the FDK Programmer’s Reference</td>
</tr>
<tr>
<td>numNewRows</td>
<td>The number of rows to add.</td>
</tr>
</tbody>
</table>

If successful, `F_ApiAddRows()` returns FE_Success.

`F_ApiAddCols()` requires you to specify the number of the reference column, whereas `F_ApiAddRows()` requires you to specify the ID of the reference row. This is because, in Frame document architecture, rows are objects. Columns are just a way of referring to a set of cells. When you create a row, the API actually creates an FO_Row object to represent the row and an FO_Cell object to represent each cell in the row. When you create a column, the API just creates FO_Cell objects and adds them to existing FO_Row objects. For more information on how tables and cells are organized, see “How the API represents tables” on page 332.

**Example**

The following code adds a column to the right of the first column and two rows below the second row in a table:

```c
F_ObjHandleT docId, tblId, row1Id, row2Id;
/* Get the document and table IDs. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tblId = F_ApiGetId(FV_SessionId, docId, FP_SelectedTbl);
/* Add the column. */
F_ApiAddCols(docId, tblId, 0, FV_Right, 1);
/* Get the ID for row 1, then traverse to the ID for row 2. */
row1Id = F_ApiGetId(docId, tblId, FP_FirstRowInTbl);
row2Id = F_ApiGetId(docId, row1Id, FP_NextRowInTbl);
/* Add the rows. */
if (row2Id) F_ApiAddRows(docId, row2Id, FV_Below, 2);
```
Creating commands, menus, and menu item separators

For information on creating and deleting commands, menus, and menu item separators, see Chapter 9, “Manipulating Commands and Menus with the API”

Deleting objects

To delete objects, use `F_ApiDelete()`.

The syntax for `F_ApiDelete()` is:

```
IntT F_ApiDelete(F_ObjHandleT docId, F_ObjHandleT objId);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document from which to delete the object</td>
</tr>
<tr>
<td>objId</td>
<td>The ID of the object to delete</td>
</tr>
</tbody>
</table>

If `F_ApiDelete()` is successful, it returns `FE_Success`. Otherwise, it returns an error code. There are a number of objects that you can’t delete. For example, you can’t delete an `FO_Doc` object or an `FO_VarFmt` object that represents a system variable format. For the list of these objects, see “F_ApiDelete()” on page 83 of the FDK Programmer’s Reference.

When you delete an object, the API automatically deletes all of that object’s child objects. For example, if you delete a frame, the API deletes all the objects in the frame. If you delete an `FO_Tbl` object, the API deletes all the `FO_Row` objects and `FO_Cell` objects in the table. Similarly, if you delete an element in a FrameMaker document, the API deletes all the descendants of that element.

Deleting flows and text frames

When you delete a flow, the API also deletes all the text frames in it (and all the paragraphs in the text frames). If you don’t want to delete a text frame when you delete a flow, you must disconnect the text frame from the flow before you delete it. To disconnect a text frame from a flow, set the text frame’s `FP_PrevTextFrameInFlow` and `FP_NextTextFrameInFlow` properties to 0.

If you delete a text frame that is not connected to another text frame, the API deletes the flow that contains it.

Deleting table columns and rows

To delete table columns and rows use these functions:

- `F_ApiDeleteCols()` to delete table columns
- `F_ApiDeleteRows()` to delete table rows
Creating and Deleting API Objects

The syntax for `F_ApiDeleteCols()` is:

```c
IntT F_ApiDeleteCols(F_ObjHandleT docId,
                     F_ObjHandleT tblId,
                     IntT refColNum,
                     IntT numDelCols);
```

This argument Means

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document containing the table.</td>
</tr>
<tr>
<td>tblId</td>
<td>The ID of the table containing the columns.</td>
</tr>
<tr>
<td>refColNum</td>
<td>The first column to delete. Columns are numbered from left to right, starting with column 0.</td>
</tr>
<tr>
<td>numDelCols</td>
<td>The number of columns to delete.</td>
</tr>
</tbody>
</table>

`F_ApiDeleteCols()` deletes the column specified by `refColNum` and `(numDelCols - 1)` columns to the right of it. If `F_ApiDeleteCols()` is successful, it returns `FE_Success`. Otherwise, it returns an error code. When you delete a table column, the API automatically deletes all the `FO_Cell` objects in the column.

The syntax for `F_ApiDeleteRows()` is:

```c
IntT F_ApiDeleteRows(F_ObjHandleT docId,
                     F_ObjHandleT tblId,
                     IntT refRowId,
                     IntT numDelRows);
```

This argument Means

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document containing the table.</td>
</tr>
<tr>
<td>tblId</td>
<td>The ID of the table containing the rows.</td>
</tr>
<tr>
<td>refRowId</td>
<td>The ID of the first row to delete.</td>
</tr>
<tr>
<td>numDelRows</td>
<td>The number of rows to delete.</td>
</tr>
</tbody>
</table>

`F_ApiDeleteRows()` deletes the row specified by `refRowId` and `(numDelRows - 1)` rows below it. If `F_ApiDeleteRows()` is successful, it returns `FE_Success`. Otherwise, it returns an error code. `F_ApiDeleteRows()` deletes only one type of row at a time. If you attempt to delete a range of rows that includes body rows and header or footer rows, `F_ApiDeleteRows()` returns an error. When you delete a table row, the API automatically deletes the `FO_Row` object and all the `FO_Cell` objects in the row.
Implicit property changes

When you create or delete an object, the API automatically updates other objects and properties that are affected. For example, if you delete a paragraph (FO_Pgf), the API automatically updates the FP.NextPgfInFlow property of the previous paragraph and the FP.PrevPgfInFlow property of the next paragraph. Figure 8.4 shows the paragraph objects in a flow before and after an FO_Pgf object is deleted.

Before

<table>
<thead>
<tr>
<th>FO_Pgf (Paragraph 1)</th>
<th>FP.NextPgfInFlow</th>
<th>FO_Pgf (Paragraph 2)</th>
<th>FP.NextPgfInFlow</th>
<th>FO_Pgf (Paragraph 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP.PrevPgfInFlow</td>
<td></td>
<td>FP.PrevPgfInFlow</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

After

<table>
<thead>
<tr>
<th>FO_Pgf (Paragraph 1)</th>
<th>FP.NextPgfInFlow</th>
<th>FO_Pgf (Paragraph 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP.PrevPgfInFlow</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.4 FO_Pgf objects in a flow before and after deleting an FO_Pgf object
Manipulating Commands and Menus with the API

This chapter describes Frame API command and menu functionality in detail. For an introduction to using commands and menus in your client’s interface, see “Using commands, menu items, and menus in your client” on page 24.

How the API represents commands and menus

The API uses an `FO_Command` object to represent each command, an `FO_Menu` object to represent each menu, and an `FO_MenuItemSeparator` object to represent each menu item separator in a FrameMaker product session.

Figure 9.1 shows a view-only menu bar, with the File menu open. Below you can see a diagram of objects for that menu bar, the menus in that menu bar, and the items in the File menu.

![Diagram of objects for a view-only menu bar](image)

**Figure 9.1** FrameMaker view-only menu bar, and the objects that represent it
IMPORTANT: FP_FirstMenuItemInMenu, FP_PrevMenuItemInMenu, and FP_NextMenuItemInMenu properties can specify menu, menu item, or menu item separator IDs.

The following table lists some of the properties of the FO_Command object that represent the Open menu item in Figure 9.1.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_CanHaveCheckMark</td>
<td>IntT</td>
<td>False</td>
</tr>
<tr>
<td>FP_Fcodes</td>
<td>F_IntsT</td>
<td>0x310</td>
</tr>
<tr>
<td>FP_KeyboardShortcuts</td>
<td>F_StringsT</td>
<td>&quot;!fo&quot;</td>
</tr>
<tr>
<td>FP_KeyboardShortcutLabel</td>
<td>StringT</td>
<td>&quot;!fo&quot;</td>
</tr>
<tr>
<td>FP_Labels</td>
<td>F_StringsT</td>
<td>&quot;Open...&quot;</td>
</tr>
<tr>
<td>FP_MenuItemIsEnabled</td>
<td>IntT</td>
<td>True</td>
</tr>
<tr>
<td>FP_Name</td>
<td>StringT</td>
<td>&quot;Open&quot;</td>
</tr>
</tbody>
</table>

You can get and set FO_Command, FO_Menu, and FO_MenuItemSeparator properties the same way you get and set the properties of other API objects. For more information on getting and setting object properties, see Chapter 5, “Getting and Setting Properties”

Lists of menus, menu items, and commands in a session

The API maintains a linked list of all the menus and menu items in a session. The FO_Session property, FP_FirstMenuItemInSession, specifies the ID of the first object in the list. The menu or menu item property, FP_NextMenuItemInSession, specifies the next object in the list. The list is not in any particular order.

The API also maintains a linked list of all the commands in a session. The FO_Session property, FP_FirstCommandInSession, specifies the ID of the first command in the list. The FO_Command property, FP_NextCommandInSession, specifies the next command in the list. Like the list of menus and menu items, this list is not in any particular order.

For a complete listing of all the available menus in the FrameMaker product see, “Getting the IDs of FrameMaker product menus and menu bars” on page 24.

Getting and setting menu and menu item properties

A single menu or command can have multiple instances in a FrameMaker product session; each FrameMaker product menu can contain an instance. For example, on UNIX platforms, the Cut command appears as a menu item on both the Edit menu and the document window pop-up menu. The API uses only one object to represent all instances of a menu, command, or menu item separator. For example, it uses only one FO_Command object to represent both instances of the Cut command on UNIX platforms.
Most properties of an FO_Menu, FO_Command, or FO_MenuItemSeparator object apply to all instances of the object. For example, if you use the following call to set the label of the Cut command:

```c
F_ObjHandleT cutCmdId;
cutCmdId = F_ApiGetNamedObject(FV_SessionId, FO_Command, "Cut");
F_ApiSetString(FV_SessionId, cutCmdId, FP_Label, "Excise");
```

the API changes the label of all instances of the Cut command to Excise.

The following properties apply only to individual instances of an FO_Menu, FO_Command, or FO_MenuItemSeparator object:

- FP_PrevMenuItemInMenu
- FP_NextMenuItemInMenu

When you get or set these properties, you must indicate which instance of the object you want to get or set them for. To do this, set the first parameter of the F_ApiGetId() or F_ApiSetId() function to the ID of the menu containing the instance.

For example, the following code gets the ID of the menu item above Cut on the Edit menu. Then it gets the ID of the menu item above Cut on the document window pop-up menu.

```c
F_ObjHandleT editMenuId, cutCmdId, docPopupMenuId,
prevItemOnEditMenuId, prevItemOnDocMenuId;
editMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu, "EditMenu");
docPopupMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu, "!MakerPopup");
cutCmdId = F_ApiGetNamedObject(FV_SessionId, FO_Command, "Cut");
prevItemOnEditMenuId = F_ApiGetId(editMenuId, cutCmdId, FP_PrevMenuItemInMenu);
prevItemOnDocMenuId = F_ApiGetId(docPopupMenuId, cutCmdId, FP_PrevMenuItemInMenu);
```

---

**Getting the IDs of commands and menus**

To manipulate a command or menu, you need its ID. If you know its name, the simplest way to get its ID is to call F_ApiGetNamedObject(). For example, the following code gets the IDs of the FrameMaker main menu bar and the File menu:

```c
```
Manipulating Commands and Menus with the API

F_ObjHandleT fileMenuId, mainMenuBarId;
mainMenuBarId = F_ApiGetNamedObject(FV_SessionId, FO_Menu, "!MakerMainMenu");
fileMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu, "FileMenu");

If a menu or command with the specified name doesn’t exist, F_ApiGetNamedObject() returns 0.

If you don’t know a command’s name, there are several lists of objects you can traverse to get its ID:

- If you know the command is on a particular menu, traverse the list of menu items on the menu.
- If you know the command is on a menu but you don’t know which menu, traverse the list of menu items in the session.
- If the command isn’t on any menu, traverse the list of commands in the FrameMaker product session.

Because there are so many FrameMaker product commands, traversing all of them can be slow. If a command is a menu item, get its ID by traversing the list of menu items on a menu or in a session instead of traversing the list of commands in the session.

For example, if you don’t know a menu item’s name, but you know that its label is Database, you can use the following code to get its ID:

F_ObjHandleT itemId;
StringT itemName;

itemId = F_ApiGetId(0, FV_SessionId, FP_FirstMenuItemInSession);
while(itemId) {
    itemName = F_ApiGetString(FV_SessionId, itemId, FP_Label);
    if(F_StrEqual(itemName, "Database")) break;
    itemId = F_ApiGetId(FV_SessionId, itemId, FP_NextMenuItemInSession);
    F_Free(itemName);
}

Determining a session’s menu configuration

The F_ApiGetNamedObject() function indicates only whether a command or menu exists. It does not indicate whether it appears on a menu. To determine whether a command or menu appears on a specific menu, call F_ApiMenuItemInMenu().

The syntax for F_ApiMenuItemInMenu() is:
F_ObjHandleT F_ApiMenuItemInMenu (F_ObjHandleT menuId,
   F_ObjHandleT menuitemId,
   BoolT recursive);

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>menuId</td>
<td>The menu or menu bar to search.</td>
</tr>
<tr>
<td>menuitemId</td>
<td>The ID of the menu item or menu to search for.</td>
</tr>
<tr>
<td>recursive</td>
<td>Flag specifying whether to search the submenus on the menu specified by menuId. Specify True to search them.</td>
</tr>
</tbody>
</table>

F_ApiMenuItemInMenu() returns the ID of the menu on which it finds the specified menu or menu item. If it finds the menu or menu item on a submenu of the menu specified by menuId, it returns the ID of the submenu.

For example, the following code determines whether the Copy menu item is on the Edit menu or any of its submenus:

```c
F_ObjHandleT copyCmdId, editMenuId, copyMenuId;

editMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu, "EditMenu");
copyCmdId = F_ApiGetNamedObject(FV_SessionId, FO_Command, "Copy");
copyMenuId = F_ApiMenuItemInMenu(editMenuId, copyCmdId, True);

if(copyMenuId == editMenuId)
   F_Printf(NULL, "Copy is on the Edit menu.");
else if (copyMenuId)
   F_Printf(NULL, "Copy is on a submenu of the Edit Menu.");
else
   F_Printf(NULL, "Copy is not on the Edit menu.");
```

If you want to enumerate the items at all hierarchical levels of a menu, keep in mind that a menu item's F_PrevMenuItemInMenu and F_NextMenuItemInMenu properties specify menu items or menus only at the same hierarchical level. To list all the menu items on a menu, you must recursively traverse each of its submenus. For example, the following function prints the IDs of all the items on a specified menu and its submenus:

```c
VoidT printMenuItemIds(menuId) {
    F_ObjHandleT menuId;
    { 
        F_ObjHandleT itemId;

        itemId = F_ApiGetId(FV_SessionId, menuId, FP_FirstMenuItemInMenu);
        while(itemId) { 
            F_Printf(NULL, "Item ID: 0x%x\n", itemId);
            if(F_ApiGetObjectType(menuId, itemId) == FO_Menu) 
```
Manipulating Commands and Menus with the API

printMenuItemIds(itemId); /* Recursive call */
itemId = F_ApiGetId(menuId, itemId,
    FP_NextMenuItemInMenu);
}
}
...

Arranging menus and menu items

The API allows you to add a command to multiple menus and to reorder and delete menus and menu items. The following sections describe this functionality in detail.

Adding a command to multiple menus

You can’t use the F_ApiDefineAndAddCommand() function discussed in “Defining commands and adding them to menus” on page 26 to add a command to multiple menus. Instead, you must use F_ApiDefineCommand() to create the command and F_ApiAddCommandToMenu() to add it to the menus.

The syntax for F_ApiDefineCommand() is:

F_ObjHandleT F_ApiDefineCommand(IntT cmd,
    StringT tag,
    StringT label,
    StringT shortcut);

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmd</td>
<td>The integer that the FrameMaker product passes to your client’s F_ApiCommand() function when the user chooses the menu item or types the keyboard shortcut for the command.</td>
</tr>
<tr>
<td>tag</td>
<td>A unique name to identify the command.</td>
</tr>
<tr>
<td>label</td>
<td>The title of the command as it appears on the menu.</td>
</tr>
<tr>
<td>shortcut</td>
<td>The keyboard shortcut sequence.</td>
</tr>
</tbody>
</table>

The syntax for F_ApiAddCommandToMenu() is:

IntT F_ApiAddCommandToMenu(F_ObjHandleT toMenuId,
    F_ObjHandleT cmdId);

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>toMenuId</td>
<td>The menu to which to add the command</td>
</tr>
<tr>
<td>cmdId</td>
<td>The ID of the command</td>
</tr>
</tbody>
</table>
For example, the following code creates a command named Grammar and adds it to the Edit and Utilities menus:

```c
#define GRAMMAR_CMD 1
F_ObjHandleT editMenuId, utilsMenuId, grammarCmdId;
editMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu, "EditMenu");
utilsMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu, "UtilitiesMenu");
grammaGrammar CmdId = F_ApiDefineCommand(GRAMMAR_CMD, "Grammar", "Grammar...","GG");
F_ApiAddCommandToMenu(editMenuId, grammarCmdId);
F_ApiAddCommandToMenu(utilsMenuId, grammarCmdId);
```

You can also use `F_ApiAddCommandToMenu()` to add FrameMaker-defined commands to multiple menus. For example, the following code adds the Compare Documents command to the Edit and Utilities menus:

```c
F_ObjHandleT editMenuId, compareCmdId, UtilsMenuId;
compareCmdId = F_ApiGetNamedObject(FV_SessionId, FO_Command, "DocCompare");
editMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu, "EditMenu");
UtilsMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu, "UtilitiesMenu");
F_ApiAddCommandToMenu(editMenuId, compareCmdId);
F_ApiAddCommandToMenu(UtilsMenuId, compareCmdId);
```

A user’s menu configuration file can also add a FrameMaker or a client command to several menus. For example, the following lines of a menu configuration file add a client-defined command named Grammar to the Edit and Utilities menus.

```conf
<Command Grammar>
<Add Grammar <Menu EditMenu>>
<Add Grammar <Menu UtilitiesMenu>>
```

### Removing menus and menu items

To remove a menu or menu item, call `F_ApiDelete()` with the first parameter set to the ID of the menu that contains the menu or menu item and the second parameter set to the ID of the menu or menu item. `F_ApiDelete()` deletes a menu or menu item from only the menu you specify. If a menu or menu item is on several menus, you must make a separate `F_ApiDelete()` call to remove it from each menu.

For example, the following code removes the Copy command from the Edit menu:

```c
```
Reordering menus and menu items

To change a menu or menu item’s position on a menu, set its `FP_NextMenuItemInMenu` or `FP_PrevMenuItemInMenu` properties to specify the IDs of other menus or menu items on the menu. You need to set only one of these properties. FrameMaker automatically sets the other one for you.

For example, if the Cut and Copy commands are on the Edit menu, you can use the following code to make Copy appear above Cut:

```c
F_ObjHandleT cutCmdId, copyCmdId, editMenuId;
ediMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu, "EditMenu");
cutCmdId = F_ApiGetNamedObject(FV_SessionId, FO_Command, "Cut");
copyCmdId = F_ApiGetNamedObject(FV_SessionId, FO_Command, "Copy");
F_ApiSetId(editMenuId, cutCmdId, FP_NextMenuItemInMenu, copyCmdId);
```

The following `F_ApiSetId()` call has the same effect as the call in the code above:

```c
F_ApiSetId(editMenuId, cutCmdId, FP_PrevMenuItemInMenu, copyCmdId);
```

To move a menu or menu item to the top of a menu, set its `FP_PrevMenuItemInMenu` property to 0. To move it to the bottom of a menu, set its `FP_NextMenuItemInMenu` property to 0. The following code moves the Copy menu item to the top of the Edit menu:

```c
F_ObjHandleT copyCmdId, editMenuId;
ediMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu, "EditMenu");
copyCmdId = F_ApiGetNamedObject(FV_SessionId, FO_Command, "Copy");
F_ApiSetId(editMenuId, copyCmdId, FP_PrevMenuItemInMenu, 0);
```

If a menu is on a menu bar, setting its `FP_PrevMenuItemInMenu` property to 0 moves it to the leftmost position on the menu bar; setting its `FP_NextMenuItemInMenu` property to 0 moves it to the rightmost position.

You can’t move a menu or menu item to another menu by setting its `FP_NextMenuItemInMenu` or `FP_PrevMenuItemInMenu` properties. Instead, you must delete it and then add it to the menu on which you want it to appear. For example, the following code moves the Font menu from the Format menu to the main menu bar:

```c
F_ObjHandleT formatMenuId, fontMenuId, mainMenuBarId;
```
Manipulating Commands and Menus with the API

.../* Delete instance of Font menu on the Format menu. */
F_ApiDelete(formatMenuId, fontMenuId);
F_ApiAddMenuToMenu(mainMenuBarId, fontMenuId);
...

Changing the menu set

The user can switch to a menu set by choosing the menu set from View>Menus. Your client

can switch menu sets programmatically by setting the session’s FP_CurrentMenuSet

property. For example, the following code switches to quick menus:

...F_ApiSetInt(0, FV_SessionId, FP_CurrentMenuSet, FV_MENU_QUICK);
...

You can’t switch to custom menus unless you have already loaded a custom menu file. To load

custom menu file, call F_ApiLoadMenuCustomizationFile().

Manipulating menu item separators

You can manipulate menu item separators (FO_MenuItemSeparator objects) the same way

you manipulate menus and menu items, with the following limitations:

● If there is more than one separator on a menu, each separator must have a different name.

● A separator can’t be the first or the last item on a menu.

● Separators can’t appear next to each other.

FrameMaker provides six predefined separators: !Separator, Separator1,

Separator2, Separator3, Separator4, and Separator5. To get the ID of one of these

separators, use F_ApiGetNamedObject() as follows:

...F_ObjHandleT separatorId;
separatorId = F_ApiGetNamedObject(FV_SessionId,
    FO_MenuItemSeparator, "Separator1");
...

You can use any predefined separator name when you add a separator to a menu. For example,
you could use the predefined separator name Separator5 for the first separator you add to a

menu. However, you should try to use the predefined separator name that corresponds to a

separator’s position among the separators on a menu. For example, the first separator on a

menu should use the predefined separator name Separator1 and the second separator
should use the predefined separator name Separator2. This makes it easier for other clients

and users who modify menu configuration files to manipulate the separators.

You can use the same separator on different menus. For example, if you create two menus that
contain two separators, you can use Separator1 and Separator2 on each menu.
Adding, moving, and deleting separators

You can add, move, and delete separators the same way you add, move, and delete commands and menu items. For example, the following code removes the separator that appears after the Conditional Text menu item on the Special menu. It then adds a separator and moves it above the Conditional Text menu item.

```c
F_ObjHandleT specialMenuId, separatorId, conditionCmdId;

specialMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu, "SpecialMenu");
separatorId = F_ApiGetNamedObject(FV_SessionId, FO_MenuItemSeparator, "Separator1");
conditionCmdId = F_ApiGetNamedObject(FV_SessionId, FO_Command, "ConditionalText");

/* Delete the separator. */
F_ApiDelete(specialMenuId, separatorId);

/* Add it back and move it below the Conditional Text item. */
F_ApiAddCommandToMenu(specialMenuId, separatorId);
F_ApiSetId(specialMenuId, separatorId, FP_NextMenuItemInMenu, conditionCmdId);

Creating separator objects

Because separators appear the same and you can use the same separator on multiple menus, you will normally need only the predefined separators. If you need additional separators, you can create them with F_ApiNewNamedObject() as follows:

```c
F_ObjHandleT separatorId;
separatorId = F_ApiNewNamedObject(FV_SessionId, FO_MenuItemSeparator, "MySeparator");
```

Getting and setting menu item labels

Most FrameMaker product menu items have only one label. For example, the label of the Cut command is always Cut. However, some menu items have different labels for different contexts. For example, the label of the TableConvert command is Convert to Table when paragraph text is selected, but Convert to Paragraphs when the insertion point is in a table or table cells are selected.

The FP_Labels property specifies the labels a menu item can have in different contexts. If a menu item has one label for all contexts, its FP_Labels property specifies only that label. Otherwise, its FP_Labels property specifies pairs of strings with the following format:
where `context` specifies a context and `label` specifies the menu item label that appears when that context is applicable. The following table lists some of the values `context` can have.

<table>
<thead>
<tr>
<th>Context value</th>
<th>When the label is displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Book</td>
<td>When a book is active</td>
</tr>
<tr>
<td>Document</td>
<td>When a document is active</td>
</tr>
<tr>
<td>Long</td>
<td>When a menu item is on a pull-down menu or the document pop-up menu</td>
</tr>
<tr>
<td>ToTable</td>
<td>When text that is not a table or text line is selected</td>
</tr>
<tr>
<td>ToText</td>
<td>When the insertion point is in a table cell or one or more table cells are selected</td>
</tr>
<tr>
<td>Short</td>
<td>When a menu item is on a pull-right menu or one of the formatting bar menus</td>
</tr>
</tbody>
</table>

For example, the strings specified by the `FP_Labels` property of the TableConvert command are:

```c
"ToTable", "Convert to Table...",
   "ToText", "Convert to Paragraphs...")
```

### Setting the labels of FrameMaker product menu items

You can change the labels of FrameMaker product menu items. If a FrameMaker product menu item has labels for different contexts, you can change only the strings that specify the labels. You can’t change the strings that specify the contexts in which the labels appear. For example, the following code changes the labels for the TableConvert command:

```c
...#
#include "fstrings.h"

F_ObjHandleT cmdId;
StringT labels[4];
F_StringsT myLabels;
...

labels[0] = (StringT) F_StrCopyString("ToTable");
labels[1] = (StringT) F_StrCopyString("Make table out of this");
labels[2] = (StringT) F_StrCopyString("ToText");
labels[3] = (StringT) F_StrCopyString("Convert to paragraphs");
myLabels.len = 4;
myLabels.val = (StringT *)labels;
F_ApiSetStrings(FV_SessionId, cmdId, FP_Labels, &myLabels);
...
Setting the labels of client-defined menu items

A client-defined menu item can have only one label for all contexts. Its FP_Labels property should specify only one string. For example, the following code sets the label of a client-defined menu item to *My Item*:

```c
F_StringsT myLabels;
F_ObjHandleT cmdId;
StringT labels[1];

labels[0] = (StringT)"My Item";
myLabels.len = 1;
myLabels.val = (StringT *)labels;
F_ApiSetStrings(FV_SessionId, cmdId, FP_Labels, &myLabels);
```

Manipulating expandomatic menu items

An *expandomatic* menu item is a dynamically determined set of menu items that FrameMaker treats as a single menu item. For example, the list of paragraph formats that appears on the lower part of the Format>Paragraphs menu is an expandomatic menu item named !ShowParagraphTags. FrameMaker currently uses the following expandomatic menu items:

<table>
<thead>
<tr>
<th>Expandomatic menu item name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!ShowCharacterTags</td>
<td>The list of character formats available in the current document</td>
</tr>
<tr>
<td>!ShowDocumentWindows</td>
<td>The list of document windows in the current session</td>
</tr>
<tr>
<td>!ShowFilesRecentlyVisited</td>
<td>The list of the last five files opened</td>
</tr>
<tr>
<td>!ShowFontChoices</td>
<td>The list of font families available in the session</td>
</tr>
<tr>
<td>!ShowImportFiles</td>
<td>The list of open files that a user can import into the current document</td>
</tr>
<tr>
<td>!ShowParagraphTags</td>
<td>The list of paragraph formats available in the current document</td>
</tr>
</tbody>
</table>

A FrameMaker product can change the contents of an expandomatic menu item during a session. For example, when the user sets the insertion point in a document, the FrameMaker product changes the !ShowParagraphTags menu item to list the paragraph formats available in the document. If the user adds or deletes a paragraph format, the FrameMaker product updates the list to reflect the change.

You can manipulate an expandomatic menu item just as you manipulate any other menu item. However, you can’t manipulate the individual items that constitute the expandomatic item. For
example, you can move or remove the entire `!ShowParagraphTags` item, but you can’t move or remove an individual item, such as Body, that appears on it.

You can get the ID of an expandomatic item with the code similar to the following:

```c
F_ObjHandleT itemId;
itemId = F_ApiGetNamedObject(FV_SessionId, FO_Command,
"!ShowParagraphTags");
```

Individual items in an expandomatic item don’t have IDs. You can determine which items an expandomatic menu item contains by getting object properties. For example, you can determine which items `!ShowFontChoices` contains by getting the session property `FP_FontFamilyNames`.

---

### Using check marks

FrameMaker displays check marks next to some menu items to indicate the state of a setting or option. For example, when borders are visible in a document, a FrameMaker product displays a check mark next to the Borders menu item. Your client can also display check marks next to its menu items. Menu items have two properties that control check marks:

- `FP_CanHaveCheckMark`, which specifies whether an item can have a check mark
- `FP_CheckMarkIsOn`, which specifies whether a check mark appears next to an item

To use a check mark with a menu item, set `FP_CanHaveCheckMark` to True. Then make the check mark visible by setting `FP_CheckMarkIsOn` to True, or invisible by setting it to False.

**IMPORTANT:** Once you set `FP_CanHaveCheckMark` to True, resetting it to False has no effect. Setting `FP_CheckMarkIsOn` has an effect only when `FP_CanHaveCheckMark` is set to True.

---

### Using context-sensitive commands and menu items

Many FrameMaker product commands and menu items change depending on the context. For example, the Group command is disabled when there are no objects selected. The API provides properties that allow you to make your client’s commands and menu items context sensitive like FrameMaker product commands. The following sections describe how to use these properties.
Enabling commands for specific contexts

The `FP_EnabledWhen` property specifies the contexts in which a command is enabled. The following table lists some of the values `FP_EnabledWhen` can have.

<table>
<thead>
<tr>
<th><code>FP_EnabledWhen</code> value</th>
<th>Context in which a menu item is active</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FV_ENABLE_ALWAYS_ENABLE</code></td>
<td>All contexts. This is the default value.</td>
</tr>
<tr>
<td><code>FV_ENABLE_ALWAYS_DISABLE</code></td>
<td>No context. Setting <code>FP_EnabledWhen</code> to this value completely disables the menu item.</td>
</tr>
<tr>
<td><code>FV_ENABLE_IS_TEXT_SEL</code></td>
<td>Text is selected in a paragraph or a graphic text line.</td>
</tr>
<tr>
<td><code>FV_ENABLE_IN_TEXT_LINE</code></td>
<td>The insertion point or selection is in a graphic text line.</td>
</tr>
</tbody>
</table>

For a complete list of the values `FP_EnabledWhen` can have, see "FO_Command" on page 523 of the FDK Programmer’s Reference.

When you create a command, `FP_EnabledWhen` has the default value, `FV_ENABLE_ALWAYS_ENABLE`. To completely disable a command, set its `FP_EnabledWhen` property to `FV_ENABLE_ALWAYS_DISABLE`. To reenable a command, reset `FP_EnabledWhen` to `FV_ENABLE_ALWAYS_ENABLE`. To enable a command only in a specific context, set its `FP_EnabledWhen` property to one of the other listed values.

For example, the following code creates a command that is enabled only when text is selected:

```
...
F_ObjHandleT cmdId;

cmdId = F_ApiDefineCommand(1,"Transpose","Transpose Words"," ");
F_ApiSetInt(FV_SessionId, cmdId, FP_EnabledWhen,
            FV_ENABLE_IS_TEXT_SEL);
...
```

If a command is a menu item, it appears dimmed when it is disabled. You can determine whether a menu item is disabled by getting its `FP_MenuItemIsEnabled` property. This is easier than getting its `FP_EnabledWhen` property and determining whether the specified context currently applies. You can’t set the `FP_MenuItemIsEnabled` property.

Using shift menu items

FrameMaker provides several *shift menu items*, menu items that change when the user holds down the Shift key. For example, when the user holds down the Shift key and displays the File menu, the label of the Close menu is Close All Open Files instead of Close. If the user chooses the menu item, the FrameMaker product closes all open files.

Shift menu items actually represent two separate commands, which are linked by their `FP_HasShiftOrUnshiftCommand` and `FP_ShiftOrUnshiftCommand` properties. For example, the Close menu item represents the commands Close and CloseAll. The following
The API allows clients to create and use shift menu items. To combine two commands into a shift menu item, you have to set the FP_HasShiftOrUnshiftCommand and FP_ShiftOrUnshiftCommand properties for only one of the commands. The API automatically sets the properties of the other command for you.

For example, the following code creates a shift menu item representing the client-defined commands, Checkin and CheckinAll. If the user displays the Edit menu normally, the label Check in File appears on it. If the user holds down the Shift key and displays the Edit menu, the label Check in All Open Files appears on it.

```c
#define CHECKIN_CMD 1
#define CHECKIN_ALL_CMD 2
F_ObjHandleT editMenuId, checkinCmdId, checkinAllCmdId;
editMenuId = F_ApiGetNamedObject(FV_SessionId, FO_Menu, "EditMenu");
checkinCmdId = F_ApiDefineAndAddCommand(CHECKIN_CMD, editMenuId, "Checkin","Check in File", "");
checkinAllCmdId = F_ApiDefineCommand(CHECKIN_ALL_CMD, "CheckinAll","Check in All Open Files","");

F_ApiSetInt(editMenuId, checkinCmdId, FP_HasShiftOrUnshiftCommand, FV_ITEM_HAS_SHIFT_COMMAND);
F_ApiSetId(editMenuId, checkinCmdId, FP_ShiftOrUnshiftCommand, checkinAllCmdId);
```

### Providing context-sensitive Help for client menu items

When a user chooses context-sensitive Help in a FrameMaker product, the cursor changes to a question mark. The API allows you to provide context-sensitive Help for your client’s menu items. The API displays the help information that you provide in a FrameMaker hypertext document.

To provide context-sensitive Help for a client-defined menu item, follow these steps:

1. Create a document containing help text for the menu item.
To allow users to navigate within the document, add hypertext markers to it. For more information on using hypertext markers, see your FrameMaker product user documentation. Avoid using numeric link names that may conflict with page numbers.

2. Use Esc Flk to lock the help document.

This allows users to click the document’s hypertext markers and prevents them from altering the document.

3. Copy the help document to the client directory.

On UNIX, the client directory is the directory specified by the <Directory> statement in the _apiclients_ file. On Macintosh, it’s the folder containing the client. On Windows, it’s the directory containing the client DLL.

When the user requests help for one of a client’s menu items, the FrameMaker product looks for the help document in the client’s directory.

4. Set the menu item’s FP_HelpLink property to the filename of the help document.

When the user requests help for the menu item, the FrameMaker product opens the help document specified by the FP_HelpLink property.

If the help information for the menu item is not on the first page of the help document, you can set the FP_HelpLink property so that the FrameMaker product jumps to a specific page or hypertext link in the document. Set the property to _document_name:linkname_or_page_, where _document_name_ is the document name, and _linkname_or_page_ is the linkname of the hypertext link or the page number to jump to.

For example, the following code sets the FP_HelpLink properties of several client-defined menu items:

```c
... 
F_ObjHandleT menuId, cmd1Id, cmd2Id;

/* Opens my.hlp when help is requested. */
F_ApiSetString(menuId, cmd1Id, FP_HelpLink, "my.hlp");

/* Opens my.hlp and jumps to mylink when help is requested. */
F_ApiSetString(menuId, cmd2Id, FP_HelpLink, "my.hlp:mylink");
... 

**IMPORTANT:** Do not set the FP_HelpLink property to a hypertext command, such as gotopage or nextpage.
10 Creating Custom Dialog Boxes for Your Client

This chapter describes how to use FrameMaker to create and modify custom dialog boxes that you can use in your client’s user interface. You can create a dialog box that contains the following items:

- Boxes
- Buttons
- Checkboxes
- Pop-up menus (with a standard appearance or drawn from bitmap images)
- Radio buttons
- Scroll bars
- Scroll lists
- Text boxes (with one or more lines)

If your client’s user interface requires only simple modal dialog boxes, you may not need to create custom dialog boxes. The API provides several simple, ready-made modal dialog boxes. For information on using these dialog boxes, see “Using API dialog boxes to prompt the user for input” on page 17.

Overview

The process of including a custom dialog box in your client involves the following general steps:

1. Create the custom dialog box.
   Creating a dialog box involves designing its layout and items and saving this information in a file format that can be used to build your FDK client. Instructions for this step are presented in this chapter.

2. Write the code in your client to manipulate the dialog box.
   After you create a custom dialog box for your client, you need to add code to your client to manipulate it. For more information, see Chapter 11, “Handling Custom Dialog Box Events”

3. Compile the dialog box with your client in the build process.
   After you write the code for your client, you can compile the code with the files containing dialog box information. Each platform (Windows, Macintosh, UNIX) uses a different build process. For more information, see the FDK Platform Guide for the appropriate platform.
This overview section describes the fundamentals behind the first step of this process, creating dialog boxes for clients. The later sections of this chapter describe the specific procedures in this step of the process.

The end of this chapter lists some helpful tips to keep in mind when editing dialog boxes.

**Designing a dialog box in FrameMaker**

You can use FrameMaker as a dialog editing application. FrameMaker can represent dialog box information as special Frame graphic objects. You can then modify the dialog box and its items just as you would modify standard Frame graphic objects.

Figure 10.1 shows a dialog box edited within FrameMaker.

![Figure 10.1 Using FrameMaker to design a dialog box](image)

**Figure 10.1 Using FrameMaker to design a dialog box**

To edit dialog box information through FrameMaker, you open a special type of file called a Frame dialog resource (DRE) file. As Figure 10.2 shows, when you open a DRE file in FrameMaker, it translates the dialog box information into a graphic representation of the dialog box.

This is similar to opening a Frame binary document in FrameMaker. When you open a Frame binary document, FrameMaker translates the document information into a graphic representation of the document.
Creating Custom Dialog Boxes for Your Client

The rest of this section describes Frame DRE files and how FrameMaker interprets these files.

**Frame DRE files**

A DRE file is a text file that uses special syntax to describe a dialog box and its items. The following lines from a DRE file illustrate how the DRE file syntax describes the OK button in a dialog box:

```xml
<Button
    <MBaseLine 10 327 92>
    <WBaseLine 59 231 53>
    <XBaseLine 63 368 64>
    <Label OK>
    <Active No>
    <HypertextHelp clnthelp:button>
> # 29
```

FrameMaker recognizes this syntax and interpret the descriptions of the dialog box and its items as special Frame graphic objects. Figure 10.3 shows the object properties for the OK button described in the previous example.
Creating Custom Dialog Boxes for Your Client

By moving and resizing these special objects and changing their properties within FrameMaker, you modify the dialog description in the DRE file. When you save a DRE file, the changes that you make are saved back to the file in the special DRE syntax.

**DRE files and platform-specific views of dialog boxes**

You can open and edit a DRE file in any version (Windows, Macintosh, UNIX) of FrameMaker. Since the geometry of a dialog box is different on each platform, the Frame DRE file format uses separate statements to store the geometry information for each platform.

```xml
<DialogBox
  <MBrect -1 -1 358 350> # Dialog box size on the Macintosh
  <WBRect -1 -1 477 465> # Dialog box size in Windows
  <XBRect -1 -1 367 388> # Dialog box size in X Windows
  <Platforms MWX> # Geometry info stored for these platforms
```

At a given time, FrameMaker uses geometry information for a specific platform to display a dialog box. The geometry information specific to one platform is called the *platform view* of a dialog box.

Since a DRE file can contain different platform views, you can edit the Windows, Macintosh, or UNIX view of a dialog box under the same version of FrameMaker. For example, in the UNIX version of FrameMaker, you can edit the UNIX, Windows, and Macintosh views of a dialog box.

Figure 10.4 illustrates how the UNIX version of FrameMaker displays different views of the same dialog box. Note that the first letter in the status bar indicates the current platform view of the dialog box.
Saving dialog box information

When you save a Frame DRE file, all the dialog box information is saved in text format in the special DRE syntax. Some versions of FrameMaker also creates platform-specific resource files for the current version you are using:

- If you are using the Windows version, FrameMaker writes out a Windows dialog resource file (.dlg) and an extra dialog information file (.xdi). The files use the same base name as the DRE file (for example, if your DRE file is named mydialog.dre, FrameMaker writes out the additional files mydialog.dlg and mydialog.xdi).

  These files are resource description files recognized by Windows and are used to compile the dialog box resources with your FDK client.

- If you are using the UNIX version, FrameMaker writes out an X resource file (.x). The files use the same base name as the DRE file. For example, if your DRE file is named mydialog.dre, FrameMaker writes out the additional file mydialog.x.

  This resource description file is recognized by X Windows and is used to display the dialog box for your FDK client.

Be sure you do not use the name of a standard FrameMaker dialog box for as the name of your dialog box. you can see the filenames for the X resource files for the standard FrameMaker dialog boxes in $FMHOME/fminit/UILanguage/xresources. Check these names to be sure you don’t make a dialog box resource with a duplicate name.
The Macintosh version of FrameMaker does not write out any platform-specific resource files.

During the build process on the Macintosh, the DRE file is converted to Macintosh resources, which are placed in the resource fork of the FDK client.

The rest of this section briefly describes how dialog boxes are included in your FDK client. For further details, see the *FDK Platform Guide* for your platform.

### Dialog box information in Windows

In the Windows build process, dialog box information needs to be provided in a Windows dialog resource file (a `.dlg` file). Additional information specific to Frame dialog boxes needs to be provided in a separate file (an `.xdi` file).

For this reason, when you save a DRE file in FrameMaker, the product also writes out the same dialog box information in a `.dlg` file and an `.xdi` file.

Figure 10.5 shows the process of creating a dialog box for a Windows client.

### Dialog box information in UNIX

In UNIX, dialog box information is read dynamically from X resource files when you invoke FrameMaker. For this reason, when you save a DRE file in FrameMaker, the product also writes out the same dialog box information in an X resource file (`.x` file).
Figure 10.6 shows the process of creating a dialog box for a UNIX client.

**Figure 10.6 Creating a dialog box for a UNIX client**

**NOTE:** If your dialog box uses scroll lists, you might want to include the single-quote character in your scroll list strings. For example, you might want to use a single quote to represent length in feet. By default, FrameMaker converts the single-quote character to an escaped double-quote. This is because single quotes are not valid in filenames. To override this default, add the following statement to the dialog box resource file along with the other statements for the given scroll list:

```
Maker.dialog.bb.itemX.isEnc: noConv
```

where `itemX` specifies the item number — for example, `Maker.dialog.bb.item7.isEnc` specifies item 7. If you place this statement at the head of the resource statements for the dialog box item, that scroll list will display single-quotes correctly.

**Dialog box information on the Macintosh**

In the Macintosh build process, dialog box information is read directly from the Frame DRE file. For this reason, no other intermediary resource file is written out when you save a DRE file in FrameMaker.

Figure 10.7 shows the process of creating a dialog box for a Macintosh client.
Creating Custom Dialog Boxes for Your Client

**FIGURE 10.7 Creating a dialog box for a Macintosh client**

**Dialog box information across platforms**

If you are writing a client for multiple platforms, you can use the same DRE file to describe a dialog box on different platforms.

*Figure 10.8* shows an example of the process for using a single DRE file to design the same dialog box on different platforms.

**FIGURE 10.8 Using the same DRE file across different platforms**
Invoking the dialog box by name

When you write the code to invoke the dialog box, use the name of the resource as an argument for opening the dialog resource.

- In Windows and on the Macintosh, the dialog resource is named after the DRE file. For example, if your DRE file is named `mydialog.dre`, the build process creates a dialog resource named `mydialog`.
- On UNIX, the dialog resource is the X resource file.

For more details on displaying dialog boxes, see “Opening dialog resources” on page 217.

How to create a dialog box

The process of creating a dialog box involves the following general steps. Each step is described in more detail in subsequent sections of this chapter.

1. **Using FrameMaker, create a new DRE file from an existing DRE file.**
   
   For details, see the next section.

2. **Design the layout of the dialog box.**
   
   For more information, see the section “Designing the layout of the dialog box” on page 189.

3. **Specify the properties of the dialog box.**
   
   For more information, see the section “Setting the properties of the dialog box” on page 193.

4. **Specify the properties of the dialog items.**
   
   For details, see the section “Setting the properties of a dialog item” on page 197.

5. **Save the new DRE file.**
   
   On some platforms, this creates platform-specific resource files. For details, see the section “Saving a DRE file” on page 204.

6. **Test the dialog box.**
   
   For more information, see the section “Testing a dialog box” on page 205.

Creating a DRE file

The first step in creating a dialog box is to create a DRE file. This file stores all the information about a dialog box and its items.
Since dialog boxes and dialog items are different objects than standard Frame graphic objects, you cannot create a new file (such as a blank portrait document, for example) and draw the dialog box and its items. You must start from an existing DRE file that already contains these objects.

Also, although you may find that you are able to copy and paste dialog items from a DRE file into a regular Frame document, you should not use a Frame document to create a dialog box. A saved Frame document cannot be converted to the dialog resources necessary to create a dialog box.

To create a new DRE file, start FrameMaker and use the DRE file as a template, as follows:

1. Choose New from the FrameMaker File menu.
   
   Using the New command ensures that you don’t overwrite the template.

2. Select an existing DRE file.

   You can use the DRE template file provided with the FDK. You can find the template file in the following locations:

   – \( \text{fdk_install_dir}/\text{samples/dre/template.dre} \) (in UNIX)
   – \( \text{fdk_install_dir: samples:dre:template.dre} \) (on the Macintosh)
   – \( \text{fdk_install_dir\samples\dre\template.dre} \) (in Windows)

   where \( \text{fdk_install_dir} \) is the directory in which you installed the FDK.

   If you have already created your own DRE files, you can select one of them.

   **IMPORTANT:** *If your API client uses custom dialog boxes, and your client will run on a Japanese system with the Japanese language set as the user interface language, then you must use a special template for creating a .dre file. In the samples/dre directory, find jtemplat.dre to use as a template for Japanese dialog boxes.*

3. Click Create.

   FrameMaker displays the DRE file in a standard Frame document window.

   The dialog box and its items appear on a single page. The dialog box is displayed as a rectangle, defining the boundaries of the dialog box.

   Each item in the dialog box is displayed as a special Frame graphic object. Although these objects appear to be composed of standard Frame graphic objects (for example, a checkbox is displayed as a rectangle and a text line), you cannot ungroup a dialog item into separate objects.

   A DRE file can describe a number of different types of dialog items, including:

   – Boxes
   – Buttons
   – Checkboxes
   – Radio buttons
Creating Custom Dialog Boxes for Your Client

- Text boxes
- Multiline text boxes
- Pop-up menus
- Image pop-up menus (pop-up menus that appear as bitmap images)
- Labels
- Scroll bars
- Scroll lists

When you select an item, the document window status bar displays information about the selected item in the following format:

\[
\text{view: item_number.item_type (related_item_number)}
\]

where:

- \text{view} is a letter specifying the current platform view of the dialog box (X for X Windows on UNIX, M for Macintosh, or W for Windows).
  
  For details on setting platform views, see “Setting the platform view of the dialog box” on page 189.
  
- \text{item_number} is the number of the selected item.
  
  In DRE files, the items in a dialog box are identified by unique numbers. Item numbers start from 0 and increase sequentially. You use these numbers in your client code to identify items in the dialog box.
  
- \text{item_type} is the item’s type.
  
- \text{related_item_number} specifies the number of a related item.

  Some types of items (such as text boxes, image pop-up menus, and radio buttons) can be related to other items. If there is no related item, the status bar displays an empty pair of parentheses.
  
  If the item type does not support related items (such as buttons or labels), the status bar does not display parentheses.
  
  For information about related items, see “Relating items in a dialog box” on page 203.

If you select more than one item, the status bar displays information on the item with the lowest number.

\text{Figure 10.9} shows the status bar when an item is selected.
Creating Custom Dialog Boxes for Your Client

**FIGURE 10.9** Item information in the status bar

Information about the selected item.
Designing the layout of the dialog box

FrameMaker displays the dialog box information graphically in a standard Frame document window. Within FrameMaker, you can:

- Set the platform view of the dialog box
- Manipulate the dialog box and its items as graphic objects
- Add new items
- Delete existing items

The rest of this section describes the specific steps for each of these tasks.

Setting the platform view of the dialog box

Before you start editing the dialog box, make sure that you are working with the platform view of the dialog box that you want. FrameMaker can display views of the dialog box as it appears on different platforms.

The status bar displays a letter identifying the current platform view of the dialog box. By default, the current platform view matches the version of FrameMaker you are using (for example, if you are using a UNIX version of FrameMaker, an opened DRE file displays the X Windows view of the dialog box). If the current platform view is not the view you want to work with, change it to the appropriate platform view.

You can change the current platform view of a dialog box by using a keyboard shortcut:

- To work with the UNIX view of the dialog box, press Esc d x.
- To work with the Macintosh view of the dialog box, press Esc d m.
- To work with the Windows view of the dialog box, press Esc d w.

FrameMaker retains the layout and geometry information for each platform view that you use.

How FrameMaker gets view information

When you change to a platform view for the first time, FrameMaker calculates the geometry information for that view. FrameMaker caches this geometry information in local memory during the session.

If you change back to that platform view later, FrameMaker does not recalculate the geometry information; instead the product reads the cached geometry information from local memory.

FrameMaker calculates geometry information for a view only if the geometry information is not already cached.

Updating views and cached information

When you switch views, you may notice that layout changes are not reflected in other views. This occurs because FrameMaker is reading cached geometry information for that view, which has not been updated with changes made to other views.
To update a view with geometry information from another view, you must make FrameMaker recalculate the geometry information. Since FrameMaker calculates geometry information only if the information is not already cached, you must clear the cached information from local memory.

To clear the cached geometry information for the current view, you reset the current platform view.

**Resetting the current platform view**

To reset the current platform view, use the keyboard shortcut Esc d r. FrameMaker clears the cached geometry information for the current view and notifies you that the geometry information is no longer cached.

When you switch to another view, FrameMaker prompts you to save the current view, since the geometry information is no longer cached. If you do not want to cache the geometry information for the current view, choose No in the alert box.

The next time you change to that view, FrameMaker recalculates the geometry information.

For example, suppose you make layout changes in the UNIX view and then change to the Windows view. If the layout displayed in the Windows view does not match the layout in the UNIX view, you need to reset the Windows view.

To update Windows view, follow these steps:

1. Press Esc d r to reset the Windows view, which is the current platform view.
   FrameMaker notifies you that the cached geometry information is cleared from local memory.
2. Press Esc d x to change back to the UNIX view.
   Since no geometry information is cached for the Windows view, FrameMaker prompts you to save the current geometry information.
3. In the alert box, choose No.
   FrameMaker does not cache any Windows geometry information and displays the UNIX view.
4. Press Esc d w to change back to the Windows view.
   Since no geometry information is cached for the Windows view, FrameMaker recalculates the geometry information, based on information from the UNIX view.
   FrameMaker displays the Windows view with the updated layout.

For a summary of keyboard shortcuts, see the section “Summary of keyboard shortcuts” on page 209.

**Manipulating the dialog box and its items**

When designing the layout of the dialog box, you can manipulate the dialog box and its items in the same way that you manipulate standard Frame graphic objects.
**Moving and resizing the dialog box and its items**

To move the dialog box, select it and drag it to its new position. To resize the dialog box, select it and drag one of its object handles.

Do not manually resize the height of a multiline text box or a scroll list. Multiline text boxes and scroll lists have special properties that define height in terms of the number of lines of text displayed. To resize a multiline text box or scroll list, set the height of the item in the Object Properties dialog box for the item. For details, see “Setting basic dialog item properties” on page 197.

Also, the length of a label, check box, or radio button is defined by the length of the text associated with the item. To change the length of any of these items, specify a shorter or longer string of text for the item.

**Using commands to modify the dialog box and its items**

You can use the following commands from the Graphics menu when working with a dialog box and its items:

- Group
- Ungroup (only on items you’ve manually grouped)
- Bring to Front
- Send to Back
- Align
- Distribute
- Scale (see the previous caveats about resizing)
- Object Properties
- Snap

Note that you cannot ungroup a single dialog item into smaller components. For example, you cannot ungroup a checkbox into a box and a label.

For example, you can use the Align command to align dialog box items. When aligning the bottoms of dialog items, note that the bottom of any item containing text is defined by the baseline of its text line.

Some commands affect the display of graphic objects in a DRE file but have no effect on the actual dialog box that you create from the file.

- You can use the Group command to group dialog items for convenience while laying out the dialog box. Although the grouping works when you are modifying the file, the grouping information is not saved in the DRE file. In other words, grouping has no effect on the appearance or functionality in the actual dialog box. For example, if you group checkboxes together, the grouping does not affect the way the checkboxes work.
- Although you can change the line, fill, and color properties of graphic objects in the DRE file, this does not change the appearance of items in the actual dialog box.
The Bring to Front and Send to Back commands do not change the appearance of items in the actual dialog box. Although you can use them while editing the DRE file, you should not allow dialog items to overlap.

The following commands from the Graphics menu have no effect on a dialog box and its items:

- Reshape
- Smooth
- Unsmooth
- Flip Up/Down
- Flip Left/Right
- Rotate
- Set Number of Sides

**IMPORTANT:** The Undo command discards only changes in size and position. The Undo command does not discard any other changes.

**Redisplaying the dialog box**

Measurements in DRE files must use whole number values. If you group items, align items, or work in a zoomed DRE file, FrameMaker might not use whole numbers for measurements. As a result, the display of the dialog box might differ from the actual measurements of the dialog box by up to 0.5 pixels.

To see the correct view of the dialog box, use the shortcut Esc d x, Esc d w, or Esc d m. This redisplays the DRE file with the correct measurements.

**Adding dialog items**

Because dialog items are special Frame graphic objects, you cannot create them by using the Tools palette. To create a dialog item, you need to select an existing item in a DRE file and copy and paste the item.

When you add an item to a dialog box, FrameMaker assigns the next highest item number to the item.

Some dialog items, such as buttons, checkboxes, and radio buttons, already contain labels. You can add a label to dialog items that don’t have labels, such as text boxes, pop-up menus, and scroll lists. **Figure 10.10** shows that the label for a text box is a separate dialog item.

**Figure 10.10 Dialog item with a separate label**
Creating Custom Dialog Boxes for Your Client

A label is considered to be a separate dialog item if it is not part of a button, checkbox, or radio button. To add a label, select a separate label, then copy and paste it.

Deleting items from a dialog box

To delete an item from a dialog box, select the item and press the Delete key.

Note that by deleting an item, you break the sequence of item numbers. For example, if you delete item 3, the sequence of item numbers skips from 2 to 4. You need to renumber the items so that the sequence is unbroken.

To renumber the items in a dialog box, follow these steps:

1. Select the dialog box.
2. From the Graphics menu, choose Object Properties.
   FrameMaker automatically renumbers the items.
3. Click OK to dismiss the Dialog Box Properties dialog box.

Setting the properties of the dialog box

The graphic object that represents a dialog box has special properties that you must set when you create a new DRE file.

To set the properties for a DRE file, follow these steps:

1. In the DRE file, select the rectangle representing the dialog box.
2. From the Graphics menu, choose Object Properties.

The Dialog Box Properties dialog box appears, displaying the properties of the dialog box. You can specify the following properties for a dialog box:

- The order of the items in the dialog box
- The title of the dialog box
- The item initially highlighted or selected by the cursor (called the first focus of the dialog box)
- The button activated by pressing the Return key (called the default button)
- The items that act as the OK, Cancel, and Help buttons
- The size and position of the dialog box (note that modal dialog boxes are always positioned in the center of the screen)

The Item Order list in the Dialog Box Properties dialog box displays a list of all the dialog items in the DRE file. Each dialog item is associated with an item number.

Note the item number of each dialog item. You use these item numbers to identify dialog items in your client code.
Specifying the focus order

When the user selects an item in a dialog box (except a label or a box), the item becomes the focus of the dialog box. For example, if the user clicks in a text box, the text box is the focus.

When the user presses the Tab key, the focus moves from one item to the next in a specific order (for example, if the cursor is in a text box and the user presses the Tab key, the focus might move to a button or pop-up menu). This order is called the focus order of a dialog box.

The focus order of a dialog box is specified by the order of items in the dialog box. For example, suppose item 4 is a text box and item 5 is a radio button. If the text box has the focus, pressing the Tab key moves the focus to the radio button.

The focus order defines the order in which items are selected when the user presses the Tab key. You can change the focus order by changing the order of items in the dialog box.

To change the focus order, follow these steps:

1. Select an item in the scroll list.
2. Click Up or Down to move the selected item up or down in the list.

To move the selected item up or down by 5 items at a time, press the Shift key while clicking Up or Down.

When you select an item in the Item Order scroll list, the corresponding graphic object in the DRE file also appears selected. This helps you to identify the item in the DRE file.
Creating Custom Dialog Boxes for Your Client

If you assign a keyboard shortcut to an item, make sure that the label containing the shortcut and the item are in sequential order. Move the label item so that it precedes the item in the Item Order scroll list.

If you do not assign keyboard shortcuts, the label and the item do not need to be in sequential order.

If you use a box item to group other items, make sure the items it contains appear immediately after it in the Item Order scroll list. Note that this order is important only if the box contains other items; if the box does not contain any items or if the box is used as a separator (see “Boxes” on page 198), the item order is not important.

**Figure 10.12 Item selected in the Item Order scroll list and the DRE file**

**Figure 10.13 Order of items in the Item Order scroll list**
Creating Custom Dialog Boxes for Your Client

Identifying basic items

Each dialog box requires some basic items. You need to specify these items as part of the properties of the dialog box. These items include:

- **Title label**
  The title label is a label that appears in the title bar of the dialog box.

- **First focus**
  The first focus of a dialog box is the item selected when the user first opens the dialog box.

- **Default button**
  The default button is a button the user can activate by pressing the Return key.

- **OK button**
  The OK button is the button that performs an action and dismisses the dialog box.

- **Cancel button**
  If you are creating a modal dialog box, identify a Cancel button. If the user closes the dialog box by clicking its close box (on the Macintosh), pressing Control-w (on Windows), or pressing Control-c (on UNIX platforms), the dialog box activates the Cancel button.

![Figure 10.14 Specifying the basic items in a dialog box](image)

To specify any of these items, type the item number in the appropriate text box. For example, to specify the Cancel button as item 4, type `4` in the Cancel Button text box.

If a dialog box doesn’t use one of these items, type `-1` in the text box for the item. For example, to create a dialog box with no Help button, type `-1` in the Help Button text box.

Specifying the size and position of a dialog box

You can change the properties of a dialog box that specify its initial size and position. You can also adjust the size of the dialog box manually by selecting its rectangle and dragging the handles.

![Figure 10.15 Specifying the size and position of a dialog box](image)
The size and position values do not affect the initial position of modal dialog boxes. Modal dialog boxes always appear in the center of the screen.

**Setting the properties of a dialog item**

Like standard Frame graphic objects, dialog items have properties. These properties define the size and position of the item and relationships with other items.

You can set the following properties for a dialog item:

- Basic properties
- Properties specific to different types of dialog items
- Keyboard shortcuts
- Relationships with other dialog items

The rest of this section describes the different types of dialog items and explains how to set the properties for these items.

**Setting basic dialog item properties**

To set or modify an item’s basic properties, follow these steps:

1. Select the item.
2. From the Graphics menu, choose Object Properties.
3. In the Dialog Item Properties dialog box, specify the properties of the item.

Figure 10.16 shows an example of the Dialog Item Properties dialog box.

![Dialog Item Properties dialog box for a text box](image)
As is the case with standard Frame graphic objects, different dialog items have different properties. Each type of dialog item has a different Dialog Item Properties dialog box. For most types of dialog items, you can set the following properties in the Dialog Item Properties dialog box:

- **Position relative to the dialog box**
  
  In the Left text box, type the offset of the item from the left side of the dialog box. In the Top or Baseline text box, type the offset of the item from the top of the dialog box.

- **Width**
  
  In the Width text box, type the width of the item.

- **Height**
  
  In the Height text box, type the height of the item.

Note that height is set to a fixed value for some items, such as text boxes, labels, and buttons. You cannot specify the height of any of these items. For these items, height is determined by the height of the text that appears in the item.

Similarly, you cannot specify the width of certain items, such as radio buttons, check boxes, and labels. For these items, width is determined by the length of the text that appears in the item.

For most types of items, the Dialog Item Properties dialog box also contains the Other Properties text box and the Stuff Item text box. The Stuff Item text box displays the number of a dialog item that is related to the current item. For instructions on relating dialog items, see “Relating items in a dialog box” on page 203.

Some item types have additional properties. Instructions on setting these properties are covered in the next section.

**Working with specific types of items**

The following sections describe each type of item and item properties not covered in the previous section. You set these properties in one of the following ways:

- Enter values in the Dialog Item Properties dialog box (for example, to specify the initial state of a checkbox)

- Select and edit the item in the DRE file (for example, to specify the text in a label, select the text and type over it)

- Write client code (for example, to specify the menu choices in a pop-up menu)

**Boxes**

A box appears as a rectangle drawn with a single black line. You can use a box to organize the items in a dialog box. The items contained within a box must have numbers that follow the item number of the box. For details, see “Specifying the focus order” on page 194.

You can also use a box as a line separator by setting its width or height to 0.
**IMPORTANT:** *Do not put a box within another box, except in cases where the inside box is a separator (a box with a height or width of 0).*

For information on setting the basic properties for this dialog item, see the section “Setting basic dialog item properties” on page 197.

**Buttons**

A button allows the user to execute a command or invoke an action from a dialog box.

The button item includes a label. To change the text in the label, select the text on the button face and type over it.

For information on setting the basic properties for this dialog item, see the section “Setting basic dialog item properties” on page 197.

**Checkboxes**

A checkbox allows the user to choose from two or three options. A checkbox can have two states (on or off) or three states (on, as is, or off). For example, checkboxes in the FrameMaker Find/Change dialog box have two states; the checkboxes in the Character Designer have three states.

Three-state checkboxes are also called *triboxes*. Triboxes and two-state checkboxes are handled as different objects internally by FrameMaker. For more information, see “How the API represents dialog boxes” on page 211.

In addition to the basic properties of the dialog item (see “Setting basic dialog item properties” on page 197), checkboxes have the following properties:

- Initial state of the checkbox
  
  In the Initial State text box, type 0 if the checkbox is initially off or 1 if it is initially on.

- Number of states for the checkbox
  
  In the States text box, type 2 if the checkbox has two states or 3 if the checkbox has three states.

The checkbox item includes a label. To change the text in the label, select the text next to the checkbox and type over it.

**Radio buttons**

A radio button allows the user to select one choice out of several choices. Each radio button belongs to a set. Only one radio button in a set can be selected at a time.

In addition to the basic properties of the dialog item (see “Setting basic dialog item properties” on page 197), radio buttons have the following properties:

- Initial state of the radio button (on or off)
  
  In the Initial State text box, type 0 if the radio button is initially off or 1 if it is initially on.

- Number of the group to which the radio button belongs
  
  In the Group text box, type the number that identifies the radio button’s group. When you assign group numbers, start with 1.
The radio button item includes a label. To change the text in the label, select the text next to the radio button and type over it. If the text is longer than one line, add another label for each additional line of text.

You can set the label of the radio button in your client code. For details, see “Labels” on page 201.

**Text boxes**

A text box allows the user to enter text. The text box item does not include a label. For instructions on adding a label for a text box, see the section “Adding dialog items” on page 192.

For information on setting the basic properties for this dialog item, see the section “Setting basic dialog item properties” on page 197.

**Multiline text boxes**

A multiline text box is a text box that displays more than one line of text. The multiline text box item does not include a label. For instructions on adding a label for a multiline text box, see the section “Adding dialog items” on page 192.

For information on setting the basic properties for this dialog item, see the section “Setting basic dialog item properties” on page 197.

**Pop-up menus**

A pop-up menu allows the user to select a setting from a list of settings. You can’t specify the list of settings in the DRE file. Instead, you must include code in your client to provide it. For more information, see “Initializing items in a dialog box” on page 218.

The pop-up menu dialog item does not include a label. For instructions on adding a label for a pop-up menu, see the section “Adding dialog items” on page 192.

For information on setting the basic properties for this dialog item, see the section “Setting basic dialog item properties” on page 197.

**Image pop-up menus**

An image pop-up menu is a pop-up menu that appears as a bitmap image of an arrow pointing downward. For an example of this bitmap image, look at the left side of the Paragraph Designer window.

In the Dialog Item Properties dialog box for this item, the name of the bitmap used is specified in the File Name text box. The name can be either `arrowdown` or `tallarrowdn`:

- In UNIX and Windows versions of FrameMaker, these two bitmaps are identical. Do not change these names.
- In Macintosh versions of FrameMaker, `arrowdown` is the same height as a label. `tallarrowdn` is the same height as a text box, which has a greater height than a label. You can change these names to specify the bitmap with the appropriate height.

If you place the image pop-up menu next to a label (for example, such as the Commands label in the Paragraph Designer), use `arrowdown`. If you place the image pop-up menu next to a text box (for example, such as the Paragraph Tag text box in the Paragraph Designer), use `tallarrowdn`. 
You can’t specify the list of settings for an image pop-up menu in the DRE file. Instead, you must include code in your client to provide it. For more information, see “Initializing items in a dialog box” on page 218.

The image pop-up menu item does not include a label. For instructions on adding a label for an image pop-up menu, see the section “Adding dialog items” on page 192.

For information on setting the basic properties for this dialog item, see the section “Setting basic dialog item properties” on page 197.

Labels
A label is a single line of text that you can use to identify other items in a dialog box. Some types of dialog items, such as radio buttons, checkboxes, and buttons, already include labels. For these items, the label is part of the dialog item. Other items, such as text boxes and pop-up menus, do not include labels. For these items, the label is a separate dialog item.

To change the text in a label, select the label’s text in the DRE file and type over it. You can also include code in your client to change a label dynamically. To do this, set the label property on the API object representing the label. For details on the properties of the API object, see “Dialog boxes” on page 531 of the FDK Programmer’s Reference. For instructions on getting and setting properties, see Chapter 5, “Getting and Setting Properties”.

If you set the text of a label in your client code, you must make sure that the DRE file defines an adequate space for the text. Otherwise, the text set by your client may appear truncated. For example, suppose you add a label to a DRE file and specify String as the text in the label. In your client code, if you set this label to String of text, the actual label in the dialog box only displays the word String and truncates the rest of the label (of text).

To prevent this, select the label and type the longest string of text set by your client code. If you do not want this string of text to appear when the dialog box is displayed, you can initialize the value of the label in your client code so that the long string is not displayed by default. For examples of initializing the values of dialog items, see “Initializing items in a dialog box” on page 218.

For information on setting the basic properties for this dialog item, see the section “Setting basic dialog item properties” on page 197.

Scroll bars
A scroll bar allows the user to choose a value within a specified range. Each end of the scroll bar represents one end of the range. You specify a scroll bar’s range in your client code. To do this, set the minimum and maximum value properties on the API object representing the scroll bar. For details on the properties of the API object, see “Dialog boxes” on page 531 of the FDK Programmer’s Reference. For instructions on getting and setting properties, see Chapter 5, “Getting and Setting Properties”.

To display the currently selected value of a scroll bar in the dialog box, add a text box or a label. The dialog box does not automatically stuff the scroll bar value into the text box or label. To do this, you must add code to your client that gets the scroll bar value and programmatically stuffs it into the text box or label.

If you resize a scroll bar by dragging on its object handles, it may appear distorted. To eliminate the distortion, use the keyboard shortcut for viewing the dialog box on the current
platform. For instructions on using this shortcut, see “Setting the platform view of the dialog box” on page 189.

To change the length of a scroll bar, type over the value in the Width or the Height text box:

- If the scroll bar is horizontal, type the length of the scroll bar in the Width text box.
- If the scroll bar is vertical, type the length of the scroll bar in the Height text box.

Note that you cannot change the width of a vertical scroll bar. Similarly, you cannot change the height of a horizontal scroll bar.

To change the orientation of a scroll bar from horizontal to vertical, drag the object handles of the scroll bar so that the height of the scroll bar is greater than the width. The scroll bar appears distorted until you use the keyboard shortcut for viewing the dialog box on the current platform. You can use a similar process to change the orientation from vertical to horizontal.

For information on setting the basic properties for this dialog item, see the section “Setting basic dialog item properties” on page 197.

**Scroll lists**

A scroll list is a list of items from which the user can select an item. You can’t specify the list of items in the DRE file. Instead, you must include code in your client to provide it. For more information, see “Initializing items in a dialog box” on page 218.

The scroll list item does not include a label. For instructions on adding a label for a scroll list, see the section “Adding dialog items” on page 192.

For information on setting the basic properties for this dialog item, see the section “Setting basic dialog item properties” on page 197.

**Specifying keyboard shortcuts for Windows versions**

In Windows, the user can activate or select certain dialog items by pressing a keyboard shortcut. The shortcut is identified by the first underlined letter in the item’s label. When the user presses the Alt key and this letter, the dialog item is activated.

Figure 10.17 illustrates how a shortcut is identified in the label of a text box.
Creating Custom Dialog Boxes for Your Client

To specify a keyboard shortcut for the Windows version of a dialog box, follow these steps:

1. In the DRE file, select the letter of the label to be used for the keyboard shortcut.
2. From the Format menu, choose Style>Underline.

If the item has a separate label item, make sure that its item number follows the label’s item number. For example if the item is a text box, make sure its item number follows the item number of its label. For more information, see “Specifying the focus order” on page 194.

Relating items in a dialog box

You can relate certain types of dialog items to other items. If two items are related, manipulating one changes the other.

For example, in the FrameMaker Paragraph Designer, the Paragraph Tag pop-up menu consists of two dialog items: a text box and an image pop-up menu. The image pop-up menu is the bitmap image of an arrow pointing downward. If the user chooses a setting from the image pop-up menu, the setting is displayed, or stuffed, in the text box. The image pop-up menu’s stuff item is the text box.

Figure 10.18 shows a text box and an image pop-up menu. The text box is the stuff item of the image pop-up menu.

Figure 10.18 Related dialog items

A text box can also be related to a radio button. If a radio button’s stuff item is a text box, turning on the radio button moves focus to the text box. If the stuff item for a text box is a radio button, typing in the text box automatically turns on the radio button.

To specify the stuff item for a dialog item, follow these steps:
1. In the DRE file, select the item.
   
   For example, if you want a text box to be the stuff item for an image pop-up menu, select
   the image pop-up menu first. The order in which you select the items is important.

2. Select the item which will be the stuff item of the first item.

3. To identify the second item you selected as the stuff item for the first item you selected,
   press Esc d f.

   FrameMaker automatically deselects the second item. The first item remains selected, and
   the status bar displays information about this item. The item number of the second item is
   displayed in parentheses, indicating that the second item is the stuff item for the first item.

   For example, suppose you select an image pop-up menu (item 18) first, then you select a text
   box (item 17). Then you use the keyboard shortcut Esc d f. The text box becomes the stuff item
   of the image pop-up menu. FrameMaker deselects the text box. The image pop-up menu
   remains selected, and the status bar displays the following information:

   X:18.ImagePopup(17)

   The number in parentheses indicates that item 17 is the stuff item of item 18.

   You can also specify the stuff item by choosing Object Properties from the Graphics menu and
   typing the number of the related item in the Stuff Item text box.

---

**Saving a DRE file**

To save the DRE file, choose the Save command from the File menu. To save it under a
different name, choose the Save As command from the File menu.

**IMPORTANT:** When naming a DRE file that you will use with UNIX versions of FrameMaker,
do not use asterisks (*), question marks (?), and extra periods (other than the
period used in the filename extension) in the filename. These symbols may
cause problems in the X resource files generated by FrameMaker.

The Save As dialog box appears with the option to save the files as a Frame dialog resource.
This option is only available when you open a DRE file in FrameMaker. If you open a Frame
binary file, this option does not appear.

In UNIX and Windows versions of FrameMaker, saving a DRE file creates additional files:

- UNIX versions of FrameMaker creates X resource files (.x files). Your FDK client uses
  these files to display the dialog boxes.

- Windows versions of FrameMaker creates Windows dialog resource files (.dlg) and extra
dialog information files (.xdi files). You compile these files with FDK client.

When FrameMaker creates these additional files, it names these files after the DRE file. For
example, in Windows versions, saving the file named mydlg.dre creates additional files
named mydlg.dlg and mydlg.xdi. In UNIX versions, saving the file mydlg.dre
creates the X resource file named mydlg.x.
Macintosh versions of FrameMaker do not create any additional files. During the build process, you convert the DRE files into Macintosh resources, which are placed in the resource fork of your FDK client.

In Windows and Macintosh clients, the dialog resource is named after the base name of the DRE file (the filename without the .dre extension). In UNIX clients, the dialog resources are the generated X resource files.

When you open the dialog resource using the `F_ApiOpenResource()` function, use the name for the resource as an argument to the function. The name of the resource is usually the base name of the DRE file.

For details on the process of including dialog boxes in your FDK client, see the *FDK Platform Guide* for your platform. For details on opening dialog resources and displaying dialog boxes, see “Opening dialog resources” on page 217.

### Testing a dialog box

In UNIX and Windows versions of FrameMaker, you can test a dialog box while you are modifying it to verify its appearance and its item focus order. You can test it as a modal dialog box or as a modeless dialog box.

- A modal dialog box prevents the user from performing any other action in FrameMaker while the dialog box is visible. For example, the Print dialog box is a modal dialog box. Until the user clicks Print or Cancel to close the dialog box, the user can’t perform any other action.

- A modeless dialog box allows the user to perform other actions in FrameMaker while the dialog box is displayed. For example, the Marker dialog box is a modeless dialog box. While the Marker dialog box is displayed, the user can do other work in a Frame document.

A DRE file does not specify whether a dialog box is modal or modeless. Your client code determines the dialog box type when it displays it. For more information, see “Displaying a dialog box” on page 219.

To test a dialog box, use one of the following keyboard shortcuts:

- To test the dialog box as modal, press Esc d t.
  
  In this mode, clicking any button dismisses the dialog box.

- To test the dialog box as modeless, press Esc d T.
  
  In this mode, clicking any button does not dismiss the dialog box. To close the dialog box, use the native window manager functionality.

**IMPORTANT:** These keyboard shortcuts only work in UNIX and Windows versions of FrameMaker. If you are using a Macintosh version, you can test the dialog box by using the MPW tool `makedbre`. For details, see the *FDK Platform Guide for the Macintosh*.

Figure 10.19 shows a DRE file and the dialog box displayed when it is tested.
Creating Custom Dialog Boxes for Your Client

In UNIX versions, while you are testing the dialog box as modal, you can use a keyboard shortcut to capture an image of the dialog box. To set this up, include the following line in your .Xdefaults file:

```
Maker.xwdCmd: xwd -id $windowid -frame > $HOME/Screen1.xwd
```

To capture an image of the dialog box, press Esc d t to display the dialog box as modal, and then press Esc d c. This invokes the `xwd` command to capture an image and save it to the file $HOME/Screen1.xwd.

---

**A simple example**

This section provides an example of how to create a simple dialog box named `pgftag`. For instructions on writing the client code to implement this dialog box, see “A simple example” on page 215.

The dialog box is shown in Figure 10.20.

---

**Figure 10.19 Dialog box being tested**

To create the sample dialog box, follow these general steps:

1. Create a new DRE file.
   
   For more information, see “Creating a DRE file” on page 185.

2. Design the layout of the dialog box.
3. Set the properties of the dialog box.

4. Save and test the DRE file.

Note that this example does not involve setting properties for specific dialog items, since the example is relatively simple.

The steps for creating the sample dialog box are described in the following sections.

**Designing the layout of the dialog box**

To design the layout of the sample dialog box, follow these steps:

1. Set the appropriate platform view of the dialog box.
   
   By default, the platform view is the same as the version of FrameMaker that you are using. To change the platform view, use the appropriate keyboard shortcut:
   
   – To work with the UNIX view of the dialog box, press Esc d x.
   – To work with the Macintosh view of the dialog box, press Esc d m.
   – To work with the Windows view of the dialog box, press Esc d w.

2. Delete all extraneous items from the DRE file, except two labels, a text box, and a button.

3. Drag the object handles of the dialog box rectangle to resize it.

4. Select the text in one of the label items and type **Apply Paragraph Tag**.

5. Select the text in the other label item and type **Paragraph Tag**.

6. Select the text in the button’s label and type **Apply**.

7. Drag the object handles of the text box and the button to resize them.

8. Position the items within the dialog box.

   **Figure 10.21** shows the DRE file with the layout of the dialog box completed.

![Completed layout of the dialog items](image)

**Figure 10.21 Completed layout of the dialog items**

9. If you intend to create the dialog box for more than one platform, change the platform view to check the layout on the other platforms.
You may need to make minor adjustments to the layout. If you do, you need to apply the changes to the other platform views. Press Esc d r to reset the other platform views.

Set the properties of the dialog box

To set the properties of the sample dialog box, follow these steps:

1. Select the dialog box in the new DRE file.
2. From the Graphics menu, choose Object Properties.
   The Item Order scroll box lists the three items remaining in this DRE file.
3. Using the Up and Down buttons, change the order of items in the dialog box so that the “Apply Paragraph Tag” label is item 0, the button is item 1, the text box is item 2, and the “Paragraph Tag” label is item 3.
   These numbers are used to identify the dialog items in the code. For details on the code used to run this example, see “A simple example” on page 215.
   Note that the label for the text box follows the text box in consecutive order. Since no keyboard shortcuts are defined in this example, the label of the text box does not need to precede the text box. The label and the text box can be in any order and do not need to be consecutive in the Item Order scroll box.
4. In the Useful Items group box, verify that the following items are specified:
   - First Focus is set to 2.
   - Title Label is set to 0.
   - Default Button is set to 1.
   - OK Button is set to 1.
   Since the dialog box has only one button, you do not need to specify the Cancel button.
5. Click OK to dismiss the Dialog Box Properties dialog box.

Saving and testing the DRE file

When you finish designing the sample dialog box, you need to test and save it:

1. To save the file as a DRE file, choose Save As from the File menu.
2. Type the name:
   pgftag.dre
   If you are in a UNIX or Windows version of FrameMaker, additional files are created:
   - UNIX versions of FrameMaker create a file named pgftag.x.
– Windows versions of FrameMaker create two files, pgftag.dlg and pgftag.xdi. You can use these files with the client code for this dialog box (see “A simple example” on page 215) to build an FDK client. For instructions on creating an FDK client that uses these files, see the FDK Platform Guide for your platform.

3. If you are in a UNIX or Windows version of FrameMaker, press Esc d t to test the dialog box.

If you are using a Macintosh version of FrameMaker, you can test the dialog box in MPW by using the makedbre tool. You can only test the dialog box after saving it. For details, see the FDK Platform Guide for the Macintosh.

**General tips for dialog editing**

When editing dialog boxes, keep the following in mind:

- A DRE file can contain only one dialog box. If the DRE file already contains a dialog box, do not paste another dialog box into the DRE file.
- In the Dialog Item Properties dialog box, FrameMaker displays size and position information in units of points. You cannot switch the units used for display, even by specifying a change in the View Options dialog box. You can, however, use any units to enter data.
- Help buttons should not be added to modeless dialog boxes. Help buttons are mainly used with modal dialog boxes. To get context-sensitive help for modeless dialog boxes, the user can choose Context-Sensitive Help from the Help menu or use the appropriate keyboard shortcut for the platform.

**Summary of keyboard shortcuts**

The following table lists the keyboard shortcuts for editing dialog boxes.

<table>
<thead>
<tr>
<th>To do this:</th>
<th>Use this shortcut:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capture an image of the dialog box while testing as modal (UNIX only)</td>
<td>Esc d c</td>
</tr>
<tr>
<td>Relate the selected items to each other</td>
<td>Esc d f</td>
</tr>
<tr>
<td>Select the first item in the draw order (usually, this is the dialog box in the DRE file)</td>
<td>Esc o F</td>
</tr>
<tr>
<td>Select the next item in the draw order</td>
<td>Esc o n</td>
</tr>
<tr>
<td>Display the properties of the selected dialog box or item</td>
<td>Esc g o</td>
</tr>
<tr>
<td>Test the dialog box as modal (UNIX and Windows only)</td>
<td>Esc d t</td>
</tr>
<tr>
<td>To do this:</td>
<td>Use this shortcut:</td>
</tr>
<tr>
<td>---------------------------------------------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Test the dialog box as modeless (UNIX and Windows only)</td>
<td>Esc d T</td>
</tr>
<tr>
<td>View the dialog box as it appears in the Macintosh versions</td>
<td>Esc d m</td>
</tr>
<tr>
<td>View the dialog box as it appears in the Windows versions</td>
<td>Esc d w</td>
</tr>
<tr>
<td>View the dialog box as it appears in the UNIX versions</td>
<td>Esc d x</td>
</tr>
<tr>
<td>Clear the geometry information for the current platform view from local memory</td>
<td>Esc d r</td>
</tr>
</tbody>
</table>
This chapter describes how to use custom dialog boxes in your client’s user interface. For instructions on creating custom dialog boxes, see Chapter 10, “Creating Custom Dialog Boxes for Your Client”

If your client’s user interface requires only simple modal dialog boxes, you may not need to create or use custom dialog boxes. The API provides several simple, ready-made modal dialog boxes. For information on using these dialog boxes, see “Using API dialog boxes to prompt the user for input” on page 17.

How the API represents dialog boxes

The API uses an FO_DialogResource object to represent each dialog resource in a FrameMaker product session. It also uses an object to represent each item in a dialog resource. The following table lists the types of dialog items and the types of objects the API uses to represent them.

<table>
<thead>
<tr>
<th>Dialog item</th>
<th>API object type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Box</td>
<td>FO_DlgBox</td>
</tr>
<tr>
<td>Button</td>
<td>FO_DlgButton</td>
</tr>
<tr>
<td>Checkbox</td>
<td>FO_DlgCheckBox</td>
</tr>
<tr>
<td>Image pop-up menu</td>
<td>FO_DlgImage</td>
</tr>
<tr>
<td>Label</td>
<td>FO_DlgLabel</td>
</tr>
<tr>
<td>Pop-up menu</td>
<td>FO_DlgPopUp</td>
</tr>
<tr>
<td>Radio button</td>
<td>FO_DlgRadioButton</td>
</tr>
<tr>
<td>Scroll list</td>
<td>FO_DlgScrollBox</td>
</tr>
<tr>
<td>Text box or multiline text box</td>
<td>FO_DlgEditBox</td>
</tr>
<tr>
<td>Three-state checkbox (tribox)</td>
<td>FO_DlgTriBox</td>
</tr>
</tbody>
</table>

The following table lists some dialog item properties.
Handling Custom Dialog Box Events

Not all types of dialog items have all of these properties. For a complete list of properties for each type of dialog item, see “Dialog boxes” on page 531 of the FDK Programmer’s Reference.

### Dialog resource and dialog item IDs

When you open a dialog resource, the API returns its ID. The API also assigns a unique ID to each item in a dialog box. To get a dialog item’s ID, call `F_ApiDialogItemId()`.

The syntax for `F_ApiDialogItemId()` is:

```c
F_ObjHandleT F_ApiDialogItemId(F_ObjHandleT dialogId, 
                                 IntT itemNum);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>dialogId</td>
<td>The ID of the dialog box containing the item</td>
</tr>
<tr>
<td>itemNum</td>
<td>The item number of the item</td>
</tr>
</tbody>
</table>

`F_ApiDialogItemId()` returns the dialog item’s ID or 0 if the item doesn’t exist.

A dialog item’s item number appears in the Dialog Object Properties window for the item when you create the dialog box. It also appears in the Frame dialog resource file following the
Handling Custom Dialog Box Events

A description of the item. For example, the following portion of a Frame dialog resource file describes a text box item with the item number 2:

```xml
<EditBox
<BaseLine 23 74 80>
<Label ImATextBox>
<StuffObject -1>
<HypertextHelp dbre.hlp:edit box>
> # 2
```

If the name of the dialog resource containing this text box is `mydlg.x`, you can use the following code to get its ID:

```c
#define EDITBOX_ITEM_NUM 2
F_ObjHandleT dlgId, editboxId;
dlgId = F_ApiOpenResource(FO_DialogResource, "mydlg");
editboxId = F_ApiDialogItemId(dlgId, EDITBOX_ITEM_NUM);
```

### Getting and setting dialog item properties

You can get and set the dialog item properties with `F_ApiGetPropertyType()` and `F_ApiSetPropertyType()` functions. When you call one of these functions, set its first parameter to the dialog resource ID and its second parameter to the ID of the item for which you want to get or set a property. For more information on using `F_ApiGetPropertyType()` and `F_ApiSetPropertyType()` functions, see Chapter 5, "Getting and Setting Properties"

For example, if you create a dialog resource named `mydlg`, which contains a checkbox with the item number 3, the following code opens the resource and turns the checkbox on:

```c
#define CHECKBOX_ITEM_NUM 3
F_ObjHandleT dlgId;
dlgId = F_ApiOpenResource(FO_DialogResource, "mydlg");
F_ApiSetInt(dlgId, F_ApiDialogItemId(dlgId, CHECKBOX_ITEM_NUM),
    FP_State, FV_DlgOptActive);
```

### Manipulating related items

When you change a dialog box programmatically, the dialog box behaves as if you are changing it interactively. For example, if you programmatically turn on one radio button in a set, the radio button in the set that was previously turned on automatically turns off. If a text box is the stuff item for a pop-up menu, when you choose a setting in the pop-up menu, the item is automatically stuffed in the text box.
Overview of using a custom dialog box in your client

To use a custom dialog box in your client, follow these general steps:

1. Call `F_ApiOpenResource()` to open the dialog resource.
   
   Set the first parameter of `F_ApiOpenResource()` to `FO_DialogResource` and the
   second parameter to the name of the dialog box. `F_ApiOpenResource()` returns the
   dialog resource’s ID.

2. Add code to initialize items in the dialog box.

   A dialog resource does not provide default settings or values for many types of dialog
   items. For example, scroll lists, pop-up menus, and image pop-up menus are empty when
   you first open the dialog resource. If you want any items to have default settings, call
   `F_ApiSetPropertyType()` functions to provide them after you call
   `F_ApiOpenResource()` but before you display the dialog box.

3. Add code to display the dialog box.

   To display a modal dialog box, call `F_ApiModalDialog()`. To display a modeless dialog
   box, call `F_ApiModelessDialog()`.

4. Add code to update the dialog box.

   If the dialog box is modeless, you may want to update it when the user changes things in
   the FrameMaker product session. For example, if the dialog box displays a scroll list of all
   the open documents in a FrameMaker product session, you may want to update the list
   whenever the user opens or closes a document.

   To update a dialog box, turn on notifications, such as `FA_Note_BackToUser` and
   `FA_Note_PostFunction`. Then add code to your client’s `F_ApiNotify()` callback to
   set item properties when it receives these notifications.

5. Add code to respond to user actions in the dialog box.

   How you handle user actions in a dialog box depends on how you display the dialog box. If
   you display it as a modeless dialog box, you should add a callback function named
   `F_ApiDialogEvent()` to your client. The FrameMaker product attempts to call this
   function whenever the user manipulates the dialog box. Your client’s
   `F_ApiDialogEvent()` function can call API functions to get or set the properties of
   items in the dialog box. It can also get and set the properties of other objects in the session
   and call functions to execute operations, such as opening and closing documents.

   If you display a dialog box as a modal dialog box, you can instruct the API to call
   `F_ApiDialogEvent()` for each dialog event just as it would for a modeless dialog box,
   or you can wait until the dialog box is closed and then check the properties of the dialog’s
   items to determine what the user changed.
Handling Custom Dialog Box Events

6. Add code to respond to the user closing the dialog box.

The user can close a dialog box by clicking its close box (on the Macintosh), pressing Control-w (on Windows platforms) or pressing Control-c (on UNIX platforms). When the user closes the dialog box, the FrameMaker product calls your client’s F_ApiDialogEvent() function. Your client may need to conduct some special processing in response to this call. For example, it may need to turn off notifications that it uses for updating the dialog box.

The following sections discuss these steps in greater detail.

A simple example

The following client implements a custom dialog box named pgftag as a modeless dialog box. For instructions on creating this dialog box, see “A simple example” on page 206.

The pgftag dialog box contains a text box that displays the paragraph tag of the paragraph containing the insertion point. The user can change the paragraph tag by typing a different tag in the text box and clicking the Apply button.

Following the code is a line-by-line description of how it works.

1. #include "fapi.h"
2. #define APPLY_BUTTON 1
3. #define TAG_FIELD 2
4. #define DLG_NUM 1
5. F_ObjHandleT dlgId = 0;
6. 7. VoidT F_ApiInitialize(init)
8. IntT init;
9. {
10.   dlgId = F_ApiOpenResource(FO_DialogResource, "pgftag");
11.   F_ApiModelessDialog(DLG_NUM, dlgId);
12.   F_ApiNotification(FA_Note_BackToUser, True);
13. }
14.
15. VoidT F_ApiNotify(notification, docId, filename, iparm)
16. IntT notification;
17. F_ObjHandleT docId;
18. StringT filename;
19. IntT iparm;
20. {
21.   F_TextRangeT tr;
22.   StringT tag;
23.
24.   /* Get tag of first paragraph in selection. */
25.   docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
26.   tr = F_ApiGetTextRange(FV_SessionId,docId, FP_TextSelection);
27.   if(!tr.beg.objId) return; /* No text selected in document. */
28.   tag = F_ApiGetString(docId, tr.beg.objId, FP_Name);
29.
Handling Custom Dialog Box Events

30. F_ApiSetString(dlgId, F_ApiDialogItemId(dlgId, TAG_FIELD),
31.     FP_Text, tag); /* Stuff tag in text box. */
32. F_Free(tag);
33. }
34. }
35. VoidT F_ApiDialogEvent(dlgNum, itemNum, modifiers)
36.   IntT dlgNum;
37.   IntT itemNum;
38.   IntT modifiers;
39. {
40.   F_TextRangeT tr;
41.   F_ObjHandleT docId;
42.   StringT tag;
43.   if (itemNum == FV_DlgClose) /* User closed dialog box. */
44.     F_ApiNotification(FA_Note_BackToUser, False);
45.   if(itemNum != APPLY_BUTTON) return; /* Apply not pressed. */
46.   tag = F_ApiGetString(dlgId, F_ApiDialogItemId(dlgId,
47.     TAG_FIELD), FP_Text);
48.   if(!tag || F_StrLen(tag) < 1) return; /* Text box empty */
49.   /* Get current selection and apply tag to first pgf in it. */
50.   docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
51.   tr = F_ApiGetTextRange(FV_SessionId,docId, FP_TextSelection);
52.   if(!tr.beg.objId) return;
53.   F_ApiSetString(docId, tr.beg.objId, FP_Name, tag);
54.   F_Free(tag);
55. }

Line 5
This line declares the ID of the dialog resource. It is declared as a global variable because the FrameMaker product does not pass dialog IDs to the F_ApiDialogEvent() callback. You need the ID to get and set properties of the items in the dialog box.

Lines 10 to 11
These lines open and display the dialog box when the FrameMaker product starts. The call to F_ApiOpenResource() opens the dialog resource. The call to F_ApiModelessDialog displays the dialog box as a modeless dialog box. DLG_NUM is a unique number used to identify the dialog box. It is passed to the F_ApiDialogEvent() callback when the user does something in the dialog box. If your client uses multiple modeless dialog boxes, you can use the dialog number passed to F_ApiDialogEvent() to determine which dialog box an event has occurred in.

Line 12
This line turns on the FA_Note_BackToUser notification. The FrameMaker product sends this notification to the client each time it finishes processing some user input. For example,
each time the user changes the insertion point or applies a paragraph format, the FrameMaker product sends this notification to the client.

**Lines 26 to 28**

These lines handle the `FA_Note_BackToUser` notification. When the client receives this notification, it is possible the user has changed the insertion point or applied a new tag to the paragraph containing the insertion point. To make sure the dialog box is displaying the correct tag, the client gets the tag of the paragraph containing the insertion point each time it receives the notification.

**Line 30**

This line uses `F_ApiSetString()` to stuff the paragraph tag into the text box. When you get or set dialog item properties, you must specify a dialog box ID and a dialog item ID. The call to `F_ApiDialogItemId()` gets the ID of the text box.

**Lines 35 to 59**

These lines define the `F_ApiDialogEvent()` callback. When a user changes an item in a client-defined dialog box, the FrameMaker product calls this function with `dlgNum` set to the dialog box’s number, `itemNum` set to the number of the item, and `modifiers` set to bit flags specifying the modifier keys the user was holding down.

**Lines 44 to 45**

These lines check `itemNum` to determine whether the user closed the dialog box. When the user closes the dialog box, the FrameMaker product sets `itemNum` to `FV_DlgClose`. If the dialog box was closed, the client no longer needs the `FA_Note_BackToUser` notification and can turn it off.

**Line 47**

This line checks `itemNum` to determine whether the user clicked Apply. If the user did anything but click Apply, the client ignores it and returns.

**Lines 49 to 58**

These lines get the string in the text box and change the tag (`FP_Name` property) of the current paragraph to it. Note that programmatically changing a paragraph tag does not change other properties of the paragraph.

---

**Opening dialog resources**

To open a dialog resource, call `F_ApiOpenResource()`.

The syntax for `F_ApiOpenResource()` is:
F_ObjHandleT F_ApiOpenResource(IntT resourceType,
                     StringT resourceName);

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceType</td>
<td>The type of resource to open. To open a dialog resource, specify FO_DialogResource.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource to open. On UNIX, specify the name of the dialog resource file. On the Macintosh and on Windows platforms, specify the resource name.</td>
</tr>
</tbody>
</table>

F_ApiOpenResource() looks for the specified dialog resource first in your client’s resources. On UNIX platforms, it looks in the directory specified by the client’s <Directory> statement in the apiclients file. On the Macintosh, it looks in the client’s resource fork. On Windows platforms, it looks in the client DLL.

**IMPORTANT:** Your client’s dialog resources must be correctly set up for the API to open them. For information on setting up dialog resources, see the FDK Platform Guide for your platform.

If F_ApiOpenResource() doesn’t find the dialog resource in your client’s resources, it looks in the FrameMaker product’s resources. If it can’t find the dialog resource in either the client or the FrameMaker product resources, it returns 0. If it finds and successfully opens the dialog resource, it returns the dialog resource’s ID.

**IMPORTANT:** To manipulate a dialog box outside of the function in which you opened it, you must save its ID to a global variable.

For example, the following code opens a dialog box named mydlg:

```
F_ObjHandleT dlgId; /* Global declaration */
.
.
dlgId = F_ApiOpenResource(FO_DialogResource, "mydlg");
.
```

### Initializing items in a dialog box

Before you display a dialog box, you can initialize the state or value of the items it contains. The dialog resource provides defaults for most items. However, it does not provide defaults for FO_DlgPopup, FO_DlgScrollBox, and FO_DlgImage items. Be sure to include code to initialize these items.

When you set the FP_Labels property of an FO_DlgPopup item, set the first string in the F_StringsT structure to the title of the pop-up menu. The title appears at the top of the menu when the user clicks on it. Although the title appears only on UNIX platforms, you must specify it on all platforms.
Handling Custom Dialog Box Events

When you first open a dialog box containing an FO_DlgPopup item, the default state of the item is 0, so the title appears as the current choice. Because the title is not a valid choice, you should initialize the FP_State property of FO_DlgPopup items to another number.

For example, the following code initializes a pop-up menu:

```c
F_ObjHandleT dlgId, popupMenuId;
F_StringsT strings;
strings.val = (StringT *) F_Alloc(3*sizeof(StringT), NO_DSE);
strings.len = 3;

strings.val[0] = F_StrCopyString("Popup title");
strings.val[1] = F_StrCopyString("PopupItem1");
strings.val[2] = F_StrCopyString("PopupItem2");
F_ApiSetStrings(dlgId, popupMenuId, FP_Labels, &strings);

/* Make the first item the default. */
F_ApiSetInt(dlgId, popupMenuId, FP_State, 1);
```

When you initialize a dialog box, you may also want to disable some of its items, making them unavailable to the user. A disabled item appears dimmed and can’t receive input focus. To disable an item, set its FP_Sensitivity property to False. For example, the following code disables a dialog item:

```c
F_ObjHandleT dlgId, itemId;
F_ApiSetInt(dlgId, itemId, FP_Sensitivity, False);
```

Displaying a dialog box

After you have opened a dialog resource, you can display it as a modal or a modeless dialog box. To display it as a modal dialog box, call F_ApiModalDialog(). To display it as a modeless dialog box, call F_ApiModelessDialog().

The syntax for F_ApiModalDialog() and F_ApiModelessDialog() is:

```c
IntT F_ApiModalDialog(IntT dlgNum,
                      F_ObjHandleT dlgId);
IntT F_ApiModelessDialog(IntT dlgNum,
                         F_ObjHandleT dlgId);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>dlgNum</td>
<td>A unique number to identify the dialog box. The API passes this number to your client’s F_ApiDialogEvent() callback when there is a user action in the dialog box. If you don’t want the API to call your client’s F_ApiDialogEvent() callback when there is a user action, set dlgNum to 0.</td>
</tr>
</tbody>
</table>
Handling Custom Dialog Box Events

**F_ApiModelessDialog()**  returns immediately. If it can’t display the dialog box, it returns an error code. Otherwise, it returns **FE_Success**.

If you call **F_ApiModalDialog()** with `dlgNum` set to 0, it does not return until the user closes the dialog box. If the user clicks Help in the dialog box, **F_ApiModalDialog()** returns a nonzero value; otherwise, it returns **FE_Success**.

If you set `dlgNum` to a nonzero value, when the user manipulates the dialog box, the API calls your client’s **F_ApiDialogEvent()** callback, just as it does for a modeless dialog box.

### Updating items in a dialog box

If you display a modeless dialog box, you may need to request and monitor notifications to update it when the user changes things in the FrameMaker product session. For example, if the dialog box displays information that is dependent on the insertion point, you should request and monitor the **FA_Note_BackToUser** or **FA_Note_PostFunction** notifications so that you can update the dialog box whenever the user changes the insertion point.

Avoid requesting more notifications than you need to update a dialog box. Also, be sure to turn off notifications after the dialog box is closed. If your client requests notifications that are issued very frequently, it can decrease FrameMaker product performance.

For example, if your dialog box includes a scroll list that displays a list of open documents in the session, you could update the list by requesting and monitoring the **FA_Note_BackToUser** notification. However, this would be inefficient because the FrameMaker product would issue notifications for many events that don’t affect the dialog box. It is much more efficient to request and monitor only the **FA_Note_PostOpenDoc** and **FA_Note_QuitDoc** notifications.

If you need to monitor notifications that are issued very frequently, such as **FA_Note_BackToUser** or **FA_Note_PostFunction**, avoid conducting extensive processing each time they are issued. If you don’t, you may decrease FrameMaker product performance.

For example, suppose you request the **FA_Note_PostFunction** notification so that you can update a dialog box when the insertion point changes. The API issues the **FA_Note_PostFunction** notification for nearly every event in a FrameMaker product session. If you update the entire dialog box every time you receive the notification, it slows the FrameMaker product. It is more efficient to first determine whether the insertion point changed, and then to update the dialog box only if it changed.
Handling user actions in dialog boxes

How you handle user actions in a dialog box depends on whether the dialog is modal or modeless. The following sections discuss how to handle user actions in each type of dialog box.

Handling user actions in a modeless dialog box

After you display a modeless dialog box, the API attempts to call a function named `F_ApiDialogEvent()` from your client whenever the user does something, such as click the mouse or press a key, in the dialog box.

Your client should define `F_ApiDialogEvent()` as follows:

```c
VoidT F_ApiDialogEvent(IntT dlgNum,
                        IntT itemNum,
                        IntT modifiers);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dlgNum</code></td>
<td>The number of the dialog box in which the user action occurred (that is, the number you specified when you displayed the dialog box with <code>F_ApiModelessDialog()</code> or <code>F_ApiModalDialog()</code>).</td>
</tr>
<tr>
<td><code>itemNum</code></td>
<td>If the user manipulated a specific dialog item, <code>itemNum</code> is a nonnegative number specifying the dialog item. If the user didn’t manipulate a specific dialog item, <code>itemNum</code> is a negative number constant specifying what the user did. For example, if the user closed the dialog box, <code>itemNum</code> is set to <code>FV_DlgClose</code>. For a list of the constants, see “Handling special events in a modeless dialog box” on page 226.</td>
</tr>
<tr>
<td><code>modifiers</code></td>
<td>Bit flags specifying which modifier keys the user was holding down when the event occurred. For a list of possible flags, see “<code>F_ApiDialogEvent()</code>” on page 93 of the FDK Programmer’s Reference.</td>
</tr>
</tbody>
</table>

Normally, you will want to include code in the `F_ApiDialogEvent()` function to check the properties of the item specified by `itemNum`. If the dialog box includes any items that the user can double-click, you may also want to check the `FO_DialogResource` property, `FP_DoubleClick`, to determine whether the user double-clicked in the dialog box.

Your `F_ApiDialogEvent()` function can include calls to any API function. It can get and set properties of objects in the dialog box and in the FrameMaker product session. It can also call functions, such as `F_ApiOpen()` or `F_ApiUpdateXrefs()`, to execute FrameMaker product operations.

For example, the following code handles events for a dialog box that contains one of each type of dialog item:
F_ObjHandleT dlgId;
...
#define BUTTON_1 1
#define CHECKBOX_1 2
#define TRIBOX_1 3
#define POPUP_1 4
#define SCROLLBOX_1 5
#define IMAGE_1 6
#define EDITBOX_1 7
#define RADIOBUTTON_1 8

VoidT F_ApiDialogEvent(dlgNum, itemNum, modifiers)
    IntT dlgNum;
    IntT itemNum;
    IntT modifiers;
{
    IntT state;
    F_ObjHandleT itemId;
    StringT text;
    F_StringsT labels;
    if (F_ApiGetInt(0, dlgId, FP_DoubleClick) == True)
        F_Printf(NULL, "The user double-clicked.\n");
    itemId = F_ApiDialogItemId(dlgId, itemNum);
    switch(itemNum)
    {
    case BUTTON_1:
    case CHECKBOX_1:
    case TRIBOX_1:
    case RADIOBUTTON_1:
        state = F_ApiGetInt(dlgId, itemId, FP_State);
        switch(state)
        {
        case FV_DlgOptActive:
            F_Printf(NULL, "%d set to on.\n", itemNum);
            break;
        case FV_DlgOptNotActive:
            F_Printf(NULL, "%d set to off.\n", itemNum);
            break;
        case FV_DlgOptDontCare:
            F_Printf(NULL, "%d set to As Is.\n", itemNum);
            break;
        }
        break;
    case IMAGE_1:
    case POPUP_1:
    case SCROLLBOX_1:
        state = F_ApiGetInt(dlgId, itemId, FP_State);
        labels = F_ApiGetStrings(dlgId, itemId, FP_Labels);
Handling user actions in a modal dialog box

The API allows you to implement two different types of modal dialog boxes:

- Single-interaction dialog boxes, which close as soon as the user clicks an item
- Multiple-interaction dialog boxes, which allow the user to manipulate one or more items without closing

For example, alert boxes are single-interaction modal dialog boxes. The API dialog boxes displayed by `F_ApiPromptInt()`, `F_ApiPromptMetric()`, and `F_ApiPromptString()` are multiple-interaction modal dialog boxes.

The following sections describe how to handle user actions in each type of modal dialog box.

## Handling user actions in single-interaction dialog boxes

To implement a dialog box as a single-interaction modal dialog box, follow these steps:

1. Display the dialog box by calling `F_ApiModalDialog()` with `dlgNum` set to 0. `F_ApiModalDialog()` does not return until the user clicks a dialog item or uses another command, such as Esc or Control-c, to close the dialog box.

2. After `F_ApiModalDialog()` returns, determine what the user did by getting the properties of items in the dialog box.

For example, the following code handles user actions in a modal dialog box that contains Cancel, Apply, and Help buttons:

```c
#define APPLY_BUTTON 1
#define CANCEL_BUTTON 2
F_ObjHandleT dlgId;

/* Open the resource and display the dialog box. */
dlgId = F_ApiOpenResource(FO_DialogResource, "singleInteract");
```
/* Determine what the user action was and respond to it. */
if(!F_ApiModalDialog(0, dlgId))
{
    if(F_ApiGetInt(dlgId, F_ApiDialogItemId(dlgId, APPLY_BUTTON),
         FP_State) == True)
        F_Printf(NULL, "Apply was clicked.\n");
    else if(F_ApiGetInt(dlgId, F_ApiDialogItemId(dlgId,
            CANCEL_BUTTON),FP_State) == True)
        F_Printf(NULL, "Cancel was clicked.\n");
    else
        F_Printf(NULL, "Dialog closed; nothing clicked.\n");
} else
{ /* User requested help; code to provide help goes here. */
}

... Handling user actions in multiple-interaction dialog boxes
To implement a dialog box as a multiple-interaction modal dialog box, follow these steps:

1. Display the dialog box by calling F_ApiModalDialog() with dlgNum set to a nonzero value.

When there is a user action in the dialog box, the API calls your client’s F_ApiDialogEvent() function, passing it the number you specified for dlgNum. Your client’s F_ApiDialogEvent() function can handle user actions in the modal dialog box the same way it handles user actions in a modeless dialog box.

2. Call F_ApiReturnValue(FR_ModalStayUp) in your client’s F_ApiDialogEvent() callback.

If you want to allow the user to click a button without closing the dialog box, call F_ApiReturnValue(FR_ModalStayUp) each time the user clicks the button.

The syntax for F_ApiReturnValue() is:
VoidT F_ApiReturnValue(IntT val);

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>Specifies a return value for the current callback. To prevent a modeless dialog box from closing, set it to FR_ModalStayUp. For a list of the other values you can specify, see “F_ApiReturnValue()” on page 268 of the FDK Programmer’s Reference.</td>
</tr>
</tbody>
</table>

3. To close the dialog box for an event that is not a button, call F_ApiClose().

For more information about F_ApiClose(), see “Closing a dialog box” on page 227.

The following code opens and displays a modal dialog box containing a checkbox. When the user clicks the checkbox, the dialog box remains on the screen. If the user clicks any other button, the dialog box closes.
Handling Custom Dialog Box Events

Handling user actions in multiple modeless dialog boxes

The API allows you to have multiple modeless dialog boxes open at the same time. To handle user actions in multiple dialog boxes, you must keep track of each dialog resource’s number and ID. The API does not pass a dialog resource’s ID to `F_ApiDialogEvent()`, so you must store each dialog resource’s ID to a global variable that you associate with the dialog resource’s number.

For example, the following code opens two dialog boxes and handles user actions in them:

```c
...  
#define DLG1_NUM 1  
#define DLG2_NUM 2  
F_ObjHandleT dlg1Id, dlg2Id; /* Global variables */  
...  
dlg1Id = F_ApiOpenResource(FO_DialogResource, "dialog1");  
dlg2Id = F_ApiOpenResource(FO_DialogResource, "dialog2");  
F_ApiModelessDialog(DLG1_NUM, dlg1Id);  
F_ApiModelessDialog(DLG2_NUM, dlg2Id);  
...  
```
Handling Custom Dialog Box Events

VoidT F_ApiDialogEvent(dlgNum, itemNum, modifiers)
    IntT dlgNum;
    IntT itemNum;
    IntT modifiers;
{
    F_ObjHandleT itemId;
    if (itemNum == FV_DlgClose) return;

    switch(dlgNum)
    {
        case DLG1_NUM:
            itemId = F_ApiDialogItemId(dlg1Id, itemNum);
            /* Code to get item properties goes here. */
            break;
        case DLG2_NUM:
            itemId = F_ApiDialogItemId(dlg2Id, itemNum);
            /* Code to get item properties goes here. */
            break;
    }
}

Handling special events in a modeless dialog box

There are several special dialog box events that don’t apply to specific dialog items. If one of these events occurs, the API sets the itemNum parameter of the F_ApiDialogEvent() function to one of the following negative integer constants:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>FV_DlgClose</td>
<td>The dialog box closed. A dialog box closes when the user makes a dialog close gesture (such as pressing Control-c), when the user exits the FrameMaker product. It also closes the dialog box when your client specifies the dialog box ID in a call to F_ApiClose(), and your code does not call F_ApiReturnValue() to set a return value of FR_ModalStayUp. The API does not specify how a dialog box is closed; it sets itemNum to FV_DlgClose regardless of how the dialog box was closed.</td>
</tr>
<tr>
<td>FV_DlgEnter</td>
<td>The user moved input focus to the dialog box. This event does not occur on UNIX platforms.</td>
</tr>
<tr>
<td>FV_DlgNoChange</td>
<td>The user pressed Shift-F8 to set all the items in a dialog box to their As Is states.</td>
</tr>
<tr>
<td>FV_DlgReset</td>
<td>The user pressed Shift-F9 to reset the items in the dialog box to the values they had the last time the user clicked Apply.</td>
</tr>
<tr>
<td>FV_DlgUndo</td>
<td>The user chose Undo (Command-z on the Macintosh or Control-z on Windows platforms). This event does not occur on UNIX platforms.</td>
</tr>
</tbody>
</table>
Handling Custom Dialog Box Events

Some FrameMaker product dialog boxes, such as the Paragraph Designer and the Character Designer, support the FV_DlgNoChange and FV_DlgReset events. If your client uses dialog boxes similar to these dialog boxes, it should include code to handle these events. For example, to make your client support the FV_DlgNoChange event, add code to do the following to a dialog box:

- Set the state of every tribox to FV_DlgOptDontCare.
- Set the text of every text box to an empty string.
- Set the state of any pop-up menus that contain an As Is item to the index of that item.

Closing a dialog box

To close a dialog box, call F_ApiClose(). The syntax for F_ApiClose() is:

```c
F_ObjHandleT F_ApiClose(F_ObjHandleT objId,
    IntT flags);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>objId</td>
<td>The ID of the dialog box to close.</td>
</tr>
<tr>
<td>flags</td>
<td>Currently an unused parameter. Set it to 0.</td>
</tr>
</tbody>
</table>

You can close a dialog box anywhere in your client code, including the F_ApiDialogEvent() callback. For example, you can call F_ApiClose() to close the dialog box after a dialog event for clicking a radio button.

However, you can call F_ApiReturnValue() to set FR_ModalStayUp in the callback for a given dialog box event. In that case, the FR_ModalStayUp overrides any call to F_ApiClose() for the same event.

**IMPORTANT:** Check the ID you pass to F_ApiClose() to make sure it is not 0. If you call F_ApiClose() with objId set to 0, it quits the Frame session, abandoning any unsaved changes.

For example, the following code closes a dialog box:

```c
... F_ObjHandleT dlgId;
... if(dlgId != 0) F_ApiClose(dlgId, 0);
... ```
This chapter provides instructions for using imported files and insets in your client. It discusses the types of imported files and insets and describes how to import files. It describes how to write a filter client, a client that translates Frame files to or from other file formats. It also discusses graphic inset editors, clients that save graphics in a format that FrameMaker can import.

## Types of imported files and insets

FrameMaker and the Frame API allow you to import graphic and text files by copy and by reference. The following sections briefly describe the types of imported files and insets.

### Imported text files

When the user imports a text file by copy, the FrameMaker product copies the file’s text into the FrameMaker product document. The FrameMaker product no longer needs the original file to display the text.

When the user imports a text file by reference, the FrameMaker product creates an object called a *text inset*. A text inset contains a locked copy of the imported text. It also references the imported text file and specifies how the text is displayed in the FrameMaker product document. The FrameMaker product uses the information in a text inset to display the inset’s text. Each time it updates a text inset, the FrameMaker product uses the text in the referenced file to replace the text in the inset.

There are several types of text insets, which correspond to the types of text files you can import. The following table lists the text inset file types and the corresponding API text inset objects.

<table>
<thead>
<tr>
<th>File type</th>
<th>API inset object that represents it</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>FO_TiText</td>
</tr>
<tr>
<td></td>
<td>FO_TiTextTable</td>
</tr>
<tr>
<td>Frame binary document</td>
<td>FO_TiFlow</td>
</tr>
<tr>
<td>MIF</td>
<td>FO_TiFlow</td>
</tr>
</tbody>
</table>
Using Imported Files and Insets

Client text insets

The API allows you to create a special type of text inset called a *client text inset*. The text for a client text inset is not directly provided by an external file. Instead, it is provided and maintained by an FDK client. The API represents each client text inset in a document with an FO_TiApiClient object. For more information on client text insets, see “Client text insets” on page 236.

Imported graphics files

When the user imports a graphics file, the FrameMaker product creates an object called a *graphic inset*. The API uses an FO_Inset object to represent each graphic inset in a document. An FO_Inset object has properties that specify aspects of how an imported graphic appears, such as its size and scaling.

If the user imports a graphics file by copy, the resulting graphic inset is called an *internal graphic inset*. An internal graphic inset contains all of an imported graphic’s data. If the user imports a graphics file by reference, the resulting inset is called an *external graphic inset*. An external graphic inset does not contain all the data for an imported graphic. Instead, its FP_InsetFile property specifies an external file, which contains the data the FrameMaker product uses to display the graphic.

For a more detailed description of graphic insets and how to manipulate them, see “Writing graphic inset editors” on page 256.

Imported PDF pages

FrameMaker provides special functionality for selecting and importing a page of a PDF file. The page displays within an anchored frame, and is treated as an imported graphic. For an example of how to do this via the FDK, see “Importing a page of a PDF file” on page 233.

Imported markup files

FrameMaker provides special functionality for importing and exporting XML and SGML documents. It provides default import and export, which can handle most markup documents. If you have complex markup, you can customize the default import and export with read/write rules or use the Structure Import/Export API to customize SGML or XML import/export clients. For more information on importing and exporting markup files, see the online manual Structure Import/Export API Programmer’s Guide.

Importing text and graphics

To import text or graphics into a FrameMaker product document, use F_ApiImport(). With F_ApiImport(), you can specify aspects of the Import operation, such as whether to import a file by reference or by copy.
The syntax for `F_ApiImport()` is:

```c
F_0bjHandleT F_ApiImport(F_0bjHandleT enclosingDocId,
    F_TextLocT *textLocP
    StringT filename,
    F_PropValsT *importParamsp,
    F_PropValsT **importReturnParamspp);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>enclosingDocId</td>
<td>The ID of the document into which to import the file.</td>
</tr>
<tr>
<td>textLocP</td>
<td>The text location at which to import the file.</td>
</tr>
<tr>
<td>filename</td>
<td>The full pathname of the file to import. For information on how to</td>
</tr>
<tr>
<td></td>
<td>specify pathnames on different platforms, see the FDK Platform</td>
</tr>
<tr>
<td></td>
<td>Guide for your platform.</td>
</tr>
<tr>
<td>importParamsp</td>
<td>A property list telling the FrameMaker product how to import the</td>
</tr>
<tr>
<td></td>
<td>file and how to respond to errors and other conditions. To use the</td>
</tr>
<tr>
<td></td>
<td>default list, specify NULL.</td>
</tr>
<tr>
<td>importReturnParamspp</td>
<td>A property list that provides information about how the FrameMaker</td>
</tr>
<tr>
<td></td>
<td>product imported the file. It must be initialized before you call F_</td>
</tr>
<tr>
<td></td>
<td>ApiImport(). For a list of properties in this property list, see &quot;F_</td>
</tr>
<tr>
<td></td>
<td>ApiImport()&quot; on page 192 of the FDK Programmer’s Reference.</td>
</tr>
</tbody>
</table>

**IMPORTANT:** Always initialize the pointer to the property list that you specify for `importReturnParamspp` to NULL before you call `F_ApiImport()`.

If you import a text file by reference, `F_ApiImport()` creates a text inset and returns its ID. Otherwise, it returns 0.

The steps for using `F_ApiImport()` are similar to the steps for calling `F_ApiOpen()` and `F_ApiSave()`. To call `F_ApiImport()`, do the following:

1. Initialize the pointer to the `importReturnParamspp` property list to NULL.
2. Create an `importParamsp` property list.
   
   You can get a default list by calling `F_ApiGetImportDefaultParams()`, or you can create a list from scratch. For a description of the default list returned by `F_ApiGetImportDefaultParams()`, see “F_ApiGetImportDefaultParams()” on page 120 of the FDK Programmer’s Reference. For information on creating a property list from scratch, see “Creating an openParamsp script from scratch” on page 50 in this manual.
3. Call `F_ApiImport()`.
4. Check the Import status.

Use `F_ApiCheckStatus()` to check the returned values in the `importReturnParamspp` list for information about how the FrameMaker product imported the file.

5. Deallocate memory for the `importParamsp` and `importReturnParamsp` property lists.

Use `F_ApiDeallocatePropVals()` to deallocate memory for the lists.

The following sections provide examples of how to import several specific types of files.

**Importing the main flow of a Frame document file**

The following code imports the main flow of a Frame document file by reference. It uses the formatting from the source document for the imported text. If the imported file isn’t a FrameMaker product document file, it displays an alert.

```c
F_PropValsT params, *returnParamsp = NULL;
F_ObjHandleT docId;
F_TextRangeT tr;
IntT i;

/* Get default import list. Return if it can’t be allocated. */
params = F_ApiGetImportDefaultParams();
if(params.len == 0) return;

/* Get current selection. Return if there isn’t one. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if(tr.beg.objId == 0) return;

/* Change property to use formatting from source document. */
i = F_ApiGetPropIndex(&params, FS_FormatImportedText);
params.val[i].propVal.u.ival = FV_SourceDoc;

F_ApiImport(docId, &tr.beg, "/tmp/frame.doc", 
&params, &returnParamsp);

if (!F_ApiCheckStatus(returnParamsp, FV_ImportedMakerDoc))
    F_ApiAlert("File wasn’t a Frame document.",
             FF_ALERT_CONTINUE_NOTE);

/* Deallocate property lists. */
F_ApiDeallocatePropVals(&params);
F_ApiDeallocatePropVals(returnParamsp);
```

Importing a graphic

The following code imports a graphic file by copy. It prevents the API from importing the file if it is not a graphic.

```c
F_PropValsT params, *returnParamsp = NULL;
F_ObjHandleT docId;
F_TextRangeT tr;
IntT i;

/* Get default import list. Return if it can’t be allocated. */
params = F_ApiGetImportDefaultParams();
if(params.len == 0) return;

/* Get current insertion point. Return if there isn’t one. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if(tr.beg.objId == 0) return;

/* Change property to import by copy. */
i = F_ApiGetPropIndex(&params, FS_HowToImport);
params.val[i].propVal.u.ival = FV_DoByCopy;

/* Change properties to disallow nongraphic files. */
i = F_ApiGetPropIndex(&params, FS_DisallowDoc);
params.val[i].propVal.u.ival = True;
i = F_ApiGetPropIndex(&params, FS_DisallowMIF);
params.val[i].propVal.u.ival = True;
i = F_ApiGetPropIndex(&params, FS_DisallowPlainText);
params.val[i].propVal.u.ival = True;

F_ApiImport(docId, &tr.beg, "/tmp/agraphic.xwd",
           &params, &returnParamsp);

if (F_ApiCheckStatus(returnParamsp, FV_BadImportFileType))
   F_ApiAlert("File isn’t importable.", FF_ALERT_CONTINUE_NOTE);

/* Deallocate property lists. */
F_ApiDeallocatePropVals(&params);
F_ApiDeallocatePropVals(returnParamsp);
```

Importing a page of a PDF file

The following code shows how to import the specified page of a PDF file. Notice that insets for PDF pages have an FP_PageNum property—no other insets created by FrameMaker use this property. You determine the page to display in the inset by setting FP_PageNum, where 0 is the first page in the PDF file.
As of this writing (FrameMaker 7.0), the order of these calls is important. You must coerce the page display by specifying the page number before setting the path for the inset, otherwise FrameMaker will display the first page of the inset. If you want to change the page it displays, you must similarly coerce FrameMaker to redisplay the inset. You can do this by setting the inset path to a dummy string, setting the new page, then setting the path to the PDF file.

```c
if(F_ApiChooseFile(
    &path, "Select a file to import.", ",", ",", FV_ChooseSelect, ",")
    return;
    if(F_StrIsEmpty(path))
        return;

    /* Now the code has a path - prompt for a page num. */
    /* If the user doesn’t cancel, create an anchored frame and */
    /* place an inset within it. Then set the inset properties. */
    /* To be useful, this code would need to set properties for */
    /* the anchored frame position and size, as well. */
    if(!F_ApiPromptInt(&pgNum, "PDF PageNum?", "1")) {
        tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
        frameId = F_ApiNewAnchoredObject(docId, FO_AFrame, &tr.beg);
        insetId = F_ApiNewGraphicObject(docId, FO_Inset, frameId);
        F_ApiSetInt(docId, insetId, FP_PageNum, pgNum-1);
        F_ApiSetString(docId, insetId, FP_InsetFile, path);
        F_ApiReformat(docId);
    }

    /* Deallocate the string. */
    F_ApiDeallocateString(&path);
)
```

**Importing a text file**

The following code imports a text file by reference into a table. It parses each paragraph in the text file into a row of cells, interpreting each tab in the paragraph as a cell separator. Notice how the code uses the FS_FileTypeHint parameter to specify the encoding for the text file.

```c
F_PropValsT params, *returnParamsp = NULL;
F_ObjHandleT docId;
F_TextRangeT tr;
IntT i;

    /* Get default import list. Return if it can’t be allocated. */
    params = F_ApiGetImportDefaultParams();
    if(params.len == 0) return;

    /* Get current insertion point. Return if there isn’t one. */
Using Imported Files and Insets

```c
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
if(tr.beg.objId == 0) return;

/* Change properties to import text into a table. */
i = F_ApiGetPropIndex(&params, FS_FileIsText);
params.val[i].propVal.u.ival = FV_DoImportAsTable;
/* The following specifies the file format as ISO encoded text. */
i = F_ApiGetPropIndex(&params, FS_FileTypeHint);
params.val[i].propVal.u.sval = F_ApiCopyString((ConStringT)"0001PGRFTXIS");
i = F_ApiGetPropIndex(&params, FS_DisallowMIF);
params.val[i].propVal.u.ival = True;
i = F_ApiGetPropIndex(&params, FS_DisallowDoc);
params.val[i].propVal.u.ival = True;
i = F_ApiGetPropIndex(&params, FS_DisallowGraphicTypes);
params.val[i].propVal.u.ival = True;
i = F_ApiGetPropIndex(&params, FS_ImportTblTag);
params.val[i].propVal.u.sval =
(StringT) F_StrCopyString("Format A");
i = F_ApiGetPropIndex(&params, FS_CellSeparator);
params.val[i].propVal.u.sval = (StringT) F_StrCopyString("\t");
F_ApiImport(docId, &tr.beg, "/tmp/mydata.txt",
params, &returnParamsp);

if (F_ApiCheckStatus(returnParamsp, FV_BadImportFileType))
    F_ApiAlert("File isn’t importable.", FF_ALERT_CONTINUE_NOTE);

/* Deallocate property lists. */
F_ApiDeallocatePropVals(&params);
F_ApiDeallocatePropVals(returnParamsp);
```

---

### Updating text insets

To update text insets, call `F_ApiUpdateTextInset()`. The syntax for `F_ApiUpdateTextInset()` is:

```c
IntT F_ApiUpdateTextInset(F_ObjHandleT docId,
    F_ObjHandleT textInsetId);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>docId</code></td>
<td>The ID of the document containing the inset.</td>
</tr>
<tr>
<td><code>textInsetId</code></td>
<td>The ID of the text inset to update. To update all the insets in the specified document, specify 0.</td>
</tr>
</tbody>
</table>
F_ApiUpdateTextInset() updates a text inset only if it is stale. The FrameMaker product determines whether a text inset is stale by comparing the modification date of the inset’s source file with the inset’s FP_LastUpdate property. To force F_ApiUpdateTextInset() to update an inset, set the inset’s FP_LastUpdate property to 0 before calling it. You do not need to unlock any insets when you call F_ApiUpdateTextInset().

**Client text insets**

Client text insets allow your client to display and dynamically update segments of locked text in a Frame document. The following sections describe how to create and update client text insets.

**Creating a client text inset**

To create a client text inset, use F_ApiNewAnchoredObject(). To add text to it, use F_ApiAddText(). After you create an inset, you may also want to set the properties listed in the following table to provide information about the inset to the user and the FrameMaker product.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_TiClientName</td>
<td>StringT</td>
<td>The registered name of your client.</td>
</tr>
<tr>
<td>FP_TiClientSource</td>
<td>StringT</td>
<td>The name that appears as the source in the Text Inset Properties dialog box.</td>
</tr>
<tr>
<td>FP_TiClientType</td>
<td>StringT</td>
<td>The name that appears as the source type in the Text Inset Properties dialog box.</td>
</tr>
<tr>
<td>FP_Name</td>
<td>StringT</td>
<td>The inset name. It is not automatically assigned by the FrameMaker product.</td>
</tr>
</tbody>
</table>

You can also use a client text inset’s FP_TiClientData property to store data, such as an SQL query string, which your client can use to update the inset.

For example, the following code creates a client text inset containing the text Inset text:

```c
F_ObjHandleT docId, insetId;
F_TextRangeT tr;

/* Add the inset at the current insertion point. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
tr = F_ApiGetTextRange(0, docId, FP_TextSelection);
if(tr.beg.objId == 0) return;
insetId = F_ApiNewAnchoredObject(docId, FO_TiApiClient, &tr.beg);
```
tr.beg.offset++; /* Increment textLoc so it’s in the inset. */

/* Unlock the inset, add text to it, and then relock it. */
F_ApiSetInt(docId, insetId, FP_TiLocked, False);
F_ApiAddText(docId, &tr.beg, "Inset text");
F_ApiSetInt(docId, insetId, FP_TiLocked, True);

/* Set some other inset info. */
F_ApiSetString(docId, insetId, FP_TiClientData, "Select...Where...");
F_ApiSetString(docId, insetId, FP_TiClientSource, "Larry’s SQL Client");
F_ApiSetString(docId, insetId, FP_TiClientType, "SQL Query");
...

**Updating a client text inset**

The API issues the following notifications when client text insets need to be updated:

- **FA_Note_UpdateAllClientTi**, when the user or an FDK client has instructed the FrameMaker product to update all the insets in the document
- **FA_Note_UpdateClientTi**, when the user or an FDK client has instructed the FrameMaker product to update one of your client’s insets

When it issues these notifications, the API sets the docId parameter of your client’s F_ApiNotify() callback to the ID of the enclosing document. For the FA_Note_UpdateClientTi notification, it also sets the iparm parameter of your client’s F_ApiNotify() callback to the ID of the inset. Your client can use the FP_LastUpdate property of an inset to determine whether it is stale.

To keep your client’s text insets updated, you should request these notifications and include code in your client’s F_ApiNotify() callback to handle them.

To modify a client text inset’s contents, you must first unlock it by setting its FP_TiLocked property to False. When you are finished modifying its contents, you should set its FP_TiLocked property back to True.

To update a client text inset, you may want to delete its current contents. The API provides a function, **F_ApiDeleteTextInsetContents()**, which makes this easier. The syntax for F_ApiDeleteTextInsetContents() is:

```c
IntT F_ApiDeleteTextInsetContents (F_ObjHandleT docId, F_ObjHandleT insetId);
```

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document containing the text inset</td>
</tr>
<tr>
<td>insetId</td>
<td>The text inset containing the text to be deleted</td>
</tr>
</tbody>
</table>
Using Imported Files and Insets

If your client is unable to update one of its client text insets, it should set the inset’s `FP_TiIsUnresolved` property to `True`.

For example, the following code handles the `FA_Note_UpdateClientTi` notification:

```c
/* Request notification. */
F_ApiNotification(FA_Note_UpdateClientTi, True);

/* F_ApiNotify() function to handle notifications. */
VoidT F_ApiNotify(notification, docId, sparm, iparm)
IntT notification;
F_ObjHandleT docId;
StringT sparm;
IntT iparm;
{
F_TextRangeT tr;
if(notification == FA_Note_UpdateClientTi)
{
    FA_errno = FE_Success; /* Initialize and check later. */
    tr = F_ApiGetTextRange(docId, iparm, FP_TextRange);

    /* Unlock inset so it can be modified.*/
    F_ApiSetInt(docId, iparm, FP_TiLocked, False);

    /* Delete existing contents and add some new stuff.*/
    F_ApiDeleteTextInsetContents(docId, iparm);
    F_ApiAddText(docId, &tr.beg, "New text");

    /* If there were errors, the inset is unresolved. */
    if(FA_errno != FE_Success)
        F_ApiSetInt(docId, iparm, FP_TiIsUnresolved, True);

    /* Relock inset.*/
    F_ApiSetInt(docId, iparm, FP_TiLocked, True);
}
}
```

Displaying a Text Inset Properties dialog box

When a user double-clicks a text inset that isn’t a client text inset, the FrameMaker product displays the Text Inset Properties dialog box. This dialog box provides information about the text inset, such as the last modification date of the inset’s source file and the date the inset was last updated. It also provides buttons for the user to execute some operations, such as convert the inset to text.

When a user double-clicks one of your client’s text insets, the Frame product does not display a Text Inset Properties dialog box. To display a Text Inset Properties dialog box, your client must request the `FA_Note_DisplayClientTiDialog` notification. If your client has requested this notification, when the user double-clicks one of its insets, the FrameMaker
Using Imported Files and Insets

product calls the client’s F_ApiNotify() callback with notification set to FA_Note_DisplayClientTiDialog and iparm set to the inset ID. When your client receives the notification, it can display its own Text Inset Properties dialog box. This dialog box does not need to appear the same as the FrameMaker product Text Inset Properties dialog box. For example, if your client updates client text insets by executing database queries, its Text Inset Properties dialog box could provide a text field for the user to enter a new query.

After your client displays its Text Inset Properties, it should call F_ApiReturnValue() with retVal set to FR_DisplayedTiDialog. This notifies the FrameMaker product that the dialog box has been displayed.

For more information on requesting and responding to notifications, see “Responding to user-initiated events or FrameMaker product operations” on page 34. For more information on using custom dialog boxes in your client, see Chapter 11, “Handling Custom Dialog Box Events”

---

Writing filter clients

You can use the FDK to create filter clients that translate Frame files to or from other file formats. The FrameMaker product calls an import filter client when the user or another client attempts to open or import a file with a specified format. It calls an export filter client when the user chooses a particular format from the Format pop-up menu of the Save As dialog box or the user or another client saves a file with a specified suffix.

**IMPORTANT:** To write clients that import and export SGML documents, use the Structure Import/Export API. For more information, see the online manual Structure Import/Export API Programmer’s Guide.

Filter clients that filter text file formats are called **text filter clients**. Filter clients that filter graphic file formats are called **graphic filter clients**. The following sections describe how to write each type of filter client and how to register filter clients.

**Writing text import filters**

The FrameMaker product invokes a text import filter in the following situations:

- The user attempts to open a file with a format that the client filters.
- The user attempts to import a file with a format that the client filters.
- Another client attempts to import or open a file with a format that the client filters.
- The FrameMaker product attempts to update a text inset that references a file with a format that the client filters.

The FrameMaker product invokes the client the same way in each of these situations. It calls the client’s F_ApiNotify() callback with notification set to FA_Note_FilterIn, docId set to the ID of the active document (if there is one), and sparm set to the pathname of the file to filter.
The client’s `F_ApiNotify()` callback should do the following to respond to the FrameMaker product’s call:

   
   The client can create the document with `F_ApiOpen()` or `F_ApiCustomDoc()`. For more information on using these functions, see “Creating documents” on page 53. If the new document is not invisible, it will cause an error.

2. Filter the contents of the specified file into the new document.
   
   The client can use Frame API calls, such as `F_ApiAddText()` and `F_ApiNewTable()`, to add content to the document. For more information on adding text and objects to a document, see Chapter 6, “Manipulating Text” and Chapter 8, “Creating and Deleting API Objects”.

   The FrameMaker product allows users and clients that call `F_ApiImport()` to specify into which flow of a document to import. The user or client can specify any flow in the document. A filter client should generally filter a file into the main flow of the document it creates. However, it can filter the file into any flow in the document. For more information on main flows, see “Main flows” on page 297.

3. Call `F_ApiReturnValue()` to indicate whether the file was successfully filtered.
   
   If the client successfully filters the file, it should call `F_ApiReturnValue(docId)`, where `docId` is the ID of the Frame document the filter created. If the client fails, it should call `F_ApiReturnValue(0)`.

   When the client’s `F_ApiNotify()` callback returns, the FrameMaker product checks the value set by the `F_ApiReturnValue()` call. If the value set by the `F_ApiReturnValue()` call is 0, the FrameMaker product displays an alert notifying the user that the file could not be opened or imported.

   If the value set by the `F_ApiReturnValue()` call is the ID of the new document the client created, what the FrameMaker product does depends on how the filter call was initiated. The following table summarizes the situations in which a filter call can be initiated and what the FrameMaker product does in each situation.

<table>
<thead>
<tr>
<th>How the filter call was initiated</th>
<th>What the FrameMaker product does</th>
</tr>
</thead>
<tbody>
<tr>
<td>The user attempted to open the filterable file.</td>
<td>Makes the document created by the filter client visible.</td>
</tr>
<tr>
<td>Another client attempted to open the filterable file (with <code>F_ApiOpen()</code>).</td>
<td>Returns the ID of the document created by the filter client to the other client’s <code>F_ApiOpen()</code> call. If the filter fails to open the filterable file, the FrameMaker product returns 0 to the other client’s <code>F_ApiOpen()</code> call. If the <code>F_ApiOpen()</code> call specified a property list with <code>FS_MakeVisible</code> set to <code>True</code>, the FrameMaker product makes the document visible.</td>
</tr>
</tbody>
</table>
### Writing graphic import filters

The FrameMaker product invokes a graphic import filter in the following situations:

- When the user attempts to open a graphic file with a format that the client filters
- When the user attempts to import a graphic file with a format that the client filters
- When another client attempts to import or open a graphic file with a format that the client filters

The FrameMaker product invokes the client the same way in each of these situations. It calls the client’s `F_ApiNotify()` callback with `notification` set to `FA_Note_FilterIn`, `docId` set to the ID of the active document (if there is one), and `sparm` set to the pathname of the file to filter.

The client’s `F_ApiNotify()` callback should do the following to respond to the FrameMaker product’s call:

<table>
<thead>
<tr>
<th>How the filter call was initiated</th>
<th>What the FrameMaker product does</th>
</tr>
</thead>
<tbody>
<tr>
<td>The user attempted to import the filterable file by copy into an existing document.</td>
<td>Copies the specified flow of the document the filter client created into the existing document. Closes the document the filter client created.</td>
</tr>
<tr>
<td>Another client attempted to import the filterable file by copy (with <code>F_ApiImport()</code>) into an existing document.</td>
<td>Creates a text inset in the existing document and then copies the specified flow of the document the filter client created into the text inset.</td>
</tr>
<tr>
<td>The user attempted to import the filterable file by reference into an existing document.</td>
<td>Closes the document the filter client created. Sets the <code>FP_TiFile</code> property of the inset to the pathname of the filterable file.</td>
</tr>
<tr>
<td>Another client attempted to import the filterable file by reference (with <code>F_ApiImport()</code>) into an existing document.</td>
<td>Returns the ID of the inset created by the filter client to the other client’s <code>F_ApiImport()</code> call. Sets the <code>FP_ImportHint</code> property of the text inset to a string specifying the filter client. This string is based on the format ID you specified when you registered your client. For more information on format IDs, see “Specifying format IDs and filetype hint strings” on page 247. For the complete syntax of import hint strings, see “Syntax of FP_ImportHint strings” on page 600 of the FDK Programmer’s Reference.</td>
</tr>
<tr>
<td>The FrameMaker product attempted to update a text inset that references the filterable file.</td>
<td>Replaces the contents of the inset with the specified flow of the document the filter client created. Closes the document the filter client created.</td>
</tr>
</tbody>
</table>

For a simple example of a text import filter, see “A simple FDE filter” on page 370.
Using Imported Files and Insets

1. Determine whether to open or import the graphic file.

   If the user or another client is attempting to open the graphic file, the FrameMaker product sets the docId parameter to 0 when it calls the client’s F_ApiNotify() callback. If the user or client is attempting to import the graphic file into an existing document, the FrameMaker product sets the docId parameter to a document ID.

2. To open the graphic file, create a new FrameMaker product document.

   The client can create the document with F_ApiOpen() or F_ApiCustomDoc(). The client can filter the imported graphic directly onto a page of the document or it can create an anchored frame for the graphic.

3. To import the graphic file into an existing document, determine where to import the graphic.

   The client should check the document specified by the docId parameter of the F_ApiNotify() call. If the document has an insertion point, the client should create an anchored frame at the insertion point to filter the graphic into. If there is a selected frame in the document, the client should filter the graphic into the frame.

4. Filter the graphic file into the FrameMaker product document.

   The client can translate the graphic file into FrameMaker product graphic objects such as ellipses, lines, and rectangles (FO_Ellipse, FO_Line, and FO_Rectangle objects). For more information on creating FrameMaker product graphic objects, see “Creating graphic objects” on page 146.

   If the graphic file contains a complex graphic, the client can create a graphic inset. Graphic insets provide graphic data in standard formats, such as TIFF or FrameVector, which the FrameMaker product can use to display and print a graphic. For more information on creating graphic insets, see “Writing graphic inset editors” on page 256.

   Note that a client does not need to be a graphic inset editor to create a graphic inset. A filter client can also create a graphic inset.

Writing export filters

The FrameMaker product invokes an export filter when the user chooses a particular format from the Format pop-up menu of the Save As dialog box or when the user or another client saves a file with a specified suffix. The FrameMaker product calls the client’s F_ApiNotify() callback with notification set to FA_Note_FilterOut, docId set to the ID of the document to filter, and sparm set to the pathname of the file to filter the document into. The client’s F_ApiNotify() callback should create the specified file if it does not already exist and filter the contents of the FrameMaker product document into it.
Writing file-to-file filters

A file-to-file filter takes a file of a specified format as input and writes a version of that file to disk in another format. This is particularly useful for filtering graphic files that will be imported by reference.

Suppose you write a standard filter that converts a graphic file to FrameVector format. When you import the graphic file by reference, the FrameVector file will actually be the referenced file. Instead, had you used a file-to-file filter for this conversion, the original graphic file would be the referenced file and FrameMaker would filter it every time you opened the document. Subsequent changes to the imported file in its native format would automatically show up in your document.

The FrameMaker product invokes a file-to-file filter either as a text import filter, a graphic import filter, a text export filter, or a graphic export filter. How the filter is invoked depends on the import or export actions made by a user or by some other client; open actions invoke an import filter, and save actions invoke an export filter.

It’s important to note that the a file-to-file filter client can contain more than one filter process. For example, a single client can contain processes for both import CGM as FrameVector and export FrameVector as CGM. Or a single text file-to-file filter client can include processes for filtering numerous versions of a given format. However, each process must be registered with a unique filter name. For more information about registering filter clients, see “Registering filters” on page 246 of this manual, and the platform guide for the platform of interest.

Data passed to F_ApiNotify() for file-to-file filters

The FrameMaker product calls the client’s F_ApiNotify() callback with notification set to FA_Note_FilterFileToFile. The value for docId is set to:

- for import filters, the ID of the active document (if there is one) or 0.
- for export filters, the document to filter, or the document containing the graphic to filter.

The value for sparm points to an F_FilterArgsT structure, which you access by casting sparm as follows (assuming argp is declared as a pointer to F_FilterArgsT):

```c
argp = (F_FilterArgsT *)sparm;
```

F_FilterArgsT is defined as:

```c
typedef struct F_FilterArgsT {
    ConStringT infile;
    ConStringT outfile;
    ConStringT clname;
    ConStringT informat;
    ConStringT outformat;
    ConStringT binname;
    F_StringsT args; /* reserved for future use; currently NULL */
} F_FilterArgsT;
```

As previously stated, a file-to-file filter client can include more than one process, but you must register each process separately, with unique filter names. clname contains the unique client name, and binname contains the client’s filename.
The values for *informat* and *outformat* come from the registration data for a particular filter process.

For both import and export filters, *infile* is the path of the file to filter, and *outfile* is the path for the resulting file. The FrameMaker product supplies these values. It’s possible that the FrameMaker product has already created a file at the location specified by *outfile*; however, your client will need to create this file if it doesn’t already exist.

The example on page 245 shows how to get the values stored in a *F_FilterArgsT* structure, and how to call *F_ChannelOpen()* to ensure you can write to *outfile*.

**How F_ApiNotify() should respond**

The client’s *F_ApiNotify()* callback should do the following to respond to FrameMaker calling a file-to-file filter:

1. Open the input file for reading, and open the output file for writing.
   
   The paths for these files are stored in the *FilterArgsT* data structure.

2. Filter the *infile* into the *outfile*.
   
   If you’re filtering a graphic file that contains a complex graphic, the client can create a graphic inset. Graphic insets provide graphic data in standard formats, such as TIFF or FrameVector, which the FrameMaker product can use to display and print a graphic. For more information on creating graphic insets, see “Writing graphic inset editors” on page 256.

   **IMPORTANT:** The *outfile* must be in a format that FrameMaker can read natively. For a document, filter the file to MIF or a FrameMaker document file. For a graphic, filter the file to MIF, FrameVector, TIFF, or any other graphic format FrameMaker can read natively.
   
   You must specify the *outfile* format when you register the filter, and your filter must create valid data for that format.

3. Set the return value to 0 (zero)
   
   Call *F_ApiReturnValue(0)*; to indicate success. Specifying any non-zero value indicates failure.

   **IMPORTANT:** File-to-file filters differ from other filters in the return value that indicates success. If you copy standardfilter code to create a file-to-file filter, be sure to change the return values so they indicate success or failure for a file-to-file filter.

Note that file-to-file filters can use *F_Progress()* to report their progress to the FrameMaker product. On Windows and Macintosh platforms, this function displays a progress indicator (a thermometer), and the user can cancel the filter by clicking the indicator’s Cancel button. On Unix, this function doesn’t display anything, but calling this function provides points in the filter process at which the user can press Control-C to cancel. If the user cancels, *F_Progress()* returns a non-zero value.
Using Imported Files and Insets

Example
The following code shows how to get values out of the F_FilterArgsT structure. Then it opens infile and writes a MIF version to outfile:

```
#include "fdetypes.h"
#include "fapi.h"
#include "fchannel.h"
#include "futils.h"
#include "fioutils.h"
#include "fmifstmt.h"

#define BUFSIZE 256

VoidT F_ApiNotify(notification, docId, sparm, iparm)
    IntT notification;
    F_ObjHandleT docId;
    StringT sparm;
    IntT iparm;
{
    FilePathT *inPath = NULL;
    FilePathT *outPath = NULL;
    ChannelT inChan = NULL;
    ChannelT outChan = NULL;
    UCharT buf[BUFSIZE];
    F_FilterArgsT *argp;
    IntT count;

    F_ApiBailOut();
    F_FdeInit();

    /*
    ** Here is where the code gets access to the filter data.
    */
    argp = (F_FilterArgsT *)sparm;

    /*
    ** Next, the code declares the necessary file I/O.
    */
    if((inPath=F_PathNameToFilePath(argp->infile,NULL,FDefaultPath))== NULL)
        return;
    /* These next 3 returns should free the preceeding paths and channels */
    if((inChan = F_ChannelOpen(inPath,"r")) == NULL)
        return;
    if((outPath=F_PathNameToFilePath(argp->outfile,NULL,FDefaultPath))==NULL)
        return;
    if((outChan = F_ChannelOpen(outPath,"a")) == NULL)
        return;

    /*
    ** Now the code sets and initializes the MIF output file.
    */
```
Using Imported Files and Insets

F_MifSetOutputChannel(outChan);
F_MifSetIndent(0);
F_MifMIFFile(7.0);

/*
 ** The code reads the input file and write MIF to the output file.
 ** This isn’t a useful example - you would need to parse the input
 ** in order to find reasonable paragraph boundaries.
 */
while(!F_ChannelEof(inChan)) {
    count = F_ReadBytes(buf, BUFSIZE-1, inChan);
    buf[count] = '\0';
    F_MifBegin("Para");
    F_MifBegin("ParaLine");
    F_MifBegin("String");
    F_MifTextString((StringT)buf);
    F_MifEnd("String");
    F_MifEnd("ParaLine");
    F_MifEnd("Para");
}

/*
 ** Now the code frees memory and sets the return value to success.
 */
F_FilePathFree(inPath);
F_ChannelClose(inChan);
F_FilePathFree(outPath);
F_ChannelClose(outChan);
F_ApiReturnValue(0);

Registering filters

For the FrameMaker product to call your client to filter files, you must register the client and
the formats it filters.

IMPORTANT: When you register a filter, the client name and the display name (the name that
appears in the Open or Save As dialog box) must be the same.

On UNIX and Windows platforms, you specify the filename extensions of the formats your
client filters in the client’s entry in the registration file. On Macintosh platforms, you specify
the file types of the formats your client filters in the client’s resource fork.

To identify your filter to the FrameMaker product, you specify a vendor ID and format ID
when you register it. The format ID is a four-character string you choose to identify the format
on all platforms. The vendor ID is a four-character string that identifies the filter vendor. The
FrameMaker product uses these IDs to identify your filter when it reimports a file imported by
reference.
File-to-file filter clients can contain more than one filter process in a single client. You must register each filter individually and give each one a unique client name, even though they are coded in the same file-to-file filter client. For example, you might have a DLL named my_filter.dll. Assume that DLL a process to convert myFormat to MIF, and another process to convert FrameMaker documents to myFormat. On Windows platforms, you would provide two entries in the FrameMaker product’s .ini file; on Unix platforms, you would provide two entries in the apiclients file; for the Macintosh, you would specify the filters individually in the client’s resource fork.

For more information on registering filter clients, see the FDK Platform Guide for your platform.

---

**Specifying format IDs and filetype hint strings**

When you register a filter, the FrameMaker product uses the information you supply to associate that filter with a specific file format. The product also uses that information to associate a filter with an imported graphic or a text inset. Internally, this information is stored in a *filetype hint*. The filetype hint includes the filter version, the vendor, and the file format this filter handles.

Assume you create a filter client that translates Himyaritic documents to English, and you assign it the format ID ‘HIM’. When you import a Himyaritic file by reference into a FrameMaker product document, the FrameMaker product creates a text inset and saves the format ID in the inset’s import hint (FP_ImportHint) property. The next time you open the document, the FrameMaker product uses the import hint to update the text inset. It looks for a client with the format ID ‘HIM’. If it finds a client with this ID, it uses that client to update the inset. Note that this mechanism works across platforms. If you move the document and text file to another platform, the hint string will still indicate the correct filter, assuming the filter is installed on the new platform.

**Specifying format IDs**

You can make up format IDs for specific file formats. However, FrameMaker reserves the following format IDs for the specified file formats.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDR</td>
<td>Corel Draw</td>
</tr>
<tr>
<td>CGM</td>
<td>Computer Graphics Metafile</td>
</tr>
<tr>
<td>CVBN</td>
<td>Corel Ventura compound document (Windows)</td>
</tr>
<tr>
<td>DCA</td>
<td>DCA to MIF (UNIX)</td>
</tr>
<tr>
<td>DIB</td>
<td>Device-independent bitmap (Windows)</td>
</tr>
<tr>
<td>DRW</td>
<td>Micrografx CAD</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>DXF</code></td>
<td>Autodesk CAD</td>
</tr>
<tr>
<td><code>EMF</code></td>
<td>Enhanced Metafile (Windows)</td>
</tr>
<tr>
<td><code>EPSB</code></td>
<td>Encapsulated PostScript Binary (Windows)</td>
</tr>
<tr>
<td><code>EPSD</code></td>
<td>Encapsulated PostScript (DCS)</td>
</tr>
<tr>
<td><code>EPSF</code></td>
<td>Encapsulated PostScript (Macintosh)</td>
</tr>
<tr>
<td><code>EPSI</code></td>
<td>Encapsulated PostScript Interchange</td>
</tr>
<tr>
<td><code>FRMV</code></td>
<td>FrameVector</td>
</tr>
<tr>
<td><code>G4IM</code></td>
<td>CCITT Group 4 to Image</td>
</tr>
<tr>
<td><code>GEM</code></td>
<td>GEM (Windows)</td>
</tr>
<tr>
<td><code>GIF</code></td>
<td>Graphic Image File Format (CompuServe)</td>
</tr>
<tr>
<td><code>HPGL</code></td>
<td>Hewlett-Packard Graphics Language</td>
</tr>
<tr>
<td><code>IAF</code></td>
<td>Interleaf compound document</td>
</tr>
<tr>
<td><code>iges</code></td>
<td>CAD format</td>
</tr>
<tr>
<td><code>IMG4</code></td>
<td>Image to CCITT Group 4 (UNIX)</td>
</tr>
<tr>
<td><code>MOOV</code></td>
<td>QuickTime Movie</td>
</tr>
<tr>
<td><code>MIAF</code></td>
<td>MIF to IAF export</td>
</tr>
<tr>
<td><code>MML</code></td>
<td>Maker Markup Language (MML)</td>
</tr>
<tr>
<td><code>MRTF</code></td>
<td>MIF to RTF export</td>
</tr>
<tr>
<td><code>MWPB</code></td>
<td>MIF to WordPerfect export</td>
</tr>
<tr>
<td><code>OLE</code></td>
<td>Object Linking and Embedding Client (Microsoft)</td>
</tr>
<tr>
<td><code>PCX</code></td>
<td>PC Paintbrush</td>
</tr>
<tr>
<td><code>PICT</code></td>
<td>QuickDraw PICT</td>
</tr>
<tr>
<td><code>PNTG</code></td>
<td>MacPaint</td>
</tr>
<tr>
<td><code>RTF</code></td>
<td>Microsoft’s RTF compound document</td>
</tr>
<tr>
<td><code>SNRF</code></td>
<td>Sun Raster File</td>
</tr>
<tr>
<td><code>TIF</code></td>
<td>Tag Image File Format</td>
</tr>
<tr>
<td><code>TREF</code></td>
<td>TROFF to MIF (UNIX only)</td>
</tr>
<tr>
<td><code>WMF</code></td>
<td>Windows Metafile</td>
</tr>
<tr>
<td><code>WDBN</code></td>
<td>MS Word compound document</td>
</tr>
</tbody>
</table>
FrameMaker does not supply filters for all of these formats on all platforms. However, you
should not use one of these format IDs unless your client filters the corresponding file format.
For the complete syntax of import hint strings, see “Syntax of FP_ImportHint strings” on
page 600 of the FDK Programmer’s Reference.

Filetype hint string syntax

The FrameMaker product uses filetype hint strings for both graphic and document or text files.
The hint strings are stored with imported graphics and with text insets. You also use hint
strings to invoke specific filters from within your API clients. For example, to save a
FrameMaker document as HTML, use the following code to specify the HTML hint string:

```c
IntT i;
F_PropValsT params;
... 
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);
params = F_ApiGetSaveDefaultParams();
i = F_ApiGetPropIndex(&params, FS_FileType)
params.val[i].propVal.u.ival = FV_SaveFmtFilter;
i = F_ApiGetPropIndex(&params, FS_SaveFileTypeHint)
params.val[i].propVal.u.sval =
    F_StrCopyString((StringT)"0001ADBEBHTML");
... 
```

The syntax of a hint string is:

```
record_vers vendor format_id platform filter_vers filter_name
```

Of these, `record_vers` `vendor` `format_id` are the minimum required to specify a filter. Each field
must specify a four-byte code, except for `filter_name` which can be up to 31 characters long. If
necessary, you must pad out each field so that it has exactly 4 Alphanumeric characters. For
example, the hint for version 1 of the XML filter must be specified as follows;

```
0001ADBEXML 00001
```

Value Description

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>WPBN</em></td>
<td>WordPerfect compound document</td>
</tr>
<tr>
<td>*WPG '</td>
<td>WordPerfect Graphics</td>
</tr>
<tr>
<td>*XWD '</td>
<td>X Windows bitmap</td>
</tr>
</tbody>
</table>

Note the space padding after the three-character code for XML.
However, the last field in a hint string does not need padding. In the above example, if you did
not specify the version number of the XML filter, XML would have been the last field, and it
would not have required padding.

`record_vers` specifies the version of the record, currently 0001.
vendor is a code specifying the filter’s vendor. The code is a string of four characters. The following table lists the possible codes.

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGRF</td>
<td>Built-in Frame filters</td>
</tr>
<tr>
<td>FAPI</td>
<td>External Frame FDK client filter</td>
</tr>
<tr>
<td>FFLT</td>
<td>External Frame filters</td>
</tr>
<tr>
<td>IMAG</td>
<td>External ImageMark filters</td>
</tr>
<tr>
<td>XTND</td>
<td>External XTND filters</td>
</tr>
</tbody>
</table>

This is not a comprehensive list of codes. Codes may be added to this list by Frame or by developers at your site.

format_id is a code specifying the format that the filter translates. The code is a string of four characters. See “Specifying format IDs” on page 247 for information about format codes and a table that lists some of the possible codes.

platform is a code specifying the platform on which the filter was run. The code is a string of four characters. The following table lists the possible codes.

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC6</td>
<td>Macintosh 68000 series</td>
</tr>
<tr>
<td>MACP</td>
<td>Power Macintosh™</td>
</tr>
<tr>
<td>WINT</td>
<td>Windows NT</td>
</tr>
<tr>
<td>WIN3</td>
<td>Windows 3.1</td>
</tr>
<tr>
<td>WIN4</td>
<td>Windows 95</td>
</tr>
<tr>
<td>OS/2</td>
<td>IBM OS/2</td>
</tr>
<tr>
<td>UNIX</td>
<td>Generic X/11 (Sun, HP)</td>
</tr>
</tbody>
</table>

filter_vers is a string of four characters identifying the version of the filter on that platform. For example, version 1.0 of a filter is represented by the string 1.0.

filter_name is a text string (up to 31 characters long) that describes the filter.

**Hint strings for the standard installation of filters**

The following tables list the minimal hint strings for all the import and export filters that ship with FrameMaker on Windows, and for importing and exporting text files. The import and export filters lists were generated against the standard installation of filters for the US English version of FrameMaker 7.0. Your installation may not include all of these filters, and you may
have other filters installed. For this reason, a code example that generates these lists follows the tables.

**Windows hint strings**
The following tables list hint strings for Windows filters:

Hint strings for import filters:

<table>
<thead>
<tr>
<th>To use this import filter:</th>
<th>Use this hint string:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corel Draw</td>
<td>0001IMAGCDR</td>
</tr>
<tr>
<td>DIB</td>
<td>0001FRAMDIB</td>
</tr>
<tr>
<td>Micrografx Drawing</td>
<td>0001IMAGDRW</td>
</tr>
<tr>
<td>AutoCAD (DWG)</td>
<td>0001IMAGDWG</td>
</tr>
<tr>
<td>AutoCAD (DXF)</td>
<td>0001IMAGDXF</td>
</tr>
<tr>
<td>Windows Enhanced Metafile</td>
<td>0001FRAMEMF</td>
</tr>
<tr>
<td>EPS/EPSF</td>
<td>0001FRAMEPSF</td>
</tr>
<tr>
<td>PDF</td>
<td>0001FRAMPDF</td>
</tr>
<tr>
<td>Frame Image</td>
<td>0001FRAMFRMI</td>
</tr>
<tr>
<td>Frame Vector</td>
<td>0001FRAMFRMV</td>
</tr>
<tr>
<td>CCITTG4</td>
<td>0001IMAGG4IM</td>
</tr>
<tr>
<td>GEM</td>
<td>0001IMAGGEM</td>
</tr>
<tr>
<td>GIF</td>
<td>0001FRAMGIF</td>
</tr>
<tr>
<td>HPGL</td>
<td>0001IMAGHPGL</td>
</tr>
<tr>
<td>IGES</td>
<td>0001IMAGIGES</td>
</tr>
<tr>
<td>JPEG</td>
<td>0001IMAGJPEG</td>
</tr>
<tr>
<td>PCX</td>
<td>0001FRAMPCX</td>
</tr>
<tr>
<td>QuickDraw PICT</td>
<td>0001IMAGPICT</td>
</tr>
<tr>
<td>Portable Network Graphics</td>
<td>0001IMAGPNG</td>
</tr>
<tr>
<td>MacPaint</td>
<td>0001FRAMPNTG</td>
</tr>
<tr>
<td>Sun Raster File</td>
<td>0001FRAMSNRF</td>
</tr>
<tr>
<td>TIFF</td>
<td>0001FRAMTIFF</td>
</tr>
<tr>
<td>Windows Metafile</td>
<td>0001FRAMWMF</td>
</tr>
<tr>
<td>Windows Metafile to FrameVector</td>
<td>0001IMAGWMF</td>
</tr>
</tbody>
</table>
To use this import filter: | Use this hint string:
---|---
WordPerfect Graphics | 0001IMAGWPG
XWD | 0001FRAMXWD
MIF | 0001FRAMMIF
MML | 0001FRAMMML
Text | 0001FRAMTEXT
Ventura Publisher | 0001FLTVCVB
Microsoft Word for Windows 6.0/7.0 | 0001AW4W0490
Microsoft Word for Windows 2.0 | 0001AW4W0441
Microsoft Word for Windows 1.0 | 0001AW4W0440
Microsoft Word Macintosh 6.0 | 0001AW4W049m
Microsoft Word DOS 5.0/6.0 | 0001AW4W0052
Microsoft Word DOS 4.0 | 0001AW4W0051
Microsoft Word DOS 3.0, 3.1 | 0001AW4W0050
Microsoft Word Mac 5.x | 0001AW4W0542
Microsoft Word Mac 4.x | 0001AW4W0541
Microsoft Word Mac 3.x | 0001AW4W0540
Microsoft RTF | 0001AW4W0191
WordPerfect DOS/Win 7.0 | 0001AW4W0482
WordPerfect DOS/Win 6.1 | 0001AW4W0481
WordPerfect DOS/Win 6.0 | 0001AW4W0480
WordPerfect DOS/Win 5.1 | 0001AW4W0071
WordPerfect DOS/Win 5.0 | 0001AW4W0070
WordPerfect Mac 3.0-3.5 | 0001AW4W0601
WordPerfect Mac 2.0-2.1 | 0001AW4W0600
WordPerfect Mac 1.0 | 0001AW4W0590
InterLeaf ASCII Format | 0001AW4W0460
DCA Revisable Form Text | 0001AW4W0150
DCA RFT (DisplayWrite 5) | 0001AW4W0151
Ami Professional 2-3.1 | 0001AW4W0331
### Hint strings for export filters:

<table>
<thead>
<tr>
<th>To use this import filter:</th>
<th>Use this hint string:</th>
</tr>
</thead>
<tbody>
<tr>
<td>FrameVector</td>
<td>0001FRAMFRMV</td>
</tr>
<tr>
<td>FrameImage</td>
<td>0001FRAMFRMI</td>
</tr>
<tr>
<td>IGES</td>
<td>0001IMAGIGES</td>
</tr>
<tr>
<td>QuickDraw PICT</td>
<td>0001IMAGPICT</td>
</tr>
<tr>
<td>EPS</td>
<td>0001IMAGEPS</td>
</tr>
<tr>
<td>TIFF</td>
<td>0001IMAGTIFF</td>
</tr>
<tr>
<td>DIB</td>
<td>0001IMAGDIB</td>
</tr>
<tr>
<td>GIF</td>
<td>0001IMAGGIF</td>
</tr>
<tr>
<td>CCITTTG4</td>
<td>0001IMAGG4IM</td>
</tr>
<tr>
<td>JPEG</td>
<td>0001IMAGJPEG</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>To use this import filter:</th>
<th>Use this hint string:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ami Professional 1</td>
<td>0001AW4W0330</td>
</tr>
<tr>
<td>Lotus 1-2-3 5.0</td>
<td>0001AW4W0204</td>
</tr>
<tr>
<td>Lotus 1-2-3 4.0</td>
<td>0001AW4W0203</td>
</tr>
<tr>
<td>Microsoft Excel 5.0</td>
<td>0001AW4W0214</td>
</tr>
<tr>
<td>Microsoft Word97/98/2000</td>
<td>0001AW4W3490</td>
</tr>
<tr>
<td>ICHITARO5</td>
<td>0001IVY JAW</td>
</tr>
<tr>
<td>ICHITARO6</td>
<td>0001IVY JBW</td>
</tr>
<tr>
<td>RTF Japanese</td>
<td>0001IVY RTF</td>
</tr>
<tr>
<td>CGM Import IsoDraw</td>
<td>0001ISO CGM</td>
</tr>
<tr>
<td>SVG</td>
<td>0001ORBLSVG</td>
</tr>
<tr>
<td>SGML</td>
<td>0001FRAMSGML</td>
</tr>
<tr>
<td>XML</td>
<td>0001FRAMXML</td>
</tr>
<tr>
<td>RTF Ver1.6</td>
<td>0001ADBIRTF</td>
</tr>
<tr>
<td>WORD97/98/2000</td>
<td>0001ADBWORD</td>
</tr>
<tr>
<td>Excel97/98/2000</td>
<td>0001ADBIXLS</td>
</tr>
</tbody>
</table>
Text import and export hint strings

The following table lists hint strings for importing and exporting text files:

<table>
<thead>
<tr>
<th>To use this import filter</th>
<th>Use this hint string:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portable Network Graphics</td>
<td>0001IMAGPNG</td>
</tr>
<tr>
<td>Windows Metafile</td>
<td>0001IMAGWMF</td>
</tr>
<tr>
<td>PDF</td>
<td>0001FRAMPDF</td>
</tr>
<tr>
<td>Microsoft RTF</td>
<td>0001AW4W0192</td>
</tr>
<tr>
<td>Microsoft Word Win 6.0/7.0</td>
<td>0001AW4W0490</td>
</tr>
<tr>
<td>Microsoft Word Mac 6.0</td>
<td>0001AW4W049m</td>
</tr>
<tr>
<td>Microsoft Word Mac 5.x</td>
<td>0001AW4W0542</td>
</tr>
<tr>
<td>Microsoft Word Mac 4.x</td>
<td>0001AW4W0541</td>
</tr>
<tr>
<td>WordPerfect DOS/Win 5.1</td>
<td>0001AW4W0071</td>
</tr>
<tr>
<td>WordPerfect Mac 3.5</td>
<td>0001AW4W0602</td>
</tr>
<tr>
<td>RTF Japanese</td>
<td>0001IVY RTFJ</td>
</tr>
<tr>
<td>CGM Export IsoDraw</td>
<td>0001ISO CGM</td>
</tr>
<tr>
<td>HTML</td>
<td>0001ADBEBHTML</td>
</tr>
<tr>
<td>XML</td>
<td>0001ADBEXML</td>
</tr>
<tr>
<td>RTF1.6</td>
<td>0001ADBIRTF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>To import or export this text:</th>
<th>Use this hint string:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain text</td>
<td>0001PGRFTEXT</td>
</tr>
<tr>
<td>Text ISO Latin 1</td>
<td>0001PGRFTXIS</td>
</tr>
<tr>
<td>Text Roman 8 (HEWLETT PACKARD UNIX)</td>
<td>0001PGRFTXRM</td>
</tr>
<tr>
<td>Text ANSI (Windows)</td>
<td>0001PGRFTANS</td>
</tr>
<tr>
<td>Text (Macintosh)</td>
<td>0001PGRFTMAC</td>
</tr>
<tr>
<td>Text ASCII</td>
<td>0001PGRFTASC</td>
</tr>
<tr>
<td>Japanese JIS</td>
<td>0001PGRFTJIS</td>
</tr>
<tr>
<td>Japanese Shift-JIS</td>
<td>0001PGRFTSJIS</td>
</tr>
<tr>
<td>Japanese EUC</td>
<td>0001PGRFTEUJ</td>
</tr>
<tr>
<td>Traditional Chinese BIG 5</td>
<td>0001PGRFTBG5</td>
</tr>
</tbody>
</table>
Using Imported Files and Insets

To generate a list of filters for a given session, you print out the list of registered import filters and the list of registered export filters. The following example gets those lists and prints their contents out to the console:

```c
VoidT printFilterStrings(VoidT)
{
    F_StringsT importFilters, exportFilters;
    UIntT i;

    F_Printf(NULL, (StringT)"\n\n ###IMPORT FILTERS###\n\n");
    importFilters = F_ApiGetStrings(0, FV_SessionId, FP_ImportFilters);
    for (i=0; i < importFilters.len; i++)
        F_Printf(NULL, (StringT)"%s\n", importFilters.val[i]);

    F_Printf(NULL, (StringT)"\n\n ###EXPORT FILTERS###\n\n");
    exportFilters = F_ApiGetStrings(0, FV_SessionId, FP_ExportFilters);
    for (i=0; i < exportFilters.len; i++)
        F_Printf(NULL, (StringT)"%s\n", exportFilters.val[i]);

    F_ApiDeallocateStrings(&importFilters);
    F_ApiDeallocateStrings(&exportFilters);
}
```

### Associations with signature bytes

Some file formats have *signature bytes*. Signature bytes are a set of bytes with a unique value and location in a particular file format. FrameMaker can use signature bytes to automatically identify a file’s format. The documentation for the file format your client converts may contain information on the signature bytes for that format.

FrameMaker allows you to associate a set of signature bytes with a specific file format. When the FrameMaker product opens a file containing the signature bytes, it assumes the file has the specified file format and calls the appropriate filter for that format. For more information on registering signature bytes, see the *FDK Platform Guide* for your platform.
Writing graphic inset editors

In addition to creating graphic insets by calling `F_ApiImport()`, you can create and maintain graphic insets with a client or application called a graphic inset editor. A graphic inset editor saves graphics in a format that FrameMaker can import. Although FrameMaker on all platforms can display graphic insets, only UNIX versions of FrameMaker support graphic inset editors.

A graphic inset maintained by a graphics inset editor doesn’t have to be a graphic like the graphics you create with a paint program. It can, for example, be a drawing created with a computer-aided drafting application or a flow chart created with a project management application. The UNIX version of FDK provides the source code for an example graphic inset editor named `fmbitmap`, which is implemented using the FrameServer interface. For more information on the location of the code, see the FDK Platform Guide for UNIX.

A graphic inset editor can be either an API client that defines an `F_ApiMessage()` callback or a FrameServer application that defines `Launch()` and `Edit()` callbacks. This chapter discusses only API graphic inset editors.

To add a graphic inset using a graphic inset editor, the user chooses Inset from the Special menu and selects an editor from the Inset Editors scroll list. The FrameMaker product starts the editor, and the user then creates the inset with the editor. The editor uses API functions to create an anchored frame, add an `FO_Inset` object to it, and set the `FO_Inset` properties to describe the inset.

FrameMaker maintains a connection, or live link, between a graphic inset and the graphic inset editor that created it. To edit a graphic inset, the user can double-click it. The FrameMaker product starts the appropriate editor and instructs it to edit the inset file.

An API graphic inset editor can update and create insets in response to other events, such as the user opening a document or choosing a menu item. It can request and respond to notifications like any other API client.

Graphic inset properties

The API represents each graphic inset with an `FO_Inset` object. An `FO_Inset` object has the properties common to all graphic objects. It also has some properties that are specific to graphic insets. The following table lists some of these properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_InsetDpi</td>
<td>IntT</td>
<td>Dots per inch (DPI). Indicates scale factor. It only applies to autosizing raster images.</td>
</tr>
<tr>
<td>FP_InsetEditor</td>
<td>StringT</td>
<td>Name of client to call to edit the inset (on UNIX platforms only).</td>
</tr>
</tbody>
</table>
In addition to the properties listed in the table above, each `FO_Inset` object has special properties called *facets*, which contain data describing the imported graphic.

Each `FO_Inset` object must have at least one of the standard graphic inset facets listed in the following table.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_InsetFile</td>
<td>StringT</td>
<td>Platform-specific pathname if the inset is an external inset, or a null string (&quot;&quot;&quot;) if it is internal. The pathname can be document-relative.</td>
</tr>
<tr>
<td>FP_InsetIsFixedSize</td>
<td>IntT</td>
<td>True if <code>FP_Width</code> and <code>FP_Height</code> are used for the graphic’s size. False if autosize is used.</td>
</tr>
<tr>
<td>FP_InsetIsFlippedSideways</td>
<td>IntT</td>
<td>True if inset is flipped sideways.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Platform on which facet format can be used</th>
<th>Facet format</th>
</tr>
</thead>
<tbody>
<tr>
<td>All platforms</td>
<td>DCS Black</td>
</tr>
<tr>
<td></td>
<td>DCS Cyan</td>
</tr>
<tr>
<td></td>
<td>DCS Magenta</td>
</tr>
<tr>
<td></td>
<td>DCS Yellow</td>
</tr>
<tr>
<td></td>
<td>CGM</td>
</tr>
<tr>
<td></td>
<td>EPSI (Encapsulated PostScript)</td>
</tr>
<tr>
<td></td>
<td>FrameImage</td>
</tr>
<tr>
<td></td>
<td>FrameVector</td>
</tr>
<tr>
<td></td>
<td>GIF</td>
</tr>
<tr>
<td></td>
<td>MacPaint</td>
</tr>
<tr>
<td></td>
<td>PCX</td>
</tr>
<tr>
<td></td>
<td>SVG</td>
</tr>
<tr>
<td></td>
<td>TIFF</td>
</tr>
<tr>
<td></td>
<td>XWD</td>
</tr>
<tr>
<td>Macintosh only</td>
<td>QuickDraw PICT</td>
</tr>
<tr>
<td></td>
<td>QuickTime</td>
</tr>
</tbody>
</table>
FrameImage facets follow Sun raster image format. FrameVector facets follow a TIFF format. For a complete description of these formats, see the online MIF Reference manual.

If an inset doesn’t have one of the facet formats listed above, FrameMaker can use a filter to convert another format into FrameImage or FrameVector formats. For example, if the inset has an HPGL facet, the FrameMaker product can convert it into FrameVector format.

FrameMaker uses an inset’s facets to display and print it. FrameMaker may use different facets to display and print a graphic. For example, the Macintosh version of FrameMaker may use the QuickDraw PICT facet for displaying a graphic and the EPSI facet for printing it.

When displaying an imported graphic, FrameMaker uses one of the following facet formats (in order of preference):

- Native platform format (QuickDraw, PICT, WMF)
- FrameVector
- FrameImage and other bitmap formats
- TIFF
- Other bitmap formats

When printing an imported graphic, FrameMaker uses one of the following facet formats (in order of preference):

- EPSI (Encapsulated PostScript)
- Native platform format (QuickDraw, PICT, WMF)
- FrameVector
- TIFF
- FrameImage and other bitmap formats

Unlike other properties, which are identified by numbers, facets are identified by names. Instead of using propIdent.num to identify a facet, the API sets propIdent.num to 0 and sets propIdent.name to the facet name. For more information on how the API represents property and property lists, see “Representing object characteristics with properties” on page 270.

In addition to the facets listed above, each FO_Inset object can have several client-specific facets. Client-specific facets contain information that your client uses. A client-specific facet can contain a complete set of data, or just a pathname to an external data file or database. You must register the names of client-specific facets with the FrameMaker product. For

<table>
<thead>
<tr>
<th>Platform on which facet format can be used</th>
<th>Facet format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows only</td>
<td>DIB</td>
</tr>
<tr>
<td></td>
<td>EMF</td>
</tr>
<tr>
<td></td>
<td>OLE</td>
</tr>
<tr>
<td></td>
<td>WMF (Windows Metafile)</td>
</tr>
</tbody>
</table>
information on registering FDK clients and facets for inset editors, see the *FDK Platform Guide* for UNIX. A graphic inset can have as many client-specific facets as you want.

Facets can specify integer (IntT), metric (MetricT), or unsigned bytes (F_UBytesT) data. Facets, such as EPSI and FrameImage, that specify multiple characters or binary data are F_UBytesT facets. F_UBytesT is defined as:

```c
typedef struct {
    UIntT len; /* The number of unsigned bytes */
    UByteT *val; /* The facet data */
} F_UBytesT;
```

### Internal and external graphic insets

There are two types of graphic insets: internal and external. You choose which type your client supports. Both types require a display and print facet and can have one or more client-specific facets.

#### Internal graphic insets

Internal graphic insets are wholly contained within a FrameMaker product document. If a graphic inset is internal, the FO_Inset object's FP_InsetFile property is set to a null string ("""). Internal graphic insets are generally more portable than external graphic insets.

To update an internal graphic inset, the user starts the FrameMaker product and initiates an event that the graphic inset editor monitors. For example, if the user double-clicks the inset, the graphic inset editor updates the inset by setting its facets.

The following EPSI facet data describes the rectangle in Figure 12.1.

![Figure 12.1 A graphic inset](image-url)
External graphic insets

External graphic insets are stored in an external file. The `FP_InsetFile` property of an external graphic inset is set to a platform-specific filename that specifies the file. For information on converting platform-specific pathnames to platform-independent (device-independent) pathnames, see Chapter 17, “Making I/O and Memory Calls Portable”

Users can update external graphic insets the same way they edit internal graphic insets. Because external graphic inset data is not contained within the FrameMaker product document, users can also edit the inset with other applications besides a graphic inset editor.

To edit an external graphic inset from FrameMaker, the user opens the document and double-clicks the inset. The FrameMaker product launches the graphic inset editor, which updates the inset by editing the inset file.
Example

Suppose the user creates a bitmap inset with a graphic inset editor named `myeditor`. The resulting `FO_Inset` object’s property list and some of its properties are shown in 261.

```
F_PropValsT
len: 12
val:

propIdent | propIdent | propIdent
num: FP_InsetEditor | num: FP_FrameParent | num: 0
name: NULL | name: NULL | name: FrameImage

propVal | propVal | propVal
valType: FT_String | valType: FT_Id | valType: FT_UBytes
u | u | u
sval: myeditor | ival: ID of frame containing the inset | NULL
```

Figure 12.2 The property list for a bitmap FO_Inset object

The `u` union for the FrameImage facet is `NULL`. The `F_PropValT` structure for `F_UBytesT` facets does not provide the facet data. It only serves as a placeholder, indicating that a facet exists. To get and set the data for `F_UBytesT` facets, you must follow special procedures, which are described in “Getting and setting facets” on page 263.

Setting up your client to create and modify graphic insets

To set up your client to create and modify graphic insets, do the following:

1. Register your client as an API client.

   You must register your client and all the facets it uses. For information on registering API clients, see the FDK Platform Guide for your platform.

2. Add an `F_ApiMessage()` callback to your client.

   When the user clicks your client in the Inset Editors scroll list or double-clicks a graphic inset, the FrameMaker product calls `F_ApiMessage()` from your client. Define `F_ApiMessage()` as follows:

   ```c
   VoidT F_ApiMessage(StringT message, /* Not used for insets */
   F_ObjHandleT docId, /* ID of document containing the inset */
   F_ObjHandleT objId); /* The ID of the inset or 0 */
   ```
Using Imported Files and Insets

Responding to the user launching your inset editor

When the user launches your graphic inset editor by choosing it from the Inset Editors scroll list, the FrameMaker product calls the editor’s F_ApiMessage() callback with objId set to 0. You can include code in F_ApiMessage() that allows the user to create a graphic inset and add it to the Frame document when your editor is launched.

To insert a graphic inset, you add an FO_AFrame object and resize it to accommodate the inset. Then you add an FO_Inset object to the frame and set its properties as needed.

The following code adds an external graphic inset (described in the file myinset.fi) when the user chooses the editor (myeditor) from the Inset Editors scroll list:

```c
... VoidT F_ApiMessage(message, docId, objId)
    StringT message;
    F_ObjHandleT docId;
    F_ObjHandleT objId; /* 0 if launched from scroll list. */
    {
        F_ObjHandleT afrmId, insetId;
        F_TextRangeT tr;
        if (!objId) {
            /* Code that allows user to create a graphic goes here. */

            /* Get the text selection and add frame at the beginning. */
            tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
            afrmId = F_ApiNewAnchoredObject(docId, FO_AFrame, &tr.beg);

            /* Code to resize and position frame goes here. */

            /* Add the inset to the frame and set its properties. */
            insetId = F_ApiNewGraphicObject(docId, FO_Inset, afrmId);
            F_ApiSetString(docId, insetId, FP_InsetFile, "/tmp/myinset.fi");
            F_ApiSetString(docId, insetId, FP_InsetEditor, "myeditor");
        }
    }
...```

Responding to the user double-clicking a graphic inset

When the user double-clicks a graphic inset for which the FP_InsetEditor property specifies your graphic inset editor, the FrameMaker product calls the editor’s F_ApiMessage() callback with objId set to the inset’s ID. Include code in F_ApiMessage() that allows the user to update the inset. If the inset is an internal inset, you update it by setting its facets.

Although your API graphic inset editor should respond when the user selects it from the Inset Editors scroll list or when the user clicks a graphic inset, it can create or update insets at any time. For example, you may want to update a document’s insets whenever the user opens the document or chooses a particular menu item.
If a graphic inset is an external graphic inset, you update it by editing the file specified by FP_InsetFile. The FrameMaker product periodically refreshes external insets against the files that describe them. If you want to ensure that the inset is refreshed immediately, you must reset the FO_Inset object’s FP_InsetFile, FP_Height, or FP_Width property.

FrameMaker product doesn’t call F_ApiMessage() only when the user double-clicks a graphic inset. It also calls F_ApiMessage() when the user clicks a hypertext marker. Therefore, make sure that the object specified by objId is a graphic inset and not a hypertext marker.

For example, the following code updates a graphic inset when the user clicks it:

```c
VoidT F_ApiMessage(message, docId, objId)
StringT message;
F_ObjHandleT docId;
F_ObjHandleT objId;
{ 
    if (!objId) { /* Code to create a new inset goes here. */
    }
    else{ /* Make sure clicked object is a graphic inset. */
        if (F_ApiGetObjectType(docId, objId) == FO_Inset){ /* Code to edit or update facets with */
            * F_ApiGet[PropertyType]ByName() and
            * F_ApiSet[PropertyType]ByName() goes here.
            */
        }
    }
}
```

### Getting and setting graphic inset properties

To get and set individual graphic inset properties identified by property numbers, you use F_ApiGetPropertyType() and F_ApiSetPropertyType() functions just as you would with any other object properties. For example, to get and set FP_InsetDpi, you use F_ApiGetInt() and F_ApiSetInt().

To get and set facets and FO_Inset property lists, follow the procedures discussed in the following sections.

### Getting and setting facets

To get and set facets, use the following functions.

<table>
<thead>
<tr>
<th>To</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query an integer facet</td>
<td>F_ApiGetIntByName()</td>
</tr>
<tr>
<td>Query a metric facet</td>
<td>F_ApiGetMetricByName()</td>
</tr>
</tbody>
</table>
### Syntax

The syntax for these functions is similar to other `F_ApiGetPropertyType()` and `F_ApiSetPropertyType()` functions, except that you must identify the property with a character string instead of an integer. For example, the syntax for `F_ApiGetUBytesByName()` is:

```c
F_UBytes *F_ApiGetUBytesByName(F_ObjHandleT docId,
                               F_ObjHandleT objId,
                               StringT *propName);
```

### Arguments and Means

<table>
<thead>
<tr>
<th>Argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document containing the inset</td>
</tr>
<tr>
<td>objId</td>
<td>The ID of the inset whose facet you want to query</td>
</tr>
<tr>
<td>propName</td>
<td>The name of the facet to query</td>
</tr>
</tbody>
</table>

For the exact syntax of the functions that get and set facets, look up the functions in Chapter 2, “FDK Function Reference,” in the *FDK Programmer’s Reference*.

Getting and setting facets takes an additional step not needed with other properties—committing the transaction. After executing a series of gets or sets for a graphic inset’s facets, commit the transaction by getting or setting a facet named "".

For example, the following code gets two facets and commits the transaction:

```c
... IntT myInt, err; MetricT myMetric; F_ObjHandleT docId, insetId;

myInt = F_ApiGetIntByName(docId, insetId, "myinteger.facet");
myMetric = F_ApiGetMetricByName(docId, insetId, "mymetric.facet");
err = F_ApiGetIntByName(docId, insetId, "");
...
```

If you are setting facets, you commit the transaction by setting a facet named "". For example:

```c
... #define in (MetricT) (72 * 65536)```

---

**To** | **Use**
---|---
Query an _UBytes facet | `F_ApiGetUBytesByName()`
Set an integer facet | `F_ApiSetIntByName()`
Set a metric facet | `F_ApiSetMetricByName()`
Set an _UBytes facet | `F_ApiSetUBytesByName()`
Using Imported Files and Insets

Getting an F_UBytesT facet

Because an F_UBytesT facet can contain large amounts of data, it is not feasible to return all the data in a single array. Calling F_ApiGetUBytesByName() returns only the next chunk of a facet’s data. To get all the data for an F_UBytesT facet, you must call F_ApiGetUBytesByName() repeatedly until F_UBytesT.len is 0. For example, the following code gets all the bytes in a facet named my.facet:

```c
F_ObjHandleT docId, insetId;
F_UBytesT aUBytes;

do {
    aUBytes = F_ApiGetUBytesByName(docId,insetId,"my.facet");
    /* Code to do something with aUBytes goes here. */

} while (aUBytes.len)
```

F_ApiGetUBytesByName(docId,insetId,""); /* Commit transaction.*/

Setting an F_UBytesT facet

To set an F_UBytesT facet that contains a relatively small chunk of data, you call F_ApiSetUBytesByName() once. To set an F_UBytesT facet that contains a large chunk of data, you must call F_ApiSetUBytesByName() multiple times, passing a small chunk of data each time. The size of the data chunk you pass depends on the platform you are using. For example, on UNIX platforms, you should use about 10K. In general, if you use a larger size chunk, you can set the facet more quickly. However, if you use too large a size, you risk exceeding the interapplication communication mechanism’s capacity.

For example, to set the EPSI facet of a graphic inset to the contents of a file named mydata, use the following code:

```c
F_ObjHandleT docId, insetId;
F_ApiSetMetricByName(docId, insetId, "mymetric.facet", 2*in);
F_ApiSetIntByName(docId, insetId, ",", 0); /* Commit */
...
Part II

Frame Product Architecture
13 Frame Session Architecture

This chapter discusses the general organization of FrameMaker product sessions from a programmer’s perspective. It provides useful background information for programmers who want to use the Frame API.

Identifying objects

The API assigns a unique ID to each object. Most API functions that manipulate objects require you to specify this ID. An object’s ID is valid only as long as the object is available in the current FrameMaker product session. For example, suppose you have a document with a rectangle drawn in it. When you open the document, the API assigns an ID to the FO_Rectangle object that represents the rectangle. As long as the document is open, the ID of the FO_Rectangle object remains the same. However, if you exit the document and then reopen it, the API may assign a new ID to the rectangle.

In addition to IDs, there are two types of identifiers that are persistent between sessions:

- Unique object names
- Unique persistent identifiers (UIDs)

Each object generally has either a FP_Name property specifying a unique object name or an FP_Unique property specifying a UID.

Unique object names

There are many types of objects that you can assign unique names to in the user interface. These objects, which are called named objects, include:

- FO_Book
- FO_CharFmt
- FO_Color
- FO_CombinedFontDefn
- FO_Command
- FO_CondFmt
- FO_ElementDef
- FO_FmtChangeList
- FO_MarkerType
- FO_MasterPage
The API provides a function named F_ApiGetNamedObject(), which gets the ID of a named object with a specified name.

**IMPORTANT:** A document can contain several flows with the same name. For example, a document can contain several A flows. To get the ID of a specific flow, first get the ID of a text frame in that flow (for example, the text frame for the current text location), and then query the text frame’s FP_Flow property.

**Unique persistent identifiers (UIDs)**

The API and MIF identify unnamed objects with UIDs. An unnamed object is an object that doesn’t have a unique name. For example, FO_Pgf objects are unnamed.

UIDs are unique within documents. An object’s UID remains the same as long as the object is in the same document. The API provides a function named F_ApiGetUniqueObject(), which gets an object’s ID from its UID.

**IMPORTANT:** If you copy an object and then paste it, the FrameMaker product considers the pasted object a new object and assigns a new UID to it. This is also true for a paragraph that is conditionalized. If the entire paragraph is of a given condition, and that condition is hidden and then shown, the paragraph will have a new UID.

**Representing object characteristics with properties**

Each object has a property list, or set of properties that represent its characteristics. Each property has a value associated with it. For example, if a paragraph has two tabs, the value of its FP_NumTabs property is 2. A property value can be more than an integer. It can also be a string, a pointer to a structure that contains a set of strings, or a variety of other things. The following table summarizes the different data types property values can be.
The API uses MetricT data to express measurement values. This manual uses constants to represent conventional measurement system units as MetricT data. For example, the constant in represents an inch and the constant pts represents a point in MetricT units. For example, five inches (5*4718592) are represented as 5*in.

For more information on the MetricT type and other data types and data structures listed in the table above, see Chapter 4, “Data Types and Structures Reference” in the FDK Programmer’s Reference.

Many property values are pointers to data structures. For example, FO_Doc objects have a property named FP_Dictionary that specifies words that the Spelling Checker will permit in a document. FP_Dictionary is an FT_Strings property. Its value is a pointer to an F_StringsT structure, which is defined as:

typedef struct {
    UIntT len; /* Number of permitted words */
    StringT *val; /* Vector of permitted words */
} F_StringsT;
Property lists

At the highest level, the API represents each object’s property list with a F_PropValsT structure. The F_PropValsT structure is defined as:

typedef struct {
    UIntT len; /* Number of properties in list */
    F_PropValT *val; /* Property-value pairs */
} F_PropValsT;

The F_PropValT structure, which provides an individual property-value pair, is defined as:

typedef struct {
    F_PropIdentT propIdent; /* The property identifier */
    F_TypedValT propVal; /* The property value */
} F_PropValT;

The F_PropIdentT structure, which identifies a property by either its property number constant (one of the constants beginning with FP_) or a property name, is defined as:

typedef struct {
    IntT num; /* The property number constant */
    StringT name; /* The property name */
} F_PropIdentT;

Most properties are identified by property number constants. Only inset facets, a special type of properties, are identified by names. For information on insets, see Chapter 12, “Using Imported Files and Insets” If a property is identified by a name, F_PropIdentT.num is 0.

The F_TypedValT structure is defined as:

typedef struct {
    IntT valType; /* The type of value. See table below */
    union {
        StringT sval; /* String value */
        F_StringsT ssval; /* Set of strings */
        F_MetricsT msval; /* Set of metrics */
        F_PointsT psval; /* Set of points */
        F_TabsT tsval; /* Set of tabs */
        F_TextLocT tlval; /* Text location */
        F_TextRangeT trval; /* Text range */
        F_ElementCatalogEntriesT csval; /* Element Catalog */
        F_AttributeDefsT adsv; /* Attribute definitions */
        F_AttributeDefsT adsv; /* Attribute values */
        F_ElementRangeT erng; /* Element range */
        F_IntsT isval; /* Set of integers */
        F_UIntsT uisval; /* Set of unsigned integers */
        IntT ival; /* Integer */
    } u;
} F_TypedValT;
The constants used in the `valType` field are described in the following table.

<table>
<thead>
<tr>
<th><code>valType constant</code></th>
<th>Property data type</th>
<th><code>u</code> field</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT_Integer</td>
<td>IntT</td>
<td>ival</td>
</tr>
<tr>
<td>FT_Ints</td>
<td>F_IntsT</td>
<td>isval</td>
</tr>
<tr>
<td>FT_Metric</td>
<td>MetricT</td>
<td>ival</td>
</tr>
<tr>
<td>FT_Metrics</td>
<td>F_MetricsT</td>
<td>msval</td>
</tr>
<tr>
<td>FT_String</td>
<td>StringT</td>
<td>sval</td>
</tr>
<tr>
<td>FT_Strings</td>
<td>F_StringsT</td>
<td>ssval</td>
</tr>
<tr>
<td>FT_Id</td>
<td>F_ObjHandleT</td>
<td>ival</td>
</tr>
<tr>
<td>FT_Points</td>
<td>F_PointsT</td>
<td>psval</td>
</tr>
<tr>
<td>FT_Tabs</td>
<td>F_TabsT</td>
<td>tsval</td>
</tr>
<tr>
<td>FT_TextLoc</td>
<td>F_TextLocT</td>
<td>tlval</td>
</tr>
<tr>
<td>FT_TextRange</td>
<td>F_TextRangeT</td>
<td>trval</td>
</tr>
<tr>
<td>FT_UIInts</td>
<td>F_UIIntsT</td>
<td>uisval</td>
</tr>
<tr>
<td>FT_UBytes</td>
<td>F_UBytesT</td>
<td>No field</td>
</tr>
<tr>
<td>FT_ElementCatalog</td>
<td>F_ElementCatalogEntriesT</td>
<td>csval</td>
</tr>
<tr>
<td>FT_AttributeDefs</td>
<td>F_AttributeDefsT</td>
<td>adsva</td>
</tr>
<tr>
<td>FT_Attributes</td>
<td>F_AttributesT</td>
<td>asval</td>
</tr>
<tr>
<td>FT_ElementRange</td>
<td>F_ElementRangeT</td>
<td>erng</td>
</tr>
</tbody>
</table>

**IMPORTANT:** Integer (IntT), metric (MetricT), and ID (F_ObjHandleT) values are all put in the `ival` field of the `u` union.

**Example**

Suppose the user creates a paragraph format named Heading, which has a 1-inch left indent and has Keep With Next Paragraph turned on. The API represents this paragraph format with an `FO_PgfFmt` object. The following are some of the object’s properties and their values.

<table>
<thead>
<tr>
<th>Property</th>
<th>Data type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_Name</td>
<td>StringT</td>
<td>Heading</td>
</tr>
<tr>
<td>FP_KeepWithNext</td>
<td>IntT</td>
<td>True</td>
</tr>
<tr>
<td>FP_LeftIndent</td>
<td>MetricT</td>
<td>1*in</td>
</tr>
</tbody>
</table>

The property list for the `FO_PgfFmt` object and the properties in the previous table are represented graphically in Figure 13.1. `FO_PgfFmt` objects have many other properties that are not shown in the illustration.
The fundamental entity in Frame architecture is a session. Each instance of a FrameMaker product that the user starts is a session. FrameMaker allows the user to have many open documents and books in a session.

Of the open documents and books in a session, only one document or book is active at a time. An open document or book is active if it has the input focus.

How the API represents sessions

The API represents each FrameMaker product session with an FO_Session object, whose properties provide the following categories of information about the session:

- System information, such as the operating system, the current FrameMaker product version, and the current directory
- The automatic save settings
- Names of fonts available on the system
- IDs of the objects that represent open and active documents and books
- Whether the FrameMaker product reformats and redisplay documents after changes have been made
Whether element reformatting and validation is turned on (for FrameMaker structured documents)

Suppose you start FrameMaker on an X Window System platform and open a document named mydoc. The API represents this session with an FO_Session object. The following are some of its properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_ProductName</td>
<td>FT_String</td>
<td>FrameMaker</td>
</tr>
<tr>
<td>FP_VersionMajor</td>
<td>FT_Integer</td>
<td>5</td>
</tr>
<tr>
<td>FP_WindowSystem</td>
<td>FT_String</td>
<td>X Windows</td>
</tr>
<tr>
<td>FP_AutoSaveSeconds</td>
<td>FT_Integer</td>
<td>300</td>
</tr>
<tr>
<td>FP_ActiveDoc</td>
<td>FT_Id</td>
<td>ID of the object that represents mydoc</td>
</tr>
</tbody>
</table>

How the API indicates which documents and books are open

The API represents a document with an FO_Doc object. The API maintains a linked list of the FO_Doc objects that represent a session’s open documents. The FO_Session property, FP_FirstOpenDoc, specifies the ID of the first FO_Doc object in the list. The FO_Doc property, FP_NextOpenDocInSession, specifies the ID of the next FO_Doc object in the list. The list of FO_Doc objects that represent open documents is not in any particular order. The FO_Doc object specified by FP_FirstOpenDoc does not necessarily represent the first document the user opened.

The API represents a book with an FO_Book object. The API also maintains the FO_Book objects that represent the session’s open books in a linked list. The FO_Session property, FP_FirstOpenBook, specifies the ID of the first FO_Book object in the list. The FO_Book property, FP_NextOpenBookInSession, specifies the next FO_Book object in the list. As with the list of FO_Doc objects, the linked list of FO_Book objects is not in any particular order.

How the API indicates which document or book is active

FO_Session has two properties, FP_ActiveDoc and FP_ActiveBook, that specify the IDs of the objects that represent the active document and the active book. Only one document or one book can be active at a time. If there is no active document or book, these properties are set to 0. Invisible documents and books can’t be active.

Example

Suppose you start FrameMaker and open the books and documents shown in Figure 13.2. The Frame API represents the session with the objects shown in Figure 13.3.
Although `Manual.book` is iconified, the API still considers it open. Although `FP_FirstOpenDoc` specifies 2Chapter, it is not necessarily the first document that was opened.

**How the API indicates which fonts are available in a session**

The following `FO_Session` properties specify which fonts are available in the current session:

- `FP_FontFamilyNames` specifies the available families, for example, Helvetica and Times.
- `FP_FontVariationNames` specifies the available variations, for example, Narrow and Oblique.
● **FP_FontWeightNames** specifies the available weights, for example, **Bold** and **Regular**.

● **FP_FontAngleNames** specifies the available angles, for example, **Italic** and **Regular**.

The **FP_FontFamilyNames**, **FP_FontVariationNames**, **FP_FontWeightNames**, and **FP_FontAngleNames** properties determine which choices appear in the Family, Weight, Angle, and Variation fields of the Character Designer and Paragraph Designer, and the pull-right menu items in the Format menu. Each of the properties specifies an **F_StringsT** structure, which is defined as:

```c
typedef struct {
    UIntT len;    /* Number of strings */
    StringT *val; /* Font names */
} F_StringsT;
```

For example, if Courier, Helvetica, and Times are the only font families available in the current session, the fields of the **F_StringsT** structure specified by **FP_FontFamilyNames** have the following values:

```plaintext
len: 4
val: {"<Reserved>" , "Courier" , "Helvetica" , "Times" }
```

Properties that specify font families, angles, weights, and variations use the index of the val array. For example, the **FO_CharFmt** property, **FP_FontFamily**, specifies the font family for a character format. Given the **F_StringsT** values shown above, if the font family for a character format is Helvetica, the value of the format’s **FP_FontFamily** property is 2.

Although a specific angle, weight, or variation may be in one of the lists described above, it may not be available for all combinations of font families, angles, weights, and variations. For example, the Bold weight may be available for Times and Helvetica, but not for Zapf Chancery. The FDK provides a convenience function named **F_ApiFamilyFonts()**, which returns all the permutations of font families, angles, weights, and variations in a FrameMaker product session. For more information, see “**F_ApiFamilyFonts()**” on page 99 of the **FDK Programmer’s Reference**.
This chapter describes Frame documents and their components and shows how the Frame API represents them.

Documents

A document is a set of pages with graphic objects and text that the user creates with FrameMaker and stores in a file.

What the user sees

When you create a new document, you can use a template to create it, or you can create a custom document. Any document can be a template. Because the FrameMaker product copies everything from a template to a new document, most users prefer to use templates containing only layout and formatting information. FrameMaker provides ready-made templates for a variety of document types.

You can’t create a document completely from scratch—the document must have a certain set of default objects for the FrameMaker product to work correctly. To ensure that all documents have this set of objects, the FrameMaker product always uses a template to create a new document. Even if you choose the Custom document option, the FrameMaker product creates the new document from a default template. On UNIX platforms, this template is named NewTemplate, and is located in $FMHOME/fmint/UILanguage/CustomDocs (where UILanguage is the language in use). On the Macintosh, the template is part of the data fork of the FrameMaker product application. If there is a template located in the appropriate folder, the FrameMaker product uses it instead. On Windows platforms, the custom document template is specified in the maker.ini file. For more information on custom documents on a specific platform, see your user documentation.

On UNIX platforms, when the user attempts to create a new document from an ASCII text file or a MIF file that doesn’t provide the necessary objects, the FrameMaker product looks for a template named xxxTemplate, where xxx is the suffix of the file being opened. First it looks in the directory from which it was started, then it looks in the user’s home directory. If the FrameMaker product can’t find xxxTemplate in either of these locations, it looks for a template named ASCIItemplate in the same locations. If it can’t find ASCIItemplate, it uses NewTemplate. On the Macintosh, the FrameMaker product looks for an ASCII template. If it can’t find one, it uses the template that is included in its data fork. On Windows, the FrameMaker product uses the ASCII template specified in the maker.ini file.

When you instruct FrameMaker to save a document, it lists the document’s objects and their properties in a file. By default, the FrameMaker product writes the information to a Frame binary format file. You can also choose to save a document as a MIF file.
## How the API represents documents

A document actually consists of much more than text and graphic objects. It includes information specifying a variety of other things, such as formatting, user preferences, and the FrameMaker product’s default behavior. The API represents the information in a document with a set of objects. The following table summarizes the information a document can contain and the objects the API uses to represent it.

<table>
<thead>
<tr>
<th>Type of information</th>
<th>Function</th>
<th>Types of objects that represent it</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global document information</td>
<td>Specifies the document’s general characteristics, some aspects of the</td>
<td>FO_Doc</td>
</tr>
<tr>
<td></td>
<td>FrameMaker product’s behavior when the document has input focus, and IDs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>of other objects that constitute the document</td>
<td></td>
</tr>
<tr>
<td>Pages</td>
<td>Organize text and graphic objects in the document</td>
<td>FO_BodyPage, FO_MasterPage, FO_RefPage, FO_HiddenPage</td>
</tr>
<tr>
<td>Graphic objects</td>
<td>Describe graphic objects in the document</td>
<td>FO_UnanchoredFrame, FO_AFrame, FO_Group, FO_Arc, FO_Rectangle, FO_Ellipse, FO_RoundRect, FO_Polyline, FO_Polygon, FO_Line, FO_TextLine, FO_TextFrame, FO_Inset, FO_Math</td>
</tr>
<tr>
<td>Text columns</td>
<td>Contain text</td>
<td>FO_SubCol</td>
</tr>
<tr>
<td>Text frames</td>
<td>Contain text</td>
<td>FO_TextFrame</td>
</tr>
<tr>
<td>Text flows</td>
<td>Specify how text frames in the document are linked</td>
<td>FO_Flow</td>
</tr>
<tr>
<td>Paragraph Catalog formats</td>
<td>Specify tags that the user can apply to a paragraph to change its</td>
<td>FO_PgfFmt</td>
</tr>
<tr>
<td></td>
<td>formatting</td>
<td></td>
</tr>
<tr>
<td>Paragraphs</td>
<td>Contain the document’s text and provide formatting information for</td>
<td>FO_Pgf</td>
</tr>
<tr>
<td></td>
<td>individual paragraphs</td>
<td></td>
</tr>
<tr>
<td>Character Catalog formats</td>
<td>Specify tags that the user can apply to a selection of characters to</td>
<td>FO_CharFmt</td>
</tr>
<tr>
<td></td>
<td>change its formatting</td>
<td></td>
</tr>
<tr>
<td>Type of information</td>
<td>Function</td>
<td>Types of objects that represent it</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>Condition formats</td>
<td>Specify tags that the user can apply to text to indicate that it belongs to a particular variation of the document</td>
<td>FO_ConfFmt</td>
</tr>
<tr>
<td>Markers</td>
<td>Describe placeholders that contain hidden text</td>
<td>FO_Marker</td>
</tr>
<tr>
<td>Marker types</td>
<td>Specifies a named category of markers</td>
<td>FO_MarkerType</td>
</tr>
<tr>
<td>Cross-reference formats</td>
<td>Specify the wording and typographic style of cross-references</td>
<td>FO_XRefFmt</td>
</tr>
<tr>
<td>Cross-reference instances</td>
<td>Describe instances of cross-references in the document</td>
<td>FO_XRef</td>
</tr>
<tr>
<td>Variable formats</td>
<td>Specify units of text and system-supplied information that the user can use multiple times in a document</td>
<td>FO_VarFmt</td>
</tr>
<tr>
<td>Variable instances</td>
<td>Describe instances of variables in the document</td>
<td>FO_Var</td>
</tr>
<tr>
<td>Footnotes</td>
<td>Describe footnotes</td>
<td>FO_Fn</td>
</tr>
<tr>
<td>Table ruling formats</td>
<td>Specify rulings and shadings that the user can apply to individual table cells</td>
<td>FO_RulingFmt</td>
</tr>
<tr>
<td>Table Catalog formats</td>
<td>Specify table formats that the user can apply to a table and that provide default numbers of columns and rows for new tables</td>
<td>FO_TblFmt</td>
</tr>
<tr>
<td>Tables</td>
<td>Describe instances of tables in the document and specify formatting information, such as alignment, ruling, and shading</td>
<td>FO_Tbl FO_Row FO_Cell</td>
</tr>
<tr>
<td>Colors</td>
<td>Specify colors that the user can apply to graphic objects and text</td>
<td>FO_Color</td>
</tr>
<tr>
<td>Text insets</td>
<td>Describe text that is imported by reference</td>
<td>FO_TiApiClient FO_TiFlow FO_TiText FO_TiTextTable</td>
</tr>
<tr>
<td>Structural element definitions</td>
<td>Specify tags that specify the organization of parts of a FrameMaker structured document</td>
<td>FO_ElementDef</td>
</tr>
<tr>
<td>Structural element instances</td>
<td>Describe instances of structural elements in a structured document</td>
<td>FO_Element</td>
</tr>
<tr>
<td>Format rules</td>
<td>Specify sets of format rule clauses</td>
<td>FO_FmtRule</td>
</tr>
</tbody>
</table>
### Frame Document Architecture

The other sections of this chapter discuss the different types of information in a document.

**How the API organizes the objects that constitute a document**

The API uses an **FO_Doc** object to organize the objects that constitute a document. **FO_Doc** objects have a number of properties that specify the IDs of other objects in the document. Many of these properties specify the ID of the first object in a linked list of objects. For example, **FP_FirstPgfFmtInDoc** specifies the first **FO_PgfFmt** object (Paragraph Catalog format) in the list of **FO_PgfFmt** objects in the document. Each **FO_PgfFmt** object has a **FP_NextPgfFmtInDoc** property that specifies the next **FO_PgfFmt** object in the list. If you want to get all the **FO_PgfFmt** objects in a document, you get the **FO_PgfFmt** object specified by **FP_FirstPgfFmtInDoc** and traverse the links to the other objects.

<table>
<thead>
<tr>
<th>Type of information</th>
<th>Function</th>
<th>Types of objects that represent it</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format rule clauses</td>
<td>Specify which formats to apply to elements in various contexts</td>
<td>FO_FmtRuleClause</td>
</tr>
<tr>
<td>Format change list</td>
<td>Specify format changes applied to an element in a specific context</td>
<td>FO_FmtChangeList</td>
</tr>
<tr>
<td>Rubi composites</td>
<td>Describe the oyamoji (base word) and rubi (phonetic spelling) of certain words in Asian text</td>
<td>FO_Rubi</td>
</tr>
<tr>
<td>Combined font definitions</td>
<td>Describe pairs of Asian and Western fonts that are treated as a single font family</td>
<td>FO_CombinedFontDefn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Document object property</th>
<th>Object that the property specifies</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FP_FirstGraphicInDoc</strong></td>
<td>The first graphic object (for example, FO_UnanchoredFrame or FO_Line) in the list of graphic objects</td>
</tr>
<tr>
<td><strong>FP_FirstColorInDoc</strong></td>
<td>The first FO_Color in the list of FO_Color objects</td>
</tr>
<tr>
<td><strong>FP_FirstPgfInDoc</strong></td>
<td>The first FO_Pgf in the list of FO_Pgf objects</td>
</tr>
<tr>
<td><strong>FP_FirstMarkerInDoc</strong></td>
<td>The first FO_Marker in the list of FO_Marker objects</td>
</tr>
<tr>
<td><strong>FP_FirstMarkerTypeInDoc</strong></td>
<td>The first FO_MarkerType in the list of marker types</td>
</tr>
<tr>
<td><strong>FP_FirstVarInDoc</strong></td>
<td>The first FO_Var in the list of FO_Var objects</td>
</tr>
<tr>
<td><strong>FP_FirstVarFmtInDoc</strong></td>
<td>The first FO_VarFmt in the list of FO_VarFmt objects</td>
</tr>
<tr>
<td><strong>FP_FirstXRefInDoc</strong></td>
<td>The first FO_XRef in the list of FO_XRef objects</td>
</tr>
<tr>
<td><strong>FP_FirstXRefFmtInDoc</strong></td>
<td>The first FO_XRefFmt in the list of FO_XRefFmt objects</td>
</tr>
<tr>
<td>Document object property</td>
<td>Object that the property specifies</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FP_FirstFnInDoc</td>
<td>The first FO_Fn in the list of FO_Fn objects</td>
</tr>
<tr>
<td>FP_FirstTblInDoc</td>
<td>The first FO_Tbl in the list of FO_Tbl objects</td>
</tr>
<tr>
<td>FP_FirstFlowInDoc</td>
<td>The first FO_Flow in the list of FO_Flow objects</td>
</tr>
<tr>
<td>FP_FirstPgFmtInDoc</td>
<td>The first FO_PgFmt in the list of FO_PgFmt objects</td>
</tr>
<tr>
<td>FP_FirstCharFmtInDoc</td>
<td>The first FO_CharFmt in the list of FO_CharFmt objects</td>
</tr>
<tr>
<td>FP_FirstCondFmtInDoc</td>
<td>The first FO_CondFmt in the list of FO_CondFmt objects</td>
</tr>
<tr>
<td>FP_FirstTblFmtInDoc</td>
<td>The first FO_TblFmt in the list of FO_TblFmt objects</td>
</tr>
<tr>
<td>FP_FirstRulingFmtInDoc</td>
<td>The first FO_RulingFmt in the list of FO_RulingFmt objects</td>
</tr>
<tr>
<td>FP_FirstSelectedGraphicInDoc</td>
<td>The first graphic object in the list of selected graphic objects</td>
</tr>
<tr>
<td>FP_MainFlowInDoc</td>
<td>FO_Flow that represents the main flow</td>
</tr>
<tr>
<td>FP_FirstElementDefInDoc</td>
<td>First structural element definition in the list of element definitions in a FrameMaker document</td>
</tr>
<tr>
<td>FP_FirstFmtChangeListInDoc</td>
<td>First format change list in the list of format change lists in a FrameMaker document</td>
</tr>
<tr>
<td>FP_FirstBodyPageInDoc or</td>
<td>The first or last FO_BodyPage in the list of FO_BodyPage objects</td>
</tr>
<tr>
<td>FP_LastBodyPageInDoc</td>
<td></td>
</tr>
<tr>
<td>FP_FirstMasterPageInDoc or</td>
<td>The first or last FO_MasterPage in the list of FO_MasterPage objects</td>
</tr>
<tr>
<td>FP_LastMasterPageInDoc</td>
<td></td>
</tr>
<tr>
<td>FP_FirstRefPageInDoc or</td>
<td>The first or last FO_RefPage in the list of FO_RefPage objects</td>
</tr>
<tr>
<td>FP_LastRefPageInDoc</td>
<td></td>
</tr>
<tr>
<td>FP_HiddenPage</td>
<td>The hidden page (FO_HiddenPage)</td>
</tr>
<tr>
<td>FP_SelectedTbl</td>
<td>The selected table object</td>
</tr>
<tr>
<td>FP_FirstTiInDoc</td>
<td>The first FO_TiApiClient, FO_TiFlow, FO_TiText, or FO_TiTextTable in the list of text insets</td>
</tr>
<tr>
<td>FP_FirstRubiInDoc</td>
<td>The first FO_Rubi in the list of rubi composites</td>
</tr>
<tr>
<td>FP_FirstCombinedFontDefnInDoc</td>
<td>The first FO_CombinedFontDefn in the list of combined font definitions</td>
</tr>
</tbody>
</table>
FP_FirstBodyPageInDoc, FP_FirstMasterPageInDoc, and FP_FirstRefPageInDoc point to the lists of pages in a document. These lists are ordered to reflect the order of the pages. All other lists (including the list of FO_Pgf objects) are not ordered. The terms first and last indicate only the position of the objects in an arbitrarily ordered list. There is no guarantee that a more recently added object will come later in a list, nor is there a guarantee that the order of a list will remain the same as the document is modified.

Global document information

FrameMaker allows you to set global document information, characteristics that apply generally to an entire document.

What the user sees

Global document information includes the following formatting characteristics:

- Document page properties, which specify the document’s page numbering and pagination style
- Document condition properties, which specify whether conditional text appears and whether formatting associated with condition tags overrides other formatting
- Document and table footnote properties, which specify the appearance of the footnotes, such as the footnote numbering and the default paragraph format
- Change bar properties, which specify the appearance and position of change bars in the document
- The current text selection or insertion point

There is also global document information that affects how the FrameMaker product behaves when the document is active. This type of global information includes:

- The document dictionary, which lists words that you want the FrameMaker product Spelling Checker to ignore
- Type-in properties, which specify whether Smart Spaces or Smart Quotes is enabled
- Equation properties, which specify default symbol sizes and fonts the FrameMaker product uses when you add equations to the document
- Printing properties, which specify the defaults that appear in the Print dialog box, such as the printer name and the range of pages to print
- View properties, which specify how the FrameMaker product displays and scrolls the document in the window
- Structure properties, which specify whether element boundaries appear and how the Element Catalog appears for a structured document in a FrameMaker session.
FrameMaker saves most of the global document information with each document. For example, if you set the zoom for a document to 140 percent and save and exit a document, the next time you open the document, the zoom will be set to 140 percent.¹

**How the API represents global document information**

The Frame API represents global document information with `FO_Doc` object properties.

**How the API represents the selection in a document**

The API uses several properties to specify what is selected in a document:

- `FP_TextSelection` specifies a structure that provides the location of the insertion point or the beginning and end of a text selection.
- `FP_FirstSelectedGraphicInDoc` specifies the ID of the first graphic in the list of selected graphics in a document.
- `FP_SelectedTbl` specifies the ID of a table that contains the insertion point or some selected cells.
- `FP_ElementSelection` specifies the range of elements selected if the document is a structured document in a FrameMaker session.

The following table summarizes the different types of selection in an unstructured document and how these properties are set to represent them.

<table>
<thead>
<tr>
<th>Selection state</th>
<th>How selection properties are set</th>
</tr>
</thead>
<tbody>
<tr>
<td>No object is selected. There is no text selection or insertion point.</td>
<td><code>FP_TextSelection</code> specifies an <code>F_TextRangeT</code> structure for which the <code>objId</code> and <code>offset</code> fields of <code>F_TextRangeT.beg</code> and <code>F_TextRangeT.end</code> are set to 0. <code>FP_FirstSelectedGraphicInDoc</code> is 0. <code>FP_SelectedTbl</code> is 0.</td>
</tr>
<tr>
<td>One or more graphic objects are selected.</td>
<td><code>FP_TextSelection</code> specifies an <code>F_TextRangeT</code> structure for which the <code>objId</code> and <code>offset</code> fields of <code>F_TextRangeT.beg</code> and <code>F_TextRangeT.end</code> are set to 0. <code>FP_FirstSelectedGraphicInDoc</code> specifies the ID of the first selected graphic in the document’s list of selected graphics. <code>FP_SelectedTbl</code> is 0.</td>
</tr>
<tr>
<td>There is an insertion point or text selection (that isn’t in a table cell, but may include table anchors).</td>
<td><code>FP_TextSelection</code> specifies the location of the text selection or insertion point <code>FP_FirstSelectedGraphicInDoc</code> is 0. <code>FP_SelectedTbl</code> is 0.</td>
</tr>
</tbody>
</table>

¹ Some platforms do not save some of the global document information such as the printing properties.
How the API represents the element selection in a structured FrameMaker document

In a structured FrameMaker document, the selection properties described in the previous section behave as they would in an unstructured document. However, structured FrameMaker documents have an additional selection property, `FP_ElementSelection`, which specifies the selection in terms of the selected element range or `F_ElementRangeT` structure.

The `F_ElementRangeT` structure is defined as:

```c
typedef struct {
    F_ElementLocT beg; /* Beginning of the element range. */
    F_ElementLocT end; /* End of the element range. */
} F_ElementRangeT;
```

The `F_ElementLocT` structure specifies a location within an element. It is defined as:

```c
typedef struct {
    F_ObjHandleT parentId; /* Parent element ID. */
    F_ObjHandleT childId; /* Child element ID. */
    IntT offset; /* Offset within child/parent element. */
} F_ElementLocT;
```

There is an insertion point or text selection within a single table cell.

- `FP_TextSelection` specifies the location of the text selection or insertion point within the cell; for example, the ID of the paragraph containing the insertion point, and the offset within that paragraph.
- `FP_FirstSelectedGraphicInDoc` is 0.
- `FP_SelectedTbl` specifies the ID of the table containing the cell.
- If the current selection is in a paragraph, the paragraph’s `FP_InTextObj` property specifies the ID of the cell that contains the selection. The cell’s `FP_CellColNum` property specifies the column number, and the cell’s `FP_CellRow` property specifies the ID of its row.
- `FP_FirstSelectedGraphicInDoc` is 0.
- `FP_SelectedTbl` specifies the ID of the table containing the cell.
- The table properties `FP_TopRowSelection` and `FP_BottomRowSelection` specify the IDs of the first and last rows containing selected cells. The `FP_LeftColNum` and `FP_RightColNum` properties of the table specify the index numbers of the outermost columns in the selection.

An entire cell or set of cells is selected.

- `FP_TextSelection` specifies an `F_TextRangeT` structure for which the objId and offset fields of `F_TextRangeT.beg` and `F_TextRangeT.end` are set to 0.
- `FP_FirstSelectedGraphicInDoc` is 0.
- `FP_SelectedTbl` specifies the ID of the table containing the cell.
- The table properties `FP_TopRowSelection` and `FP_BottomRowSelection` specify the IDs of the first and last rows containing selected cells. The `FP_LeftColNum` and `FP_RightColNum` properties of the table specify the index numbers of the outermost columns in the selection.

a. If an entire cell is selected, there is no text selection.

How the API represents the element selection in a structured FrameMaker document

In a structured FrameMaker document, the selection properties described in the previous section behave as they would in an unstructured document. However, structured FrameMaker documents have an additional selection property, `FP_ElementSelection`, which specifies the selection in terms of the selected element range or `F_ElementRangeT` structure.

The `F_ElementRangeT` structure is defined as:

```c
typedef struct {
    F_ElementLocT beg; /* Beginning of the element range. */
    F_ElementLocT end; /* End of the element range. */
} F_ElementRangeT;
```

The `F_ElementLocT` structure specifies a location within an element. It is defined as:

```c
typedef struct {
    F_ObjHandleT parentId; /* Parent element ID. */
    F_ObjHandleT childId; /* Child element ID. */
    IntT offset; /* Offset within child/parent element. */
} F_ElementLocT;
```
The following table summarizes the different types of selection in a structured FrameMaker document and how the fields of the `F_ElementRangeT` structure specified by the `FP_ElementSelection` property are set to represent them.

<table>
<thead>
<tr>
<th>Selection state</th>
<th>What the fields of the <code>F_ElementRangeT</code> structure specify</th>
</tr>
</thead>
</table>
| No object is selected. There is no text selection or insertion point.           | `beg.parentId`: 0  
|                                                                                | `beg.childId`: 0  
|                                                                                | `beg.offset`: 0  |
| One or more graphic objects are selected.                                       | `end.parentId`: 0  
|                                                                                | `end.childId`: 0  
|                                                                                | `end.offset`: 0  |
| There is an insertion point or text selection within an element that has no subelements. | `beg.parentId`: ID of the element containing the insertion point or selection  
|                                                                                | `beg.childId`: ID of the child element immediately following the insertion point or the beginning of the text selection  
|                                                                                | `beg.offset`: offset of the beginning of the selection or insertion point from the beginning of the element containing it  
|                                                                                | `end.parentId`: ID of the element containing the insertion point or selection  
|                                                                                | `end.childId`: ID of the child element immediately following the insertion point or the end of the text selection  
|                                                                                | `end.offset`: offset of the end of the selection or insertion point from the beginning of the element containing it  |
| An entire element or range of elements (excluding the highest level element) is selected. | `beg.parentId`: ID of the element containing the first selected element  
|                                                                                | `beg.childId`: ID of the first selected element  
|                                                                                | `beg.offset`: 0  
|                                                                                | `end.parentId`: ID of the element containing the first selected element  
|                                                                                | `end.childId`: ID of the sibling element following the last selected element, or 0 if there is no sibling element following the last selected element  
|                                                                                | `end.offset`: 0  |
| The highest level element is selected.                                          | `beg.parentId`: 0  
|                                                                                | `beg.childId`: ID of the highest-level element  
|                                                                                | `beg.offset`: 0  
|                                                                                | `end.parentId`: 0  
|                                                                                | `end.childId`: 0  
|                                                                                | `end.offset`: 0  |

**Example**

Suppose you create the document shown in Figure 14.1.
The API represents the document with an `FO_Doc` object. The following table lists some of its properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_Name</td>
<td>String</td>
<td>C:\Sales\Training\1Chapter</td>
</tr>
<tr>
<td>FP_ViewBorders</td>
<td>IntT</td>
<td>True</td>
</tr>
<tr>
<td>FP_ViewRulers</td>
<td>IntT</td>
<td>True</td>
</tr>
<tr>
<td>FP_ViewPageScrolling</td>
<td>IntT</td>
<td>FV_SCROLL_VERTICAL</td>
</tr>
<tr>
<td>FP_Zoom</td>
<td>Metric</td>
<td>1 &lt;&lt; 16</td>
</tr>
<tr>
<td>FP_IsIconified</td>
<td>IntT</td>
<td>False</td>
</tr>
<tr>
<td>FP_ViewTextSymbols</td>
<td>IntT</td>
<td>True</td>
</tr>
<tr>
<td>FP_IsOnScreen</td>
<td>IntT</td>
<td>True</td>
</tr>
</tbody>
</table>

**Pages**

Frame documents have three kinds of visible pages: body pages, master pages, and reference pages.

**What the user sees**

With FrameMaker, the user can change any of the visible pages.
Body pages

Body pages are what a user normally thinks of as the document’s pages. They organize the text and graphic objects that appear in the body of a document.

Master pages

Master pages control the layout of body pages. Each body page is associated with one master page, and each master page can be associated with zero or more body pages. A master page provides the following for the body page:

● The text frame layout, which defines the number, size, and placement of the page’s text frames and the column layout within each text frame

● The page background, which includes graphic objects and text frames (such as headers and footers) with unnamed flows

By default, single-sided documents have at least one master page, which is named Right. Double-sided documents have two master pages, named Right and Left. FrameMaker allows you to add your own custom master pages to both single-sided and double-sided documents.

When FrameMaker adds a body page, it uses a left or right master page object as a background. It also copies all the text frames with named flows from that master page to the body page. Once the FrameMaker product copies these text frames to the body page, they are independent of the text frames on the master page from which they were copied. If you change the text frames, it does not affect the master page.

A body page’s background appears when you view a body page on the screen or print it. However, the background is part of the master page and not the body page. The FrameMaker product superimposes the body page on the background for displaying and printing. If you go to the master page and change the graphic objects that constitute the background, the changes appear when you view or print the body pages associated with the master page.

Reference pages

Reference pages can contain named graphic frames. Named graphic frames provide decoration, such as a thick line ruling, for paragraphs in the body pages. The Paragraph Designer provides two settings, Frame Above and Frame Below, that allow you to specify the named graphic frames you want to appear above or below a paragraph.

Reference pages can also contain special flows that control the appearance of generated documents. For example, a Table of Contents document normally has a flow named TOC on one of its reference pages.

How the API represents pages

FrameMaker represents body pages, master pages, and reference pages with FO_BodyPage, FO_MasterPage, and FO_RefPage objects, respectively.

In addition to these pages, a document can also have a hidden page, which stores hidden conditional text. The user cannot see or directly modify hidden pages. FrameMaker represents each hidden page with an FO_HiddenPage object.
Page objects have a number of common properties. These properties specify the following characteristics of a page:

- The dimensions of the page
- Its type (body, master, reference, or hidden)
- Its numbering
- IDs of the objects that represent its page frame and its sibling pages

A page object does not actually contain the text and graphic objects that appear on a page. Instead, it has a property named `FP_PageFrame`, which specifies the ID of a page frame. A page frame is an invisible unanchored frame whose dimensions match those of the page. (For more information on unanchored frames, see “Graphic objects” on page 293.) The API represents a page frame with an `FO_UnanchoredFrame` object. This `FO_UnanchoredFrame` object has properties that specify the IDs of the first and last objects in the linked list of API graphic objects that appear directly on the page.

Suppose you create a body page with a single text frame as shown in Figure 14.2.

![Figure 14.2 Single text frame on a body page](image)

FrameMaker automatically creates and destroys the page frame when it creates and destroys the page. The page frame is not accessible to the user. However, as a developer, you need to use it to get the objects on a page.
How the API organizes pages

The API maintains the different types of visible pages in a document (body pages, master pages, and reference pages) in separate linked lists. FO_Doc objects have the following properties that specify the first and last page object in each list:

- FP_FirstBodyPageInDoc and FP_LastBodyPageInDoc
- FP_FirstMasterPageInDoc and FP_LastMasterPageInDoc
- FP_FirstRefPageInDoc and FP_LastRefPageInDoc

Each page object has two properties, FP_PagePrev and FP_PageNext, that specify the IDs of the page objects before and after it in the list. When you delete a page, the API removes the object that represents it and updates the FP_PagePrev and FP_PageNext properties for all the FO_Page objects before and after it.

FO_Doc objects also have a property named FP_CurrentPage that specifies the ID of the current page. The current page is the page that appears on the screen. If more than one page appears on the screen, it is the page that appears with a dark border around it.

Suppose you create a double-sided document that has three body pages, two master pages (Left and Right), and a single reference page, as shown in Figure 14.4. The current page is the Right master page.

FrameMaker organizes the objects as shown in Figure 14.5.
**FIGURE 14.5  Example document and page objects**

**How the API represents hidden pages**

If the user chooses to hide conditional text in the document, the FrameMaker product adds a hidden page to the document to store the hidden text. A document can have only one hidden page. FrameMaker represents the hidden page with an `FO_HiddenPage` object. The `FO_Doc` property `FP_HiddenPage` specifies its ID. The hidden page has only one text flow, named HIDDEN. For more information on how the API represents hidden conditional text, see “How the API represents hidden conditional text” on page 318.

**How the API represents master pages**

Both single-sided and double-sided documents have default master pages, named Right and Left. In single-sided documents, the Left master page is not visible to the user. However, you can get and set its properties with the API.
Graphic objects

A graphic object is anything the user can draw with the Tools palette, or an imported graphic.

What the user sees

A graphic object can be:

- An anchored frame, which is a container for graphic objects that is tied to a specific location in text.
- An unanchored frame, which is a container for graphic objects that is not tied to a specific location in text.
- A simple geometric shape, which is a line, an arc, a rectangle, a rounded rectangle, an ellipse, a polyline, or a polygon.
- A group, which is an invisible graphic object that holds together a set of other graphic objects.
- A text line, which is a single line of text that isn’t in a paragraph or flow (for more information on text lines, see “Text” on page 311).
- A text frame, which is a container for text in a flow (for more information on text frames, see “Text” on page 311).
- An imported graphic such as a bitmap or a PostScript file created with another application.
- An inset or imported graphic.
- A math equation, which describes a formatted equation.

You can draw a graphic object directly on a page in a document. A graphic object drawn directly on a page doesn’t move as you edit the text around it. You can also draw a graphic object inside an anchored or unanchored frame. When you move a frame, all the graphic objects inside it move with it. You can nest frames; that is, you can draw a frame within a frame within a frame.

Draw order

The graphic objects in a frame have a back-to-front order or *draw order* that specifies the order in which the FrameMaker product draws them. By default, the draw order is the same as the order in which you created the objects. When graphic objects overlap, the ones in front (at the end of the draw order) obscure those in back. You can change the draw order by selecting a graphic object and choosing Front or Back from the Tools palette.

Groups

You can create a group from one or more graphic objects. This allows you to manipulate them as a single object. When you resize the group, the FrameMaker product automatically resizes the group’s component objects proportionally.
Anchored frames

You can draw graphic objects in anchored frames, which are tied to text symbols named anchor symbols (⊥). An anchor symbol (and its anchored frame) moves with the text to which it is attached. You can specify a variety of parameters that determine where a frame appears in relation to its anchor symbol. For example, it can be below the line containing the anchor symbol or at the bottom of the text frame containing the anchor symbol. Unlike other graphic objects, the anchored frame cannot be drawn directly on a page or into another frame; it can only be created in text.

How the API represents graphic objects

The API represents each type of graphic object with a different type of API object. For example, it represents polygons with FO_Polygon objects and text frames with FO_TextFrame objects.

All types of API graphic objects² have properties that provide the following information:
- The graphic object’s format—that is, characteristics such as its fill pattern and border width
- The graphic object’s location and angle
- IDs of the graphic object’s parent, sibling, and child objects

Some format properties do not affect some graphic objects. For example, an FO_Rectangle object, like all other objects, has an FP_ArrowType property. This property can have a value, but that value will not affect the appearance of the rectangle that the object represents.

All types of API graphic objects also have several properties that are specific to them. For example, FO_Arc objects have a property named FP_Theta that specifies an arc’s theta value.

Suppose you create the arrow shown in Figure 14.6.

---

2. This manual uses the term API graphic object to refer to objects (such as FO_Polygon and FO_TextFrame objects) that the API uses to represent the graphic objects (such as polygons and text frames) that appear on a page.
The API represents the arrow with an FO_Line object, whose properties include those shown in the following table.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_FrameParent</td>
<td>F_ObjHandleT</td>
<td>ID of the frame containing the arrow</td>
</tr>
<tr>
<td>FP_Pen</td>
<td>IntT</td>
<td>FV_FILL_BLACK</td>
</tr>
<tr>
<td>FP_LocX</td>
<td>MetricT</td>
<td>.25*in</td>
</tr>
<tr>
<td>FP_LocY</td>
<td>MetricT</td>
<td>1.125*in</td>
</tr>
<tr>
<td>FP_HeadArrow</td>
<td>IntT</td>
<td>True</td>
</tr>
<tr>
<td>FP_ArrowType</td>
<td>IntT</td>
<td>FV_ARROW_HOLLOW</td>
</tr>
</tbody>
</table>

How the API organizes graphic objects

The API maintains each API graphic object in at least two linked lists:

- The list of all the API graphic objects in the document
  
  For convenience, the API maintains a linked list of all the API graphic objects in a document. The FO_Doc property FP_FirstGraphicInDoc specifies the ID of the first object in the list. API graphic objects have a property named FP_NextGraphicInDoc, which specifies the ID of the next API graphic object in the list. If you traverse this list, you will cover every graphic object in a document. The order of the list is completely random.

- The list of API graphic objects in the graphic object’s parent frame

  Each API graphic object (except an anchored frame and a page frame) has exactly one parent frame object. The parent frame is the frame that contains the graphic object. The API maintains a linked list of the child objects in each frame. FO_UnanchoredFrame and FO_AFrame objects have FP_FirstGraphicInFrame and FP_LastGraphicInFrame properties, which specify the first and last objects in the list of their child objects. All API graphic objects have FP_PrevGraphicInFrame and FP_NextGraphicInFrame properties, which specify the objects before and after them in the list. The order of the objects in the linked list is the same as the draw order of the graphic objects in a frame.

Like the frames they represent, API frame objects can be nested: that is, an FO_UnanchoredFrame or FO_AFrame object can be the parent of another FO_UnanchoredFrame object. Every API graphic object (except an object that represents a page frame or an anchored frame) is a descendant of exactly one API page frame object.

Suppose you create a page that contains:

- An unanchored frame that contains an oval, a rectangle, and a text frame with some text in it
- A text line that overlaps the unanchored frame, but is drawn directly on the page

FrameMaker organizes these objects as shown in Figure 14.7.
How the API represents groups

The API represents a grouped set of graphic objects with an FO_Group object. It maintains the objects that constitute a group in a linked list. The FO_Group properties, FP_FirstGraphicInGroup and FP_LastGraphicInGroup, specify the first and last objects in the list. Each graphic object has FP_PrevGraphicInGroup and FP_NextGraphicInGroup properties, which specify the objects before and after it in the list.

Grouping graphic objects does not affect their position in the linked list of API graphic objects in a frame. That is, it does not affect their position in the draw order. Group objects themselves have an arbitrary position in the draw order.

How the API represents selections of graphic objects

The FO_Doc property, FP_FirstSelectedGraphicInDoc, specifies the ID of the object that represents the first selected graphic object in a document. If more than one graphic object is selected, the API forms a linked list of the API graphic objects that represent the selected graphic objects. API graphic objects have an FP_NextSelectedGraphicInDoc property that specifies the ID of the next selected graphic object. The order of the list is not necessarily the same as the order in which the graphic objects were selected.
Although `FP_FirstSelectedGraphicInDoc` is a document property, you can only select graphic objects that are within the same frame.

**Flows**

FrameMaker uses flows to connect text frames in a document.

**What the user sees**

A flow tells the FrameMaker product where to put additional text when a text frame is full. In a simple document, there may be only one flow associated with the body pages. In complex documents such as newsletters, you may create multiple flows that have connected text frames on different pages.

![Figure 14.8 Multiflow document](image)

**Main flows**

Each document has a main flow. Normally, the FrameMaker product treats the main flow like any named flow in the document. However, there are cases when the FrameMaker product treats the main flow specially:

- When you generate a table of contents or an index, the FrameMaker product puts the generated text into the main flow of the generated document.
- When you run Compare Documents, the FrameMaker product puts the Summary text into the main flow.

Usually the main flow is the default flow for the current language. For example, if the current language is English, the main flow is `A`.

If there are several Autoconnect flows in the document with the default flow tag, the main flow is the one in the backmost text frame on the frontmost body page.
How the API represents flows

The API represents a flow with an FO_Flow object, whose properties provide the following information:

- The flow’s format characteristics, such as the feathering and whether Autoconnect is enabled
- The IDs of the first and last FO_TextFrame objects in the flow
- The ID of the next FO_Flow object in the document

How the API organizes flows

The API maintains a document’s FO_Flow objects in a linked list. The FO_Doc property, FP_FirstFlowInDoc, specifies the ID of the first FO_Flow object in the list. FO_Flow objects have a property named FP_NextFlowInDoc, which specifies the next FO_Flow object in the list. The order of the list is random; it does not correspond to the order in which the flows appear in the document.

The API also maintains the objects that represent a flow’s text frames in a linked list. The FO_Flow properties, FP_FirstTextFrameInFlow and FP_LastTextFrameInFlow, specify the first and last FO_TextFrame objects in the list. Each FO_TextFrame object has an FP_PrevTextFrameInFlow property and a FP_NextTextFrameInFlow property, which specify the previous and next FO_TextFrame objects in the list. For more information on how flows, text frames, and paragraphs are organized, see “How the API organizes paragraphs” on page 304.

Suppose you create the document shown in Figure 14.9. The document has two flows: a main flow, named A, and a second flow that is unnamed. The A flow connects a two-column text frame on the first page and a two-column text frame on the second page. The unnamed flow appears only on the first page and has only one text frame.

Figure 14.9 shows how the API organizes the objects that represent the flows and text frames shown in Figure 14.9.
Although the FP_FirstFlowInDoc property specifies the FO_Flow object for the unnamed flow in Figure 14.10, it could also specify the FO_Flow object that represents Flow A. The determination of which flow is first in the list is completely random.

The API uses FO_SubCol objects to represent the column formatting of text as follows:

- Contiguous paragraphs in each column of a text frame are within a single FO_SubCol object.
- Contiguous paragraphs within sidehead area are within a single FO_SubCol.
- Each contiguous series of paragraphs that spans columns and/or sideheads is represented by a single FO_SubCol object. Following paragraphs that do not span columns and sideheads begin a new group of FO_SubCol objects. For example, Figure 14.11 shows a page that has seven FO_SubCol objects—two groups of three, plus one for the heading that spans all columns.
The FO_TextFrame properties, FP_FirstSubCol and FP_LastSubCol, specify the first and last FO_SubCol objects in a specific text frame. Each FO_SubCol object has an FP_PrevSubCol property and a FP_NextSubCol property, which specify the previous and next FO_SubCol objects in the flow. Each FO_SubCol object also has a FP_ParentTextFrame property, which specifies the text frame it is in. If a text frame has only one column, its FP_FirstSubCol and FP_LastSubCol properties both specify the ID of the FO_SubCol object that represents it.

Figure 14.12 shows how the API organizes the objects that represent the two-column text frame on the first page of the document in Figure 14.9.

In FrameMaker, flows can be structured. If a flow is structured, the FO_Flow object that represents it has a FP_HighestLevelElement property that specifies the ID of the root element.

For information on how the API organizes paragraphs and text in flows, text frames, and columns, see “How the API organizes paragraphs” on page 304.

---

**Paragraph Catalog formats**

Each document has a Paragraph Catalog containing one or more Paragraph Catalog formats.

**What the user sees**

Each Paragraph Catalog format specifies aspects of a paragraph’s appearance, such as its indents, line spacing, and default font. Each format has a name or tag, which usually corresponds to a type of paragraph, such as title, body, or heading. To make a paragraph’s appearance conform to a format, you apply the format to the paragraph. You can apply the same format to multiple paragraphs to ensure consistency in your document.
You can add formats to the Paragraph Catalog or modify or delete formats that are already in it.

**How the API represents Paragraph Catalog formats**

FrameMaker represents each Paragraph Catalog format with an `FO_PgfFmt` object, whose properties provide the following information:

- The name of the paragraph format
- Formatting information
- The ID of the next `FO_PgfFmt` object in the document

Suppose you create the paragraph format described in the Paragraph Designer in Figure 14.13.

![Figure 14.13 Paragraph Designer](image)

The API represents this paragraph format with an `FO_PgfFmt` object. The following table lists some of its properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_Name</td>
<td>StringT</td>
<td>Step</td>
</tr>
<tr>
<td>FP_AutoNumString</td>
<td>StringT</td>
<td>&lt;n+&gt;.\t</td>
</tr>
<tr>
<td>FP_AutoNumChar</td>
<td>StringT</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>FP_PgfIsAutoNum</td>
<td>IntT</td>
<td>True</td>
</tr>
<tr>
<td>FP_NextPgfFmtInDoc</td>
<td>F_ObjHandleT</td>
<td>ID of the next Paragraph Catalog format in the document’s list of Paragraph Catalog formats</td>
</tr>
</tbody>
</table>

The `FP_AutoNumChar` property specifies the character format to be applied to the string specified by `FP_AutoNumString`. When the default font is used, `FP_AutoNumChar` is set to a null string (""").
How the API organizes Paragraph Catalog formats

The API organizes the FO_PgfFmt objects in a document in a linked list. The FO_Doc property, FP_FirstPgfFmtInDoc, specifies the first FO_PgfFmt object in the list. FO_PgfFmt objects have an FP_NextPgfFmtInDoc property, which specifies the ID of the next FO_PgfFmt object in the list. The order of the list does not correspond with the order in which the formats appear in the Paragraph Catalog.

Paragraphs

A paragraph can be a body of text, a title, or an item in a list.

What the user sees

You can type a paragraph in a text frame, a footnote, or a table cell.

Every paragraph has a paragraph format consisting of:

- A tag, which is the name of a format stored in the Paragraph Catalog
- Formatting information, which is the same information that a Paragraph Catalog format provides, such as indents and leading

Every paragraph starts with a tag and formatting information that matches a Paragraph Catalog format. There are several ways you can change a paragraph’s format:

- Apply a different Paragraph Catalog format to the paragraph.
  
  When you do this, the FrameMaker product changes the paragraph’s formatting information to match that of the Paragraph Catalog format. This process is known as tagging.

- Change the paragraph’s formatting information.
  
  This does not affect the Paragraph Catalog format that you tagged the paragraph with. For example, if you tag a paragraph with a tag named indentbody that specifies a 1-inch indent and subsequently change the paragraph’s indent to 2 inches, the indentbody format and other paragraphs tagged as indentbody still have a 1-inch indent. This change is a format override, and it applies only to that paragraph instance.

- Change the Paragraph Catalog format’s formatting information.
  
  FrameMaker allows you to update all the paragraphs that are tagged with the format you changed. You can choose whether you want to retain format overrides when FrameMaker updates all paragraphs in the document with the same tag.

How the API represents paragraphs

FrameMaker represents each paragraph with an FO_Pgf object, whose properties provide the following information:
● The ID of the text frame and text column containing the paragraph
● The paragraph’s formatting information (the same set of properties that a Paragraph Catalog format provides)
● The paragraph’s tag
● The IDs of sibling FO_Pgf objects
● A flag indicating whether the paragraph has been successfully spell-checked since the last change was made to it

Each paragraph object also contains an F_TextItemsT structure, which represents the text in the paragraph. For more information about text and the F_TextItemsT structure, see “How the API represents text” on page 312.

Suppose you create the paragraph shown in Figure 14.14.

![Figure 14.14 A paragraph](image)

The API represents the paragraph with an FO_Pgf object. The following table lists some of its properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_Name</td>
<td>StringT</td>
<td>Step</td>
</tr>
<tr>
<td>FP_AutoNumString</td>
<td>StringT</td>
<td>&lt;n+&gt;.\t</td>
</tr>
<tr>
<td>FP_PgfIsAutoNum</td>
<td>IntT</td>
<td>True</td>
</tr>
<tr>
<td>FP_PgfNumber</td>
<td>StringT</td>
<td>1.</td>
</tr>
<tr>
<td>FA_LeftIndent</td>
<td>MetricT</td>
<td>0</td>
</tr>
<tr>
<td>FP_InTextFrame</td>
<td>F_ObjHandleT</td>
<td>ID of the text frame the paragraph starts in</td>
</tr>
<tr>
<td>FP_InTextObj</td>
<td>F_ObjHandleT</td>
<td>ID of the subcolumn (FO_SubCol object) the paragraph starts in</td>
</tr>
</tbody>
</table>

A paragraph’s FP_InTextObj property does not always specify the ID of a subcolumn. If the paragraph appears in a table cell, it specifies the ID of the FO_Cell object representing the cell. If the paragraph appears in a footnote, FP_InTextObj specifies the ID of the FO_Fn object representing the footnote.
How to apply formats to paragraphs
To apply a format from the paragraph format catalog to a specific paragraph object, first get the ID of the FO_Pgf object in question. Then loop through the document looking for the FO_PgfFmt object with a name that matches the tag you want to apply to the paragraph. Then use F_ApiGetProps() to get the list of properties from the FO_PgfFmt object, and use F_ApiSetProps() to set the property list to the FO_Pgf in question.

How the API organizes paragraphs
The API maintains FO_Pgf objects in two linked lists:
- The list of all FO_Pgf objects in a document
- The list of FO_Pgf objects in a flow

The list of paragraphs in a document
The FO_Doc property, FP_FirstPgfInDoc, specifies the first FO_Pgf object in the list of FO_Pgf objects in a document. Each FO_Pgf object has an FP_NextPgfInDoc property, which specifies the next FO_Pgf object in the list. The order of the list of FO_Pgf objects in a document does not necessarily correspond to the actual order of the paragraphs in the document.

The list of paragraphs in a flow
FO_Flow objects do not have a property that specifies the first FO_Pgf object in a flow. To find the first FO_Pgf object in the flow, you must find the first FO_TextFrame object in the flow. Then you must get the FO_Pgf object specified by the FO_TextFrame object's FP_FirstPgf property. In some cases, the first text frame in the flow may not contain any paragraphs. You must traverse subsequent text frames and check them to see if they contain any paragraphs.

Each FO_Pgf object has FP_PrevPgfInFlow and FP_NextPgfInFlow properties, which specify the IDs of the FO_Pgf objects before and after it in the flow. To get the paragraphs in a flow in order, you traverse these properties.

It is possible for a paragraph to begin in one text frame and end in another. When this happens, the ID of the FO_Pgf is specified by the FP_LastPgf property of the text frame in which it begins and the FP_FirstPgf property of the text frame in which it ends.

Suppose you create two text frames and connect them with a flow. The first text frame has two paragraphs in it; the second paragraph continues into the next text frame as shown in Figure 14.15.

![Figure 14.15 Flow with two text frames](image)

The API organizes the objects that represent the flow, text frames, and paragraphs as shown in Figure 14.16.
FIGURE 14.16 Objects that represent a flow with text frames and paragraphs

Like text frames, subcolumns (FO_SubCol objects) have FP_FirstPgf and FP_LastPgf properties, which specify the first and last paragraphs in them. If a paragraph begins in one text column and ends in another, the paragraph's ID is specified by the FP_LastPgf property of the subcolumn in which it begins and the FP_FirstPgf property of the subcolumn in which it ends.

Character Catalog formats

Each document has a Character Catalog containing one or more character formats.

What the user sees

Each character format has a name (or tag), which usually corresponds to a type of text, such as Emphasis, Usertype, or Booktitle. It also contains information about how text should look,
such as its font family, weight, and angle. To make the appearance of a set of characters conform to a Character Catalog format, you apply the format to the set of characters.

When you apply a character format to a set of characters, it overrides the default font setting of the paragraph format. If you reapply a Paragraph Catalog format to the paragraph, it does not affect the format of the characters that you previously tagged with the character format.

FrameMaker allows you to create character formats that override only some properties of the text to which they are applied. To leave one of the current text properties intact, you set the corresponding character format property to As Is. The Character Designer indicates the As Is state with the words As Is or a grayed (or stippled) checkbox.

You can add character formats to the Character Catalog or modify or delete formats that are already in it.

**How the API represents Character Catalog formats**

FrameMaker represents each Character Catalog format with an FO_CharFmt object, whose properties provide the following information:

- The name of the character format
- Character formatting information, such as the font family, angle, and weight
- Whether the character format’s formatting overrides the default formatting of the text that the format is applied to
- The ID of the next FO_CharFmt object in the document

**How the API represents fonts**

FO_Session objects have properties (such as FP_FontFamilyNames) that provide arrays of the names of the font families, variations, angles, and weights available in the current session. These lists are referenced by F_StringsT structures.

F_StringsT is defined as:

```c
typedef struct {
    UIntT len; /* Number of strings */
    StringT *val; /* Array of strings */
} F_StringsT;
```

For example, if Bold and Regular are the only font weights available in the current session, the fields of the F_StringsT structure specified by the FO_Session property, FP_FontWeightNames, have the following values:

```c
len: 3
val: {"<None>"", "Regular", "Bold"}
```

To set a character format’s weight to Bold in this session, you set its FP_FontWeight property to 2.

For more information on session font properties, see “How the API indicates which fonts are available in a session” on page 276.

You can also use the following properties to specify a font:
● **FP_FontPlatformName** specifies a font name that uniquely identifies the font on a specific platform.

● **FP_FontPostScriptName** specifies the name given to a font when it is sent to a PostScript printer (specifically, the name that is passed to the PostScript FindFont operator before any font coordination operations).

The PostScript name is unique for all PostScript fonts, but may not be available for fonts that have no PostScript version.

For the same Asian font, the PostScript name can be different on different platforms. This is because they might have slightly different extensions to the character mappings. For example, Ryumin-Light on the Macintosh is `Ryumin-Light-83pv-RKSJ-H`, while it is `Ryumin-Light-90ms-RKSJ-H` on a Windows system. Since these are the same fonts, FrameMaker will treat these as the same PostScript name. To do so, FrameMaker ignores the following keywords in PostScript names:

- 83pv
- 90pv
- 90ms
- Ext
- Add
- NWP

The **FP_FontPlatformName** property specifies a platform-specific ASCII string that uniquely identifies a font for a particular platform. The string consists of several fields separated by periods. On the Macintosh platform, it has the following syntax:

```
M.FontName.StyleFlags
```

<table>
<thead>
<tr>
<th>This field</th>
<th>Represents</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>Platform designator</td>
</tr>
<tr>
<td>FontName</td>
<td>Macintosh Resource Manager font name (for more information, see your Macintosh documentation)</td>
</tr>
<tr>
<td>StyleFlags</td>
<td>Macintosh font styles; you can use zero or more of the following flags:</td>
</tr>
<tr>
<td></td>
<td>B (Bold)</td>
</tr>
<tr>
<td></td>
<td>I (Italic)</td>
</tr>
<tr>
<td></td>
<td>C (Condense)</td>
</tr>
<tr>
<td></td>
<td>E (Extend)</td>
</tr>
<tr>
<td></td>
<td>P (Plain, no flags set)</td>
</tr>
<tr>
<td></td>
<td>You cannot use the C and E flags for the same font.</td>
</tr>
</tbody>
</table>

The following strings are valid representations of the Macintosh font Helvetica Narrow Bold Oblique:

- `M.Helvetica.BIC`
- `M.B Helvetica Bold.IC`
- `M.NI Helvetica Narrow Oblique.B`
- `M.NB1 Helvetica Narrow BoldObl.P`

On Windows platforms, the string you specify for **FP_FontPlatformName** has the following syntax:
The following strings are valid representations of the Windows font, Helvetica Narrow Bold Oblique:

W.Helvetica-Narrow.I.700
W.Helvetica.I.700.Narrow

When reading in a document, FrameMaker determines a font name by checking font properties in the following order:

- **FP_FontPlatformName**
- **Combination of** 
  - **FP_FontFamily**
  - **FP_FontVariation**
  - **FP_FontWeight**
  - **FP_FontAngle**
- **FP_FontPostScriptName**

Your clients do not need to use all three methods to change fonts. You should always specify the PostScript name, if it is available. Use the platform name only if your client is for a specific platform.

**How the API represents As Is settings**

**FO_CharFmt** objects use two properties to represent a font characteristic: one to represent the characteristic’s As Is state and one to represent the characteristic itself.

For example, **FP_UseFontWeight** specifies whether the character format’s font weight overrides the default font weight of the text that the format is applied to. **FP_FontWeight** specifies the character format’s font weight. If **FP_UseFontWeight** is True, the font weight specified by **FP_FontWeight** overrides the default font weight for the text. If **FP_UseFontWeight** is False (As Is), **FP_FontWeight** does not affect the text’s font weight.

If an **FP_UseCharacteristic** property is False, the character format’s property list includes only the **FP_UseCharacteristic** property. It doesn’t include the **FP_Characteristic** property for the characteristic (since this property is not used).

Suppose you create the character format specified in the Character Designer in **Figure 14.17**.
The API represents the character format with an FO_CharFmt object. The following table lists some of its properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_CharTag</td>
<td>StringT</td>
<td>booktitle</td>
</tr>
<tr>
<td>FP_FontAngle</td>
<td>IntT</td>
<td>Index of Italic font angle</td>
</tr>
<tr>
<td>FP_useFontAngle</td>
<td>IntT</td>
<td>True</td>
</tr>
<tr>
<td>FP_useFontFamily</td>
<td>IntT</td>
<td>False</td>
</tr>
<tr>
<td>FP_useFontVariation</td>
<td>IntT</td>
<td>False</td>
</tr>
<tr>
<td>FP_useFontWeight</td>
<td>IntT</td>
<td>False</td>
</tr>
<tr>
<td>FP_useUnderline</td>
<td>IntT</td>
<td>False</td>
</tr>
<tr>
<td>FP_FontSize</td>
<td>MetricT</td>
<td>36*pts</td>
</tr>
<tr>
<td>FP_useFontSize</td>
<td>IntT</td>
<td>True</td>
</tr>
</tbody>
</table>

This character format overrides the default font angle setting and the size of the text to which it is applied. It does not override any of the text’s other default characteristics.

**How the API organizes Character Catalog formats**

The API organizes the formats in a document’s Character Catalog in a linked list. FO_Doc objects have an FP_FirstCharFmtInDoc property that specifies the first FO_CharFmt object in the list. FO_CharFmt objects have an FP_NextCharFmtInDoc property, which specifies the ID of the next FO_CharFmt object in the list. The order of the list does not correspond to the order in which the formats appear in the Character Catalog.

**Condition Formats**

FrameMaker provides condition formats that allow the user to selectively show or hide text in a document.
What the user sees

To selectively show and hide text, you create a condition format (or tag) and apply it to selections of text. For example, you can create a condition tag named Comment and apply it to all the comments you add to a document. You can then instruct the FrameMaker product to hide all the text with the Comment tag when you print a final draft of the document. A document can have multiple condition tags.

FrameMaker allows you to specify a format override, or a special style and color for a condition. For example, you can make all text tagged with the Comment condition underlined and red.

How the API represents condition formats

The API represents each condition format with an FO_CondFmt object, whose properties provide the following information:

- The condition name
- Whether text tagged with the condition is currently visible
- The format overrides
- The ID of the next condition format in the document

The API represents the condition setting of a location in text as a text property. For more information on text properties, see “How the API represents text” on page 312.

If you choose to hide a condition tag, the FrameMaker product moves text with that tag to a hidden page and replaces it with markers. For more information on hidden conditional text, see “How the API represents hidden conditional text” on page 318.

Suppose you create the condition tag specified in the Edit Condition Tag dialog box shown in Figure 14.18.

![Edit Condition Tag dialog box](image)

The API represents the condition with an FO_CondFmt object. The following table lists some of its properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_Name</td>
<td>StringT</td>
<td>Comment</td>
</tr>
<tr>
<td>FP_NextCondFmtInDoc</td>
<td>F_ObjHandleT</td>
<td>ID of next condition in list of document’s conditions</td>
</tr>
</tbody>
</table>
How the API organizes condition formats

The API organizes a document’s FO_CondFmt objects in a linked list. The FO_Doc property FP_FirstCondFmtInDoc specifies the first FO_CondFmt object in the list. FO_CondFmt objects have a property named FP_NextCondFmtInDoc, which specifies the ID of the next FO_CondFmt object in the list. The order of the list does not correspond to the order in which the formats appear in the Conditional Text window.

Text

The user can type text into a text line or a paragraph in a text frame, table cell, or footnote.

What the user sees

FrameMaker allows you to insert things, such as anchored frames, footnotes, tables, and cross-references into text. The point at which you insert these things is called an anchor. FrameMaker represents an anchor with an anchor symbol (⊥) on the screen. This symbol is not visible if the Text Symbols view option is turned off. The anchor moves with the text to which it is attached.

All text has a set of properties that specify the following information about it:

- A tag, or the name of a character format stored in the Character Catalog
- Formatting information (the same information that a Character Catalog format provides, such as the font family and size)
- A set of conditional text formats that apply to it

These properties are called text properties.

Just as you can override a Paragraph Catalog tag by changing an individual paragraph’s format, you can also override a Character Catalog tag by changing the properties of a selection of text.

You can also apply one or more conditions to a selection of text. This allows you to hide or display the text for particular versions of a document.
How the API represents text

The API represents the text in each paragraph or graphic text line with an F_TextItemsT structure, which is defined as:

```c
typedef struct {
    UIntT len; /* The number of text items */
    F_TextItemT *val; /* Array of text items */
} F_TextItemsT;
```

The API represents an individual text item with an F_TextItemT structure, which is defined as:

```c
typedef struct {
    IntT offset; /* Characters from the beginning */
    IntT dataType;  /* The type of text item, e.g. FTI_String */
    union {
        StringT sdata; /* String if type is FTI_String */
        IntT idata; /* An ID if the item specifies an object */
    } u;
} F_TextItemT;
```

The `offset` value specifies the distance between the start of the text item and the beginning of the text line or paragraph. This distance is measured in the number of characters (both regular characters and anchor symbols).

Each of the following constitutes a separate text item:

- A string of characters with common text properties
  A text item can contain a string that is as long as a line of text. However, the API uses a separate text item for each section of the text that has different text properties. If a single property (such as the font weight, font angle, or condition format) is different, the API starts a new text item. So a single line of text may require several text items to represent it.

- The beginning or end of a line, paragraph, flow, column, page, or structural element
  The API uses text items to indicate the beginning or end of the various entities that organize text. Most of these text items specify the ID of an object. Text items that indicate the end of a line specify whether the line end is a regular, hyphenated, or hard line end.

- An anchor for a table, footnote, marker, cross-reference, variable, or anchored frame
  The API represents tables, footnotes, markers, cross-references, variables, and anchored frames with separate objects. It uses a text item to represent the anchor for each of these entities. The text item specifies the ID of the object. For example, the API represents a table with an FO_Tbl object. It uses a table anchor (FTI_TblAnchor) text item to indicate where the table occurs in the text.

- A text properties change
  This type of text item identifies the point in text at which the text properties change. It specifies flags that indicate which text properties differ from the properties of the text immediately preceding the text item.
The following table lists the values the `F_TextItemT.dataType` field can have and the types of data the corresponding text item provides.

<table>
<thead>
<tr>
<th>Text item type (dataType)</th>
<th>What the text item represents</th>
<th>Text item data</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTI_TextObjId</td>
<td>The object to which the offsets of all the text items are relative</td>
<td>ID of an FO_Pgf, FO_Cell, FO_TextLine, FO_TiApiClient, FO_TiFlow, FO_TiText, or FO_TiTextTable</td>
</tr>
<tr>
<td>FTI_String</td>
<td>A string of characters with the same condition and character format</td>
<td>A character string</td>
</tr>
<tr>
<td>FTI_LineBegin</td>
<td>The beginning of a line</td>
<td>Nothing</td>
</tr>
<tr>
<td>FTI_LineEnd</td>
<td>The end of a line and the line end type</td>
<td>If the line end is a normal line end, 0; if it is a forced line end, the FTI_HardLineEnd flag is set; if it is a hyphen line end, the FTI_HyphenLineEnd flag is set</td>
</tr>
<tr>
<td>FTI_PgfBegin</td>
<td>The beginning of a paragraph</td>
<td>ID of an FO_Pgf</td>
</tr>
<tr>
<td>FTI_PgfEnd</td>
<td>The end of a paragraph</td>
<td>ID of an FO_Pgf</td>
</tr>
<tr>
<td>FTI_FlowBegin</td>
<td>The beginning of a flow</td>
<td>ID of an FO_Flow</td>
</tr>
<tr>
<td>FTI_FlowEnd</td>
<td>The end of a flow</td>
<td>ID of an FO_Flow</td>
</tr>
<tr>
<td>FTI_PageBegin</td>
<td>The beginning of a page</td>
<td>ID of an FO_BodyPage, FO_HiddenPage, FO_MasterPage, FO_RefPage</td>
</tr>
<tr>
<td>FTI_PageEnd</td>
<td>The end of a page</td>
<td>ID of an FO_BodyPage, FO_HiddenPage, FO_MasterPage, FO_RefPage</td>
</tr>
<tr>
<td>FTI_TextFrameBegin</td>
<td>The beginning of a text frame</td>
<td>ID of an FO_TextFrame</td>
</tr>
<tr>
<td>FTI_TextFrameEnd</td>
<td>The end of a text frame</td>
<td>ID of an FO_TextFrame</td>
</tr>
<tr>
<td>FTI_SubColBegin</td>
<td>The beginning of a column</td>
<td>ID of an FO_SubCol</td>
</tr>
<tr>
<td>FTI_SubColEnd</td>
<td>The end of a column</td>
<td>ID of an FO_SubCol</td>
</tr>
<tr>
<td>FTI_FrameAnchor</td>
<td>An anchored frame</td>
<td>ID of an FO_AFrame</td>
</tr>
<tr>
<td>FTI_FnAnchor</td>
<td>A footnote</td>
<td>ID of an FO_Fn</td>
</tr>
<tr>
<td>FTI_TblAnchor</td>
<td>A table</td>
<td>ID of an FO_Tbl</td>
</tr>
</tbody>
</table>
### Text item type (dataType)

<table>
<thead>
<tr>
<th>Text item type (dataType)</th>
<th>What the text item represents</th>
<th>Text item data</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTI_MarkerAnchor</td>
<td>A marker</td>
<td>ID of an FO_Marker</td>
</tr>
<tr>
<td>FTI_XRefBegin</td>
<td>The beginning of a cross-reference instance</td>
<td>ID of an FO_XRef</td>
</tr>
<tr>
<td>FTI_XRefEnd</td>
<td>The end of a cross-reference instance</td>
<td>ID of an FO_XRef</td>
</tr>
<tr>
<td>FTI_VarBegin</td>
<td>The beginning of a variable instance</td>
<td>ID of an FO_Var</td>
</tr>
<tr>
<td>FTI_VarEnd</td>
<td>The end of a variable instance</td>
<td>ID of an FO_Var</td>
</tr>
<tr>
<td>FTI_TextInsetBegin</td>
<td>The beginning of a text inset</td>
<td>ID of an FO_TiApiClient, FO_TiFlow, FO_TiText, or FO_TiTextTable</td>
</tr>
<tr>
<td>FTI_TextInsetEnd</td>
<td>The end of a text inset</td>
<td>ID of an FO_TiApiClient, FO_TiFlow, FO_TiText, or FO_TiTextTable</td>
</tr>
<tr>
<td>FTI_ElementBegin</td>
<td>The beginning of a container element</td>
<td>ID of an FO_Element</td>
</tr>
<tr>
<td>FTI_ElementEnd</td>
<td>The end of a container element</td>
<td>ID of an FO_Element</td>
</tr>
<tr>
<td>FTI_ELEMPrefixBegin</td>
<td>The beginning of an element’s prefix</td>
<td>ID of an FO_Element</td>
</tr>
<tr>
<td>FTI_ELEMPrefixEnd</td>
<td>The end of an element’s prefix</td>
<td>ID of an FO_Element</td>
</tr>
<tr>
<td>FTI_ELEMSuffixBegin</td>
<td>The beginning of an element’s suffix</td>
<td>ID of an FO_Element</td>
</tr>
<tr>
<td>FTI_ELEMSuffixEnd</td>
<td>The end of an element’s suffix</td>
<td>ID of an FO_Element</td>
</tr>
<tr>
<td>FTI_CharPropsChange</td>
<td>A change in the text properties</td>
<td>Flags indicating which properties have changed (see the table below)</td>
</tr>
<tr>
<td>FTI_RubiCompositeBegin</td>
<td>The beginning of a rubi composite (and the beginning of oyamoji text).</td>
<td>ID of an FO_Rubi</td>
</tr>
<tr>
<td>FTI_RubiCompositeEnd</td>
<td>The end of a rubi composite.</td>
<td>ID of an FO_Rubi</td>
</tr>
<tr>
<td>FTI_RubiTextBegin</td>
<td>The beginning of rubi text (and the end of oyamoji text).</td>
<td>ID of an FO_Rubi</td>
</tr>
<tr>
<td>FTI_RubiTextEnd</td>
<td>The end of rubi text.</td>
<td>ID of an FO_Rubi</td>
</tr>
</tbody>
</table>
The following table lists the bit flags that a client can bitwise AND with the `idata` field of an `FTI_CharPropsChange` text item and the types of text property changes each flag indicates. For example, to determine if the font family changed, bitwise AND the `FTF_FAMILY` flag with the `idata` field.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FTF_FAMILY</code></td>
<td>The font family has changed.</td>
</tr>
<tr>
<td><code>FTF_VARIATION</code></td>
<td>The font variation has changed.</td>
</tr>
<tr>
<td><code>FTF_WEIGHT</code></td>
<td>The font weight has changed.</td>
</tr>
<tr>
<td><code>FTF_ANGLE</code></td>
<td>The font angle has changed.</td>
</tr>
<tr>
<td><code>FTF_UNDERLINING</code></td>
<td>The underlining has changed.</td>
</tr>
<tr>
<td><code>FTF_STRIKETHROUGH</code></td>
<td>The strikethrough characteristic has changed.</td>
</tr>
<tr>
<td><code>FTF_OVERLINE</code></td>
<td>The overline characteristic has changed.</td>
</tr>
<tr>
<td><code>FTF_CHANGEBAR</code></td>
<td>The change bars have changed.</td>
</tr>
<tr>
<td><code>FTF_OUTLINE</code></td>
<td>The outline characteristic has changed.</td>
</tr>
<tr>
<td><code>FTF_SHADOW</code></td>
<td>The shadow characteristic has changed.</td>
</tr>
<tr>
<td><code>FTF_PAIRKERN</code></td>
<td>The pair kerning has changed.</td>
</tr>
<tr>
<td><code>FTF_SIZE</code></td>
<td>The font size has changed.</td>
</tr>
<tr>
<td><code>FTF_KERNX</code></td>
<td>The kern-x characteristic has changed.</td>
</tr>
<tr>
<td><code>FTF_KERNY</code></td>
<td>The kern-y characteristic has changed.</td>
</tr>
<tr>
<td><code>FTF_SPREAD</code></td>
<td>The font spread has changed.</td>
</tr>
<tr>
<td><code>FTF_COLOR</code></td>
<td>The color has changed.</td>
</tr>
<tr>
<td><code>FTF_CHARTAG</code></td>
<td>The Character Catalog format has changed.</td>
</tr>
<tr>
<td><code>FTF_CAPITALIZATION</code></td>
<td>The capitalization has changed.</td>
</tr>
<tr>
<td><code>FTF_POSITION</code></td>
<td>The character position has changed.</td>
</tr>
<tr>
<td><code>FTF_CONDITIONTAG</code></td>
<td>The condition tag has changed.</td>
</tr>
<tr>
<td><code>FTF_STRETCH</code></td>
<td>Font stretch value has changed</td>
</tr>
<tr>
<td><code>FTF_LANGUAGE</code></td>
<td>Character language has changed</td>
</tr>
<tr>
<td><code>FTF_TSUME</code></td>
<td>Tsume setting has changed</td>
</tr>
<tr>
<td><code>FTF_IIF</code></td>
<td>An internal flag having to do with asian text. input. If there is a non-zero value for this flag, a front end processor is controlling that text; you should not modify the associated text item.</td>
</tr>
<tr>
<td><code>FTF_ENCODING</code></td>
<td>The text encoding has changed.</td>
</tr>
</tbody>
</table>
Figure 14.19 shows a paragraph and the text items the API uses to represent the paragraph’s text.

There are several important things to note about the text items shown in Figure 14.19:

- Because the string "This " and the string "is a marker." have different font weights, there are separate text items for them.
- The FTI_CharPropsChange text item indicates that the text properties have changed; the FTF_WEIGHT flag that it specifies indicates that the font weight has changed.
- The marker anchor is counted in the offset.

**How the API represents special characters**

The API uses the FrameMaker product character set. Some characters are either reserved by the C language or belong to the higher ASCII range. To represent these characters in a string, use octal (\) or hexadecimal (\x) sequences.

<table>
<thead>
<tr>
<th>Character</th>
<th>Hexadecimal representation</th>
<th>Octal representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>\x3e</td>
<td>\76</td>
</tr>
<tr>
<td>&quot; (straight double quotation mark)</td>
<td>\x22</td>
<td>\42</td>
</tr>
<tr>
<td>“ (left curved quotation mark)</td>
<td>\xd2</td>
<td>\322</td>
</tr>
<tr>
<td>” (right curved quotation mark)</td>
<td>\xd3</td>
<td>\323</td>
</tr>
</tbody>
</table>

For a complete list of the characters in the FrameMaker product character set and the corresponding hexadecimal codes, see your Frame product user’s manual. If you are using
ANSI C, you can use these hexadecimal codes or their octal equivalents. If you are not using ANSI C, you must use octal (\) sequences.

Suppose you want to represent the following text in the API:

This is an em dash —

If you are not using ANSI C, you must specify the string This is an em dash \321. If you are using ANSI C, you can also specify the string This is an em dash \xd1.

**How the API represents text properties**

The FTI_CharPropsChange text item only indicates that particular text properties *have* changed. It does not indicate what they have changed to. The API provides a function named F_ApiGetTextProps(), which allows you to retrieve the text properties for individual characters in text. You cannot retrieve the text properties for a range or selection of text, because they may be different for individual characters within the selection. You can, however, set the text properties for a range of text. For examples of how to get and set text properties, see “Getting and setting text formatting” on page 123.

Suppose you retrieve the text properties at the insertion point shown in Figure 14.20.

**Figure 14.20 Text containing insertion point**

The following are some of the properties of the text at the insertion point.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_CharTag</td>
<td>StringT</td>
<td>booktitle</td>
</tr>
<tr>
<td>FP_FontFamily</td>
<td>IntT</td>
<td>Index of Times font (for more information on font name indexes, see “How the API indicates which fonts are available in a session” on page 276)</td>
</tr>
<tr>
<td>FP_FontWeight</td>
<td>IntT</td>
<td>Index of Bold font weight</td>
</tr>
<tr>
<td>FP_FontAngle</td>
<td>IntT</td>
<td>Index of Regular font angle</td>
</tr>
<tr>
<td>FP_InCond</td>
<td>F_IntsT</td>
<td>NULL</td>
</tr>
</tbody>
</table>

If the conditions Comment and MyComment are applied to the text location, the FP_InCond property specifies an F_IntsT structure with the following values:

len: 2
val: { Comment_ID, MyComment_ID }

where Comment_ID and MyComment_ID are the IDs of the FO_CondFmt objects that represent the Comment and MyComment condition formats.
How the API represents hidden conditional text

The user can choose to hide all the text with a specified condition format. If a document has hidden conditional text, the FrameMaker product automatically adds a hidden page to it. This hidden page is completely invisible to the user. It has a single flow, named HIDDEN.

When the user chooses to hide text with a condition format, the FrameMaker product removes each block of text with that condition format and inserts a Conditional Text marker (type 10) in its place. This marker text consists of a plus sign (+) and a five-digit integer. The FrameMaker product places the blocks of hidden text in the HIDDEN text flow on the hidden page. The text begins with a Conditional Text marker containing a minus sign (−) and the integer. It ends with another Conditional Text marker containing an equal sign (=) and the integer. If the hidden conditional text doesn’t span paragraphs, it appears in one paragraph. If the hidden conditional text spans paragraphs, each paragraph of conditional text constitutes a separate paragraph in the HIDDEN flow.

Suppose you have a body page with some conditional text and some unconditional text. The condition tag’s format overrides specify that the text appears underlined, as shown in Figure 14.21.

If you hide the text, the body page appears as shown in Figure 14.22.

If you could see the hidden page and the text in the HIDDEN flow, it would appear as shown in Figure 14.23. The numbers in the markers that represent the hidden conditional text on the body page correspond to the numbers in the markers on the hidden page.

FIGURE 14.21 Body page with conditional and unconditional text

FIGURE 14.22 Body page with the conditional text hidden

FIGURE 14.23 Hidden conditional text on the hidden page
Markers

Markers are anchored objects that store data and associate that data with specific locations in the text. Various features in FrameMaker may refer to a marker, or you can use markers to store data for your FDK clients.

What the user sees

You can use markers to mark entries for a table of contents or an index. A marker’s position in text is indicated by a marker symbol. A marker contains text, which appears in the Marker window when you select the marker and choose Marker from the Special menu.

Any number of marker types can be defined for a document; 11 of them are predefined by the FrameMaker product as a standard list of marker types, and the others are defined by the user. The list of defined marker types is saved with the document.

How the API represents markers

The API represents each marker with the following:

- An FTI_MarkerAnchor text item that specifies the ID of an FO_Marker object
- An FO_Marker object

FO_Marker properties provide the following information:

- The marker type; the Id of an FO_MarkerType object
- The text the marker contains
- The ID of the next FO_Marker object in the document
- The location of the marker in text
- The element ID of the marker, if it is a structured marker in a FrameMaker document
- If included, the number of a marker type in versions earlier than 5.5; when opening the document in FrameMaker 5.5, this maps the old numbered marker type to the new named marker type

FO_MarkerType properties provide the following information:

- The ID of the next FO_MarkerType object in the document
- The name of the marker type, as it appears in the user interface
- The internal name of the marker type (usually the same as the name that appears in the user interface)
- If included, the number of a marker type in versions earlier than 5.5; when opening the document in FrameMaker 5.5, this maps the old numbered marker type to the new named marker type
- Whether the marker type appears in the user interface, whether it is saved with the document, and whether the marker type can be deleted
The **FO_Doc** property, **FP_MarkerTypeNames**, specifies an **F_StringsT** structure which provides the list of marker types available in the current document. The document object also has an **FP_FirstMarkerTypeInDoc** property as an entry into the document’s list of marker types.

Given a marker type name, you can use **F_ApiGetNamedObject()** to get the ID of the associated **FO_MarkerType**. The following code returns the ID of the index marker type:

```c
F_ObjHandleT docId, markerId;

/* Get ID of the active document. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

markerId = F_ApiGetNamedObject(docId, FO_MarkerType, (StringT) "Index");
```

**Figure 14.24** shows an index marker anchor and the text item that represents it.

---

![Marker Anchor with Text Item](image)

**Figure 14.24** A marker anchor and the text item that represents it

The following table lists some of the properties of the **FO_Marker** object specified by **FTI_MarkerAnchor** in **Figure 14.24**.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_MarkerTypeId</td>
<td>F_ObjHandleT</td>
<td>ID of the FO_MarkerType for &quot;Index&quot;</td>
</tr>
<tr>
<td>FP_MarkerText</td>
<td>StringT</td>
<td>presidents:past</td>
</tr>
<tr>
<td>FP_NextMarkerInDoc</td>
<td>F_ObjHandleT</td>
<td>ID of the next FO_Marker object in the document</td>
</tr>
</tbody>
</table>
Adding marker types to documents

To add a marker type to a document, use `F_ApiNewNamedObject()`. Once you have the new marker type’s ID, you can set any properties that you want to be different from the default values.

```c
F_ObjHandleT docId, myMarkerTypeId;

/* Get ID of the active document. */
docId = F_ApiGetId(0, FV_SessionId, FP_ActiveDoc);

/* Create the marker type. */
myMarkerTypeId = F_ApiNewNamedObject(docId, FO_MarkerType,
                                      (StringT) "MyMarkerType");
```

To delete a marker type from a document, pass the document ID and the marker type ID to `F_ApiDelete()`.

Mapping old marker types to named marker types

In versions of FrameMaker earlier than 5.5, marker types were identified by number. Type 1 through Type 10 were reserved for FrameMaker, and given specific names; Header/Footer $1, Header/Footer $2, etc. Type 11 through Type 25 were for custom marker types.

`FO_MarkerType` and `FO_Marker` objects have an `FP_OldTypeNum` property that maps the named marker type to what was a numbered marker type in earlier documents. In this way, you can ensure that your client handles markers in legacy data the way you want.

For example, assume your client adds a marker type named MyMarkerType to a document, and you set the `FP_OldTypeNum` property of MyMarkerType to 11. The user might import or paste text from an older document into the document with MyMarkerType. If the older text has markers of type 11 in it, they will come into the new document as MyMarkerType.

The standard list of marker types

Every document includes a set of required marker types; Header/Footer $1, Header/Footer $2, Index, Comment, Subject, Author, Glossary, Equation, Hypertext Cross-Ref, and Conditional Text. These are required markers, and cannot be deleted.

You can add an existing public marker type to the standard list by setting the name string to the `FP_AddMarkerTypeToStandardMarkers` property of the current session object. Once you add a marker type to this list, it remains for the entire session; you must quit the session to remove it.

```c
F_ApiSetInt(0, FV_SessionId, FP_OldTypeNum, (IntT) 17);
F_ApiSetString(0, FV_SessionId,
               FP_AddMarkerTypeToStandardMarkers, (StringT) "MyMarkerType");
```
This example first sets a session integer for FP_OldTypeNum to 17. This ensures that for the rest of the current session, markers of type 17 (from earlier documents) will come into new documents as markers of type MyMarkerType.

If the the FP_OldTypeNum you specify is taken, your new marker type will not be added to the list of standard marker types. To confirm that your marker type was added to the standard list, get the FP_MarkerNames property from the FV_SessionId object.

It’s possible for the FP_OldTypeNum you specified to be taken; another API client may have already used that value when assigning a marker type to the standard list. For example, HTML export in FrameMaker 5.5 is performed by a client that adds the HTML Macro marker type to the standard list. The value of that marker’s FP_OldTypeNum is 11. After that client is initialized, no other clients can use the same value for FP_OldTypeNum when assigning a marker to the standard list.

### Cross-reference formats

When you insert a cross-reference in a document, you choose a cross-reference format that specifies the information provided by the cross-reference.

### What the user sees

A cross-reference format consists of a combination of text and cross-reference building blocks. Cross-reference building blocks are variables that provide information, such as the current page number or filename.

Each document has a catalog of cross-reference formats. You can add or delete formats from this catalog.

### How the API represents cross-reference formats

FrameMaker represents each cross-reference format with an FO_XRefFmt object, whose properties provide the following information:

- The name of the cross-reference format
- A string containing the cross-reference’s text and cross-reference building blocks
- The ID of the next FO_XRefFmt object in the document

Suppose you create a cross-reference format named See Heading & Page as shown in Figure 14.25.
The following table lists some properties of the FO_XRefFmt object that represents this cross-reference format.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_Name</td>
<td>StringT</td>
<td>See Heading &amp; Page</td>
</tr>
<tr>
<td>FP_Fmt</td>
<td>StringT</td>
<td>See '&lt;$paratext&gt;' on page'&lt;$pagenum&gt;.</td>
</tr>
<tr>
<td>FP_NextXRefFmtInDoc</td>
<td>F_ObjHandleT</td>
<td>ID of the next FO_XRefFmt object in the document</td>
</tr>
</tbody>
</table>

### Cross-references

A cross-reference refers to a specific location, known as a *source*, within the current document or another document. The source can be either a cross-reference marker (a type 9 marker) or a unique string of text.

### What the user sees

When you insert a cross-reference, you select a cross-reference format, which specifies the information provided by the cross-reference. For more information on cross-reference formats, see “Cross-reference formats” on page 322.

### How the API represents cross-reference instances

The API represents each cross-reference instance with the following:

- FTI_XRefBegin and FTI_XRefEnd text items that specify the ID of the FO_XRef object
Frame Document Architecture

- An FTI_String text item, which provides the text that appears where the cross-reference is inserted
- An FO_XRef object

FO_XRef properties provide the following information:
- The ID of an FO_XRefFmt object
- The ID of the next FO_XRef object in the document
- The name of the file in which the cross-reference source is located
- The element ID of the cross-reference, if it is in a structured flow in a FrameMaker document

Suppose you insert the cross-reference shown in Figure 14.26, using the See Heading & Page cross-reference format shown in Figure 14.25.

![Figure 14.26 A cross-reference and the text items that represent it](image)

The following are some properties of the FO_XRef object that represents the cross-reference.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_XRefFmt</td>
<td>F_ObjHandleT</td>
<td>ID of the FO_XRefFmt object representing the See Heading &amp; Page cross-reference format</td>
</tr>
<tr>
<td>FP_NextXRefInDoc</td>
<td>F_ObjHandleT</td>
<td>ID of the next FO_XRef object in document</td>
</tr>
<tr>
<td>FP_XRefFile</td>
<td>StringT</td>
<td>An empty string (&quot;&quot;)</td>
</tr>
</tbody>
</table>

**Variable formats**

The user can insert variables that represent a variety of information, such as the page number or the date, into text. The information a variable provides is specified by a variable format.

**What the user sees**

Each variable format can specify a combination of text and building blocks. Building blocks are FrameMaker product-defined variables that you can string together.
There are six principal classes of variable formats:
- Page number
- Date
- Filename
- Table
- Running header or footer
- User

Each of these classes has a unique set of building blocks. You cannot use a building block from one class in another class. For example, you cannot use a date building block in a page number variable format.

**How the API represents variable formats**

FrameMaker represents each variable format with an `FO_VarFmt` object, whose properties provide the following information:
- The name of the variable format
- The list of building blocks and text strings
- The type of variable it is (for example, page count or user variable)
- ID of the next `FO_VarFmt` object in the document

Suppose you edit the Creation Date (Long) variable format so that its definition is as shown in Figure 14.27.

![Figure 14.27 Creation Date (Long) variable definition](image)

The following are some properties of the `FO_VarFmt` object that represents the Creation Date (Long) variable format.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_Fmt</td>
<td>StringT</td>
<td>&lt;$monthname&gt;, &lt;$daynum&gt;, &lt;$year&gt;</td>
</tr>
<tr>
<td>FP_SystemVar</td>
<td>IntT</td>
<td>FV_VAR_CREATION_DATE_LONG</td>
</tr>
</tbody>
</table>
Variables

The user can insert variables in text. There are some restrictions on inserting some variable formats. For example, you can insert current page number, running header, and running footer variables only in an untagged flow on a master page.

What the user sees

The information an instance of a variable provides depends on its variable format. For example, if a variable’s format is Page Count and the current document has 27 pages, each time the variable occurs in text, it appears as 27.

How the API represents instances of variables

The API represents each variable instance with the following:

- An `FTI_VarBegin` text item and an `FTI_VarEnd` text item that specify the ID of an `FO_Var` object
- An `FTI_String` text item that provides the text that appears where the variable is inserted
- An `FO_Var` object

`FO_Var` properties provide the following information:

- The ID of an `FO_VarFmt` object
- ID of the next `FO_Var` object in the document
- The element ID of the variable, if it is a structured variable in a FrameMaker document

Figure 14.28 shows an instance of the Creation Date (Long) variable and the text items that represent it.
The following table lists some of the properties of the \texttt{FO_Var} object specified by \texttt{FTI_VarBegin} and \texttt{FTI_VarEnd} in Figure 14.28.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{FP_VarFmt}</td>
<td>\texttt{F_ObjHandleT}</td>
<td>ID of the \texttt{FO_VarFmt} object that represents the Creation Date (Long) variable format</td>
</tr>
<tr>
<td>\texttt{FP_NextVarInDoc}</td>
<td>\texttt{F_ObjHandleT}</td>
<td>ID of the next \texttt{FO_Var} object in the document</td>
</tr>
</tbody>
</table>

**Footnotes**

A footnote is a type of special text column that appears at the bottom of a page.

**What the user sees**

A footnote reference (or anchor) appears in the main text as a number, letter, or special character. A footnote is visually separated from the main text by a separator (usually a horizontal line).

The Footnote Properties dialog box allows you to change characteristics that apply to all the footnotes in a document, such as the type of numbering or special characters used to represent the anchor and the height of the footnote column.

**How the API represents footnotes**

When the user chooses the Footnote command, the FrameMaker product inserts a footnote anchor. It also creates a text frame with invisible borders at the bottom of the text frame in which the footnote was inserted. The user types the footnote text into the footnote text frame.

Characteristics, such as the footnote anchor’s numbering type, are represented as document properties because they apply to all the footnotes in a document and not just individual footnote instances. For more information on the document properties that govern footnote characteristics, see “How the API represents documents” on page 280.

The API represents each footnote anchor with an \texttt{FTI_FnAnchor} text item, which specifies the ID of the \texttt{FO_Fn} object that represents the footnote. \texttt{FO_Fn} properties provide the following information:

- The footnote number
- The ID of the text frame in which the footnote text appears
- The ID of the next footnote in the list of footnotes in the document
- The IDs of the first and last paragraphs containing the footnote’s text
- The element ID of the footnote, if it is a structured footnote in a FrameMaker document
Figure 14.29 shows a footnote and the text item that represents it.

![Footnote example]

The footnote text is actually in its own text column.\$1. Which appears down here.\$

**Figure 14.29** A footnote and the text item that represents it

The following table lists the properties of the FO_Fn object specified by the FTI_FnAnchor text item in Figure 14.29.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_InTextObj</td>
<td>F_ObjHandleT</td>
<td>The ID of the subcolumn (FO_SubCol) in which the footnote appears.</td>
</tr>
<tr>
<td>FP_FnNum</td>
<td>IntT</td>
<td>0.</td>
</tr>
<tr>
<td>FP_PrevFn</td>
<td>F_ObjHandleT</td>
<td>ID of the previous footnote in the text frame (in this case, NULL).</td>
</tr>
<tr>
<td>FP_NextFn</td>
<td>F_ObjHandleT</td>
<td>ID of the next footnote in the text frame (in this case, NULL).</td>
</tr>
<tr>
<td>FP_NextFnInDoc</td>
<td>F_ObjHandleT</td>
<td>ID of the next footnote in the document.</td>
</tr>
<tr>
<td>FP_FirstPgf</td>
<td>F_ObjHandleT</td>
<td>ID of the first paragraph (FO_Pgf) in the footnote.</td>
</tr>
<tr>
<td>FP_LastPgf</td>
<td>F_ObjHandleT</td>
<td>ID of the last paragraph (FO_Pgf) in the footnote (in this example, it is the same as the one specified by FP_FirstPgf).</td>
</tr>
</tbody>
</table>

Although FP_FnNum specifies an integer, the number that appears in the document can be one of several ordinal or special characters. For example, if you set the document’s FP_FnNumStyle property to FV_FN_NUM_ALPHA_UC, an A would appear instead of the I in the body text and at the beginning of the footnote.

To get all the paragraphs in a footnote, you traverse the FP_NextPgfInFlow and FP_PrevPgfInFlow properties, just as you would to get the paragraphs in any other flow.

---

**Ruling Formats**

Each document has a Ruling Catalog containing several ruling formats.
What the user sees

Rulings are the lines that border a table cell or an entire table. A ruling format specifies a line type (such as Thin or Thick) and the gap between the line and the cell contents.

You can specify rulings for an entire table in the Table Designer or for individual table cells in the Custom Ruling and Shading dialog box. FrameMaker provides default rulings, such as Thick and Thin. You can change these rulings or create your own.

How the API represents ruling formats

The API represents a ruling format with an `FO_RulingFmt` object, whose properties provide the following information:

- The name of the ruling format
- Its line width
- The gap between lines if the ruling specifies double lines
- The ID of the next `FO_RulingFmt` object in the document

Suppose you create the Medium ruling format shown in Figure 14.30.

The API represents this ruling format with an `FO_RulingFmt` object. The following table lists some of its properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_Name</td>
<td>StringT</td>
<td>Medium</td>
</tr>
<tr>
<td>FP_RulingPenWidth</td>
<td>MetricT</td>
<td>2*pts</td>
</tr>
<tr>
<td>FP_RulingGap</td>
<td>MetricT</td>
<td>0</td>
</tr>
<tr>
<td>FP_RulingLines</td>
<td>IntT</td>
<td>2</td>
</tr>
</tbody>
</table>

How the API organizes ruling formats

The API organizes the `FO_RulingFmt` objects in a document in a linked list. The `FO_Doc` property, `FP_FirstRulingFmtInDoc`, specifies the first `FO_RulingFmt` object in the list. Each `FO_RulingFmt` object has an `FP_NextRulingFmtInDoc` property, which specifies
the ID of the next \texttt{FO\_RulingFmt} object in the list. The order of the list does \textit{not} correspond to the order in which the formats appear in the Custom Ruling and Shading dialog box.

---

**Table Catalog formats**

Each document has a Table Catalog containing table formats.

**What the user sees**

When you create a new table, you specify a format from the Table Catalog. The format provides the following information:

- The Table Catalog format name
- Format characteristics, such as the table position, alignment, and rulings
- The number of columns and rows

If you tag an existing table with a Table Catalog format, the Table Catalog format provides only the format name and the format characteristics for the table; it does not affect the number of columns or rows.

After you have created a new table or tagged an existing table, you can change the number of columns or rows or the format without affecting the Table Catalog tag. You can also instruct the FrameMaker product to apply the changes to the Table Catalog tag and other tables tagged with the format. You can modify or delete formats that are already in the Table Catalog, or you can add new formats.

**How the API represents Table Catalog formats**

FrameMaker represents each Table Catalog format with an \texttt{FO\_TblFmt} object, whose properties provide the following information:

- The name of the table format
- Format characteristics
- The default number of initial columns and rows
- The ID of the next \texttt{FO\_TblFmt} object in the document

Suppose you create the Table Catalog format described in the Table Designer and the Insert Table dialog box in Figure 14.31.
The following table lists some of the properties of the `FO_TblFmt` object that represents Format A.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_Name</td>
<td>StringT</td>
<td>Format A</td>
</tr>
<tr>
<td>FP_TblInitNumCols</td>
<td>IntT</td>
<td>5</td>
</tr>
<tr>
<td>FP_TblInitNumHRows</td>
<td>IntT</td>
<td>1</td>
</tr>
<tr>
<td>FP_TblColRuling</td>
<td>F_ObjHandleT</td>
<td>ID of <code>FO_RulingFmt</code> object that represents the Thin line ruling</td>
</tr>
<tr>
<td>FP_TblBodyRowRuling</td>
<td>F_ObjHandleT</td>
<td>ID of <code>FO_RulingFmt</code> object that represents the Thin line ruling</td>
</tr>
<tr>
<td>FP_TblHFSeparatorRuling</td>
<td>F_ObjHandleT</td>
<td>ID of <code>FO_RulingFmt</code> object that represents the Double line ruling</td>
</tr>
</tbody>
</table>

**How the API organizes Table Catalog formats**

The API organizes the formats in the Table Catalog in a linked list. The `FO_Doc` object property, `FP_FirstTblFmtInDoc`, specifies the ID of the first `FO_TblFmt` object in the list. Each `FO_TblFmt` object has an `FP_NextTblFmtInDoc` property, which specifies the ID of the next `FO_TblFmt` object in the list. The order of the list does not correspond to the order in which the formats appear in the Table Designer.
Tables

FrameMaker allows you to insert tables into text. When you insert a table, a table anchor symbol (⊥) appears on the screen at the point where you inserted it.

What the user sees

Tables are useful for organizing information in cells arranged in rows and columns. Tables can have titles and heading, body, and footing rows. FrameMaker automatically repeats table titles and heading and footing rows on each page of a table.

Each cell in a table is actually a type of text frame. It can contain text and nearly anything you insert in text, such as an anchored frame or a marker. You cannot insert another table directly into a table cell.

Like a paragraph, each table has a tag and a format. The tag is the name of a Table Catalog format. A table format specifies the layout characteristics of a table, such as its position in a text frame, its alignment, and the rulings and shadings of its columns and rows. The table format can specify different rulings for the different types of rows (for example, the body, heading, and footing rows).

You can override a Table Catalog format by changing an individual table’s format. You can also override an individual table’s format by specifying a custom ruling, shading, or color for an individual cell or set of cells in the table. If you retag a table with a Table Catalog format after you have specified custom ruling for some of its cells, it does not affect the custom ruling for those cells.

How the API represents tables

The API represents a table anchor with an FTI_TblAnchor text item. For more information on text items, see “How the API represents text” on page 312. Each FTI_TblAnchor text item specifies the ID of an FO_Tbl object.

The API represents the table itself with the following objects:

- An FO_Tbl object
- One or more FO_Row objects
- One or more FO_Cell objects

FO_Tbl and table formats

FO_Tbl properties provide the following information:

- The table format tag (name)
- Formatting (such as alignment and rulings)
- The number of columns and rows
- The ID of the paragraph in the table’s title
● IDs of FO_Row objects that represent the first and last rows in the table
● ID of the next FO_Tbl object in the document
● The element IDs of the table, table title, table heading, table body, table footing elements if the table is a structured table in a FrameMaker document

FO_Tbl formatting properties are the same as FO_TblFmt formatting properties, except they do not include properties (such as FP_TblInitNumHRows) that specify the initial numbers of rows or columns.

The table title

If a table has a title, the FO_Tbl properties, FP_FirstPgf and FP_LastPgf, specify the IDs of the first and last FO_Pgf objects in the title.

Rows

The API represents each row in a table with an FO_Row object. FO_Row properties provide the following information about a table row:
● Its type (heading, body, or footing)
● Whether it is kept with the previous row, the next row, or both when a page break occurs within the table
● Its maximum and minimum allowable height
● The IDs of the FO_Row objects that represent the rows before and after it in the table
● The ID of the FO_Cell object that represents the first (leftmost) cell in the row
● The conditions that apply to the row
● The element ID of the row, if it is a structured row in a FrameMaker document

If a row has conditions applied to it, its FP_InCond property specifies an F_IntsT structure that includes the IDs of the FO_CondFmt objects that represent the conditions.

Cells

The API represents each cell in a table with an FO_Cell object, whose properties provide the following information:
● The ruling and shading that the cell inherits from the table format
● Custom ruling and shading
● Flags that indicate whether the cell’s custom shading and fill override the table’s shading and fill
● IDs of sibling FO_Cell objects
● IDs of the first and last paragraphs in the cell
● The element ID of the cell, if it is a structured cell in a FrameMaker document

The FO_Cell object properties, FP_FirstPgf and FP_LastPgf, specify the IDs of the first and last paragraphs in the cell. If there is more than one paragraph in the cell, each paragraph’s FP_PrevPgfInFlow and FP_NextPgfInFlow properties specify the IDs of the paragraphs before and after it.
FO_Cell objects have two properties for each ruling. For example, the properties for the top ruling are FP_CellDefaultTopRuling and FP_CellOverrideTopRuling. The default ruling is the ruling that the cell inherits from the FO_Tbl object that contains it. For example, the FP_CellDefaultTopRuling property for a cell in a body row inherits the value of the FP_TblBodyRowRuling property in the FO_Tbl object that contains it. An override ruling is a ruling that the user specifies in the Custom Ruling and Shading dialog box for an individual cell. If a value is specified for an override ruling, it overrides the default ruling.

The FO_Cell properties, FP_CellOverrideShading and FP_CellOverrideFill, specify the cell’s custom shading and fill. If the cell’s custom shading and fill override the table’s default shading and fill, FP_CellUseOverrideFill and FP_CellUseOverrideShading are True.

How the API organizes the objects that represent tables

Figure 14.32 shows a table anchor, a table, and the text item that represents the anchor.

![Figure 14.32 A table and the text item that represents its anchor](image)

The API represents the table with the objects shown in Figure 14.33.
Figure 14.33 Objects that represent a table
The following table lists some of the FO_Tbl object’s properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_TblTag</td>
<td>StringT</td>
<td>Format A</td>
</tr>
<tr>
<td>FP_TblTopRuling</td>
<td>F_ObjHandleT</td>
<td>NULL</td>
</tr>
<tr>
<td>FP_TblHFSeparatorRuling</td>
<td>F_ObjHandleT</td>
<td>ID of FO_RulingFmt that represents Double line</td>
</tr>
<tr>
<td>FP_TblBodyRowRuling</td>
<td>F_ObjHandleT</td>
<td>ID of FO_RulingFmt that represents Thin line</td>
</tr>
<tr>
<td>FP_TblBodyFirstFill</td>
<td>IntT</td>
<td>0</td>
</tr>
<tr>
<td>FP_FirstRowInTbl</td>
<td>F_ObjHandleT</td>
<td>ID of FO_Row that represents the heading row (row 1)</td>
</tr>
<tr>
<td>FP_LastRowInTbl</td>
<td>F_ObjHandleT</td>
<td>ID of FO_Row that represents the footing row (row 3)</td>
</tr>
</tbody>
</table>

The following are some of the properties of the FO_Row object that represents the table’s heading row.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_PrevRowInTbl</td>
<td>F_ObjHandleT</td>
<td>NULL</td>
</tr>
<tr>
<td>FP_NextRowInTbl</td>
<td>F_ObjHandleT</td>
<td>ID of FO_Row that represents row 2</td>
</tr>
<tr>
<td>FP_RowType</td>
<td>IntT</td>
<td>FV_ROW_HEADING</td>
</tr>
<tr>
<td>FP_RowKeepWithNext</td>
<td>IntT</td>
<td>True</td>
</tr>
<tr>
<td>FP_FirstCellInRow</td>
<td>F_ObjHandleT</td>
<td>ID of FO_Cell that represents the R1-C1 cell</td>
</tr>
</tbody>
</table>

The following are some properties of the FO_Cell object that represents the R2-C2 cell. For the cell’s override fill and rulings to override the fill and ruling provided by the table’s format, the FP_CellUseOverrideCharacteristic properties must be set to True.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_CellOverrideFill</td>
<td>IntT</td>
<td>5</td>
</tr>
<tr>
<td>FP_CellUseOverrideFill</td>
<td>IntT</td>
<td>True</td>
</tr>
<tr>
<td>FA_CellDefaultLeftRuling</td>
<td>F_ObjHandleT</td>
<td>ID of FO_RulingFmt that represents the Medium line ruling</td>
</tr>
<tr>
<td>FP_CellOverrideLeftRuling</td>
<td>F_ObjHandleT</td>
<td>ID of FO_RulingFmt that represents the Thick line ruling</td>
</tr>
</tbody>
</table>
How the API represents straddle table cells

When the user straddles a set of table cells, the FrameMaker product links all of the paragraphs in the cells. It changes the FP_FirstPgf and FP_LastPgf properties of the first cell (topmost and leftmost) so that it specifies the first and last paragraphs of the new linked list of paragraphs. All the other cells specify paragraph IDs of zero. It changes the properties of the first FO_Cell object in the straddle as listed in the following table.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_CellDefaultBottomRuling</td>
<td>F_ObjHandleT</td>
<td>ID of FO_RulingFmt that represents the Medium line ruling</td>
</tr>
<tr>
<td>FP_CellOverrideBottomRuling</td>
<td>F_ObjHandleT</td>
<td>ID of FO_RulingFmt that represents the Thick line ruling</td>
</tr>
</tbody>
</table>

The straddle uses the custom rulings and shadings of the first cell. When the user unstraddles the cells, the FrameMaker product leaves all the paragraphs that were in the straddle in the first cell. It gives each other cell a new empty paragraph. It leaves the original custom rulings and shadings of each cell intact.

Suppose you straddle both cells in a table row that has two cells. Each cell contains a single paragraph before you straddle them. Figure 14.34 shows how the FO_Cell objects appear before and after they are straddled and unstraddled. The FrameMaker product automatically inserts a new paragraph in the second cell after you unstraddle the cells.
You can assign spot colors to text and objects in a document, and you can print process color separations. You can also set up color views to specify which colors are visible in a document.
What the user sees

FrameMaker provides a set of default colors. You can also define your own colors and store them in the document’s Color Catalog. The FrameMaker product provides three color models for creating your own colors: CMYK, RGB, and HLS. It also provides eight color libraries; Crayon, DIC, FOCOLTONE, Greys, MUNSELL, PANTONE®®, TOYO, and TRUMATCH.

How the API represents colors

The API represents each default color and each user-defined color with an FO_Color object. Tints are special FO_Color objects because each tint includes a reference to a base color, which is itself an FO_Color object. For a tint, the FP_TintBaseColor property returns the object ID of the base FO_Color object.

Paragraph formats, graphic objects, and other objects to which you can apply a color have properties that specify the ID of an FO_Color object. For example, all graphic objects have an FP_Color property that specifies the ID of an FO_Color object.

The properties of FO_Color objects provide the following information:

- The name of the color
- The color library and associated pigment used for the color
- The CMYK values of the color
- Tint base color and percentage
- Whether the color overprints or knocks out when printing

Library colors, FP_FamilyName, and FP_InkName

When you specify a color from a library, the FP_Cyan, FP_Magenta, FP_Yellow, and FP_Black properties are set to represent the library color. If you later set FP_FamilyName and FP_InkName to NULL, the CMYK settings remain the same, but the FO_Color object no longer defines a library color.

Both FP_FamilyName and FP_InkName are required to uniquely define a library color. The order in which you set the values of these properties is important. You must set a valid value for FP_FamilyName before you set FP_InkName. If you try to set the ink name when the family name is set to NULL, F_ApiSetString() returns an error of FE_NoColorFamily.

When you set a value for FP_FamilyName, two things can happen:

- If the current value for FP_InkName specifies a valid ink for the newly set FP_FamilyName, then FP_InkName does not change.
- If the current value for FP_InkName does not specify a valid ink for the newly set FP_FamilyName, then the value for FP_InkName automatically changes to the first ink name for the new color family.

If you set FP_FamilyName to a color family that is not installed on your system, F_ApiSetString() returns an error of FE_BadFamilyName. If you set FP_InkName to a
name that is not included in the current family, F_ApiSetString() returns an error of FE_BadInkName.

If you set one of either FP_FamilyName or FP_InkName to NULL, then the other property value automatically changes to NULL.

**Formal color library names and ink names**

Note that you must specify the family name as the formal color library name, including the registered trademark symbol. For example, the following sets the color to MUNSELL® Book of Color; note the code (\xa8) for the “®” character.

F_ApiSetString(docId, baseId, FP_FamilyName, "MUNSELL\xa8 Book of Color");

When specifying an ink name, you don’t necessarily provide the full ink name as described in the color library’s reference material. Some ink names have prefixes or suffixes that are not used by the API.

The following table lists the formal name for each color library that FrameMaker supports, along with an example of a legal string to specify an ink name via the FDK:

<table>
<thead>
<tr>
<th>Color library name</th>
<th>Ink name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crayon</td>
<td>Apricot</td>
</tr>
<tr>
<td>DIC COLOR GUIDE SPOT</td>
<td>2298p*</td>
</tr>
<tr>
<td>FOCOLTONE</td>
<td>1070</td>
</tr>
<tr>
<td>Greys</td>
<td>49% Grey.prcs</td>
</tr>
<tr>
<td>MUNSELL® High Chrome Colors</td>
<td>2.5R 7:10</td>
</tr>
<tr>
<td>MUNSELL® Book of Color</td>
<td>2.5R 9:1</td>
</tr>
<tr>
<td>PANTONE® Coated</td>
<td>Yellow 012</td>
</tr>
<tr>
<td>PANTONE® ProSim</td>
<td>Process Yellow</td>
</tr>
<tr>
<td>PANTONE® Uncoated</td>
<td>Yellow 012</td>
</tr>
<tr>
<td>PANTONE ProSim EURO®</td>
<td>Process Yellow</td>
</tr>
<tr>
<td>PANTONE® Process CSG</td>
<td>1-1</td>
</tr>
<tr>
<td>PANTONE® Process Euro</td>
<td>E 1-1</td>
</tr>
<tr>
<td>TOYO COLOR FINDER</td>
<td>0001pc*</td>
</tr>
<tr>
<td>TRUMATCH 4-Color Selector</td>
<td>1-a</td>
</tr>
</tbody>
</table>
Tinted colors

In FO_Color objects that are tints, the following properties have no meaning:

- FP_FamilyName
- FP_InkName
- FP_Cyan
- FP_Magenta
- FP_Yellow
- FP_Black

Changing these properties in a tinted color will turn the FO_Color object into an untinted color. If you want to change the hue of a tinted FO_Color object, you must select a new base color or change the hue of the base color.

Also, you cannot change FP_ColorPrintCtl and FP_ColorViewCtl in a tinted color; if you try to change them, the FDK returns an error of FE_TintedColor. To change these properties, you must change them in the tint’s base color.

**FP_TintPercent**

You can set FP_TintPercent to a metric value from 0.00 to 100.0 (representing 0% to 100%), or to FV_COLOR_NOT_TINTED. If you set it to FV_COLOR_NOT_TINTED, then FP_TintBaseColor automatically changes to FV_NO_BASE_COLOR.

When you set a percentage value for FP_TintPercent, if FP_TintBaseColor was set to FV_NO_BASE_COLOR, then it automatically changes to the object ID for the color Black.

**FP_TintBaseColor**

Every tint has a base color. Note that you cannot use a tint as a base color for some other tint. If you set the base color to FV_NO_BASE_COLOR, then the FP_TintPercent for the current FO_Color object is set to FV_COLOR_NOT_TINTED.

When you set a valid color for FP_TintBaseColor, if FP_TintPercent was initially set to FV_COLOR_NOT_TINTED, then it will automatically be set to the metric value of 100.0 (for 100%). Be sure to change the tint percent if you want less than 100%.

Reserved colors

FrameMaker has eight reserved colors. FO_Color objects have a read-only property named FP_ReservedColor to specify whether the object represents a reserved color or not. Unless the color is one of the eight reserved colors, FP_ReservedColor will always be FV_COLOR_NOT_RESERVED. FP_ReservedColor can have one of the following values:

- FV_COLOR_NOT_RESERVED
- FV_COLOR_CYAN
- FV_COLOR_MAGENTA
- FV_COLOR_YELLOW
- FV_COLOR_BLACK
- FV_COLOR_WHITE
- FV_COLOR_RED
- FV_COLOR_GREEN
- FV_COLOR_BLUE
For a reserved color, all the properties are read-only except FP_ColorOverPrint, FP_ColorPrintCtl, and FP_ColorViewCtl. If you try to change any of the read-only properties, the FDK returns an error of FE_ReservedColor (except for properties that normally return FE_ReadOnly for unreserved colors).

### Structural element definitions

A structured FrameMaker document\(^3\) has an Element Catalog, which contains structural element definitions and named format change lists.

There are two ways to test whether a document is structured via the API. To test whether a document contains structure elements, get the FP_HighestLevelElement property for the main FO_Flow object in the document. To test whether the document contains an element catalog, get the FP_FirstElementDefInDoc property for the FO_Doc object. If you get legal values for these properties, then the document contains structure elements or an element catalog.

### What the user sees

Each structural element definition has a name (tag), which usually corresponds to a type of document component or structural element, such as Section, List, Quotation, or BodyPara. An element definition specifies an element’s relationship to other elements in a structured document. An element definition can also contain formatting information about the element.

The parts of an element definition that specify an element’s format are known as the **format rules**.

The part of an element definition that specifies a container element’s contents is known as a **content rule**. The content rule includes the following:

- A **general rule**, which specifies what elements are inside the container and in what order
- A list of **inclusions**, which specifies other elements that can appear anywhere in a container or the elements it includes (its **descendants**)
- A list of **exclusions**, which specifies elements that cannot appear in a container or in its descendants

Element definitions also specify **attribute definitions**, which describe attributes or separate units of information that the user can store with an element. An attribute definition can specify that an attribute is required for all elements with the element definition. It can also provide a list of the values an attribute can have, as well as a default value.

---

\(^3\) Structured documents are only available the Structured FrameMaker product interface.
How the API represents structural element definitions

FrameMaker represents each element definition with an FO_ElementDef object. FO_ElementDef properties provide the following information about an element definition:

- Its name
- Its format rules
- Comments which describe its use
- Its attribute definitions
- Its content rule
- The type of element it defines (for example, a container or a system variable)
- Flags indicating whether the element definition is defined in the Element Catalog and whether it can be used as the highest-level element for a flow
- The ID of the next FO_ElementDef object in the document
- Initial structure rules for automatic insertion of child elements
- Initial pattern rules for table components

The API uses an FO_FmtRule object to represent each of an element definition’s format rules. The FO_ElementDef object has the following properties that specify an element definition’s format rules:

- FP_FirstPgfRules
- FP_LastPgfRules
- FP_ObjectFmtRules
- FP_PrefixRules
- FP_SuffixRules
- FP_TextFmtRules

Each of these properties specifies an F_IntsT structure, which provides a list of FO_FmtRule IDs.

For example, suppose you create the element definition shown in Figure 14.35.

---

**Figure 14.35 Element definition for Item element**

- **Element (Container):** Item
- **General rule:** <TEXT>
- **Text format rules**
  - Element paragraph format: item
  1. If context is: BulletList
     - Numbering properties
     - Autonumber format: \b\t
  Else, if context is: NumberList
     - Numbering properties
     - Autonumber format: <n+>\t
- **Comment:** For normal body paragraphs.
The API represents the element definition in Figure 14.35 as an `FO_ElementDef` object with the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_Name</td>
<td>StringT</td>
<td>Item.</td>
</tr>
<tr>
<td>FP_ElementPgfFormat</td>
<td>StringT</td>
<td>Item.</td>
</tr>
<tr>
<td>FP_NextElementDefInDoc</td>
<td>F_ObjHandleT</td>
<td>ID of the next <code>FO_ElementDef</code> object in the Element Catalog.</td>
</tr>
<tr>
<td>FP_ElementInCatalog</td>
<td>IntT</td>
<td>True.</td>
</tr>
<tr>
<td>FP_GeneralRule</td>
<td>StringT</td>
<td>&lt;TEXT&gt;.</td>
</tr>
<tr>
<td>FP_ObjectType</td>
<td>IntT</td>
<td>FV_FO_CONTAINER.</td>
</tr>
<tr>
<td>FP_Comment</td>
<td>StringT</td>
<td>For normal body paragraphs.</td>
</tr>
<tr>
<td>FP_TextFmtRules</td>
<td>F_IntsT</td>
<td>The ID of the element definition’s text format rule. For a list of this format rule’s properties, see “How the API represents format rules and format rule clauses” on page 345.</td>
</tr>
</tbody>
</table>

**Format rules and format rule clauses**

An element definition can contain several format rules, each of which can contain several format rule clauses.

**What the user sees**

Format rules and format rule clauses allow the template builder to specify the formats an element has in specific circumstances. A format rule can be either a context rule or a level rule.

A context rule contains clauses that specify an element’s formatting based on its parent and sibling elements. For example, one clause of a format rule could specify that a Para element has the FirstBody paragraph format if it is the first child of a Heading element. Another clause could specify that a Para element has the Body paragraph format in all other contexts.

A level rule contains clauses that specify an element’s formatting based on the level to which it is nested within specific types of ancestor elements. For example, one clause of a level rule could specify that a Para element appears in 12-point type if has only one Section element among its ancestors. Another clause could specify that a Para element appears in 10 point type if there are two Section elements among its ancestors.

A format rule clause can use any of the following to specify an element’s formatting in specific contexts:
● A formatting tag or name, such as a paragraph tag, a character tag, or a marker name
● A subformat rule
● A format change list
● A named format change list

How the API represents format rules and format rule clauses

The API uses an FO_FmtRule object to represent each format rule in an element definition, and an FO_FmtRuleClause object to represent each format rule clause in a format rule. Each FO_FmtRule object has an FP_FmtRuleClauses property, which specifies its format rule clause.

FO_FmtRule properties provide the following information about a format rule:

● A list of its format rule clauses
● An indication of whether it is a context rule or a level rule
● If the format rule is a level rule, the element tags to count among the element’s ancestors and the tag at which to stop counting

FO_FmtRuleClause properties provide the following information about a format rule clause:

● A flag indicating how the rule clause specifies formatting, such as a paragraph tag or a format change list
● The formatting tag or name, subformat rule, or change list the rule clause uses to specify the element’s formatting
● The circumstances under which the rule clause applies: if it is in a context rule, the context; if it is in a level rule, the level
● The context label

For example, the element definition shown in Figure 14.35 on page 343 includes a single format rule. The following table lists some of the properties of that format rule.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_ElemeDef</td>
<td>F_ObjHandleT</td>
<td>ID of the FO_ElementDef object that contains the format rule.</td>
</tr>
<tr>
<td>FP_FmtRuleClauses</td>
<td>F_IntsT</td>
<td>The IDs of the format rule clauses in the format rule. For a list of the rule clause’s properties, see the table below.</td>
</tr>
<tr>
<td>FP_FmtRuleType</td>
<td>IntT</td>
<td>FV_CONTEXT_RULE.</td>
</tr>
</tbody>
</table>
The `F_IntsT` structure specified by the `FO_FmtRule` object’s `FP_FmtRuleClauses` property provides an array, which includes the IDs of two format rule clauses. The following table lists some of the properties of the first format rule clause.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_ContextLabel</td>
<td>StringT</td>
<td>BulletList.</td>
</tr>
<tr>
<td>FP_FmtChangeList</td>
<td>F_ObjHandleT</td>
<td>The ID of the format change list (FO_FmtChangeList object) that is applied to the element when the specified context is valid. For a list of the change list’s properties, see “How the API represents format change lists” on page 346.</td>
</tr>
<tr>
<td>FP_FmtRule</td>
<td>F_ObjHandleT</td>
<td>ID of the <code>FO_Rule</code> object that contains the format rule clause.</td>
</tr>
<tr>
<td>FP_RuleClauseType</td>
<td>IntT</td>
<td>FV_RC_CHANGELIST.</td>
</tr>
</tbody>
</table>

**Format change lists**

A format change list describes a set of changes to paragraph format properties.

**What the user sees**

A format rule clause can use format change lists to specify how a paragraph format changes when the format rule clause applies. A change list can specify a change to just a single paragraph property, or it can specify changes to a long list of properties.

A change list can specify absolute values or relative values. For example, it can specify that the paragraph left indent is one inch, or it can specify that it is one inch greater than the inherited left indent.

A change list can be named or unnamed. A named change list appears in the Element Catalog. Format rule clauses that use a named change list specify its name (or tag). Multiple rule clauses can specify the same named change list. An unnamed change list appears in a rule clause. It is used only by the rule clause in which it appears.

**How the API represents format change lists**

The API uses an `FO_FmtChangeList` object to represent each change list in a document. `FO_FmtChangeList` properties provide the following information about a change list:
● Its name if it is a named change list
● The ID of the next change list in the document’s list of change lists
● A paragraph format tag if the change list specifies one

A change list has one property for each paragraph format property it changes. For example, if it changes only the first indent, it has the properties described above and just an FP_FirstIndent property. If it changes the space below and the leading, it has the properties described above and the FP_SpaceBelow and FP_Leading properties.

If a change list changes a paragraph property to an absolute value, the property it uses has the same name as the corresponding paragraph format property (for example, FP_FirstIndent). If the change list changes a property with a relative value, the property it uses has the name of the corresponding paragraph format property with the word Change appended to it (for example, FP_FirstIndentChange).

For example, the format rule clause in the element definition in Figure 14.35 on page 343 includes an unnamed change list. The following table lists the change list’s properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_Name</td>
<td>StringT</td>
<td>NULL</td>
</tr>
<tr>
<td>FP_NextFmtChangeListInDoc</td>
<td>F_ObjHandleT</td>
<td>ID of the next FO_FmtChangeList object in the document</td>
</tr>
<tr>
<td>FP_PgfCatalogReference</td>
<td>StringT</td>
<td>NULL</td>
</tr>
<tr>
<td>FP_AutoNumString</td>
<td>StringT</td>
<td>\b\t</td>
</tr>
</tbody>
</table>

**Structural elements**

Structured FrameMaker documents contain structural elements, which are instances of structural element definitions.

**What the user sees**

Each structural element is a component of a document. A structural element can consist of one or more paragraphs, a text range, one or more child elements, or anything you can insert in text (such as variables or tables).

Each structural element has an element definition specifying what its format and contents should be. Elements in a document can have the same element definition. For example, a document may have several elements with a Para element definition. For more information on element definitions, see “Structural element definitions” on page 342.

The elements in a container element are called its child elements. Child elements can also be containers; container elements can be nested. The element definition’s general rule specifies a container’s allowable child elements or text and the order in which they should occur.
A container element can violate its content rule by omitting required child elements, by including excluded child elements, or by having the elements in the wrong order. If a container element obeys its content rule, it is said to be valid.

Elements can also have attributes, which correspond to SGML or XML attributes. An attribute can be a defined attribute, which is defined in the element’s element definition, or an undefined attribute, which is not defined in the element’s element definition.

**How the API represents structural elements**

FrameMaker represents a structural element with an FO_Element object, whose properties provide the following information:

- Its attributes
- The ID of the object that represents its element definition
- The IDs of its parent and immediate sibling elements
- The IDs of the first and last FO_Element objects in the linked list of its child FO_Element objects
- Whether the element is collapsed
- Whether the element is valid; and if it is invalid, the reasons it is invalid
- The ID of the object associated with the FO_Element object, if the element is a noncontainer element, such as a marker or a system variable
- The element’s context label
- The format rule clauses that apply to the element

**How the API represents a structural element’s validity**

An element can be invalid in several ways. For example, its parent’s content rule may not allow it, or it may contain a child element that is not allowed. FO_Element objects have validation properties that indicate the extent of an element’s validity. For example, an element has a property named FP_ElementIsInvalidInParent, which is set to True if the element is not allowed by its parent element.

**Changing an element**

When using FrameMaker, an author can select elements and wrap, merge, or change them. The API provides functions to wrap and merge elements directly; for example, F_ApiMergeIntoFirst(). However, there is no corresponding function to directly change an element. Changing an element corresponds to a user selecting an element in the document, selecting an element name in the Element catalog, and then clicking Change on the element catalog.

To change an element via the API, you must change the FP_ElementDef property of the FO_Element object. You can traverse the list of element definitions in the document by starting with the FP_FirstElementDefInDoc property of the FO_Doc object, and then using the FP_NextElementDefInDoc property of the resulting FO_ElementDef object. You can identify the element definition by using its FP_Name property.
This chapter describes books and discusses how the Frame API represents them.

What the user sees

A book maintains a collection of documents that are known as components. The book helps you organize and format these component documents. It also enables you to create generated files, such as tables of contents and indexes. A book does not contain the component document files. It contains references to the component document files, in an ordered list; such a reference is called a book component.

Each component contains its own setup data such as pagination and numbering. For example, each component contains properties to determine whether its page and paragraph numbering continues from the previous document or restarts at 1, and whether the document starts on a left or right page.

The component properties should not differ from the corresponding properties in the document file. However, by setting the values in one but not both, a client could set up a component with different numbering properties than the corresponding document file. Subsequent book updates will make the numbering properties match.

For example, the paragraph numbering for a set of documents in files might be set to restart at 1. When the user adds the documents to the book, then each component will have the same numbering properties. Your client could loop through a book and set the numbering for each component to continue from the previous file. In this case, when your client (or the user) updates the book, the FrameMaker product will change the numbering for the document files so they match the component numbering. For more information about how component and document numbering properties interact, see your Frame product user’s manual.

IMPORTANT: A book component can be a document saved in any file format. The FDK can only modify document objects in documents that were saved in FrameMaker binary (FASL) files, but you can use channels to open MIF or text files and modify them.

How the API represents books

The API represents each book with an FO_Book object which can contain one or more FO_BookComponent objects.

FO_Book objects have properties that provide the following information:
● Whether the book has been modified
● Display properties such as book window size and location, text to show for each component, and text in the book’s status line
● Whether or not the book is view-only, and view-only display properties
● Selection state; whether the book icon is selected, the first selected component, or the range of selected structure elements in the book
● Properties that determine how to print the book and save it as PDF
● For structured books, structure properties such as the element catalog for the book and the ID of the highest level element in the book

An `FO_BookComponent` object represents an individual book component. It has properties that provide the following information:

● The name of the document represented by the component
● The IDs of the next and previous component in the book and the next selected component in the book
● Whether the component is generated, and the type of generated file; in other words, whether the component is a specific type of list or index
● Whether to include the component in print, update, and import formats operations
● The list of paragraph format tags the product uses to generate a list from this component
● The ID of the parent book
● Numbering and pagination properties for the component; these properties may differ from the document’s specific set of numbering properties
● For structured books, the structural element representing the book component

Suppose you create the book in Figure 15.1. The book is named `C:\MyDocs\book1.book`, and has three documents; `myDoc1.fm`, `myDoc2.fm`, and `myDoc3.fm`.

![Figure 15.1 A book with three components](image-url)
The API represents mybook.book with one FO_Book object and three FO_BookComponent objects. The following table lists some of the FO_Book object’s properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_Name</td>
<td>StringT</td>
<td>C:\MyDocs\book1.book</td>
</tr>
<tr>
<td>FP_NextOpenBookInSession</td>
<td>F_ObjHandleT</td>
<td>0</td>
</tr>
<tr>
<td>FP_FirstComponentInBook</td>
<td>F_ObjHandleT</td>
<td>ID of FO_BookComponent object for myDoc1.fm</td>
</tr>
<tr>
<td>FP_FirstSelectedComponentInBook</td>
<td>F_ObjHandleT</td>
<td>ID of FO_BookComponent object for myDoc3.fm</td>
</tr>
<tr>
<td>FP_StatusLine</td>
<td>StringT</td>
<td>Empty string (&quot;&quot;); the status line currently displays no text</td>
</tr>
<tr>
<td>FP_TypeOfDisplayText</td>
<td>IntT</td>
<td>FV_BK_FILENAME</td>
</tr>
</tbody>
</table>

The following code shows how to get properties from the selected book component. First it gets the active book, and then the first selected component in the active book. For the book in Figure 15.1, the component would be for myDoc3.fm. For the selected component, the code prints out the method the component uses to compute footnote numbering. Then, if the footnote numbering uses custom characters for footnotes (daggers, etc.), the code prints out the custom numbering string.

```c
VoidT F_ApiCommand(command)
{
    IntT command;

    F_ObjHandleT bookId, compId;
    StringT numString;

    bookId = F_ApiGetId(0, FV_SessionId, FP_ActiveBook);
    compId = F_ApiGetId(FV_SessionId, bookId, FP_FirstSelectedComponentInBook);

    F_Printf(NULL, "\n CompName is: %s",
            F_ApiGetString(bookId, compId, FP_Name));

    switch(F_ApiGetInt(bookId, compId, FP_FnNumComputeMethod)) {
    case FV_NUM_RESTART:
        F_Printf(NULL, "\nFn Compute: FV_NUM_RESTART");
        break;
    case FV_NUM_CONTINUE:
        F_Printf(NULL, "\nFn Compute: FV_NUM_CONTINUE");
    }
}```
break;
case FV_NUM_PERPAGE:
    F_Printf(NULL, "\nFn Compute: FV_NUM_PERPAGE");
    break;
case FV_NUM_READ_FROM_FILE:
    F_Printf(NULL, "\nFN Compute: FV_NUM_READ_FROM_FILE");
    break;
default:
    F_Printf(NULL, "\nFn Num Compute Method: UNKNOWN");
    break;
}
if(F_ApiGetInt(bookId, compId, FP_FnNumStyle)
    == FV_FN_NUM_CUSTOM) {
    F_Printf(NULL, "\nFn Num Style: FV_FN_NUM_CUSTOM");
    numString = F_ApiGetString(
        bookId, compId, FP_FnCustNumString);
    F_Printf(NULL, "\n Cust Str: %s", numString);
    F_ApiDeallocateString(&numString);
}

How the API organizes book components

The API organizes the FO_BookComponent objects that represent a book’s components in a linked list. The FO_Book object’s FP_FirstComponentInBook property specifies the first FO_BookComponent object in the list. Each FO_BookComponent object has FP_PrevComponentInBook and FP_NextComponentInBook properties that specify the IDs of the previous and next FO_BookComponent objects in the list. The order of the list is the same as the order of the components in the book.

Suppose you create the book shown in Figure 15.1 on page page 352. The API represents this book with the objects shown in Figure 15.2 on page page 355.
FIGURE 15.2 Objects that represent a book and its components

How the API represents structured books

If a book is a FrameMaker structured book, it has an FP_HighestLevelElement property, which specifies the ID of the FO_Element object that represents the root element. Each component in the book also has an FP_ComponentElement property, which specifies the FO_Element object ID for that component’s structure element.

Creating new books and components

To create a new book, use F_ApiNewNamedObject(). If you pass an empty string for the object name, the FrameMaker product creates an untitled book.

To insert a new book component in a book, use F_ApiNewSeriesObject(). This creates a new book component, but the component has no document file assigned to it. If the user tries to open such a component’s file, the FrameMaker product will alert the user that no file exists to match the component name. When you insert a new book component, you should assign a document to it.

For more information on F_ApiNewSeriesObject(), see “Creating series objects” on page 149.

The following code creates an untitled book with one component that represents a newly created, custom document file. After the code successfully saves the new document, it creates a book component and assigns the document name to the new component.
#define in ((MetricT) 65536*72)
...

VoidT F_ApiCommand(command)
    IntT command;
{
    F_ObjHandleT bookId, docId, compId;
    StringT s;

    F_ApiBailOut();
    bookId = F_ApiNewNamedObject(FV_SessionId, FO_Book, (StringT)"");
    /* First create an 8.5 x 11 custom document. */
    docId = F_ApiCustomDoc(F_MetricFractMul(in,17,2), 11*in, 1,
        F_MetricFractMul(in,1,4), in, in, in, in, in,
        FF_Custom_SingleSided, True);
    /* Save the doc, prompting for filename */
    F_ApiSimpleSave(docId, (StringT)"", True);
    /* If file has name, make a component */
    s = F_ApiGetString(FV_SessionId, docId, FP_Name);
    if(F_StrLen(s)) {
        compId = F_ApiNewSeriesObject(
            bookId, FO_BookComponent, 0);
        F_ApiSetString(bookId, compId, FP_Name, s);
        F_ApiDeallocateString(&s);
    }
}

---

**Updating a book**

After making changes in the documents or components of a book, your client must update the book to ensure all references are valid. For example, if you change the page numbering properties, you must update the book to ensure cross-references indicate the correct numbering. You can update a book via the `F_ApiUpdateBook()` command or the `F_ApiSimpleGenerate()` command.

With `F_ApiUpdateBook()`, you can specify aspects of the Update operation, such as whether to update books with view-only documents. You can specify all aspects of the operation, or you can specify some aspects and allow the user to decide others. For example, you can instruct the FrameMaker product to post an alert if the book contains a MIF file.

To use `F_ApiUpdateBook()`, you should first understand property lists and how to manipulate them directly. For more information on this subject, see “Representing object characteristics with properties” on page 270 and “Manipulating property lists directly” on page 93.

The syntax for `F_ApiUpdateBook()` is:
ErrorT F_ApiUpdateBook(F_ObjHandleT bookId, 
    F_PropValsT *updateParamsp, 
    F_PropValsT **updateReturnParamspp);

<table>
<thead>
<tr>
<th>This argument</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>docId</td>
<td>The ID of the document or book to save.</td>
</tr>
<tr>
<td>updateParamsp</td>
<td>A property list that tells the FrameMaker product how to update the book and how to respond to errors and other conditions. Use F_ApiGetUpdateBookDefaultParams() or F_ApiAllocatePropVals() to create and allocate memory for this property list. To use the default list, specify NULL.</td>
</tr>
<tr>
<td>updateReturnParamspp</td>
<td>A property list that returns information about how the FrameMaker product updated the book.</td>
</tr>
</tbody>
</table>

**IMPORTANT:** Always initialize the pointer to the property list that you specify for updateReturnParamspp to NULL before you call F_ApiUpdateBook().

To call F_ApiUpdateBook(), do the following:

1. Initialize the pointer to the updateReturnParamspp property list to NULL.
2. Create an updateParamsp property list.
   - You can get a default list by calling F_ApiGetUpdateBookDefaultParams(), or you can create a list from scratch.
3. Call F_ApiUpdateBook().
4. Check the Update status.
   - Check the returned values in the updateReturnParamspp list for information about how the FrameMaker product updated the book.
5. Deallocate the updateParamsp and updateReturnParamspp property lists.

Steps 2, 4, and 5 are discussed in the following sections.

**Creating an updateParamsp script with F_ApiGetUpdateBookDefaultParams()**

The API provides a function named F_ApiGetUpdateBookDefaultParams() that creates a default updateParamsp property list. If you are setting a number of properties, it is easiest to use this function get the default property list and then change individual properties as needed.

The syntax for F_ApiGetUpdateBookDefaultParams() is:
F_PropValsT F_ApiGetUpdateBookDefaultParams();
The following table lists some of the properties in the property list returned by `F_ApiGetUpdateBookDefaultParams()`. The first value listed for each property is the default value for the property. You can change any property in the list to use its other legal values.

<table>
<thead>
<tr>
<th>Property</th>
<th>Meaning and possible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>FS_AlertUserAboutFailure</td>
<td>Specifies whether to notify the user if something unusual occurs during the update operation</td>
</tr>
<tr>
<td></td>
<td>False: don’t notify user</td>
</tr>
<tr>
<td></td>
<td>True: notify user</td>
</tr>
<tr>
<td>FS_MakeVisible</td>
<td>Make newly generated files (lists and indexes) visible</td>
</tr>
<tr>
<td></td>
<td>True: make visible</td>
</tr>
<tr>
<td></td>
<td>False: don’t make visible</td>
</tr>
<tr>
<td>FS_ShowBookErrorLog</td>
<td>Specifies whether to use the book error log to display warnings.</td>
</tr>
<tr>
<td></td>
<td>False: don’t display book error log; display warnings in the console</td>
</tr>
<tr>
<td></td>
<td>True: display the book error log</td>
</tr>
</tbody>
</table>

For the complete list returned by `F_ApiGetUpdateBookDefaultParams()`, see “F_ApiGetUpdateBookDefaultParams()” on page 187 of the *FDK Programmer’s Reference*.

For example, to get a default `updateParamsp` property list and modify it so that it instructs `F_ApiUpdate()` to show the book error log, use the following code:

```c
... F_PropValsT params; ErrorT err;
... /* Get the default parameter list. */
params = F_ApiGetUpdateBookDefaultParams();
/* Get the index for the error log property, */
/* then set the property to True. */
i = F_ApiGetPropIndex(&params, FS_ShowBookErrorLog);
params.val[i].propVal.u.ival = True;
...```

The API allocates memory for the property list created by `F_ApiGetUpdateBookDefaultParams()`. Use `F_ApiDeallocatePropVals()` to free the property list when you are done with it.
Creating an updateParams script from scratch

If you want to specify only a few properties when you call `F_ApiUpdateBook()`, it is most efficient to create a property list from scratch. To create the property list, you must allocate memory for it, and then set up the individual properties.

Use the API convenience function, `F_ApiAllocatePropVals()`, to allocate memory for the property list. For example, the following code creates an `updateParams` property list that will instruct `F_ApiUpdateBook()` to display the error log:

```c
#DEFINE ERR_LOG 0

F_PropValsT params, *returnParamsp = NULL;
... /* Allocate memory for the list. */
params = F_ApiAllocatePropVals(1);
/* Set up FS_ShowBookErrorLog property and set it to True. */
params.val[ERR_LOG].propIdent.num = FS_ShowBookErrorLog;
params.val[ERR_LOG].propVal.valType = FT_Integer;
params.val[ERR_LOG].propVal.u.ival = True;
... /* When you're finished, free the F_PropValsT */
F_ApiDeallocatePropVals(&params)
```

Checking update status

`F_ApiUpdateBook()` stores a pointer to a property list in `updateReturnParamspp`; the list contains one property which contains flags to indicate the status. For a list of the possible flags, see “`F_ApiUpdateBook()`” on page 338 of the `FDK Programmer's Reference`.

To determine if a particular `FS_UpdateBookStatus` bit is set, use `F_ApiCheckStatus()`. For example, the following code determines if an Update operation was canceled because the current book contains duplicate files (components that refer to the same file):

```c
F_PropValsT params, *returnParamsp = NULL;
F_ObjHandleT bookId;

/* Get the active book, update, and check status. */
bookId = F_ApiGetId(0, FV_SessionId, FP_ActiveBook);
params = F_ApiGetUpdateBookDefaultParams();
F_ApiUpdate(bookId, &params, &returnParamsp);
if (F_ApiCheckStatus(returnParamsp, FV_DuplicateFileInBook))
    F_ApiAlert("Duplicate files in book.", FF_ALERT_CONTINUE_NOTE);

/* Deallocate property lists. */
F_ApiDeallocatePropVals(&params);
F_ApiDeallocatePropVals(returnParamsp);
...`
The API provides a utility function named `F_ApiPrintUpdateStatus()`, which prints the save error values to the console platforms. For more information, see “F_ApiPrintUpdateBookStatus()” on page 255 of the *FDK Programmer’s Reference*.

**Example**

The following code updates the currently active book. The update operation will display the error log for any error conditions, will allow inconsistent numbering properties, and (since the code allows inconsistent numbering) will not update the numbering in the book. It then prints out the update status. Finally, the code deallocates the property lists that it used to update the book.

```c
#include "futils.h"

IntT i;
UCharT msg[1024];
F_PropValsT params, *returnParamsp = NULL;
F_ObjHandleT bookId;

params = F_ApiGetUpdateBookDefaultParams();
i = F_ApiGetPropIndex(&params, FS_ShowBookErrorLog);
params.val[i].propVal.u.ival = True;
i = F_ApiGetPropIndex(&params, FS_AllowInconsistentNumProps);
params.val[i].propVal.u.ival = FV_DoOK;
i = F_ApiGetPropIndex(&params, FS_UpdateBookNumbering);
params.val[i].propVal.u.ival = False;

err = F_ApiUpdateBook(bookId, &params, &returnp);
F_ApiPrintUpdateBookStatus(returnp);
F_ApiDeallocatePropVals(&params);
F_ApiDeallocatePropVals(returnp);
```

---

**Using the book error log**

When updating a book, the FrameMaker product posts errors to a book error log. The error log is a FrameMaker document that lists error conditions and includes hypertext links to offending locations in the book’s document files.

By default, FDK clients post book errors to the console. However, your clients can post errors to the log, and can include hypertext links in those messages.
Displaying the error log for book updates

By default, the FrameMaker product displays update errors in the console. You direct the FrameMaker product to display the error log via the property list you pass to F_ApiUpdateBook(). In that list, set the FS_ShowBookErrorLog flag to True. For more information about the update book properties, see “Creating an updateParamsp script with F_ApiGetUpdateBookDefaultParams()” on page 357.

Writing messages to the error log

The FrameMaker product includes an API client that writes messages to the error log. To write a message to the error log, you must use F_ApiCallClient().

The syntax for the client call is:

F_ApiCallClient("BookErrorLog", "log -b=[bookId] -d=[docId] -o=[objId] --[text]");

where:

- BookErrorLog is the name of the client to call.
- log identifies this as a log message.
- -b is either the book ID or a document ID; typically the active book.
- -d is either a document ID or an object ID; typically a document associated with a book component.
- -o is an object in the document represented by the -d argument. If you pass both a document ID and an object ID, the call adds a hypertext link, from the error message to the object you specified.
- -text is the text of the message to appear in the log. To post a time stamp in the message, pass the FM_PRINT_DATESTAMP token as the message string.

The call creates a unique log for each book or document ID you pass in the -b argument; if you pass 0 for a book ID, you will create a log that is not associated with any book; all calls with the 0 book ID will go to that log file.

When you pass a document ID for the -d argument, the call creates an entry with the document’s pathname. It then indents all contiguous entries with the same document ID under that document’s pathname. This continues until you pass a different document ID. If you pass 0 for the -d argument, the call will not indent the errors.

If you don’t have the document ID, you can specify log entry indenting under a filename via the text you pass for the log message. To do this, you precede the log message with a filename, followed by a carriage return. This method creates an indented section each time you pass a filename and carriage return, even if you pass the same filename in a series of log entries.

For example, if you passed the following to the BookErrorLog client in two consecutive calls:

"filename.fm\012Here is my first Log Message"
"filename.fm\012Here is my second Log Message"

The BookErrorLog client would create the following messages:
filename.fm
Here is my first Log Message
filename.fm
Here is my second Log Message

Example

The following code shows a function that posts messages to a log, with or without a time stamp; if you pass valid ID’s for all the ID arguments, the log message will include a hypertext link to the specified object in the specified document:

```c
VoidT ReportError(F_ObjHandleT docId, F_ObjHandleT objId,
                   ConStringT errmsg, BoolT dateStamp)
{
    F_ObjHandleT bookId;
    StringT log_msg = F_StrNew((UIntT)256);
    bookId = F_ApiGetId(0, FV_SessionId, FP_ActiveBook);
    if(dateStamp) {
        F_Sprintf(log_msg, "log -b=%d -d=%d -o=%d --%s",
                  bookId, docId, objId, (StringT)"FM_PRINT_DATESTAMP");
    } else {
        F_Sprintf(log_msg, "log -b=%d -d=%d -o=%d --%s",
                  bookId, docId, objId, errmsg);
    }
    F_ApiCallClient("BookErrorLog", log_msg);
    F_ApiDeallocateString(&log_msg);
}
```
Part III

Frame Development Environment (FDE)
This chapter provides an overview of how the FDE works and how to use it to write portable FDK clients. It also provides a simple example: a portable filter that you can run right away.

The FDE helps make your clients portable by providing platform-independent alternatives to platform-specific input/output (I/O), string, and memory schemes. With the FDE, you can run your client on all FrameMaker products with minimal effort.

The FDE also provides libraries of utility functions that are useful for filter development.

**How the FDE works**

The FDE consists of the following:

- A virtual environment
- Utility libraries

Figure 16.1 shows the components of the FDE and their relationship to a client. All of the platform-specific code is contained within the virtual environment.

---

**Figure 16.1  Relationship of the FDE to a client**

- Platform-specific code (FDE)
- Platform-independent code (clients and libraries)
The virtual environment

The FDE’s virtual environment abstracts the functionality of platform-dependent compilers, operating systems, and C language libraries. It provides the following facilities to replace platform-dependent schemes:

- I/O channels
  Channels are an abstraction of I/O streams. Instead of directly using the input, output, and temporary files of a particular environment, you use the virtual environment’s channels.

- Memory handling
  The virtual environment provides platform-independent memory allocation and deallocation functions for pointers and handles.

Utility libraries

The FDE utility libraries provide the following types of functions:

- Platform-independent replacements for C language library functions, such as `strcpy()` or `strlen()`

- MetricT type functions for converting and manipulating FrameMaker product MetricT values
  The MetricT data type is used in the API to express measurements, such as font sizes and line widths. It is also used in MIF `<MathFullForm>` statements. For more information on the MetricT data type, see “MetricT values” on page 675 of the FDK Programmer’s Reference.

- MIF writing functions for directing output to a MIF output channel, setting indent levels for the channel, and writing a variety of simple MIF statements

- I/O functions for writing and reading from input channels when byte swapping is necessary

How to make your client portable

To use the FDE to make your client portable, follow these general steps:

1. Add a call to `F_FdeInit()` to initialize the FDE environment.
   To initialize the FDE environment, call `F_FdeInit()` as follows:
   ```
   F_FdeInit();
   ```

2. Replace C primitive data types with FDE types.
   Replace any C primitive data types, such as `char` or `int`, with FDE types, such as `UCharT` or `IntT`. 
3. Replace I/O and memory calls in your client with calls to the FDE’s virtual environment.

   Use the FDE virtual environment’s channel I/O and memory functions to make your I/O and memory calls platform independent. For a description of the virtual environment and its I/O and memory facilities, see Chapter 17, “Making I/O and Memory Calls Portable”

4. Replace string, character, and other platform-specific C library function calls with corresponding calls to FDE utility library functions.

   For information on the utility library functions, see Chapter 2, “FDK Function Reference” of the FDK Programmer’s Reference

5. Include the appropriate FDE header files.

   All FDE clients must include the fdetypes.h header file. If you are using functions from the FDE utility libraries, you must also include the header files for these functions after fdetypes.h. For example, you must include fstrlist.h if you use any of the string list functions. If you need to include other C library header files, they must precede all FDE header files.

6. Compile your client and link the FDK library with it.

   The following sections describe Steps 2, 3, and 4 in greater detail.

### Replacing C primitive data types with FDE types

To ensure portability across different platforms and compilers, the FDE uses substitutes for C language primitive data types. These types are defined in the f_types.h header file. The following table lists the FDE data types and their equivalents.

<table>
<thead>
<tr>
<th>FDE data type</th>
<th>Equivalent type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddrT</td>
<td>char*, void*</td>
<td>Unsigned 4 bytes</td>
</tr>
<tr>
<td>BoolT</td>
<td>long</td>
<td>Signed 4 bytes</td>
</tr>
<tr>
<td>ByteT</td>
<td>char</td>
<td>Signed 1 byte</td>
</tr>
<tr>
<td>CharT</td>
<td>char</td>
<td>Signed 1 byte</td>
</tr>
<tr>
<td>ConStringT</td>
<td>const unsigned char*</td>
<td>Pointer</td>
</tr>
<tr>
<td>ErrorT</td>
<td>long</td>
<td>Signed 4 bytes</td>
</tr>
<tr>
<td>FunctionT</td>
<td>Function pointer (returns IntT)</td>
<td>Unsigned 4 bytes</td>
</tr>
<tr>
<td>GenericT</td>
<td>char*, void*</td>
<td>Signed 4 bytes</td>
</tr>
<tr>
<td>IntT</td>
<td>long</td>
<td>Signed 4 bytes</td>
</tr>
<tr>
<td>MetricT</td>
<td>long</td>
<td>Signed 4 bytes</td>
</tr>
<tr>
<td>NativeDoubleT</td>
<td>double</td>
<td>Signed 4 bytes (platform dependent)</td>
</tr>
</tbody>
</table>
To ensure that your client does not use platform-specific data types or functions, the FDE redefines them. If a client that includes the `fdetypes.h` header file uses a platform-specific data type, the FDE will provide an equivalent type that is platform-independent. A table below illustrates the equivalent types and their sizes:

<table>
<thead>
<tr>
<th>FDE data type</th>
<th>Equivalent type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>NativeIntT</td>
<td>int</td>
<td>Signed 4 bytes (platform dependent)</td>
</tr>
<tr>
<td>NativeCharT</td>
<td>char</td>
<td>Signed 1 byte (platform dependent)</td>
</tr>
<tr>
<td>NativeULongT</td>
<td>unsigned long</td>
<td>Unsigned 4 bytes (platform dependent)</td>
</tr>
<tr>
<td>NativeLongT</td>
<td>long</td>
<td>Signed 4 bytes (platform dependent)</td>
</tr>
<tr>
<td>ProcedureT</td>
<td>Procedure pointer</td>
<td>Unsigned 4 bytes</td>
</tr>
<tr>
<td>PByteT</td>
<td>int, char</td>
<td>Signed 4 bytes</td>
</tr>
<tr>
<td>PCharT</td>
<td>int, char</td>
<td>Signed 4 bytes</td>
</tr>
<tr>
<td>PRealT</td>
<td>double</td>
<td>Signed 4 bytes</td>
</tr>
<tr>
<td>PShortT</td>
<td>int, short</td>
<td>Signed 4 bytes</td>
</tr>
<tr>
<td>PtrT</td>
<td>char*, void*</td>
<td>Unsigned 4 bytes</td>
</tr>
<tr>
<td>PUByteT</td>
<td>unsigned int, unsigned char</td>
<td>Unsigned 4 bytes</td>
</tr>
<tr>
<td>PCharT</td>
<td>unsigned int, unsigned char</td>
<td>Unsigned 4 bytes</td>
</tr>
<tr>
<td>PShortT</td>
<td>unsigned int, unsigned short</td>
<td>Unsigned 4 bytes</td>
</tr>
<tr>
<td>RealT</td>
<td>float</td>
<td>Signed 4 bytes</td>
</tr>
<tr>
<td>ShortT</td>
<td>short</td>
<td>Signed 2 bytes</td>
</tr>
<tr>
<td>StrBuffT</td>
<td>char [STRBUFFSIZE+1]</td>
<td>256 bytes</td>
</tr>
<tr>
<td>StringT</td>
<td>unsigned char*</td>
<td>Pointer</td>
</tr>
<tr>
<td>UByteT</td>
<td>unsigned char</td>
<td>Unsigned 1 byte</td>
</tr>
<tr>
<td>UCharT</td>
<td>unsigned char</td>
<td>Unsigned 1 byte</td>
</tr>
<tr>
<td>UIntT</td>
<td>unsigned long</td>
<td>Unsigned 4 bytes</td>
</tr>
<tr>
<td>UShortT</td>
<td>unsigned short</td>
<td>Unsigned 2 bytes</td>
</tr>
<tr>
<td>VoidT</td>
<td>void</td>
<td>None</td>
</tr>
</tbody>
</table>

a. `PByteT`, `PCharT`, `PRealT`, `PShortT`, `PUByteT`, `PCharT`, and `PShortT` are used to suppress compiler errors if a function is not declared in ANSI format. They are used only for function arguments.

To ensure that your client does not use platform-specific data types or functions, the FDE redefines them. If a client that includes the `fdetypes.h` header file uses a platform-specific data type, the FDE will provide an equivalent type that is platform-independent. A table below illustrates the equivalent types and their sizes:

<table>
<thead>
<tr>
<th>FDE data type</th>
<th>Equivalent type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>NativeIntT</td>
<td>int</td>
<td>Signed 4 bytes (platform dependent)</td>
</tr>
<tr>
<td>NativeCharT</td>
<td>char</td>
<td>Signed 1 byte (platform dependent)</td>
</tr>
<tr>
<td>NativeULongT</td>
<td>unsigned long</td>
<td>Unsigned 4 bytes (platform dependent)</td>
</tr>
<tr>
<td>NativeLongT</td>
<td>long</td>
<td>Signed 4 bytes (platform dependent)</td>
</tr>
<tr>
<td>ProcedureT</td>
<td>Procedure pointer</td>
<td>Unsigned 4 bytes</td>
</tr>
<tr>
<td>PByteT</td>
<td>int, char</td>
<td>Signed 4 bytes</td>
</tr>
<tr>
<td>PCharT</td>
<td>int, char</td>
<td>Signed 4 bytes</td>
</tr>
<tr>
<td>PRealT</td>
<td>double</td>
<td>Signed 4 bytes</td>
</tr>
<tr>
<td>PShortT</td>
<td>int, short</td>
<td>Signed 4 bytes</td>
</tr>
<tr>
<td>PtrT</td>
<td>char*, void*</td>
<td>Unsigned 4 bytes</td>
</tr>
<tr>
<td>PUByteT</td>
<td>unsigned int, unsigned char</td>
<td>Unsigned 4 bytes</td>
</tr>
<tr>
<td>PCharT</td>
<td>unsigned int, unsigned char</td>
<td>Unsigned 4 bytes</td>
</tr>
<tr>
<td>PShortT</td>
<td>unsigned int, unsigned short</td>
<td>Unsigned 4 bytes</td>
</tr>
<tr>
<td>RealT</td>
<td>float</td>
<td>Signed 4 bytes</td>
</tr>
<tr>
<td>ShortT</td>
<td>short</td>
<td>Signed 2 bytes</td>
</tr>
<tr>
<td>StrBuffT</td>
<td>char [STRBUFFSIZE+1]</td>
<td>256 bytes</td>
</tr>
<tr>
<td>StringT</td>
<td>unsigned char*</td>
<td>Pointer</td>
</tr>
<tr>
<td>UByteT</td>
<td>unsigned char</td>
<td>Unsigned 1 byte</td>
</tr>
<tr>
<td>UCharT</td>
<td>unsigned char</td>
<td>Unsigned 1 byte</td>
</tr>
<tr>
<td>UIntT</td>
<td>unsigned long</td>
<td>Unsigned 4 bytes</td>
</tr>
<tr>
<td>UShortT</td>
<td>unsigned short</td>
<td>Unsigned 2 bytes</td>
</tr>
<tr>
<td>VoidT</td>
<td>void</td>
<td>None</td>
</tr>
</tbody>
</table>

a. `PByteT`, `PCharT`, `PRealT`, `PShortT`, `PUByteT`, `PCharT`, and `PShortT` are used to suppress compiler errors if a function is not declared in ANSI format. They are used only for function arguments.
type or function, the compiler issues an error message when you attempt to compile it. For example, if your client declares the following variable:

```c
char ch;
```

the compiler issues an error message similar to the following:

```c
#error ! Non_FDE_token "char" ! ch;
```

To avoid these error messages, you can:

- Use the FDE substitute for the platform-dependent data type or function. For example, use `UCharT` instead of `char`.
- Add the following code above the `#include "fdetypes.h"` statement:
  ```c
  #define DONT_REDEFINE
  ``
  This prevents the FDE 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:
  ```c
  #undef char
  ``
  This allows the FDE to generate errors if your client uses any other platform-specific types.

### Replacing I/O and memory calls

The following table lists some commonly used I/O and memory calls and the FDE functions you can replace them with:

<table>
<thead>
<tr>
<th>Function</th>
<th>FDE substitute</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fclose()</code></td>
<td>F_ChannelClose()</td>
</tr>
<tr>
<td><code>fopen()</code></td>
<td>F_ChannelOpen()</td>
</tr>
<tr>
<td><code>fwrite()</code></td>
<td>F_ChannelWrite()</td>
</tr>
<tr>
<td><code>alloc()</code></td>
<td>F_Alloc()</td>
</tr>
<tr>
<td><code>free()</code></td>
<td>F_Free()</td>
</tr>
<tr>
<td><code>printf()</code></td>
<td>F_Printf()</td>
</tr>
<tr>
<td><code>sprintf()</code></td>
<td>F_Sprintf()</td>
</tr>
</tbody>
</table>

Some FDE functions have slightly different parameters or return values than the corresponding platform-specific I/O and memory functions. For example, `F_Alloc()` has a parameter that `alloc()` doesn’t have. Before using an FDE I/O or memory function, look it up in Chapter 2, “FDK Function Reference” of the *FDK Programmer’s Reference*.


**Replacing C library calls**

The following table lists some commonly used C library functions and the FDE functions you can replace them with:

<table>
<thead>
<tr>
<th>Function</th>
<th>FDE substitute</th>
</tr>
</thead>
<tbody>
<tr>
<td>strcmp()</td>
<td>F_StrEqual() or F_StrCmp()</td>
</tr>
<tr>
<td>strlen()</td>
<td>F_StrLen()</td>
</tr>
<tr>
<td>strcpy()</td>
<td>F_StrCpy()</td>
</tr>
<tr>
<td>strcat()</td>
<td>F_StrCat()</td>
</tr>
</tbody>
</table>

Some FDE functions have slightly different parameters or return values than the corresponding C library functions. For example, F_StrCpy() returns VoidT, while strcpy() returns a pointer. Before using an FDE library function, look it up in Chapter 2, “FDK Function Reference” of the *FDK Programmer’s Reference*.

**A simple FDE filter**

The following client filters a text file into a Frame document. Following the code is a line-by-line description of how it works.

```c
1. #include "fdetypes.h"
2. #include "fapi.h"
3. #include "fchannel.h"
4. #include "fmemory.h"
5. #include "fmetrics.h"
6. #include "futils.h"
7. #include "fioutils.h"
8.
9. #define BUFFERSIZE 1025
10. #define in (MetricT)(72*65536)
11. 12. VoidT F_ApiNotify(notification, docId, sparm, iparm)
13.     IntT notification;
14.     F_ObjHandleT docId;
15.     StringT sparm;
16.     IntT iparm;
17. {
18.         FilePathT *path;
19.         ChannelT chan;
20.         F_TextLocT tl;
21.         StringT buf;
22.         IntT count;
23.         24.         F_FdeInit();
25.         buf = F_Alloc(BUFFERSIZE, NO_DSE);
```
if (buf == NULL) {
    F_ApiBailOut();
    F_ApiReturnValue(0);
    return;
}

path = F_PathNameToFilePath(sparm, NULL, FDefaultPath);
if (path == NULL) return;
if((chan = F_ChannelOpen(path,"r")) == NULL) return;

docId = F_ApiCustomDoc(F_MetricFractMul(in, 17, 2),
    11*in, 1, F_MetricFractMul(in, 1, 4), in,
    in, in, FF_Custom_SingleSided, False);

tl.objId = F_ApiGetId(FV_SessionId, docId, FP_MainFlowInDoc);
tl.offset = 0;

while(!F_ChannelEof(chan)) {
    count = F_ReadBytes(buf, BUFFERSIZE-1, chan);
    buf[count] = '\0';
    tl = F_ApiAddText(docId, &tl, buf);
}

F_ApiReturnValue(docId);
F_FilePathFree(path);
F_Free(buf);
F_ChannelClose(chan);
F_ApiBailOut();

Lines 1 to 11

These lines include header files and define constants for the client.

Lines 12 to 56

These lines define the F_ApiNotify() callback, which the FrameMaker product calls when the user attempts to open or import a filterable file. You specify which file types are filterable when you register the filter. For more information on registering the filter, see “Compiling and running the example FDE filter” on page 373.

When the user or a client attempts to open or import a filterable file, the FrameMaker product calls the F_ApiNotify() callback with notification set to FA_Note_FilterIn and filename set to the name of the file the user is attempting to open or import.

For more information on F_ApiNotify() and other Frame API functions, see the FDK Programmer’s Reference.
Line 24

F_FdeInit() initializes the FDE. You must initialize the FDE before calling any FDE functions.

Lines 25 to 32

The FDE memory allocation function, F_Alloc(), allocates a buffer for reading text from the file. The NO_DSE flag instructs the function to return NULL if it is unable to allocate memory for the buffer.

The F_ApiReturnValue(0) call notifies the FrameMaker product that the filter was unable to filter the specified file. After the filter returns, the FrameMaker product displays an alert informing the user that the filter could not filter the file.

Lines 33 to 34

F_PathNameToFilePath() converts the platform-specific pathname specified by filename to a platform-independent filepath. The FDefaultPath flag instructs F_PathNameToFilePath() to use the filenaming conventions of the current platform to interpret the platform-specific pathname. Because the pathname specified by filename is absolute, F_PathNameToFilePath() ignores the second parameter (the anchor).

Lines 35 to 36

F_ChannelOpen() opens the file specified by the platform-independent filepath. The "r" instructs F_ChannelOpen() to open the file for reading only.

Lines 37 to 40

The filter needs to create a new FrameMaker product document for the filterable file’s contents. To do this, it calls F_ApiCustomDoc().

The parameters of F_ApiCustomDoc() specify the dimensions and layout of the new document. For example, the first parameter specifies the page width, the MetricT equivalent of 8.5 inches. To get the MetricT equivalent of fractions such as 8.5 inches, you can’t use simple multiplication and division. You must use an FDE metric function, such as F_MetricMul(). For example, to get the MetricT equivalent of 8.5 inches, you can’t use the expression 17/2*in. You must use F_MetricFractMul(in, 17, 2).

The last parameter of the F_ApiCustomDoc() call specifies that the new document is not visible. If another FDK client initiates the open operation, it may keep the document invisible after the filter has added the filterable file’s contents to it. If the user initiates the open operation, the FrameMaker product automatically makes the document visible when the filter returns.
Note that you could alternately open an existing document to use as a template for the filtered data. In that case, you should still be sure to open it silently using F_ApiOpen(), do not use F_ApiSimpleOpen().

**Lines 44 to 50**

These lines read text from the input channel and add it to the FrameMaker product document. F_ReaderBytes() reads text from the input channel into the buffer until it reaches the end of the file. F_ApiAddText() adds the text in the buffer to the Frame document, starting at the current insertion point.

**Line 51**

This line sets the filter’s return value to the ID of the document the filter created. This notifies the FrameMaker product that the filter opened the file successfully.

**Lines 52 to 56**

These lines clean up and free resources used by the filter. The calls to F_FilePathFree() and F_Free() free resources used by the filepath and the text buffer; F_ChannelClose() closes the input channel; and F_ApiBailOut() exits the filter.

**Compiling and running the example FDE filter**

The source code for the example filter and a makefile or project file are provided online with the FDK. To compile the sample filter, use your platform’s make or build utility. For the location of example files and instructions on compiling and linking them, see the [FDK Platform Guide](#) for your platform.

To run the example filter, follow these general steps:

7. Register it as required by your platform’s implementation of the FDK:

- If you are working on a Macintosh computer, copy or move the compiled example filter to the FrameMaker product Filters folder. The sample resource file that you compile with the filter instructs the FrameMaker product to call the filter any time the user attempts to open or import a file with the type KURT.

- If you are working on a Windows platform and you have compiled your client into a DLL named filter.dll and copied or moved it to the FrameMaker product filters directory, add the following line to the [APIClients] section of your product.ini file:

  KurtWrite=TextImport,kurt,"KURT",Kurt,all,filters\filter.dll,krt

This instructs the FrameMaker product to call the filter when the user attempts to open or import a file with a .krt extension.
If you are working on a UNIX platform with US English as the user interface language, and the pathname of your compiled client is /usr/fdk/samples/filter/filter, you can register it by creating a text file named ~/fminit/usenglish/APIArea/apiclients with the following lines in it:

```xml
<APIclient
  <Name KurtWrite Files Import>
  <Type TextImport>
  <Suffix krt>
  <Facet kurt>
  <FormatId KURT>
  <VendorId FAPI>
  <Directory /usr/fdk/samples/filter>
  <CommandLine $FAPIDIR/filter>
>
```

This instructs the FrameMaker product to call the filter when the user attempts to open or import a file with a .krt extension. The FDK sets the environment variable $FAPIDIR to the directory specified by the <Directory> statement.

For more information on how to register filters, see the FDK Platform Guide for your platform.

1. Create a sample Text Only file.

   If you are working on a UNIX or a Windows platform, give the file a filename with an .krt extension. If you are working on a Macintosh computer, change the file’s type to KURT.

2. Start the FrameMaker product.

3. Open the file you created.

The FrameMaker product calls the filter, which creates a new document and adds the text from the file to it.
Making I/O and Memory Calls Portable

This chapter discusses the FDE virtual environment functions you can use to replace the platform-specific I/O, assertion handler, and memory calls in your client.

For lists of the FDE virtual environment functions, see Chapter 1, “Function Summary” of the FDK Programmer's Reference. For the complete description of a function, look it up in Chapter 2, “FDK Function Reference,” in the FDK Programmer’s Reference.

Initializing the FDE

Before you call any FDE functions, you should initialize the FDE. To initialize the FDE, call F_FdeInit() as follows:

```c
F_FdeInit();
```

**IMPORTANT:** Each time your client bails out and reinitializes, it should reinitialize the FDE by calling F_FdeInit().

Using platform-independent representations of pathnames

The FDE allows you to specify pathnames with a platform-independent representation called a filepath. The FDE uses the data type FilePathT to specify a filepath.

Converting pathnames to filepaths

The FDE provides functions that allow you to convert platform-specific pathname strings to filepaths and filepaths back to platform-specific pathname strings. For example, the following code converts the UNIX pathname /tmp/myfile to a filepath:

```c
FilePathT *path;
path = F_PathNameToFilePath("/tmp/myfile", NULL, FUnixPath);
F_FilePathFree(path);
```

**IMPORTANT:** If you call a function typed FilePathT*, you must use F_FilePathFree() to free the returned pointer when you are done with it.

The following code converts the filepath created above back to a UNIX pathname:
You don’t have to convert a filepath back to the platform-specific pathname it was originally created from; you can convert it to a pathname for any platform the FDE supports. For example, the following code converts the filepath created above to a Windows pathname:

```
FilePathT *path;
StringT pathname;

pathname = F_FilePathToPathName(path, FDosPath);
```

To make a `F_FilePathToPathName()` and `F_PathNameToFilePath()` call platform-independent, set the second parameter to `FDefaultPath`. This instructs the function to use the pathname conventions of the platform the client is currently running on. For example, the following code converts a filepath to a pathname for the current platform:

```
FilePathT *path;
StringT pathname;

pathname = F_FilePathToPathName(path, FDefaultPath);
```

FDE filepath functions, such as `F_PathNameToFilePath()` and `F_FilePathToPathName()`, have arguments that specify a path type. These arguments are typed `PathEnumT`. `PathEnumT` is defined as:

```
typedef enum{
    FDefaultPath /* Platform the client is running on */
    FUnixPath /* UNIX */
    FMacPath /* Macintosh */
    FDosPath /* Windows */
    FDIPath /* Device-independent */
} PathEnumT
```

The following table provides examples of the path types.

<table>
<thead>
<tr>
<th>Path type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUnixPath</td>
<td>/MyDirectory/MySubdirectory/MyFile</td>
</tr>
<tr>
<td>FMacPath</td>
<td>HardDrive:MyDirectory:MySubdirectory:MyFile</td>
</tr>
<tr>
<td>FDosPath</td>
<td>c:\mydirect\mysubdir\myfile</td>
</tr>
<tr>
<td>FDIPath</td>
<td>&lt;r&gt;&lt;c&gt;MyDirectory&lt;c&gt;MySubdirectory&lt;c&gt;MyFile</td>
</tr>
</tbody>
</table>
Device-independent pathnames have the following format:
<code>name<code>name<code>name... where code identifies the role of the component in the pathname and name is the name of a component in the pathname. The following table lists codes and their meanings.

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Root of the file tree</td>
</tr>
<tr>
<td>c</td>
<td>Component</td>
</tr>
<tr>
<td>u</td>
<td>Up one level in the file tree</td>
</tr>
</tbody>
</table>

For example, you can express the following UNIX pathname:
/MyDirectory/MySubdirectory/MyFile
as the following device-independent, absolute pathname:
<r><c>MyDirectory<c>MySubdirectory<c>MyFile
or as the following device-independent, relative pathname:
\<c>MyFile
For more information on how FrameMaker specifies device-independent pathnames, see the online MIF Reference.

**Manipulating filepaths**

The FDE provides functions that allow you to manipulate filepaths. For example, F_DeleteFile() deletes a file or directory, F_FilePathProperty() checks file permissions, and F_FilePathGetNext() allows you to traverse all the files in a directory. These functions are platform-independent substitutes for functions such as remove(), _access(), and _fstat().

**Making I/O portable with channels**

To make your client’s I/O portable, you use channels. Channels are an abstraction of platform-specific files or file systems. The FDE provides functions to manipulate channels. For example, F_ChannelOpen() opens a channel, F_ChannelRead() reads from a channel, F_ChannelWrite() writes to a channel, and F_ChannelClose() closes a channel. These functions are substitutes for platform-specific functions, such as fopen(), fread(), fwrite(), and fclose(). All I/O channels in the FDE are buffered internally.

To manipulate a file with FDE channel functions, you must first convert the file’s pathname to a filepath and then open the filepath with F_ChannelOpen(). For example, the following code opens the file Macintosh HD:myfile for reading.
Making I/O and Memory Calls Portable

ChannelT chan;
FilePathT *path;

path = F_PathNameToFilePath((StringT)"Macintosh HD:myfile",
                            NULL, FMacPath);
if((chan = F_ChannelOpen(path,"r")) == NULL)
{
    F_Printf(NULL, "Couldn't open file.\n");
    return;
}

Assertion-handler functions

Your client can register an assertion handler and perform its own error handling with the FDE function F_SetAssert(). When the client’s assertion handler returns, the FDE’s assertion handler is called to clean up the system and exit the client properly.

To use the FDE assertion-handler functions, you must include fdetypes.h and fassert.h in your client.

Making memory allocation portable

The FDE provides a set of functions you can substitute for your client’s platform-specific memory allocation and deallocation function calls. These functions accommodate the different memory models on UNIX, Macintosh, and Windows platforms.

The Macintosh does not use a hardware memory management unit. Instead, the operating system periodically compacts memory to free up more space. To accommodate this model, the Macintosh uses relocatable memory. It allows you to work with handles, which are pointers to pointers to relocatable memory. Each handle points to a master pointer, which in turn points to the memory block. If the operating system decides to move the memory, it moves the block and then updates the master pointer to point to the new location. The old memory location can then be overwritten. The handle, however, remains the same. So your client is not affected by the change in the absolute location of the memory.

Sometimes you may need to use a pointer directly into absolute memory. To create this pointer, you must first lock the memory to tell the operating system that it should not relocate it. You can then safely use any absolute pointer into the block without fear of the memory being relocated. After you are done with the pointer, you unlock the memory, allowing the operating system to relocate it. The Macintosh also provides nonrelocatable memory that you can access directly.

To develop portable clients that use large memory blocks, you should use handle-based memory management. There are some trade-offs between using pointers and handles. Handles may slow down the access to memory. Pointers may fragment the heap space, especially if you
use them on the Macintosh with memory blocks larger than 512K. In general, you should use
pointers for small memory allocations and handles for large allocations.

Many FDE memory allocation functions, such as `F_Alloc()`, provide a `flags` argument
that specifies what to do if memory can’t be allocated. If you set this argument to `DSE` and
memory can’t be allocated, the FDE calls a function that you register by calling
`F_SetDSExit()`. If you set `flags` to `NO_DSE` and memory allocation is unsuccessful, the
memory allocation function returns `NULL`. For more information, see “F_SetDSExit()” on
page 439 of the *FDK Programmer’s Reference*.

**Allocating memory with handles**

The FDE provides functions that allow you to allocate and deallocate memory with handles.
For example, `F_AllocHandle()` allocates a handle, `F_ClearHandle()` initializes a
handle’s block of data, and `F_FreeHandle()` frees memory allocated to a handle.

Handle memory functions that are typed `ErrorT` return `FdeSuccess` if they are
successful. Other handle memory functions return `NULL` if they are unable to comply with a
request.

To use the FDE memory functions, you must include `fdetypes.h` and `fmemory.h` in your
program.

**Allocating memory with pointers**

The FDE provides functions that allow you to allocate and deallocate memory with pointers.
For example, `F_Alloc()` allocates memory, `F_ClearPtr()` initializes a pointer’s block
of data, and `F_Free()` frees memory allocated to a pointer.

Pointer memory functions that are not typed `ErrorT` return `NULL` if they are unable to
comply with a request. Functions that are typed `ErrorT` return `FdeSuccess` if they are
successful.

If you are a UNIX programmer, you should avoid static memory allocation because it
consumes valuable data and heap space on the Windows and Macintosh platforms.

To use the FDE memory functions, you must include `fdetypes.h` and `fmemory.h` in your
program.

**Error and progress reporting**

The virtual environment provides functions that allow you to report error and progress status.
For example, `FWarning()` prints a warning message to the console window on UNIX
platforms, or to the error log window on the Macintosh or the Frame console on Windows.

To use the FDE progress reporting functions, you must include `fdetypes.h` and
`fprogs.h` in your program.
This chapter describes FDE utility libraries:

- The string library provides platform-independent equivalents for many of the functions in `<string.h>`.
- The string list library provides routines for creating a list of strings and manipulating the strings.
- The character library provides platform-independent equivalents for some of the functions in `<ctype.h>` that can be used on Frame characters.
- The I/O library provides functions to read data from or write data to channels and performs byte swapping when it is necessary.
- The hash library provides functions to create a hash table and manipulate the cells in it.
- The metric library provides functions for manipulating and converting MetricT values (values using Frame’s internal representation of measurements).
- The MIF data structures and macros provide data structures and macros to represent and manipulate the statements described in the online MIF Reference.
- The MIF library provides functions that help you write nicely formatted MIF to a channel.
- The simple MIF library provides functions to write individual MIF statements to a channel.

For lists of FDE utility library functions, see Chapter 1, “Function Summary” in the FDK Programmer’s Reference. For the complete description of a function, look it up in Chapter 2, “FDK Function Reference” in the FDK Programmer’s Reference.

### String library

The string library provides functions for allocating and manipulating strings. For example, `F_StrNew()` allocates a string, `F_StrCmp()` compares two strings, and `F_StrAlphaToInt()` converts an alphanumeric string to an integer.

The FDE uses the StringT type for strings. StringT is an array of UCharT. When you specify a string size in an FDE string function, you must include the terminating 0 in the size. The string library provides functions for allocating, manipulating, and freeing strings.

**IMPORTANT:** Use `F_Free()` or `F_ApiDeallocateString()` to free StringT strings. See “F_Free()” on page 386 of the FDK Programmer’s Reference and “F_ApiDeallocateStructureType()” on page 74 of the FDK Programmer’s Reference.

Most FDE string functions have the prefix F_Str. To use FDE string functions, you must include fdtypes.h and fstrings.h in your program and call F_FdeInit().
The string list library

The string list library provides routines for creating a string list and manipulating the strings in it. For example, `F_StrListNew()` allocates a string list, `F_StrListInsert()` inserts a string into a list, and `F_StrListSort()` sorts a string list.

String lists are typed `StringListT`. All the functions in the string list library are prefixed with `F_StrList`. To use the FDE string list functions, you must include `fdetypes.h` and `fstrlist.h` in your program.

Character library

The character library provides routines that convert and manipulate Frame characters. For example, `F_CharIsAlphabetic()` determines whether a character is alphabetic, `F_CharToLower()` converts a character to lowercase, and `F_CharToUpper()` converts a character to uppercase. The FDE implements all character library functions as macros.

The character library functions are all prefixed with `F_Char`. To use the FDE character functions, you must include `fdetypes.h` and `fcharmap.h` in your client.

**IMPORTANT:** Before you use the character library functions, you must call `F_FdeInit()` to initialize the character library.

The I/O library

The I/O library provides functions that allow your client to read data from and write data to channels. These functions can swap bytes when reading or writing on a channel. For example, it may be necessary to swap bytes in either of the following cases:

- The platform your client is running on is little-endian and the channel you are reading from, or writing to, isn’t.

- The channel is little-endian but the platform isn’t.

When you use the I/O library functions, you can specify whether an input or output channel’s byte ordering is little-endian or big-endian by calling `F_SetByteOrder()` or `F_ResetByteOrder()`. The I/O functions use this information to determine whether byte swapping is necessary. By default, the FDE assumes the channel and the platform are consistent. If you do not specify whether the input or output channel order is little-endian, the functions do not swap bytes.

When you use the FDE I/O functions, you should take care of alignment issues yourself. FDE I/O functions return `0` if they reach the end of a file or an error occurs.

To use the FDE channel utility functions, you must include `fdetypes.h` and `fioutils.h` in your program.
The hash library

The hash library provides functions to create a hash table and manipulate the cells in it. For example, F_HashCreate() creates a hash table, F_HashSet() adds an entry to a hash table, and F_HashGet() fetches an entry from a hash table. All the functions in the library are prefixed by F_Hash. To use any FDE hash functions, you must include fdetypes.h and fhash.h in your program.

The hash table is stored in a data structure of type HashTableT. This data structure is opaque to you; you must use the F_Hash routines to manipulate a hash table.

Creating hash tables

You create a hash table via F_HashCreate(), which is defined as follows:

```c
HashT F_HashCreate(StringT name, /* Name of the table */
                 IntT minSize, /* Minimum size of the table */
                 PShortT keyLen, /* Size of keys */
                 GenericT notFound, /* Returned if searched key not found */
                 /* Determine if cell can be reused */
                 BoolT (*deadQuery)(GenericT),
                 /* Called when cell is deleted */
                 VoidT (*removeNotify)(GenericT),
                 /* Converts key to string*/
                 Void(*T stringifyMe)(PtrT, UCharT *));
```

- Use minSize to suggest the amount of space the FDE should allocate for the hash table. You can pass a value of 0 which notifies the FDE to use its own calculations for memory allocation.
- For non-string keys, you specify a value for the size of key. For keys that are strings, you specify KEY_IS_STRING; in that case each key is a nul-terminated string.
- F_HashGet() is a routine that searches for a key and returns the associated data. If it can’t find the specified key, it still must return a value; the notFound argument specifies what value to return when F_HashGet() doesn’t find the key.
- deadQuery specifies a callback to give you the opportunity to determine the validity of a cell. You should only specify a function for this argument if your code can make this determination; if the function returns True, the cell will be marked for deletion. As the FDE hash routines maintain the table, they will call this function to determine whether they can delete the cell’s contents. Normally, it is best to specify 0 for this argument, letting the FDE manage the hash table on its own.
- removeNotify specifies a callback to invoke whenever an F_Hash function or the FDE removes a cell. A typical use for this is to deallocate structures that were allocated for the cell’s data. If you don’t need to clean up memory for any cells, pass 0.
- stringifyMe specifies a procedure that turns non-string keys into strings. This is most useful for debugging.
Structures and pointers in keys

The routines F_HashSet(), F_HashGet(), and F_HashRemove() each receive a key as an argument. These routines work best with flat keys such as integers or strings. It is safest to use flat keys instead of structures for keys; if necessary you can always use F_Sprintf() to put structure fields into a string.

The internal hash routine used by these functions examines the bytes in the key. If the key is a structure containing pointers, it only checks the pointers, not the pointed-to data. Further, it examines every byte in the key. If you use structures for keys, you should be sure to clear the memory first, then assign values to the structure members, then set the item in your hash table.

For example, use F_ClearPtr(&myStructKey, sizeof(myStructKey)); when myStructKey will be a key for a table cell.

When you pass a key to F_HashSet(), the function actually creates a copy of the key. However, it only copies the string, or the number of bytes specified in F_HashCreate() as the key length. If your key is a structure that contains pointers, F_HashSet() will create copies of the pointers, but not copies of the pointed-to data.

Metric library

The Frame API uses the MetricT type to specify measurements, such as tab offsets and font sizes. MIF also uses it in <MathFullForm> statements. MetricT values should not be confused with the metric system. For more information on metric values, see “MetricT values” on page 675 of the FDK Programmer’s Reference.

The metric library provides a set of operations that allow you to manipulate metric values without converting them to other units of measurement. For example, F_MetricFloat() converts a real number to a metric number, F_MetricFractMul() multiplies a metric number by a fraction, and F_MetricToFloat() converts a metric number to a real number.

All the functions in the metric library are prefixed with F_Metric. To use the FDE metric functions, you must include fdetypes.h and fmetrics.h in your program.

MIF data structures and macros

The FDE provides data structures that represent the statements described in the online manual MIF Reference. It also provides convenience macros that help you get and set fields in these structures. This saves you the effort of developing your own data structures and macros. For the location of the files containing the MIF data structures and macros, see the FDK Platform Guide for your platform.

MIF data structures adhere to the following naming conventions:

- Structure names contain the prefix Mif, the statement name, and the suffix Struct. For example, the FDE represents the <TextFlow> statement with the MifTextFlowStruct structure.
● Structure type names use the suffix `StructT`. For example, the structure type name for `MifTextFlowStruct` is `MifTextFlowStructT`.

● Each of the substatements in a MIF statement is represented as a field, whose name corresponds to the name of the substatement. For example, the FDE represents the statement `<Marker <MType N> <MText string>>` with the structure:

```c
typedef struct MifMarkerStruct {
    IntT MType;
    StringT MText;
} MifMarkerStructT;
```

● If a substatement is a structure, a pointer in the structure points to the structure that represents the substatement. For example, the FDE represents the statement `<Para <Pgf>...>` with:

```c
typedef struct MifParaStruct {
    MifPgfStructT *Pgf;
    ...
} MifParaStructT;
```

● If a substatement is a list of structures, it is represented as a link structure. The link structure’s name contains a `Mif` prefix, the substatement name, and a `StructL` suffix. The link structure type includes the suffix `StructLT`. A pointer in the structure points to the link structure of the substatement. For example, the FDE represents the statement `<AFrames <Frame>...>` with:

```c
typedef struct MifAFrameStruct {
    struct MifFrameStructL *Frame;
} MifAFrameStructT;
```

where the field `Frame` contains a list of all the frames associated with `AFrame`.

To get or set a field in a MIF data structure, use MIF macros. MIF macro names contain the `Mif` prefix, the structure name, the access type (Get or Set), and the structure field name. For example, to get the `MType` field of a `<Marker...>` statement, use the macro named `MifMarkerGetMType(mif_marker_ptr)`. When you call MIF macros, you must specify pointers to MIF data structures.

The names of MIF macros for list structures comprise the `Mif` prefix, the substatement name, the string `List`, the access type (Get or Set), and the structure field name. For example, to get `<Polygon... <Point>...`, you use `MifPointListGetNext`, `MifPointListGetPrev`, and `MifPointListGetPoint`.

Because most MIF data structures contain pointers to other structures, you should dynamically allocate memory space for all data structures to minimize errors.

---

The MIF library

The MIF library functions help write formatted MIF statements to a channel. For information on functions that write individual MIF statements, see “Simple MIF library” on page 387.
The MIF library maintains its own output channel and indent information. Before you call any MIF library function, you must:

1. Call `F_ChannelOpen()` to open a channel.
2. Call `F_MifSetOutputChannel()` to set the channel as the MIF output channel.
3. Call `F_MifSetIndent()` to set the channel’s indent level.

After you finish writing to the channel, be sure to call `F_ChannelClose()` to close it.

For example, to create a MIF file, use code similar to the following:

```c
#include "fdetypes.h"
#include "fapi.h"
#include "fchannel.h"
#include "futils.h"
#include "fioutils.h"
#include "fmifstmt.h"

FilePathT *path;
ChannelT chan;

path = F_PathNameToFilePath((StringT)"my.mif",
    NULL, FDefaultPath);
if((chan = F_ChannelOpen(path,"w")) == NULL) return;
F_MifSetOutputChannel(chan);
F_MifSetIndent(0);
F_MifMIFFile(5.0);
F_MifComment((StringT) "Generated by KurtWrite");
F_MifNewLine();
F_ChannelClose(chan);
```

This code creates a MIF file named `my.mif` in the default directory (the current directory on UNIX, the FrameMaker product directory on Windows, or the FrameMaker product Modules folder on the Macintosh) with the following contents:

```xml
<MIFFile 7.00 > # Generated by KurtWrite
```

Some MIF library functions use the type `MifUnitT`, which is an enumerated type specifying measurement units. It can have the following values.

**Table 18.1**

<table>
<thead>
<tr>
<th>MifUnitT value</th>
<th>Measurement unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIFUnitIn</td>
<td>Inches</td>
</tr>
<tr>
<td>MIFUnitCm</td>
<td>Centimeters</td>
</tr>
</tbody>
</table>
The FDE uses MifUnitT to generate MIF statements that include decimal values. If MIFUnitDef is specified, no unit symbol is generated.

All the MIF library functions are prefixed with F_Mif. To use the FDE MIF functions, you must include fdetypes.h and fmifstmt.h in your program.

### Simple MIF library

The simple MIF library functions are useful for writing individual MIF statements. Each function's name corresponds to the MIF statement that it writes.

If a MIF statement uses an enumerated value string, such as Left, Right, or Center, you construct a constant from that string by prefixing it with MIF. For example, the enumerated value strings for the <DParity> MIF statement are FirstLeft and FirstRight. When you call the corresponding MIF library function, F_MifDParity(), you can specify either MIFFirstLeft or MIFFirstRight for the DParity argument. If you aren’t sure if a certain constant exists, you can check for it in the fmiftype.h header file.

For the syntax and description of MIF statements, see the online MIF Reference.

All simple MIF library functions have the prefix F_Mif. To use them, you must include fdetypes.h and fmifstmt.h in your program.

### Table 18.1

<table>
<thead>
<tr>
<th>MifUnitT value</th>
<th>Measurement unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIFUnitMm</td>
<td>Millimeters</td>
</tr>
<tr>
<td>MIFUnitPca</td>
<td>Picas</td>
</tr>
<tr>
<td>MIFUnitPt</td>
<td>Points</td>
</tr>
<tr>
<td>MIFUnitDd</td>
<td>Didots</td>
</tr>
<tr>
<td>MIFUnitCc</td>
<td>Ciceros</td>
</tr>
<tr>
<td>MIFUnitDef</td>
<td>Default unit</td>
</tr>
</tbody>
</table>

The FDE uses MifUnitT to generate MIF statements that include decimal values. If MIFUnitDef is specified, no unit symbol is generated.
Glossary

This glossary contains words used in the FDK Programmer’s Guide. For other references providing more information about a term, see the index.

**anchored frame**
A container that is tied to a specific location in the text. An anchored frame moves with the text as the anchor symbol (¶) moves. See also unanchored frame.

**bail out**
The process by which an API application can exit and free system resources.

**body page**
A printable page in a document. See also master page.

**building blocks**
Text strings that define a cross-reference format, variable definition, index entry, or other item. For example, in a running header (Running H/F) variable, the building block `<$curpagenum>` is a building block that specifies the current page number.

**callback**
An application-defined function that responds to a call, such as an initialization call, from FrameMaker.

**channel**
A platform-independent abstraction of an input or output stream.

**child graphic object**
A graphic object that belongs to a particular group or frame.

**draw order**
The order in which FrameMaker draws graphic objects (that is, the back-to-front order). By default, the draw order is the same as the order in which you draw the graphic objects. When graphic objects overlap, the ones in the front (at the end of the draw order) obscure those in back.

**f-codes**
Hexadecimal function codes that specify individual user actions, such as cursor movement or text entry.

**flow**
An abstract notion of where text in a document can flow. Flows connect text frames. If text in an autoconnected frame reaches the end of a text frame, FrameMaker automatically creates a new text frame and connects it. In structured documents, a structured flow contains structural elements.

**footnote reference**
An anchor that appears in the main text as a number, letter, or special character.

**getting properties**
Using API functions to determine and get one or more characteristics of an object.
<table>
<thead>
<tr>
<th><strong>global document information</strong></th>
<th>Characteristics that apply to an entire document.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>graphic object</strong></td>
<td>Anything that the user can create with the Tools palette. Specifically, a graphic object can be an anchored frame, an unanchored frame, a geometric shape (a line, arc, rectangle, rounded rectangle, oval, polyline, or polygon), a group of other graphic objects in a frame together, a text line, a text frame, an imported graphic, an equation, or an inset.</td>
</tr>
<tr>
<td><strong>hidden page</strong></td>
<td>The page where FrameMaker stores hidden conditional text. A document can have only one hidden page. The API represents each hidden page with an <code>FO_HiddenPage</code> object.</td>
</tr>
<tr>
<td><strong>ID</strong></td>
<td>Identifier for an API object. IDs are typed as <code>F_ObjHandleT</code> (32-bit integers).</td>
</tr>
<tr>
<td><strong>inset</strong></td>
<td>An imported image created with an inset editor (a specially modified external application that can be launched from FrameMaker).</td>
</tr>
<tr>
<td><strong>MIF</strong></td>
<td>Maker Interchange Format, a set of ASCII statements that describes a FrameMaker document or book.</td>
</tr>
<tr>
<td><strong>named graphic frames</strong></td>
<td>Frames on reference pages that contain a graphic decoration, such as a line ruling for paragraphs in the body pages or a graphic that appears at the top of the page. The Paragraph Designer provides two settings, Frame Above and Frame Below, to specify the named frames that appear above or below a paragraph.</td>
</tr>
<tr>
<td><strong>named objects</strong></td>
<td>API objects identified by a unique name, for example, <code>FO_PgfFmt</code> and <code>FO_CharFmt</code> objects.</td>
</tr>
<tr>
<td><strong>object</strong></td>
<td>The API representation of basic entities in FrameMaker. For example, an <code>FO_Pgf</code> object represents a paragraph and an <code>FO_Rectangle</code> object represents a rectangle.</td>
</tr>
<tr>
<td><strong>page frame</strong></td>
<td>An invisible unanchored frame whose dimensions match those of a page. The API represents a page frame with an <code>FO_UnanchoredFrame</code> object.</td>
</tr>
<tr>
<td><strong>parent frame</strong></td>
<td>The frame containing a graphic object. See also page frame.</td>
</tr>
<tr>
<td><strong>properties</strong></td>
<td>The characteristics of API objects. Different types of objects have different properties. For example, <code>FO_Rectangle</code> objects have properties to represent a rectangle's height and width. These properties are named <code>FP_Height</code> and <code>FP_Width</code> respectively.</td>
</tr>
<tr>
<td><strong>property value</strong></td>
<td>Each property has a value associated with it. For example, if a paragraph has two tabs, the value of its <code>FP_NumTabs</code> property is 2.</td>
</tr>
<tr>
<td>term</td>
<td>definition</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>reference page</td>
<td>A nonprinting page containing named frames that can be used above and below paragraphs, or above footnote text. A reference page can also contain special flows that control the appearance of generated files such as indexes and lists. See also master page, body page, and named frames.</td>
</tr>
<tr>
<td>series object</td>
<td>Objects that the API keeps in ordered series. Series objects include FO_Pgf, FO_BodyPage, and FO_BookComponent objects.</td>
</tr>
<tr>
<td>session</td>
<td>Each time you start FrameMaker you are starting a session.</td>
</tr>
<tr>
<td>type-in properties</td>
<td>Font characteristics and conditional text properties that apply to new text typed at the insertion point.</td>
</tr>
<tr>
<td>unique persistent identifier (UID)</td>
<td>An integer that uniquely identifies an object within a document. An object’s UID remains the same from one FrameMaker session to the next.</td>
</tr>
<tr>
<td>virtual environment</td>
<td>FDE functionality that abstracts the functionality of platform-dependent compilers, operating systems, and C libraries.</td>
</tr>
</tbody>
</table>
Symbols

$FMHOME directory, getting G:96
+ (plus sign), conditional text marker prefix G:318
− (minus sign), conditional text marker prefix G:318
.dlg files (Windows) G:181, G:182
.x files (UNIX) G:181, G:182, G:185
.xdi files (Windows) G:181, G:182
<$endtrange R:407
= (equal sign), conditional text marker prefix G:318
\!, Escape symbol G:26
† (dagger symbol), indicating read-only properties R:495

A
aborting commands G:37
active books G:274
  API indication of G:275
active documents G:274
  API indication of G:275
getting the IDs of G:10
getting the pathname of G:97
Adobe Acrobat
  book properties governing R:498, R:507
document properties governing R:541
alert boxes G:17 to G:18
displaying G:7, G:11, G:17 to G:18
  function for R:5, R:43 to R:44
illustrated G:18, R:43 to R:44
type constants for (table) G:18
alerts
  table of G:18
  See also alert boxes
Alt key, keyboard shortcuts (Windows) G:202
anchored frames G:294
  API representation of G:295, G:312
creating G:148 to G:149, R:217
  functions for R:14, R:209 to R:212
defined G:293, G:389
locating G:294
properties of G:295, R:596 to R:597
See also unanchored frames
anchored objects G:148, G:312, R:209, R:210
creating G:148 to G:149
  functions for R:14, R:209 to R:212
default properties of G:148
See also anchored frames
anchors G:311, G:312
  See also anchor symbol; anchored frames; anchored objects
ANSI C, specifying special characters G:316
API (Frame Application Program Interface) G:ix, G:3 to G:12, R:xv
  advantages and limitations of G:xi, R:xvii
closing connection with R:318
disabling G:16
error protection feature G:81
initializing connection with R:330 to R:331
manipulating commands and menus G:161 to G:176
servicing calls to R:277 to R:278
See also API clients; API error codes; API functions
API clients G:3
bailing out of G:15, G:43 to G:44, G:375, G:389
  function for R:11, R:24, R:48
calling another client, function for R:24, R:49 to R:51
canceling operations, function for R:24
closing objects with G:4
closing FDK client connection with the API R:318
communication with the FrameMaker product G:4
communication with the user G:4
compiling G:8
creating G:6 to G:12
  example G:9 to G:12
creating custom dialog boxes for G:177 to G:209
current
generating directory containing R:24, R:59 to R:60
generating registered name of R:24, R:60
disabling G:16
executing commands in G:7, G:45 to G:79
for filters G:5, G:15, G:36
freesing memory from R:11, R:24, R:48
including header files in G:8
initializing G:7, G:13 to G:16, G:375
initializing FDK client connection with the API R:330 to R:331
modes of interaction G:17
property manipulation capabilities of G:81
registering with the FrameMaker product G:8 to G:9, G:12
responding to events initiated by G:40
running G:12
running with different FrameMaker product interfaces G:6
setting up user interfaces for G:7, G:17 to G:44
special types of G:5 to G:6
suspending operation of, functions for R:17, R:328, R:344 to R:345
user configuration of the user interface G:30 to G:31
API error codes
naming convention for G:xii, R:xix
printing current error status R:98, R:248
table of R:693 to R:697
API filters. See filters
API functions
alphabetical listing of R:27 to R:493
for creating objects (tables) G:143 to G:145, R:14
declarations of G:8
determining the availability of G:6
disadvantage of F_ApiFcodes() G:75
executing commands with G:7, G:40, G:45 to G:79
for getting and setting properties (table) G:90
header file containing G:8
naming convention for G:xii, R:xix
scriptable G:xii, G:45, G:46, G:93, R:xix
simple G:45
topical listing of R:1 to R:24
utility R:24 to R:25
See also individual API functions in the FDK
Reserved Names Index
API graphic objects. See graphic objects
API header file, including in API clients G:8
API objects. See objects (API)
API scriptable functions G:45, G:46
properties of
naming convention for G:xii, R:xix
setting G:93
API simple functions G:45
API structures. See data structures
API utility functions R:24 to R:25
-apiclient client_name command line option G:15
apiclients file G:8
Apple event handlers
determining if installed R:28 to R:29
installing R:29 to R:30
removing R:30 to R:31
applications. See API clients; FDE clients
arc properties R:598
As Is settings (Character Designer) G:306
API representation of G:308 to G:309
ASCII characters (special and higher), specifying G:316 to G:317
ASCIITemplate G:279
assertion-failure handlers, registering R:437 to R:438
assertion-handler functions G:378, R:437 to R:438
attribute definitions G:342
function for getting G:90
attribute definitions (of structural element definitions)
function for getting R:108 to R:110
function for setting R:278 to R:280
attributes
function for getting G:90
attributes (of structural element definitions)
function for getting R:110 to R:112
function for setting R:280 to R:282
attributes in markup G:348
Autoconnect setting (for flows) G:297
automatic updating of objects and properties G:81, G:160
autonumbering properties
of paragraph formats (FO_PgfFmt) R:623
of paragraphs (FO_Pgf) R:613 to R:614

B
bailing out (of API clients) G:15, G:43 to G:44, R:48
defined G:389
function for R:11, R:24, R:48
basename, of a specified filepath R:376 to R:377
batching changes G:96
big-endian byte order G:382, R:438
binary files
and Frame DRE files G:178
importing as text G:229
bit flags. See flags
body pages  G:288
  adding to documents  G:11, G:149 to G:150, G:289
                      function for  R:220 to R:221
  API representation of  G:289
  defined  G:389
  organization of objects on  G:290
                     illustrated  G:290
  properties of  G:291 to G:292, R:610
  See also pages
book components  G:351
  creating  G:355 to G:356, R:212 to R:213
  function for adding to documents  R:220 to R:221
  as organized in books  G:354 to G:355
  properties of  G:351 to G:355, R:503 to R:514
                  structure properties of  R:514
book structure properties  R:503
books  G:275, G:351 to G:362
       active  G:274, G:275
       Adobe Acrobat properties  R:498
       API representation of  G:351 to G:355
                     illustrated  G:355
       checking the Update status of  G:359 to G:360
            closing  G:66 to G:67
                     function for  R:2, R:61 to R:62
       comparing  G:68 to G:70
                     function for  R:2, R:65 to R:66
       components of  G:351
       creating  G:146, G:355 to G:356
       documents as set up in  G:351
       error log  G:360 to G:362
       functions applicable to  R:2
       general properties  R:495 to R:496
       generating files for  G:71
                     function for  R:2, R:321 to R:322
       importing formats from documents into  G:72
                     function for  R:2, R:323 to R:325
       making active  G:95
       open, API indication of  G:275
       opening  G:46 to G:53
                     functions for  R:2, R:238 to R:244,
                        R:326 to R:327
       organization of documents in  G:351
       as organized in sessions (illustrated)  G:276
       printing  G:57 to G:59
                     function for  R:2
       print properties  R:502 to R:503
properties of  G:351 to G:355, R:495 to R:503
       getting and setting  G:105
       saving  G:59 to G:66
                     functions for  R:2, R:272 to R:274,
                        R:327 to R:328
       setting status bars  G:98
       setting title bars  G:98
       structured  G:355
       updating  G:356 to G:360
       updating files for, function for  R:321 to R:322
       updating files of, functions for  R:2
       View Only properties  R:507
  See also book components
Bourne shell  R:333
boxes, in dialog boxes  G:198
building blocks
       cross-reference  G:322
       defined  G:389
       variable  G:324 to G:325
buttons
       in dialog boxes  G:199
       getting/setting the state of, function for  R:5
       properties of  G:199, R:533
bytes
       I/O library functions for
                       ordering  G:382, R:435, R:438
                       reading  R:427 to R:428
                       writing  R:489 to R:490
  See also  longs (4 bytes); shorts (2 bytes)

C
C language library functions, FDE replacements
       for  G:370, G:381 to G:387
C language programs, writing with the Frame API  G:ix, R:xv
C library header files, including in clients  G:8, G:367
callback functions
       client-defined  R:1
       defined  G:389
Cancel buttons
       of dialog boxes  G:196
       displaying  G:18
cells
       adding text to  G:122
       API representation of  G:333 to G:334, G:336
       custom shading and filling  G:333 to G:334
selecting, function for R:17, R:22, R:201 to R:202
selection state of G:286
straddling G:75 to G:79, G:337 to G:338
function for R:22, R:331 to R:332
illustrated G:338
as text frames G:332
unstraddling G:75 to G:79, G:337 to G:338
function for R:22, R:336 to R:337
illustrated G:338
See also rows
close bar properties. See document properties
close bars (for documents)
document properties for G:284
function for clearing R:7, R:58
close lists
document properties for R:566
channels G:377, G:389
functions for G:377, R:350
appending the contents of one to another R:350
closing G:377, R:351
determining offsets on R:358
flushing buffered output into R:352
indicating the ends of R:352
making temporary R:353
opening G:377, R:353
printing formatted output to R:424
reading G:377, R:356
reading bytes from R:427
reading formatted input R:436
returning the next byte of R:355
returning the size of R:357
setting offsets on R:357
writing to G:377, R:359
I/O channels (FDE) G:366
Character Catalog G:305
Character Catalog formats G:280, G:305 to G:309
API representation of G:280, G:306
applying G:306
to text G:124
creating G:306, G:308
as organized in documents G:309
overriding G:311
tags for G:305
See also Character Designer; text format; text properties
Character Designer, As Is settings G:306
cracter formats, applying to text G:124
cracter formats (of tagged characters), reapplying
Paragraph Catalog formats and G:306
cracter library (FDE) G:381, G:382
cracter strings. See strings
characteristics (of objects). See properties characters
FDE functions for G:381, G:382, R:360 to R:369
converting to lowercase G:382, R:368
converting to uppercase G:382, R:369
determining type of character G:382, R:360 to R:368
specifying special and higher ASCII G:121, G:316 to G:317, R:41
tagging G:306
characters
FDE functions for R:369
check marks, for menu items G:173
checkboxes
in dialog boxes G:199
getting/setting the state of, function for R:5
properties of G:199, R:533
tri-state G:199, R:537
child API objects
automatic deletion of G:158
declared G:389
as organized in frames G:147, G:295
child elements (structural) G:347
traversing G:121
child graphic objects. See child API objects
clearing text selections G:127, R:3, R:23, R:56 to R:58
client data, associating with structural elements G:110
client resources, opening, function for R:245
client text insets G:230, G:236 to G:238
creating G:236 to G:237
updating G:237 to G:238
See also text insets
client-defined callback functions R:1
client-defined dialog boxes, functions for R:1, R:5, R:93 to R:94
client_name, defined G:8
clients. See API clients
clipboard functions. See Frame Clipboard functions
Close Command, compared with F_ApiClose() G:67

closing
  channels G:377, R:351
dialog boxes G:227, R:5
documents and books G:66 to G:67, R:2, R:6,
  R:61 to R:62
file handles R:377
CMYK color model G:339
color models G:339
color views G:339
  bit positions representing (illustrated) R:520
colors G:281, G:338
  API representation of G:281, G:339
color models G:339
  as organized in documents G:339
properties of G:339, R:519 to R:520
columns. See table columns; text columns; text frames
combined font definitions
  properties for R:522
combined fonts
  angles, API specification of R:8
command shortcuts. See keyboard shortcuts
commands
  aborting G:37
adding to menu bars, function for R:75 to R:77
adding to menus G:26 to G:28
  functions for R:3, R:35 to R:37, R:75 to R:77,
  R:81
adding to multiple menus G:166 to G:167
API functions for R:3
API representation of G:161 to G:163
arranging on menus, function for R:35
in the client user interface G:24 to G:31
client-defined, responding to R:3
context-sensitive G:173 to G:175
creating G:158
  function for R:3
defining G:26 to G:28
  functions for R:75 to R:77, R:81 to R:82
determining existence of, function for R:3
getting IDs of G:163 to G:164
  function for R:3
linked lists of G:162
manipulating, with the API G:161 to G:176
properties of R:523 to R:528
removing, function for R:3
responding to user choices of G:29
See also Frame commands; Frame product
commands
committing the transaction (in querying and setting
facets) R:129
common graphic object properties R:592 to R:595
Compare Documents
  running G:297
  specifications for (illustrated) G:70
comparing documents and books G:68 to G:70
compiling
  API clients G:8
custom dialog boxes G:177
component documents. See book components
components (of books). See book components
condition tags G:309 to G:311
conditional text
  appearance of (illustrated) G:318
  API representation of G:281, G:309 to G:311
  as organized in documents G:311
  properties of G:284, G:309 to G:311, R:522,
  R:528
markers
  illustrated G:318
  prefixes for G:318
conditional text formats. See conditional text
conditional text markers. See conditional text
text conditions
  applying to text G:311
  properties for R:528
See also conditional text
consstants
defining in MetricT values R:675 to R:677
See also initialization constants; notification
  constants
See also property value types; and individual
consants in the FDK Reserved Names Index
container elements (structural) G:347
content rules (of structural element definitions) G:342
context rules G:344
context-sensitive commands G:173 to G:175
context-sensitive help
  for menu items G:175 to G:176
context-sensitive menu items G:173 to G:175
Continue buttons, displaying G:18
conventions, as used in this manual G:xii to G:xiv,
  R:xix to R:xxi
coordinate pairs
    structure for individual  R:683
    structure for sets  G:271, R:684
    See also  points (F_PointsT) properties
coordinates, measuring  R:683

copying
    API structures  R:4, R:69
    the contents of handles  R:372
    memory blocks  R:373
    properties  G:102, R:297
    text selections  G:127, R:3, R:23, R:68 to R:69

creating
    API clients  G:6 to G:12
    API objects  G:7, G:143 to G:145
    documents  G:53 to G:57, G:279
        functions for  G:53, G:56, R:6, R:70 to R:72, R:238 to R:244, R:325 to R:326
        the F_ApiInitialize() function  G:13, G:35
    cross-reference building blocks  G:322
    cross-reference formats  G:281, G:322 to G:323, R:531
        API representation of  G:281, G:322 to G:323
        illustrated  G:323
        properties of  G:322 to G:323, R:531
    cross-reference instances  G:323 to G:324, R:530
        properties of  G:323 to G:324, R:530

cross-references
    API representation of  G:281, G:323 to G:324
    API representation of the beginning of  G:314, R:167, R:687
    API representation of the end of  G:314, R:167, R:687
    creating  G:148 to G:149
    function for  R:209 to R:210
    illustrated  G:324
    properties of  G:323 to G:324, R:529 to R:531
    updating, function for  R:7, R:343
    See also  cross-reference formats; cross-reference instances

current directory, getting  G:96

current page, defined  G:291

custom dialog boxes  G:177 to G:227
    creating  G:177 to G:209
    using  G:211 to G:227
    See also  dialog boxes

custom documents, creating  G:55 to G:56, G:57

Custom Ruling and Shading dialog box  G:329

Customizing FrameMaker  G:31

cutting text selections  G:127, R:3, R:72 to R:73

D

dagger symbol (†), indicating read-only properties  R:495
dash patterns  R:595
    setting  R:294, R:595
data structures  R:677 to R:691
    copying, functions for  R:4, R:69
    functions applicable to  R:4
    MIF  G:384 to G:385
    See also  individual structures in the FDK Reserved Names Index
data types  G:3 to G:4, R:675 to R:677
    for object IDs  G:3, G:6, G:271, G:390
    primitive (C language)  G:6, G:367 to G:369, R:675
    FDE substitutes for  G:6, G:367 to G:369, R:675
    for property values (tables)  G:273
    See also  data structures; filepaths; IDs (of API objects); integers; MetricT values
    See also  primitive data types (C language); string lists; strings; and individual data types in the FDK Reserved Names Index
default button, of dialog boxes  G:196
default font properties
    of paragraph formats (FO_PgfFmt)  R:623 to R:625
    of paragraphs (FO_Pgf)  R:614 to R:616
default objects  G:279
default print settings  R:319
default rulings, for cells in tables  G:334
default scripts, creating and customizing  G:46
defined attributes  G:348
deleting
    API objects  G:158 to G:159, R:14, R:83 to R:90
    dialog box items  G:193
    facets  R:9, R:86
    files  G:377, R:372
    flows  G:158
    hash tables  R:392
    menu item separators  G:170
    page frames  R:84
    rows  G:158 to G:159, R:22, R:84, R:87 to R:88
    structural elements  R:337 to R:338
    table columns  G:158 to G:159, R:22, R:84
Index - Subjects

tables R:22, R:83

text frames G:158
text insets R:24
text selections R:23, R:88 to R:89
undefined attributes R:91
dialog box items. See dialog items

Dialog Box Properties dialog box G:193
illustrated G:194
dialog boxes G:17 to G:24, G:177 to G:227
adding dialog items to G:192
alert G:17 to G:18
API representation of G:211 to G:213
basic items in G:196
Cancel button G:196
capturing screenshots of G:206
client-defined, functions for R:1, R:5, R:93 to R:94
closing G:227
function for G:227, R:5
compiling G:177
creating G:185 to G:188
custom
creating G:177 to G:209
using G:211 to G:227
custom modal G:17
default button G:196
deleting dialog items from G:193

designing the layout of G:189 to G:193, G:207
displaying G:219
during initialization G:13
functions for G:7, G:219, R:5 to R:6,
R:53 to R:56
editing G:209
editing dialog box information G:179 to G:181,
G:209
file selection G:21 to G:22
first focus G:196
getting IDs of items in, function for R:5, R:94

handling special events G:226
handling user actions in G:221 to G:227
initializing items in G:218 to G:219
input G:18 to G:21
integer prompt G:18 to G:20
invoking by name G:185
keyboard shortcuts (Windows) G:202
labels in G:201

on Macintosh computers G:180 to G:181,
G:189 to G:190, G:207, G:209
manipulating G:190 to G:192
metric prompt G:18 to G:21
modal G:17, G:177, G:197, G:205 to G:206, G:211,
G:219, R:206 to R:207
modeless G:17, G:205 to G:206, G:219,
R:208 to R:209
modifying G:191 to G:192
moving G:191
multiple modeless G:225 to G:226
for multiple platforms G:184
multiple-interaction G:223, G:224 to G:225
OK button G:196
platform-specific views of G:180 to G:181
predefined R:5
properties of G:208, R:531 to R:537
setting G:193 to G:197, G:208
resizing G:191
saving dialog box information G:181 to G:185
scroll bars in G:201
scroll list G:22 to G:24, R:5
scroll lists in G:202
single-interaction G:223
specifying item order in G:194 to G:195
specifying position G:196
specifying size G:196
specifying the focus order G:194 to G:195
string prompt G:18 to G:20
testing G:205 to G:206, G:209
title label G:196
on UNIX platforms G:180 to G:181,
G:189 to G:190, G:207
updating items in G:220
on Windows platforms G:180 to G:181,
G:189 to G:190, G:207

See also dialog items; dialog resources; DRE (Frame
dialog resource) files

Dialog Item Properties dialog box G:197, G:209
dialog items G:196
adding to dialog boxes G:192
boxes G:198
buttons G:199
checkboxes G:199
deleting G:193
disabling G:219
IDs of G:212 to G:213
initializing G:218 to G:219
labels G:201
manipulating G:190 to G:192
modifying G:191 to G:192
moving G:191
multiline text boxes G:200
properties of G:211
    getting G:213
    setting G:213
radio buttons G:199
resizing G:191
scroll bars G:201
scroll lists G:202
setting properties of G:197 to G:204, G:208
specifying order in dialog boxes G:194 to G:195
as stuff items G:203 to G:204, G:213
text boxes G:200
triboxes G:199, R:537
updating G:220
See also dialog boxes; dialog resources
dialog resources G:212 to G:213
    opening G:217 to G:218
See also dialog boxes; dialog items
dictionaries (document) G:284
    adding words to R:303 to R:305
directories
    functions for
        allocating and returning handles for R:380
        creating R:400
        getting current G:96, R:24, R:59 to R:60
        renaming R:434
        returning file path for R:421 to R:422
        returning file path of parent R:381
        returning identification information R:388
        returning information about R:381
        returning the next file in R:379 to R:380
disabling the API G:16
document dictionaries. See dictionaries (document)
document properties
Adobe Acrobat R:541
change bar G:284, R:549
condition G:284, R:550
equation G:284, R:550 to R:551
flow R:559
footnote G:284, G:327, R:557 to R:559
general R:546 to R:548
getting and setting G:97 to G:99
hypertext R:551
menu bar R:557
object pointer G:282 to G:284, R:538 to R:540
page G:284, R:559 to R:561
print G:284, R:561 to R:563
selection R:564
table footnote G:284, R:572
type-in G:126, G:284, G:391, R:573 to R:575
typographic R:576
view G:284, G:288, R:576 to R:578
view-only R:578 to R:579
See also documents
document reports G:5
    bailing out G:15
    initializing G:15
document structure properties G:284, R:566
document window, scrolling to a specified text
    range R:276 to R:277
document windows, manipulating G:97
documents G:279 to G:348
    active G:10, G:83, G:274, G:275
    adding body pages to G:11, G:289
    API representation of G:275, G:288
    clearing change bars in, function for R:7
closing G:66 to G:67
    function for R:6, R:61 to R:62
comparing G:68 to G:70
    function for R:6, R:65 to R:66
destroying items in G:143
delaying reformattting of G:97
formatting G:70 to G:74, G:96
    functions for G:77, R:7, R:25
disabling items in G:143
functions applicable to R:6 to R:7
generating the pathname of G:97
importing formats from G:72
    function for R:7, R:192 to R:197, R:323 to R:325
importing Frame product documents
    main flow of G:232
properties for (table) R:124
    information included in (table) G:280 to G:281
landscape G:57
making active G:95
making invisible G:98
master pages for G:289, G:292
next, API indication of G:83
open
  API indication of G:83, G:275
  getting the IDs of G:83 to G:84
    illustrated G:275 to G:276
opening with F_ApiOpen() G:48 to G:53
  function for R:2, R:238 to R:244
opening with F_ApiSimpleOpen() G:46 to G:47
  function for R:2, R:326 to R:327
organization of API objects in G:82, G:84, G:282 to G:284
organization of graphic objects in G:295 to G:296
organization of pages in G:291
  illustrated G:292
organization of paragraphs in G:304
as organized in books G:351
as organized in sessions (illustrated) G:276
portrait G:57
printing G:57 to G:59, R:319
  document properties for G:284, R:561 to R:563
    function for R:6, R:319 to R:320
properties of R:537 to R:579
redisplaying, function for R:7, R:263 to R:264
reformatting G:70 to G:74, G:96
  functions for G:77, R:7, R:25
refreshing G:96
rehyphenating words in, function for R:7, R:265
saving G:59 to G:66, G:279
  functions for R:6, R:272 to R:274, R:327 to R:328
selection state, properties for G:285 to G:286
setting status bars G:98
setting title bars G:98
structured G:342
Table of Contents G:289
updating G:70 to G:74
  functions for G:77, R:7
See also document properties
DOSS platforms, memory management on G:378 to G:379
double-byte strings
  FDE functions for
    determining if a prefix of another string R:463
    determining if a suffix of another string R:464
double-precision numbers, converting strings to R:492
double-sided documents, master pages for G:289, G:292
draw order G:100, G:293, G:389
  grouping and G:296
  moving graphic objects in G:100
DRE (Frame dialog resource) files G:179
  capturing screenshots of G:206
  contents of G:186
  creating G:185 to G:188
  deleting items from G:207
  for different platforms G:181 to G:185
  editing G:179, G:209
  editing dialog box information G:179 to G:181, G:209
example of G:206
  and Frame binary files G:178
  keyboard shortcuts for G:209
  opening G:179
  and platform-specific views of dialog boxes G:180 to G:181
  saving dialog box information G:181 to G:185
  setting properties of G:193, G:208
  testing G:205, G:208
See also dialog boxes; dialog items
DSE (direct straight exit) function, setting R:439

E

Edit Condition Tag dialog box, illustrated G:310
Edit menu
  adding menus to G:28
  getting the ID of G:26
    name used to specify G:25, R:76
Element Catalog G:342
  function for getting entries G:90
  structure for entry sets G:271, R:682
  structure for individual entries R:682
Element menu, name used to specify G:25
ellipse properties  R:598
encoded text  G:132 to G:142
  encodings for fonts  G:135 to G:137
  encodings for the current session  G:137
  in text items  G:140
  initializing encoding data  G:134 to G:135
  inspecting and manipulating  G:138 to G:139
  parsing  G:139 to G:140
  Setting the U/I encoding  G:134 to G:135
special issues  G:140 to G:142
  decimal tabs  G:141
  ghost encodings  G:141
  internal strings  G:141
  offsets into strings  G:141
  using encoding data  G:134 to G:137
equal sign (=), conditional text marker prefix  G:318
equation properties.  See document properties
equations  G:293
  document properties for  G:284, R:550 to R:551
  properties of  R:603
  resetting settings in, function for  R:7, R:266
error codes.  See API error codes
  handling  G:45
  See also API error codes; status flags
Esc shortcuts  G:26
Escape symbol (\!)  G:26
events
  initiated by API clients, responding to  G:40
  monitoring by the FrameMaker product of  G:43
  notification points in  G:34 to G:36, R:227 to R:232
  requesting notification of  G:13, G:34 to G:36
  function for  R:25, R:226 to R:233
  responding to notification of  G:15, G:34, G:36 to G:42
  function for  R:1, R:233 to R:237
exclusions (in content rules), lists of  G:342
executable name, defined  G:8
  exiting clients  G:378
  FDE function for  R:374
  See also bailing out (of API clients)
extendomatic menu items, manipulating  G:172 to G:173
export filters.  See filters
expressions, testing if true, function for  R:348
  See also graphic insets

F
F_ naming convention  G:xii, R:xix
F_Api naming convention  G:xii, R:xix
F_ApiSetElementRange() function  G:120
F_Char naming convention  G:382
F_Hash naming convention  G:383
F_Metric naming convention  G:384
F_Mif naming convention  G:387
f_stdio.h header file  R:27, R:45
f_str.h header file  G:367
f_types.h header file  G:367
F_UBytesT facets
  getting  G:265
  function for  G:265, R:9, R:183 to R:184
  setting  G:265 to ??, R:316
  function for  G:265, R:9, R:316 to R:317
FA_Init naming convention  G:xiii, R:xix
FA_Note naming convention  G:xiii, R:xix
facets
  client-specific  G:258
  deleting, function for  R:9, R:86
  getting/setting, functions for  G:263
  identifying  G:272
  of imported graphics  G:257 to G:259
  querying  R:183
  functions for  R:129, R:133, R:183
facilities (of the FDE virtual environment)  G:366
fapi.h header file  G:8, G:370, R:27
fassert.h header file  G:378
fcharmap.h header file  G:382
  for common user actions (tables)  G:74
  executing  G:74 to G:75
  function for  R:7, R:101
  and invisible documents  G:98
f codecs.h file  G:39
FDE (Frame Development Environment)  G:x, G:365 to G:387, R:xvi
  components of  G:365
  illustrated  G:365
I/O channels  G:366
initializing  G:366, G:375, R:374
memory handling G:366
as related to API clients (illustrated) G:365
replacements for C language library functions G:366, G:381 to G:387
specifying filepaths G:375 to G:377
utility libraries G:366, G:381 to G:387
virtual environment G:366 functions G:375 to G:379
See also FDE clients; FDE functions
FDE clients
function for displaying warning messages G:379, R:488
function for exiting G:378, R:374
function for reporting progress in G:379
including header files in G:367
FDE data types. See data types
FDE filters, example G:370 to G:374
FDE functions G:375 to G:379, G:381 to G:387
assertion handler G:378, R:348
channel G:377, R:350
character G:381, G:382
for replacing C library functions G:370
hash G:381, G:383
header file containing G:367
I/O (filepath) G:375 to G:377
I/O (read/write) G:366, G:381, G:382
memory allocation and deallocation G:378 to G:379
metric G:366, G:384, R:675 to R:677
naming convention for G:xii, R:xix
for replacing I/O and memory calls G:369
simple MIF G:381, G:387
string G:381
string list G:381, G:382
virtual environment functions G:375 to G:379
See also individual FDE functions in the FDK
Reserved Names Index
FDE header files
including in FDE clients G:367
See also individual FDE header files
FDE I/O utility functions
for handling filepaths G:375 to G:377
for reading and writing data G:366, G:381, G:382
for replacing common I/O functions G:369
FDE utility libraries G:366, G:381 to G:387
functions provided in G:381 to G:387
FDE utility library functions G:366, G:381 to G:387
FDE virtual environment G:366
FDK (Frame Developer’s Kit) G:ix, R:xv
documentation G:xi to G:xii, R:xvii
FDK clients, writing with the Frame API G:ix, R:xv
FDK Platform Guide G:xii, R:xviii
FDK Programmer’s Reference, summary of contents G:xii, R:xviii
fdhash.h header file G:383
File menu
name used to specify G:25, R:76
view-only menu bar G:161
file operations, functions for (table) R:6
file selection dialog boxes
displaying G:21 to G:22
function for R:5
illustrated G:22
file types, checking during Open operations G:41
filenames
specifying in opening R:326
See also filepaths
filepaths
converting pathnames to G:375 to G:377
converting to pathnames G:375
basenames for R:376
converting DOS-specific to FDE G:375
converting FDE filepaths to UNIX pathnames G:375
converting Macintosh-specific to FDE G:375 to G:377
converting MIF platform-independent to FDE G:375 to G:377
converting platform-specific to platform-independent G:375, R:422 to R:423
converting UNIX-specific to FDE G:375 to G:377
creating from pathnames G:375 to G:377, R:422 to R:423
filepath information R:383 to R:384
freeing resources associated with G:375, R:378
guessing path type R:424
platform-dependent R:382 to R:383
platform-independent G:375 to G:377, R:387, R:422 to R:423
returning a copy of a specified filepath R:378
returning file or directory identification information R:388
manipulating G:377
files
checking the Open status of G:51 to G:52
checking the Save status of G:64 to G:66
DRE G:179
functions for
deleting G:377, R:372
executing R:107
importing text R:126 to R:127
renaming R:434
returning file path for R:421 to R:422
returning identification information R:388
returning information about R:381
table of operations on R:6
generating for books G:71
header G:8, G:367
imported G:229 to G:235
importing, text and graphics G:230 to G:235
MIF G:x, G:xi, R:xvi, R:xvii
opening G:41 to G:42, R:326
saving G:35
uncompressing G:42
filetype hint strings
for importing and exporting text files G:254
printing out a list for installed filters G:255
syntax for G:249
fill patterns R:595
illustrated R:595
setting G:11
filters G:5
bailing out G:15
compiling and running G:373 to G:374
example FDE G:370 to G:374
for graphic insets G:241 to G:250, R:600 to R:602
initializing G:15
making portable
example G:370 to G:374
registering G:15
requesting notification for G:36
find and replace, API function for R:23
fiutils.h header file G:382
first focus, of dialog boxes G:196
first initialization G:14
flags
for alert boxes (table) G:18
for Frame Clipboard functions G:128 to G:129
table of G:128
for FS_ImportStatus (table) R:195 to R:196
naming convention for G:xiii, R:xix
for text item types G:111
tables of G:315, R:167
See also status flags; and individual flags in the FDK Reserved Names Index
flow properties. See document properties
flows G:280, G:297 to G:300, G:389
API representation of G:280, G:298
API representation of the beginning of G:313, R:166, R:686
API representation of the end of G:313, R:166, R:686
Autoconnect setting for G:297
creating G:145, G:150
deleting G:158
document properties for R:559
HIDDEN G:292, G:318
illustrated G:297, G:298, G:304 to G:305
main G:297
importing G:232
organization of paragraphs in G:304
as organized in documents G:298 to G:300
illustrated G:298
special G:289
structure properties of G:581
structured G:300
See also text frames
fmbatch utility G:x, R:xvi
advantages and limitations of G:xi, R:xvii
fmemory.h header file G:379
fmetrics.h header file G:384
fmifuext.h header file G:387
FO_ naming convention G:xiii, R:xix
focus order, of a dialog box G:194 to G:195
font encoding
functions for
getting the encoding for a given ID R:385
getting the ID of a named encoding R:384
getting
function for R:117 to R:118
font encoding data
  initializing R:375
font encoding for dialog boxes R:375
font family encoding
  getting function for R:115
fonts
  angles, API specification of G:276 to G:277, R:8
  API representation of G:306 to G:308
  API specification of G:276 to G:277, R:8
conventions used in this manual G:xiv, R:xx
families, API specification of G:276 to G:277, R:8
  names
    determination of G:306 to G:308
    platform G:307 to G:308
    PostScript G:307 to G:308
    specifying G:306 to G:309
text encoding, API specification of R:8
variations, API specification of G:276 to G:277, R:8
weights, API specification of G:276 to G:277,
  G:316, R:8
footnote anchors
  API representation of G:312, G:313, R:166, R:687
  defined G:389
  illustrated G:328
Footnote command G:327
Footnote Properties dialog box G:327
footnote references. See footnote anchors
footnote separators G:327
footnote text frames G:327
footnotes G:281, G:327 to G:328
  API representation of G:281, G:327 to G:328
document properties for G:284, G:327, R:572
footnote separators G:327
  illustrated G:328
properties of G:327 to G:328, R:581 to R:582
text frames for G:327
See also footnote anchors
forking a new process, function for R:107
format change list properties R:582 to R:589
  adding G:106
  getting and setting G:106
  removing G:107
format change lists G:346
  adding properties G:106
  advanced properties R:582
  API representation of G:346
Asian character spacing properties R:583
autonumber properties R:584
basic properties R:584
changing G:106
creating G:152
  functions for R:221 to R:224
font properties R:585
general properties R:582
getting the IDs of named R:136
pagination properties R:588
properties of R:582 to R:589
removing properties G:107
table cell properties R:589
Format menu, name used to specify G:25, R:76
format rule clause properties
  setting G:107
format rule clauses G:344
  API representation of G:345
  creating G:152
    functions for R:221 to R:224
  setting properties G:107
format rules G:344
  API representation of G:345
  creating G:152
    functions for R:221 to R:224
format rules (of structural element definitions) G:342
  specifying G:342
formats
  copying G:102, R:297
  importing from documents G:72
  function for R:7, R:323
See also Character Catalog formats; conditional text, formats; cross-reference formats; Paragraph Catalog formats; properties; ruling formats; Table Catalog formats; variable formats
formatted anchored objects
  creating G:148 to G:149
  function for R:209 to R:210
formatting documents G:70 to G:74, G:96
  functions for G:77, R:7, R:25
FP_ImportHint property (FO_Inset)
  syntax of FP_ImportHint strings G:249 to G:255
FP_TintPercent G:341
fprogs.h header file G:379
Frame Above setting (Paragraph Designer) G:289,
  G:390
frame anchors, API representation of G:312, G:313,
  R:166, R:687
Frame API. See API (Frame Application Program Interface)
Frame Application Program Interface. See API (Frame Application Program Interface)
Frame Below setting (Paragraph Designer) G:289, G:390
Frame binary documents, importing as text G:229
Frame Clipboard functions G:127 to G:129
   flags for G:128 to G:129
   table of G:128
   manipulating the Clipboard stack G:129, R:248, R:261 to R:262
   saving contents of Clipboard G:129
Frame commands
   replacing with your own commands G:30
   See also commands
Frame Development Environment. See FDE (Frame Development Environment)
Frame dialog resource (DRE) files. See DRE (Frame dialog resource) files
Frame product commands
   functions for executing G:7, G:40, G:45 to G:79
   See also commands; Frame product operations
Frame product documents, properties for importing (table) R:124
Frame product menu bars. See menu bars
Frame product menus
   getting IDs of G:24 to G:26
   replacing with your own menus G:30
   setting menu item labels G:171
   table G:25
   See also menus
Frame product operations
   initiating G:40
   notification in G:34 to G:36
   Open G:41 to G:42
   responding to G:34 to G:42
   Save G:41, G:60
   suspending, functions for R:17, R:328, R:344 to R:345
   See also Frame product commands
Frame product sessions. See sessions
Frame products
   defined G:xiii, R:xix
Frame tools
   advantages and limitations of (table) G:xi, R:xvii
   choosing G:xi, R:xvii
FrameBuilder
   documents
      importing element definitions from G:77
      manipulating elements in G:76 to G:77
FrameImage facets G:257, G:259, R:183, R:316
FrameMaker G:258
FrameMaker product
   client initialization by G:5
   events monitoring by G:43
   getting the name of current product interface G:94
   languages available in R:635
   list of products R:636
   regaining control from, function for R:25, R:334
   registering API clients with G:8 to G:9, G:12
   returning control to, function for R:96
   servicing calls to the API, function for R:277 to R:278
frames
   anchored G:293, G:294, G:389
   API representation of G:295
   lists of graphic objects in G:295
   nesting G:293
   organization of graphic objects in G:295 to G:296
   illustrated G:295
   properties of G:295
   unanchored G:293, R:608
   See also anchored frames; text frames
FrameVector facets G:257
FS_UpdateBookStatus property (F_ApiUpdateBook())
   status flags of R:339
fstrings.h header file G:381
fstrlist.h header file G:382
functions. See API functions; FDE functions
G
   general rules (in content rules) G:342, G:347
   general structural element properties R:638
   geometric shapes, defined G:293
Gets R:164
   getting
      API object types G:84
      attribute definitions R:108 to R:110
      attributes R:110 to R:112
IDs G:82 to G:89, G:163 to G:164, G:270
the insertion point G:114 to G:117
pathnames G:97
property lists G:92 to G:94
text G:111 to G:113
  in text ranges G:118
text properties G:123 to G:124, G:317
text selections G:114 to G:117
UIDs G:89, G:270
valid elements R:112
global document information G:280, G:284 to G:288
  API representation of G:280, G:285 to G:288
  defined G:390
  saving G:284, G:285
global variables
effect of bailing out on G:44
  for error conditions G:45, R:693
graphic inset editors G:256 to ??
  responding to user launching of G:262
  See also graphic insets
graphic insets G:230, G:256 to ??
  client-specific facets G:258
  creating G:261
    function for G:256, R:15
  external G:230, G:260 to G:261
  facets of G:257 to G:259
  filters for G:241 to G:250, R:600 to R:602
  functions applicable to R:9
  internal G:230, G:259 to G:260
  live links to graphic inset editors G:256
  modifying G:261
    getting/setting G:263 to ??
  responding to user clicking of, function for R:9
  responding to user double-clicking of G:262 to G:263
  See also graphics
graphic objects G:280, G:293 to G:297, R:592 to R:608
  API representation of G:280, G:294 to G:297
  API representation of selections of G:296
  changing size/location within a frame G:99
  creating G:146 to G:148
    function for R:14, R:216 to R:218
  default properties of G:147
defined G:294, G:390
drawing G:293 to G:294
  in frames G:293
  getting the IDs of selected G:87
    function for R:17, R:119 to R:120
  grouping G:293, G:296
  manipulating G:87
  moving in the draw order G:99 to G:102
  moving to different frames or pages G:101
  as organized in documents G:84, G:295 to G:296
  as organized in frames G:295 to G:296
    illustrated G:295
  on pages G:293
  common R:592 to R:595
  copying between graphic objects G:102
  getting and setting G:99 to G:102
  selecting G:296
  traversing lists of G:84
  See also graphic insets; groups (of graphic objects); objects
graphics G:293
  displaying imported G:258
  importing G:230 to G:233
    filters for G:241 to G:250, R:600 to R:602
    function for R:192 to R:197
  printing imported G:258
  properties of imported G:247 to G:250, R:125, R:599 to R:602
  See also graphic insets
Graphics menu
  name used to specify G:25, R:76
  using commands with dialog boxes G:191 to G:192
Group command (Graphics menu), grouping dialog box items G:191
grouping
dialog box items G:191
graphic objects G:101
groups (of graphic objects)
  API representation of G:296
  defined G:293
  properties of G:296, R:599
  resizing G:293
H handles
allocating and deallocating memory with G:379, R:432
allocating and returning, for directories R:380
allocating new G:379
clearing the data of G:379, R:370
closing a file handle R:377
copying the contents of R:372
determining if contents equal R:389
determining size of R:389
freeing memory allocated to G:379, R:386
locking R:399
resetting R:435
returning size of R:389
unlocking R:488
using in memory management G:378 to G:379
hash functions G:381, G:383, R:390 to R:397
hash library (FDE). See hash functions
hash tables
functions for R:390
adding entries to G:383
creating G:383, R:390
deleting R:392
enumerating entry keys and data R:393
finding and printing specified entries R:395
getting entries from G:383, R:394
removing R:394
header files
including in clients G:8, G:367
See also individual header files
help. See context-sensitive help
Help menu, name used to specify G:25, R:76
hidden conditional text, API representation of G:318
hidden pages G:289, G:292, G:318
API representation of G:289
defined G:390
properties of G:289, R:611
HIDEN text flows G:292, G:318
hiding text G:309 to G:311
HLS color model G:339
hypertext commands G:31
setting up G:32
simulating user-invoked R:192
hypertext markers
illustrated G:33
responding to user clicking G:32, G:32 to G:34, R:192
function for R:1, R:206
hyphenation. See paragraph formats (FO_PgfFmt), hyphenation properties of; paragraphs (FO_Pgf), hyphenation properties of
I, J I/O calls
making portable G:367, G:375 to G:379
replacing with FDE functions G:369
I/O channels. See channels
ID (ObjHandleT) properties
function for getting R:15, R:119 to R:120
function for setting R:16, R:285
identifiers. See IDs (of API objects); UIDs (unique IDs)
identifying objects G:269 to G:270
IDs (of API objects) G:3
for active documents or books G:275
data type for G:3, G:6, G:271, G:390
defined G:390
for dialog boxes R:5, R:94
for dialog items G:212 to G:213
of Frame product menus G:24 to G:26
getting G:82 to G:89, G:269 to G:270
for graphics G:285
of menu bars G:24 to G:26
for tables G:285
See also ID (ObjHandleT) properties; UIDs (unique IDs)
illegal values, protection against G:81
image pop-up menus
in dialog boxes G:200
properties of G:200, R:534
import filters. See filters
imported graphics. See graphic insets
importing
element definitions G:77
formats from documents G:72
function for R:192 to R:197
main flow of a Frame document file G:232
properties for import operations
(tables) R:120 to R:127
Index - Subjects

text and graphics G:230 to G:235
See also graphic insets; insets; text insets
importing, element definitions G:77
inclusions (in content rules), lists of G:342
indentation. See paragraph formats (FO_PgfFmt), indentation properties of; paragraphs (FO_Pgf), indentation properties of indexes, generating G:297
initialization callback function G:7, G:10, R:198 to R:199
creating G:13, G:35
definition of G:13
initialization calls G:5
responding to G:13
function for R:1, R:198 to R:199
initialization constants
naming convention for G:xiii, R:xix
tables of G:14, R:198
initialization flags. See initialization constants
initialization types G:14 to G:16, R:198
initializing
after bail out G:15
of document reports G:15
FDE (Frame Development Environment) G:366, G:375, R:374
FDE font encoding data R:375
of filters G:15
first G:14
preventing G:16
take-control G:14
types of G:14 to G:16, R:198
See also initialization callback function; initialization calls; initialization constants
input, prompting users for G:7, G:17 to G:24
input dialog boxes
displaying G:18 to G:21
See also dialog boxes
input/output calls. See I/O calls
Insert Table dialog box, illustrated G:331
insertion point
getting G:114 to G:117
function for R:181 to R:182
selection state of G:285, G:287
setting G:114 to G:117
function for R:17, R:313 to R:314
text containing G:317
inset facets. See facets
insets G:229 to ??, G:293
defined G:293, G:390
properties of G:247 to G:250, R:599 to R:602
responding to users clicking, function for R:206
See also facets; graphic insets; text insets
integer (IntT) properties
function for getting R:16, R:128 to R:129
function for setting R:16, R:287 to R:288
integer array (F_IntsT) properties
function for getting R:15, R:131 to R:132
function for setting R:16, R:289 to R:290
integer arrays
structure for G:271, R:682
See also integer array (F_IntsT) properties
integer facets
function for getting R:9, R:129 to R:130
function for setting R:9, R:288 to R:289
integer prompt dialog boxes
displaying G:18 to G:20, R:257
function for R:5, R:257 to R:258
illustrated G:20, R:258
integers
data type for G:3, G:271, G:367, G:675
FDE function for converting and appending to strings R:446, R:493
prompting users for G:18 to G:20, R:257
function for R:5, R:257 to R:258
See also integer (IntT) properties; integer arrays
interactive mode, in simple functions G:45, G:47
interfaces, setting up for API clients G:7, G:17 to G:44
internal graphic insets G:230, G:259 to G:260
See also graphic insets
IntT facets. See integer facets
invisible documents G:98

K
keyboard shortcuts G:26
adding to menus G:26
in dialog boxes (Windows) G:202
for dialog editing G:209
prefix for coding G:26
L

labels
  adding to dialog box items G:193
  of checkboxes G:199
  of client-defined menu items G:172
  in dialog boxes G:201
  of Frame product menu items G:171
  of menu items G:170 to G:172
  properties of R:534
  of radio buttons G:200
landscape documents, creating G:57
language strings
  functions for R:397
  table of R:397
languages. See Frame products, languages available in;
  language strings; paragraph formats
  (FO_PgfFmt), language property of
languages. See paragraphs (FO_Pgf), language properties of
Left master page G:292
level rules G:344
libfde.a library file G:367
libraries. See FDE utility libraries
line breaks, specifying G:121
line properties R:602
line spacing. See paragraph formats (FO_PgfFmt), line spacing property of;
paragraphs (FO_Pgf), line spacing property of
lines
  dashed R:595
  properties of R:602
  See also text lines
linked lists
  of commands G:162
  in Frame product architecture G:82
  of menu items G:162
  of menus G:162
  order of objects in G:84
lists. See linked lists
little-endian byte order G:382, R:435, R:438
lock count (for handles) R:399
longs (4 bytes)
  FDE function for reading (and swapping) R:428 to R:429
  FDE function for writing (and swapping) R:490

M

Macintosh computers
  compiling and running FDE filters G:373
dialog boxes on G:180 to G:181, G:189
disabling clients G:16
image pop-up menus G:200
memory management on G:378 to G:379
registering clients with the FrameMaker
  product G:8, G:12
  saving global document information G:285
take-control initialization G:15
main flows G:297
makefiles G:8
Maker Interchange Format. See MIF (Maker Interchange Format)
manipulating
  asian text G:131 to G:142
document windows G:97
  expandomatic menu items G:172 to G:173
  menu item separators G:169 to G:170
  properties G:89 to G:95
  property lists G:93 to G:94
  selected graphic objects G:87
text G:111 to G:129
marker anchors
  API representation of G:312, G:314, R:166, R:687
  illustrated G:320
marker types
  API representation of G:281
  properties of R:609
Marker window G:319
markers G:281, G:319 to G:320
  Adding marker types to documents G:321
  API representation of G:281, G:319 to G:320
  conditional text (illustrated) G:318
  creating G:148 to G:149
  functions for R:14, R:210 to R:212
Mapping old marker types to named marker
  types G:321
  properties of G:319 to G:320, R:608
  The standard list of marker types G:321
  See also hypertext markers
master pages G:289, G:292
  API representation of G:289, G:292
  creating. See named objects, creating
default G:292
  properties of G:290, G:291 to G:292, R:612
Index - Subjects

master pointer  G:378
  See also  pointers
math equations.  See  equations
memory
  allocating, functions for  R:32 to R:33, R:45 to R:46, R:431
  allocating blocks of  R:32, R:349, R:431
  allocating for properties  G:94
  allocating for property lists  G:50
    API function for  R:11, R:45 to R:46
  allocating for text items, API function for  R:46
  allocating with handles, FDE functions for  G:379, R:32 to R:33
  allocating with pointers, FDE functions for  G:379, R:373
  API functions applicable to  R:11, R:32 to R:33, R:45 to R:46, R:74
  comparing contents of blocks, function for  R:427
  deallocating, functions for  R:74
  deallocating from properties  G:94
    function for  R:74
  deallocating from property lists, API function for  R:11, R:74
  deallocating from text items  G:112
    API function for  R:11
    FDE functions applicable to  G:378 to G:379, R:349, R:370
  freeing from clients, API function for  R:11, R:24, R:48
  freeing from handles, FDE function for  G:379, R:370, R:386
  freeing from pointers, FDE function for  G:379, R:370, R:386
  making calls portable  G:367, G:378 to G:379
  preventing the relocation of, FDE function for  R:399
  See also  handles; memory blocks (associated with pointers); memory management
memory blocks (associated with pointers), FDE functions for  G:379, R:370, R:373
  allocating  G:379, R:32
  clearing  G:379, R:370
  comparing  R:427
  copying  R:371
  duplicating  R:372
  freeing  G:379, R:386
  reallocating  R:431
memory calls
  making portable  G:367, G:378 to G:379
  replacing with FDE functions  G:369
memory management
  on DOS platforms  G:378 to G:379
  on Macintosh computers  G:378 to G:379
  on UNIX platforms  G:378 to G:379
  using handles and pointers  G:378 to G:379, R:370, R:372
menu bars
  adding commands to, function for  R:75 to R:77
  adding menus to  G:13, G:27, R:79
    function for  R:3, R:37 to R:39, R:82
  arranging menus on  G:166 to G:170, R:3
determining if a menu is on, function for  R:203 to R:204
  getting IDs of  G:24 to G:26, G:27
  strings specifying (tables)  G:25
  view-only  ?? to G:162
menu customization file, loading, function for  R:3, R:200
menu item separators
  adding, moving, and deleting  G:170
  creating  G:158
  creating separator objects  G:170
  manipulating  G:169 to G:170
  properties of  R:523, R:528
menu items
  adding to menus  G:26 to G:28, R:35
    function for  R:35
  arranging  G:166 to G:170, R:3
  check marks  G:173
  in the client user interface  G:24 to G:31
  context-sensitive  G:173 to G:175
  context-sensitive help for  G:175 to G:176
  creating menu item separators  G:158
determining if on menu, function for  R:203 to R:204
  expandomatic  G:172 to G:173
  getting labels  G:170 to G:172
  getting properties of  G:162
  linked lists of  G:162
  manipulating menu item separators  G:169 to G:170
  properties of  R:523 to R:528
  removing  G:167
    function for  R:3
  reordering  G:168 to G:169
  responding to user choices of  G:29
    function for  R:1, R:64
  setting properties of  G:162
  shift  G:174 to G:175
menu properties. See document properties
menus
  adding commands to G:26, G:27 to G:28, R:3
    functions for R:35 to R:37, R:75 to R:77, R:81
  adding keyboard shortcuts to G:26
  adding menu items to G:26, G:27 to G:28, R:35
    function for R:35
  adding pull-right menus to G:27
  adding to menu bars G:13, G:27, R:37
    functions for R:3, R:37 to R:39, R:77 to R:80, R:82
  adding to other menus G:27
    functions for R:3, R:37 to R:39, R:77 to R:80, R:82
  adding to pop-up menus G:27
API functions applicable to R:3
API representation of G:161 to G:163
arranging G:166 to G:170, R:3
arranging commands in, function for R:35
arranging menu items in G:166 to G:170, R:3
changing the menu set G:169
checking for the existence of, function for R:3, R:203
in the client user interface G:24 to G:31
creating G:158
  functions for R:3
defining G:27
determining configuration of G:164 to G:166
determining existence of, function for R:3
determining if a menu item is on, function for R:203 to R:204
determining if on menu bar, function for R:203 to R:204
getting IDs of G:24 to G:26, G:27, G:163 to G:164
  function for R:3
getting properties of G:162
linked lists of G:162
loading custom menu files G:169
manipulating, with the API G:161 to G:176
  multiple, adding commands to G:166 to G:167
  properties of R:523, R:528
pull-down G:27
removing G:167
  function for R:3
reordering G:168 to G:169
setting properties of G:162
See also Frame product menus
messages, handling G:45
metric (MetricT) properties
  function for getting R:16, R:132 to R:133
  function for setting R:16, R:291 to R:292
metric facets
  function for querying R:9, R:133 to R:134
  function for setting R:9, R:292 to R:293
  comparing metric numbers R:401
  computing square roots R:406
  computing squares R:406
  constraining angles R:401
  constructing metric numbers from fractions R:404
  converting metric numbers to real numbers R:407
  converting real numbers to metric numbers R:403
  dividing metric numbers R:402
  multiplying metric numbers R:403, R:405
  normalizing angles R:405
metric library (FDE). See metric functions
metric prompt dialog boxes
  displaying G:18 to G:21, R:258
    function for R:5, R:258 to R:260
  illustrated G:21, R:260
metric values. See MetricT values
metrics (F_MetricsT) properties
  function for getting R:16, R:135 to R:136
  function for setting R:16, R:293 to R:295
MetricT facets. See metric facets
MetricT functions. See metric functions
MetricT values G:271, G:384, R:675
  defining constants in R:675 to R:677
  functions for G:366, G:384, R:675 to R:677
  converting real numbers to G:384
  converting to real numbers G:384
  for linear measurements (table) R:676
  for percentages and angles R:676
  prompting users for G:18 to G:21, R:258 to R:260
    function for R:5, R:258 to R:260
  specifying in code R:675 to R:677
  structure for G:271, R:683
See also metric (MetricT) properties; metrics (F_MetricsT) properties
MIF (Maker Interchange Format) G:x, G:390, R:xvi
  advantages and limitations of G:xi, R:xvii
  importing MIF files as text G:229
See also MIF functions; MIF statements; MIF write output channel
Index - Subjects

MIF <MathFullForm> statement R:603
MIF data structures G:384 to G:385
  setting fields in G:385
MIF files G:x, G:xi, R:xvi, R:xvii
  simple G:381, G:385, R:418
MIF library (FDE) G:385 to G:387, R:418
  See also MIF functions; MIF write output channel
MIF macros G:384 to G:385
MIF statements
  functions for G:385
    ending R:410
    starting new R:408
  substatements in G:385
  See also MIF write output channel
MIF write output channel, functions for G:386
  indenting R:411 to R:413
  returning the current indent level R:411
  setting G:386, R:415
  setting the indent level G:386, R:414
  writing a blank space to R:416
  writing a comment string to R:408
  writing a decimal to R:409
  writing a new line to R:414
  writing a tab space to R:416
  writing a text string to R:417
  writing an integer to R:413
MifUnitT values G:386, R:409
minus sign (−), conditional text marker prefix G:318
mmlimport filter G:5
  displaying, function for G:219
  displaying dialog resources as, function for R:206
  handling user actions in G:223 to G:225
  multiple-interaction G:223, G:224 to G:225
  single-interaction G:223
  testing G:205
modeless dialog boxes G:17, G:205 to G:206
  displaying, function for G:219
  displaying dialog resources as, function for R:208 to R:209
  handling user actions in G:221 to G:223, G:225 to G:226
  multiple G:225 to G:226
  testing G:205
multiline text boxes, in dialog boxes G:200
multiple modeless dialog boxes, handling user actions in G:225 to G:226
multiple-interaction dialog boxes G:223
  handling user actions in G:224 to G:225

N
named graphic frames G:289, G:390
named objects G:87 to G:89, G:269 to G:270, R:136 to R:137
  creating G:145
    function for R:14, R:218 to R:220
  defined G:390
  getting the IDs of G:87 to G:89, G:269 to G:270
    function for R:15, R:136 to R:137
  naming conventions, as used in this manual G:xii, R:xix
  NewTemplate template G:279
No buttons, displaying G:18
-noapi start-up option G:16
NoName elements R:347
noninteractive mode, in simple functions G:46, G:47
  notices. See alert boxes; alerts
  notification (of events)
    nonspecific G:38 to G:40
      in Open operations G:41 to G:42
      requesting G:13, G:34 to G:36
        function for R:25, R:226 to R:233
      requesting for filters G:36
      responding to G:15, G:34 to G:42
        function for R:1, R:233 to R:237
      when updating dialog items G:220
  notification constants G:35, G:38, R:227 to R:232
  naming convention for G:xiii, R:xix
  notification points G:34 to G:36, G:38, R:227 to R:232
  numbering paragraphs. See paragraph numbering

O
object IDs. See IDs (of API objects)
object pointer properties. See document properties
object properties. See properties
object types G:3
  getting G:84
    function for R:24, R:138
  naming convention for G:xiii, R:xix
Index - Subjects

See also individual object types in the FDK Reserved Names Index

objects (API)  G:3, R:495 to R:673
  automatic updating of  G:81, G:160
  child  G:295
  child graphic  G:389
  color property of  G:339
  creating  G:7, G:143 to G:158
    functions for (tables)  G:143 to G:145, R:14
  data type for IDs of  G:3, G:6, G:271, G:390
  default  G:279
  defined  G:390
  deleting  G:158 to G:159
    functions for  R:14, R:83 to R:90
  determining if valid, function for  R:24, R:237
  document lists of  G:284
  document properties for  G:282 to G:284
  frame  G:295
  getting text in, function for  R:23
  getting the IDs of  G:82 to G:89, G:269 to G:270
    functions for  R:15
  getting the IDs of selected  G:86 to G:87, G:285
  grouping  G:101
  identifying  G:269 to G:270
  moving between frames  R:286
  named  G:145, G:269 to G:270, G:390
  nondeletable  G:158
    listed  R:84
  order of in linked lists  G:84
  as organized in documents  G:82, G:84,
    G:282 to G:284
  properties of  R:495 to R:673
  selection state of, properties for  G:285 to G:286
  session  G:10
  traversing lists of  G:84 to G:86
  unnamed  G:270
  validating IDs for, function for  R:24, R:237
  See also  graphic objects; object types; text objects;
    and individual types of objects

offset value (F_TextItemT structure)  G:312

OK buttons
  of dialog boxes  G:196
  displaying  G:18

online help.  See context-sensitive help

Open dialog box  R:326
  displaying  G:21 to G:22, R:53 to R:56

Open operation
  notification in  G:41 to G:42

Open script property lists.  See openParamsp property lists

Open status (of files)
  checking  G:51 to G:52
  printing the values of  G:52
  See also  status flags

opening
  channels  G:377, R:353
  documents and books  G:46 to G:53, R:2,
    R:238 to R:244, R:326
  files  G:41 to G:42

openParamsp property lists
  allocating memory for  G:50 to G:51
  creating
    with
    F_ApiGetOpenDefaultParams()  G:49 to G:
      50
    from scratch  G:50 to G:51
  deallocating memory for  G:52 to G:53

P

page backgrounds  G:289

page frames  G:147, G:290
  API representation of  G:290
  defined  G:290, G:390
  deleting  R:84

page properties.  See document properties

page range, printing  R:148

pages  G:280, G:288 to G:292
  API representation of  G:280, G:289 to G:290
  API representation of the beginning of  G:313, R:166,
    R:686
  API representation of the end of  G:313, R:166,
    R:687
  applying page’s layout to another page  R:47
  backgrounds of  G:289
  current  G:291
  displaying particular  G:99
  document lists of  G:284
  document properties for  G:284, R:559 to R:561
    properties of  G:290, R:611
  master  G:289
    properties of  G:290, G:291 to G:292, R:612
as organized in documents G:291
   illustrated G:292
   pagination of G:284
   properties of G:290 to G:292, R:610 to R:612
   illustrated G:290, G:292
   reference G:289, G:391
   properties of G:290, G:291 to G:292, R:612
   text frame layouts of G:289
   visible G:288
See also body pages; frames
PANTONE color model G:339
paragraph breaks, specifying G:121
Paragraph Catalog G:300
Paragraph Catalog formats G:102, G:280, G:300 to G:302
   API representation of G:280, G:300 to G:302
   applying to paragraphs G:89, G:102 to G:103, G:302
   changing G:91, G:302
   changing individual paragraph formats and G:302
   creating G:145
   as organized in documents G:302
   overriding G:302
   properties of G:301 to G:302, R:621 to R:629
   reapplying G:306
   tags for G:300, G:302
paragraph default font properties
   (FO_Pgf) R:614 to R:616
paragraph default font properties
   (FO_PgfFmt) R:623 to R:625
Paragraph Designer
   illustrated G:301
   specifying named frames with G:289, G:390
paragraph formats (FO_PgfFmt) R:621 to R:629
   autonumbering properties of R:623
   changing G:302 to G:305
   creating G:145
   default font properties of R:623 to R:625
   hyphenation properties of R:625
   indentation properties of R:626
   language property of R:626
   line spacing property of R:627
   object pointer property of R:627
   placement properties of R:627 to R:628
   reference frame properties of R:628
   spell-checking language property of R:626
   tab properties of R:16, R:629
   table cell properties of R:628
   tag properties of R:629
   word spacing properties of R:629
See also Paragraph Catalog formats; paragraph properties; paragraphs (FO_Pgf)
paragraph numbering, function for restarting R:7, R:267
paragraph properties G:102, G:302 to G:303, R:613 to R:621
   applying to Paragraph Catalog formats G:102 to G:103
   getting and setting G:102 to G:105
   illustrated G:305
See also paragraph formats (FO_PgfFmt); paragraphs (FO_Pgf)
paragraph reference frame properties (FO_Pgf) R:620
paragraphs (FO_Pgf) G:280, G:302 to G:305
   adding text to, function for R:23
   adding to documents G:149 to G:150
   function for R:220 to R:221
   API representation of G:280, G:302 to G:305
   API representation of the beginning of G:313, R:166, R:686
   API representation of the end of G:313, R:166, R:686
Asian character spacing properties of R:613
   autonumbering properties of R:613 to R:614
   changing the formats of G:302 to G:305
   general properties of R:617
   getting in order G:86
   hyphenation properties of R:617 to R:618
   identifier properties of R:618
   indentation properties of R:619
   language properties of R:618
   line spacing property of R:619
   object pointer properties of R:620
   as organized in documents G:304 to G:305
   as organized in flows G:304
   illustrated G:305
   placement properties of R:619 to R:620
   restarting numbering, function for R:7, R:267
   spell-checking property of R:618
   tab properties of R:16, R:621
   table cell properties of R:616 to R:617
   tag properties of R:621
   tagging G:302
   traversing G:84, G:103
   word spacing properties of R:621
See also Paragraph Catalog formats; paragraph formats (FO_PgfFmt); paragraph properties paragraphs (FO_PgfFmt)

Asian character spacing properties of R:622

parent frames (of graphic objects), defined G:295, G:390

parent’s content rules (of structural element definitions) G:348

pasting text selections G:127, R:23, R:246 to R:247

path types G:376

See also filepaths; pathnames

pathnames

conventions used in this manual G:xiv, R:xx

converting FDE filepaths to G:375

converting to FDE filepaths G:375 to G:377

See also filepaths; path types

pen patterns R:595

illustrated R:595

persistent identifiers. See UIDs (unique IDs)

platform names (for fonts) G:307 to G:308

Macintosh specification string for G:307

Windows specification string for G:307 to G:308


platform-independent code, in the FDE (illustrated) G:365

platform-specific code, in the FDE (illustrated) G:365

platform-specific views of dialog boxes G:180 to G:181

plus sign (+), conditional text marker prefix G:318

pointers

functions for

allocating and deallocating memory R:32

freeing memory allocated to R:386


master pointer G:378

points. See coordinate pairs

points (F_PointsT) properties

function for getting R:16, R:146 to R:148

function for setting R:16, R:295 to R:297

polygon properties R:16, R:603

polyline properties R:16, R:604

pop-up menus

adding menus or menu items to G:27, R:79

in dialog boxes G:200

getting/setting the state of, function for R:5

properties of G:200, R:535

See also image pop-up menus


portrait documents, creating G:57

PostScript names (for fonts) G:307 to G:308

primitive data types (C language)

API substitutes for G:6, R:675

FDE substitutes for G:6, G:367 to G:369, R:675

Print command, replacing G:30

print properties. See document properties

print settings, changing G:58

printer state, function for getting R:148

printer state, function for setting R:297

printing

books, function for G:57 to G:59, R:2

current API error status, function for R:4, R:98, R:248

documents G:57 to G:59

function for R:6, R:319 to R:320

properties for specifying R:561 to R:563

formatted output, functions for R:424, R:440

imported graphics, facets for G:258

properties G:284

property-value pairs, functions for R:5

status flags, functions for R:4, R:250

text in text items, functions for R:5, R:253 to R:254

values in specified property lists, function for R:252

values of specified properties, function R:251 to R:252

progress

function for showing R:426

project files G:8

properties G:3 to G:4

allocating memory for G:94

API functions applicable to R:15 to R:17

of API objects R:495 to R:673

automatic updating of G:81, G:160

batching changes in G:96

copying G:102, R:297

data types for (tables) G:270, G:273

deallocating memory from G:94, R:11

function for R:11

defined G:3, G:390


API functions for (table) G:90, R:15

for menus and menu items G:162

getting values of, function for R:190 to R:191
Index - Subjects

- identifying G:272
  - structure for G:272, R:684
- implicit changes in G:160
- for import operations (tables) R:120 to R:127
- manipulating G:89 to G:95
- naming convention for G:xiii, R:xix
- object changes possible with G:81
- order of, in property lists G:93
- page G:289 to G:292, R:610 to R:612
- printing values of specified, function for R:251 to R:252
- querying G:84 to G:89
  - function for R:15, R:190 to R:191
- read-only G:81, R:495
- representing object characteristics with G:270 to G:274
- setting G:7, G:37, G:45, G:89 to G:92, G:95 to G:105
  - API error protection feature G:81
  - API functions for G:89, R:15 to R:17, R:300 to R:301
  - of dialog boxes G:193 to G:195, G:208
  - of dialog items G:197 to G:204, G:208
  - individually G:89 to G:92
  - for menus and menu items G:162
  - steps for G:81
- type-in G:126, G:284, G:391, R:573 to R:575

See also document properties; property lists; property values; individual types of properties generally under individual object types in this index

See also individual properties by name in the FDK Reserved Names Index

- property data types, tables of G:270, G:273
- property identifier (structure) G:272, R:684
- property lists G:3, G:81, G:270, G:272 to G:274
  - allocating memory for, function for R:11, R:45 to R:46
  - API representation of G:272
deallocating memory from, function for R:11, R:74
  - getting G:92 to G:94
  - function for R:16, R:150 to R:151
  - getting particular properties in G:93
  - illustrated G:274
  - manipulating directly G:93 to G:94
  - Open script G:49 to G:50

  order of properties in G:93
  printing values in specified, function for R:252
  for scriptable functions G:46
  setting G:92 to G:94
  - function for R:16, R:297
  - setting, function for R:16
  - structure for G:272, R:684

  property names, identifying properties by G:272
  property number constants, identifying properties by G:272
  property value constants, naming convention for G:xiii, R:xix

  property value data types. See property data types
  property value type constants. See property value types
  property value types, tables of G:273, R:690
  property values G:3 to G:4, G:270 to G:271, G:390
  data types for (tables) G:270, G:273
  structure for individual R:689
types of (tables) G:273, R:690
  union for sets R:690

  property-value pairs
  - function for getting indexes of R:24, R:149
  - functions for printing R:5, R:252
  - structure for individual G:272, R:684
  - structure for sets G:272, R:684

  pull-down menus
  - adding pull-right menus to R:79
  - adding to menu bars R:79
  - creating G:27

  pull-right menus
  - adding to other menus R:79
  - creating G:27

Q

querying
  - facets R:183 to R:184
  - properties G:84 to G:89
  - quick keys, implementing G:42 to G:43, R:262
  - quick-key interface G:42 to G:43
  - function for implementing R:262
  - quitting Frame sessions G:67
  - function for R:17

FDK Programmer’s Reference & Guide
radio buttons
  in dialog boxes G:199
  getting/setting the state of, function for R:5
read-only properties G:81
  symbol indicating R:495
rectangle properties R:604
reference frames
  function for resetting R:7, R:266
  properties of R:620, R:628
reference page properties R:612
reference pages G:289, G:391
  API representation of G:289
  creating. See named objects, creating properties of G:290, G:291 to G:292, R:612
reformatting documents G:70 to G:74, G:96
  functions for G:77, R:7, R:25
registered names, of current client R:24, R:60
registering
  API clients with the FrameMaker product G:8 to G:9, G:12
  filters G:15
rehyphenating words, function for R:7
return values of functions. See individual functions in the FDK Reserved Names Index
RGB color model G:339
Right master page G:292
rounded rectangle properties R:604
rows
  adding to tables G:156 to G:157
  function for R:14, R:22, R:39 to R:41
API representation of G:333
  deleting from tables G:158 to G:159
  functions for R:22, R:84, R:87 to R:88
  getting the IDs of selected G:87
  properties of G:333, G:336, R:651 to R:652
  selecting G:118
  See also cells
rubi
  documents
    properties for formatting R:564
rubi composites
  properties of R:630
Rubi groups, creating G:131
rubi text
  API representation of the beginning of R:171, R:172
  API representation of the ending of R:171
Ruling Catalog G:328
ruling format properties G:329, R:630
ruling formats G:281, G:328 to G:330
  API representation of G:281, G:329 to G:330
  illustrated G:329
  as organized in documents G:329
  properties of G:329, R:630
rulings G:328
  default G:334
  override G:334
  See also ruling formats
S
Save dialog box, displaying G:21 to G:22, R:53 to R:56
Save operation
  repeating G:41
  specifying aspects of G:60
Save operation property lists. See saveParamsp property lists
Save status (of files)
  checking G:64 to G:66
  See also status flags
saveParamsp property lists
  creating
    with
      F_ApiGetSaveDefaultParams() G:61 to G:64
    from scratch G:64
saveParamsp scripts. See saveParamsp property lists
saveReturnParamsp property lists G:64
saving
  contents of Clipboard G:129
  dialog box information G:181 to G:185
  files G:35
  Frame DRE files G:181
  global document information G:284, G:285
  screen display, updating/not updating G:97
  screen flicker, avoiding G:96
scriptable functions (API) G:45, G:46
  naming convention for properties of G:xii, R:xix
  setting properties using G:93
scripts G:46
See also property lists
scroll bars
changing orientation of G:202
changing the length of G:202
in dialog boxes G:201
displaying the value of G:201
properties of G:201, R:536
resizing G:201
scroll list dialog boxes
displaying G:22 to G:24
function for R:5, R:274 to R:276
getting/setting the state of, function for R:5
illustrated G:24
scroll lists
in dialog boxes G:202
properties of R:536 to R:537
selected objects, getting the IDs of G:86 to G:87
selected structural elements
API representation of G:286
deleting R:337
demoting, function for R:92 to R:93
getting the IDs of G:120
merging into the first element, function for R:204 to R:205
merging into the last element, function for R:205
traversing G:120
selected text. See text selections
selection properties. See document properties
selections
API functions applicable to R:17
properties applicable to G:285 to G:286
series objects G:149 to G:150, R:220
creating G:11, G:149 to G:150
function for R:14, R:220 to R:221
defined G:391
sessions G:274, G:391
active documents and books in G:275 to G:276
API representation of G:274 to G:277
illustrated G:276
connecting the calling process to, functions for R:67, R:95, R:106
determining menu configuration of G:164 to G:166
only ID of G:11, G:82
open documents and books in G:275
illustrated G:275 to G:276
properties of G:10, G:83, G:275, G:276 to G:277,
R:631 to R:638
getting and setting G:95 to G:97
quitting Frame sessions G:67
function for R:17
setting
attribute definitions R:278 to R:280
attributes R:280 to R:282
byte ordering (on I/O channels) G:382
channel offsets R:357, R:358
dash patterns R:294, R:595
element range G:282 to R:284
fill patterns G:11
the insertion point G:114 to G:117
properties G:7, G:37, G:45, G:89 to G:92,
G:95 to G:105, G:162
property lists G:92 to G:94
text properties G:93, G:124 to G:126, G:317
text selections G:114 to G:117
type-in properties G:126
SGML attributes G:348
shift menu items G:174 to G:175
shortcuts. See keyboard shortcuts
shorts (2 bytes)
FDE function for reading (and swapping) R:430
FDE function for writing (and swapping) R:491
sibling elements (structural), traversing G:121
simple functions (API) G:45
simple MIF functions G:381, G:387
simple MIF library (FDE). See simple MIF functions
single-interaction dialog boxes G:223
handling user actions in G:223
single-sided documents, master pages for G:289, G:292
sleep functions R:17, R:328, R:344 to R:345
special characters, specifying G:121, G:316 to G:317,
R:41
Special menu
adding commands to G:29
name used to specify G:25, R:76
specified persistent identifiers. See UIDs (unique IDs)
spell-checking paragraphs, property for R:618
spot color views, bit positions representing
(illustrated) R:520
spot colors G:338
square roots, function for returning R:491
static memory allocation (in UNIX) G:379
status bars, setting G:98
status bits, function for checking  R:6, R:52 to R:53
status flags
  for FS_ImportStatus (table)  R:195 to R:196
when books (F_ApiSave())
  function for printing  R:4
when importing documents or books
  (F_ApiImport()), function for printing  R:4, R:250
when opening documents or books
  (F_ApiOpen())  R:240 to R:243
  function for printing  R:4, R:250 to R:251
when saving documents or books
  (F_ApiSave())  R:273 to R:274
  function for printing  R:4, R:253
when updating books (F_ApiUpdateBook())  R:339
  function for printing  R:255 to R:256
straddling cells  G:75 to G:79, G:337 to G:338
  API representation of  G:337 to G:338
  function for  R:22, R:331 to R:332
  illustrated  G:338
string functions  G:381, R:18 to R:20, R:440 to R:465
  allocating space for  G:381
  specifying string size in  G:381
  See also string lists; strings
string library (FDE).  See string functions
string list library (FDE)  G:381, G:382
string lists
  data type for  G:382
FDE functions for  G:381
  allocating new  G:382, R:474
  appending strings to  G:382, R:466
  concatenating  R:466
  copying  R:468
  copying strings from  R:467
  freeing strings in  R:469
  inserting strings into  G:382, R:472
  positioning strings in  R:476
  removing strings from  R:475
  returning first strings in  R:469, R:471
  returning last strings in  R:473
  returning specified strings in  R:470
  returning the number of strings in  R:474
  sorting  R:477
See also strings

string prompt dialog boxes
  displaying  G:18 to G:20, R:260
  function for  R:5, R:260 to R:261
  illustrated  G:20, R:261
strings
  API representation of  G:312, G:313, R:166, R:686
  copying, functions for  R:4, R:69
data type for  G:3, G:271, G:381, R:675
FDE functions for  G:381, R:440
  allocating new  G:381, R:478
  appending characters to  R:444
  appending double-byte characters to  R:445
  appending integers to  R:446
  appending to string lists  R:466
  comparing  G:381, R:455
  comparing characters in  R:450
  concatenating  R:443, R:446
  concatenating double-byte strings  R:447
  converting alphanumeric to integers  G:381, R:441
  converting alphanumeric to real numbers  R:442
  converting integers to  R:446
  converting to double-precision numbers  R:492
  converting to long integers  R:493
  copying  R:452
  copying double-byte strings  R:454
  copying from string lists  R:467
  copying one to another  R:453
  determining if a prefix of another string  R:478
  determining if a suffix of another string  R:484
  finding double-byte characters in  R:449, R:480
  finding one in another  R:484
  finding single characters in  R:444, R:448, R:480
  freeing strings in string lists  R:469
  freeing the memory used by  G:381
  inserting into string lists  R:472
  printing formatted output to  R:440
  reading formatted input from  R:441
  removing characters from  R:482
  removing from string lists  R:475
  removing leading spaces from  R:482
  removing trailing spaces from  R:483
  returning strings in string lists  R:470
  returning text tokens in  R:485
returning the length of  R:465
returning the length of double-byte  R:465
reversing characters in  R:481
truncating  R:486
truncating double-byte strings  R:487
prompting users for  G:18 to G:20, R:260
function for  R:5, R:260 to R:261
properties (F_StringsT) of
function for getting  R:16, R:160 to R:161
function for setting  R:16, R:303 to R:305
properties (StringT) of
function for getting  R:16, R:158 to R:159
function for setting  R:16, R:301 to R:303
See also string lists
structural element CSS properties  R:641
structural element definition properties
getting and setting  G:108
API representation of  G:281, G:343 to G:347
getting and setting properties  G:108
importing  G:77
function for  R:322
and NoName elements R:347
properties of  G:342, R:645 to R:647
structural element ID properties  R:642
structural element Namespace properties  R:643
structural element properties G:348, R:638 to R:644
general  R:638
structural element range properties
function for getting  R:113 to R:115
function for setting  R:282 to R:284
structural element validation properties  R:643 to R:644
structural elements G:281, G:342, G:347
API representation of  G:281
API representation of the beginning of  G:314, R:687
API representation of the end of  G:314, R:687
API representation of the validity of  G:348
creating  G:145, G:150 to G:151, R:213, R:347
functions for  R:213, R:347 to R:348
deleting, function for  R:337 to R:338
demoting, function for  R:92 to R:93
determining formatting  G:108
determining if text nodes  R:96
document properties for  G:284, R:566
getting the IDs of selected
function for  R:113 to R:115
manipulating  G:76 to G:77
NoName  R:347
preventing validation of  G:110
promoting, function for  R:256 to R:257
properties of  G:348, R:638 to R:644
selecting  G:120
setting the IDs of selected
function for  R:282 to R:284
specifying client data for  G:110
traversing  G:105
structure properties  R:566
structured books, API representation of  G:355
structured documents  G:342
structured flows  G:300
Structured FrameMaker
documents  G:347
improving the performance of  G:110
Structured FrameMaker commands
executing  G:76 to G:77
structured properties  G:284
structures.  See data structures
stuff items, dialog items as  G:203 to G:204, G:213
style conventions, as used in this
manual  G:xiii to G:xiv, R:xix to R:xxi
system resources, freeing  G:15, G:43 to G:44

T
table anchors, API representation of  G:312, G:313, R:167, R:687
Table Catalog  G:330
Table Catalog formats G:281, G:330 to G:331
API representation of  G:281, G:330 to G:331
illustrated  G:331
as organized in documents  G:331
overriding  G:332
properties of  G:330 to G:331, G:332
versus table properties  G:332
tagging existing tables with  G:330
See also table formats
table cells.  See cells
table columns
adding to tables  G:156 to G:157
function for  R:14, R:22, R:34 to R:35
Index - Subjects

- deleting from tables G:158 to G:159
  function for R:22, R:84
- as references to cells G:157
- selecting G:118

Table Designer
- illustrated G:331
- specifying ruling formats G:329
- table footnote properties. See document properties

Table formats G:330 to G:331
- basic properties R:658
- changing G:330, G:332
- general properties R:659
- new table properties R:660
- overriding G:332
- properties of R:658 to R:662
- ruling properties R:660 to R:661
- shading and color properties R:661 to R:662
See also Table Catalog formats

Table menu, name used to specify G:25, R:76

Table of Contents documents
- generating R:321
- TOC flows on G:289

Table ruling formats. See ruling formats

Table selections, getting and setting G:118

- tables G:281, G:332 to G:338
  - adding rows to G:156 to G:157
    function for R:14, R:22, R:39 to R:41
  - adding table columns to G:156 to G:157
    function for R:14, R:22, R:34 to R:35
  - API representation of G:281, G:332 to G:338
    - basic properties R:653
    - creating G:148 to G:149, G:154 to G:156
      functions for R:15, R:22, R:224 to R:226
    - deleting R:83
      function for R:22, R:83
  - deleting rows from G:158 to G:159
    functions for R:22, R:84, R:87 to R:88
  - deleting table columns from G:158 to G:159
    function for R:22, R:84, R:85 to R:86

- document footnote properties for G:284
- general properties R:654
- getting the IDs of selected G:87

- objects representing (illustrated) G:334
- organization of (illustrated) G:335
- properties of G:332 to G:338, R:638 to R:658
  - versus Table Catalog format properties G:332
- retagging G:332

ruling properties R:655 to R:656
- selecting G:118
- selecting elements in structured G:119
- selection properties R:656
- shading and color properties R:656 to R:657
- sorting data (qsort) R:27 to R:28
- tagging G:332
- title properties R:657
- titles of G:333
See also cells; hash tables; rows; ruling formats; table columns

tabs
- adding G:103 to G:105
- function for getting R:16, R:162 to R:164
- function for setting R:16, R:305 to R:307
- structure for G:104, G:271, R:685
See also paragraph formats (FO_PgfFmt), tab properties of; paragraphs (FO_Pgf), tab properties of

Tag area, choosing commands in G:42, R:3

tagging
- characters G:306
- paragraphs G:302
- tables G:332

take-control initialization G:14

templates, creating documents from G:53 to G:54, G:279
- function for R:325 to R:326
text G:111, G:311 to G:318
- adding G:123
  function for R:23, R:41 to R:42
to table cells G:122
- API functions applicable to R:23
- API representation of G:312 to G:318
- applying character formats G:124
- applying conditions to G:126
- conditional G:318
  illustrated G:318
deleting
  functions for R:88 to R:90
- getting G:111 to G:113
  functions for R:23, R:24, R:164 to R:173
  in text ranges G:118, R:23
- hiding G:309 to G:311
- importing G:230 to G:235
  function for R:192 to R:197
manipulating G:111 to G:129
printing, functions for R:5, R:253 to R:255
properties of R:669 to R:671
setting properties G:124 to G:126, R:669
setting type-in properties G:126
See also conditions; flows; strings; text boxes; text columns; text files; text frames; text insets; text items; text lines
See also text locations; text properties; text ranges; text selections; variables
text boxes
in dialog boxes G:200
getting/setting the state of, function for R:5
properties of R:534
text columns
API representation of the beginning of G:313, R:166, R:687
API representation of the end of G:313, R:167, R:687
creating G:150
function for R:14, R:216 to R:218
as organized in flows, illustrated G:298, G:305
properties of R:520 to R:521
See also text frames
text conditions. See conditions
text entry, client response to G:38 to G:40
text files
importing G:229, G:230, G:234
properties for (table) R:126 to R:127
text flows. See flows
text format G:311
See also Character Catalog formats; text properties
text frames
API representation of G:294
API representation of the beginning of G:313, R:167
API representation of the end of G:313, R:167
cells as G:332
connecting G:297 to G:300
defined G:293
for footnotes G:327
as organized in flows G:298
properties of R:605 to R:606
text frame layouts G:289
See also footnotes
text insets G:229 to G:230, G:281
adding text to, function for R:24
API functions applicable to R:24
API representation of G:281
API representation of the beginning of G:314, R:167
API representation of the end of G:314, R:167
client text insets G:230, G:236 to G:238
common properties R:662 to R:666
creating, functions for R:15, R:24
deleting text from, function for R:24, R:90
deleting, function for R:24
getting default property list for use with, function for R:24
getting text from, function for R:24
manipulating G:236
properties for text imported by an FDK client R:666
properties for text imported from a Frame product R:667
properties for text imported from a MIF file R:667
properties for text imported from a text file R:667 to R:668
properties of R:662 to R:668
updating stale, function for R:24
updating, function for R:7
text item structures G:112, G:312 to G:316, R:686 to R:689
allocating memory for, API function for R:46
deallocating memory from G:112, R:165, R:170
functions for R:11, R:46, R:165, R:170
text item types
getting G:111 to G:113
tables of G:313 to G:314, R:686 to R:689
text items G:112, G:312 to G:316, R:164, R:170
API representation of G:112, G:312 to G:316
arrays of, allocating memory for R:46
for paragraphs (illustrated) G:316
printing text in, functions for R:5, R:253 to R:254
See also text; text item structures; text item types
text line end type, API representation of G:313, R:166, R:686
text lines
API representation of the beginning of G:313, R:166, R:686
API representation of the end of G:313, R:166, R:686
defined G:293
properties of R:606 to R:607
text locations (F_TextLocT) G:114
properties of
functions for getting R:16, R:23, R:178 to R:179
functions for setting R:16, R:23, R:308 to R:309
structure for G:114, G:271, R:689
text nodes  R:96

text objects, API representation of  G:313, R:167

text properties  G:311, G:317, R:669 to R:671

  API representation of  G:312, G:317

  API representation of changes in  G:312, G:314, G:317, R:166

  changing individual  G:125

  getting  G:123 to G:124, G:317

    functions for  R:16, R:23, R:179 to R:180, R:669

  getting the value of, function for  R:182 to R:183 at the insertion point  G:317

  keeping intact  G:306

  setting  G:93, G:124 to G:126, G:317


  type-in  G:126, G:284, G:391, R:573 to R:575

  See also  text locations (F_TextLocT); text ranges (F_TextRangeT)

text ranges (F_TextRangeT)  G:114 to G:117, R:689

  centering, function for  R:51 to R:52

  getting properties of  G:114 to G:117

    function for  R:16, R:23, R:181 to R:182

  getting text in  G:118, R:181 to R:182

    function for  R:23

  setting properties of  G:114 to G:117

    functions for  R:17, R:23, R:311 to R:316

  structure for  G:114 to G:115, G:271, R:689

text selections

  clearing  G:127

    function for  R:3, R:56 to R:58

  Clipboard functions and  G:127 to G:129, R:3

    copying  G:127

      function for  R:3, R:68 to R:69

    cutting  G:127

      function for  R:3, R:72 to R:73

  deleting, function for  R:3, R:88 to R:89

  getting  G:114 to G:117

    function for  R:181 to R:182

  getting the text in, function for  R:173 to R:176

  pasting  G:127

    function for  R:3, R:23, R:246 to R:247

  selection state of  G:285, G:287

  setting  G:114 to G:117

    functions for  R:17, R:313 to R:314

  See also  text; text items

  Text Symbols view option  G:311

  text tokens, in strings  R:485

title bars, setting  G:98

  title label, of dialog boxes  G:196

  TOC flows  G:289

  topical listing of API functions  R:1 to R:24

  transactions, committing (in getting and setting facets)  R:129, R:288

  traversing

    lists of document objects  G:84

    lists of objects of one type  G:85

    ordered lists of objects  G:85 to G:86

    paragraphs  G:84, G:103

    starting  G:83

  tri-state check boxes

    properties of  R:537

    See also  triboxes

  triboxes

    in dialog boxes  G:199, R:537

    properties of  G:199, R:537

  type-in properties  G:126, G:284, R:573 to R:575

    defined  G:391

    setting  G:126

  typedefs, naming convention for  G:xiii, R:xix

  types.  See  data types; file types; object types

  typographic conventions used in this manual  G:xiv, R:xx

  typographic properties  R:576

U

  UByteT data, structure for  R:691

  UIDs (unique IDs)  G:89, G:270, G:391

    defined  G:391

    getting  G:89, G:270

      function for  R:15, R:185 to R:186

    See also  IDs (of API objects)

  unanchored frames

    defined  G:293

    properties of  G:295, R:608

  uncompressing files  G:42

  undefined attributes  G:348

  Undo command (Graphics menu)  G:192

  unions, for sets of property values  R:690

  unique identifiers.  See  UIDs (unique IDs)

  unique object names  G:269

  UNIX pathnames, conventions used in this manual  G:xiv, R:xx

  UNIX pathnames.  See  filepaths; pathnames
UNIX platforms
  capturing screenshots of dialog boxes G:206
  compiling and running FDE filters G:374
  dialog boxes on G:180 to G:181, G:189 to G:190, G:207
  disabling clients G:16
  image pop-up menus G:200
  memory management on G:378 to G:379
  registering clients with the FrameMaker product G:8, G:12
  saving DRE files G:204
  take-control initialization G:15
  See also filepaths
unnamed objects G:270
unstraddling cells G:75 to G:79, G:337 to G:338
  function for R:22, R:336 to R:337
  illustrated G:338
Update status (of books)
  checking G:359 to G:360
updateParamsp property lists
  creating with
    F_ApiGetUpdateBookDefaultParams() G:357 to G:358
  from scratch G:359
updateReturnParamspp property lists G:359
updating
  cross-references R:343
  documents and books G:70 to G:74
    functions for G:77, R:7
  objects and properties automatically G:81, G:160
  variables R:342
user actions
  common (tables) G:74
  in dialog boxes, handling G:221 to G:227
  user input, simulating G:74 to G:75
user interfaces
  setting up for API clients G:7, G:17 to G:44
  user configuration of G:30 to G:31
  using hypertext commands in G:31 to G:34, R:192
user-initiated events, responding to G:34 to G:42
users
  configuring the user interface G:30 to G:31
  f-codes for common actions of (tables) G:74
  prompting for input G:7, G:17 to G:24
    functions for R:257 to R:261
  responding to actions of G:7, G:29
  returning control to, function for R:25, R:96
  simulating the input of G:74 to G:75
utility functions.  See API utility functions; FDE utility library functions
utility libraries.  See FDE utility libraries

V
valid elements
  function for getting R:112
valType constants, tables of G:273, R:690
values.  See property values
variable format building blocks G:324 to G:325
variable formats G:281, G:324 to G:326
  API representation of G:281, G:325 to G:326
  building blocks for G:324 to G:325
  classes of G:325
  definitions of (illustrated) G:325
  properties of G:325, R:672
  See also variables
variables G:281, G:326 to G:327
  API representation of G:281, G:326 to G:327
  API representation of the beginning of G:314, R:167, R:687
  API representation of the end of G:314, R:167, R:687
  creating G:148 to G:149
    function for R:209 to R:210
  illustrated G:326
  inserting G:324, G:326
  properties of G:326 to G:327, R:672
  restrictions on inserting G:326
  updating, function for R:7, R:342
  See also variable formats
View menu, name used to specify G:25, R:76
view-only menu bar ?? to G:162
view-only properties.  See document properties
virtual environment, defined G:391
virtual environment.  See FDE virtual environment

W
warnings
  displaying G:17 to G:18, G:45, G:379, R:488
  See also API error codes
Windows platforms
  compiling and running FDE filters G:373
Index - Subjects

dialog boxes on  G:180 to G:181, G:189
disabling clients  G:16
image pop-up menus  G:200
keyboard shortcuts in dialog boxes  G:202
memory management on  G:378 to G:379
registering clients with the FrameMaker product  G:8, G:12
take-control initialization  G:15
word spacing. See paragraph formats (FO_PgfFmt), word spacing properties of; paragraphs (FO_Pgf), word spacing properties of
wrdcnt document report  G:5
Writing export filters  G:242
Writing file-to-file filters  G:243
Writing graphic import filters  G:241
Writing text import filters  G:239

X
XML attributes  G:348
XMP Metadata  R:497, R:540
xxxTemplate  G:279

Y, Z
Yes buttons, displaying  G:18
Symbols

!BookMainMenu string constant G:25, R:78
!CustomMakerMainMenu string constant R:78
!MakerMainMenu string constant G:25, G:27, R:78
!MakerPopup string constant R:78
!QuickBookMainMenu string constant G:25, R:78
!QuickMakerMainMenu string constant G:25, G:27, R:78
!QuickMakerPopup string constant R:78
!ShowCharacterTags expandomatic item G:172
!ShowDocumentWindows expandomatic item G:172
!ShowFilesRecentlyVisited expandomatic item G:172
!ShowFontChoices expandomatic item G:172
!ShowImportFiles expandomatic item G:172
!ShowParagraphTags expandomatic item G:172
!StructureViewMainMenu string constant R:78
!ViewerPopup string constant R:78
!ViewOnlyMainMenu string constant G:26, R:78
$FMHOME directory, getting G:96

A, B
Addt data type G:367
AEClient AppleScript client R:30, R:31
alloc() function G:369
BoolT data type G:367, R:675
ByteT data type G:367, R:675

C, D, E
ChannelT structure R:677
CHAR_APPLY_TAG f-code G:39
CharT data type G:367
ConStringT data type G:367, R:675
DDQSort() function R:27
DocFP_XmlEncoding property (FO_Book) R:505
DocFP_XmlFileEncoding property (FO_Book) R:505, R:570
DSE flag (FDE functions) G:379, R:439
ErrorT data type G:367, G:379, R:675
F G:356, G:357

F_
F_AEGetEventHandler() function R:1, R:28
F_AEInstallEventHandler() function R:1, R:29
F_AERemoveEventHandler() function R:1, R:30
F_Alloc() function G:369, G:372, G:379, R:12, R:32
F_AllocHandle() function G:379, R:12, R:32
F_ApDeallocatePropVals() function G:71
F_ApiAddCols() function G:156, R:14, R:22, R:34
F_ApiAddCommand() function R:35
F_ApiAddCommandToMenu() function G:166, R:3, R:35, R:81
F_ApiAddMenu() function R:37
F_ApiAddMenuToMenu() function R:3, R:37, R:82
F_ApiAddRows() function G:156, R:14, R:22, R:39
F_ApiAddText() function G:121, G:236, G:373, R:23, R:24, R:41
F_ApiAlert() function G:7, G:17, R:5, R:43
F_ApiAlive() function R:44
F_ApiAllocatePropVals() function G:46, G:49, G:50, R:11, R:45
F_ApiAllocateTextItems() function R:11, R:46
F_ApiApplyPageLayout() function R:7, R:47
F_ApiBailOut() function G:15, G:43, G:373, R:11, R:24, R:48, R:374
F_ApiCallClient() function G:361, R:24, R:49
F_ApiClipBoard() function R:51
F_ApiChooseFile() function G:21, R:5, R:53
F_ApiClear() function G:127, R:3, R:23, R:56
F_ApiClearAllChangeBars() function G:73, R:7, R:58
F_ApiClientDir() function R:24, R:59
F_ApiClientName() function R:24, R:60
F_ApiConnectToSession() function R:67, R:95, R:106

Index - Reserved Names
Index - Reserved Names

F_ApiCopy() function G:127, R:3, R:23, R:68
F_ApiCopyAttribute() function R:69
F_ApiCopyAttributeDef() function R:69
F_ApiCopyAttributeDefs() function R:69
F_ApiCopyAttributes() function R:69
F_ApiCopyElementCatalogEntries() function R:69
F_ApiCopyFonts() function R:69
F_ApiCopyInts() function R:4, R:69
F_ApiCopyMetrics() function R:4, R:69
F_ApiCopyPoints() function R:4, R:69
F_ApiCopyProp() function R:69
F_ApiCopyPropVal() function R:4, R:70
F_ApiCopyPropVals() function R:4, R:70
F_ApiCopyStructure() function R:69
F_ApiCopyTab() function R:4, R:70
F_ApiCopyTabs() function R:4, R:70
F_ApiCopyTextItem() function R:70
F_ApiCopyTextItem() function R:70
F_ApiCopyTextItems() function R:4, R:70
F_ApiCopyUBytes() function R:4, R:70
F_ApiCopyUInts() function R:4, R:70
F_ApiCopyVal() function R:4, R:70
F_ApiCustomDoc() function G:53, G:55, G:372, R:6, R:70
F_ApiCut() function G:127, R:3, R:23, R:72
F_ApiDeallocateAttribute() function R:74
F_ApiDeallocateAttributeDef() function R:74
F_ApiDeallocateAttributeDefs() function R:74
F_ApiDeallocateAttributes() function R:74
F_ApiDeallocateElementCatalogEntries() function R:74
F_ApiDeallocateTab() function R:11, R:74
F_ApiDeallocateTabs() function R:11, R:74
F_ApiDeallocateTextStyle() function R:74
F_ApiDeallocateTextItem() function R:74
F_ApiDelete() function G:158, G:167, R:3, R:15, R:22, R:24, R:83
F_ApiDeleteCols() function G:158, R:22, R:84, R:85
F_ApiDeletePropByName() function R:9, R:86
F_ApiDeleteRows() function G:158, R:22, R:84, R:87
F_ApiDeleteText() function G:121, R:23, R:88
F_ApiDeleteTextInsetContents() function G:237, R:24, R:90
F_ApiDeleteUndefinedAttribute() function R:91
F_ApiDemoteElement() function R:92
F_ApiDoneCommand() function G:212, G:217, R:5, R:94
F_ApiDoneCommand() function R:25, R:96
F_ApiElementDefIsText() function R:96
F_ApiErr() function R:96
F_ApiFamilyFonts() function G:74, R:7, R:101
F_ApiFcodes() function G:74, R:7, R:101
F_ApiFind() function R:23, R:102
F_ApiFindXSessionAddress() function R:106
F_ApiForkAndExec() function R:107
F_ApiGetAttributeDefs() function R:108
F_ApiGetAttributes() function R:110
F_ApiGetElementCatalog() function R:112
F_ApiGetElementRange() function G:120, R:113
F_ApiGetEncodingForFamily() function R:8, R:20, R:115
F_ApiGetEncodingForFont() function R:8, R:20, R:117
F_ApiGetId() function G:11, G:82, G:90, G:163, R:15, R:17, R:119
Index - Reserved Names

F_ApiGetImportDefaultParams() function R:6, R:24, R:120
F_ApiGetInt() function G:7, G:90, R:5, R:16, R:128, R:151
F_ApiGetIntByName() function G:263, R:9, R:129
F_ApiGetInts() function G:90, R:15, R:131
F_ApiGetMetric() function G:90, R:16, R:135
F_ApiGetMetricByName() function G:263, R:9, R:133
F_ApiGetObjectType() function G:84, R:24, R:138
F_ApiGetPoints() function G:90, R:16, R:146
F_ApiGetPrinterState() function R:148
F_ApiGetPropIndex() function G:93, R:24, R:149
F_ApiGetProps() function G:92, G:102, R:16, R:150
F_ApiGetPropVal() function R:15, R:151
F_ApiGetSaveDefaultParams() function G:61, R:6, R:24, R:152
F_ApiGetStrings() function G:90, R:16, R:99, R:115, R:160
F_ApiGetSupportedEncodings() function R:8, R:20, R:161
F_ApiGetTabs() function G:90, G:103, R:16, R:162
F_ApiGetText() function G:111, G:118, R:23, R:24, R:164
F_ApiGetText2() function R:170
F_ApiGetTextForRange() function G:118, R:23, R:173
F_ApiGetTextForRange2() function R:174
F_ApiGetTextLoc() function G:90, R:16, R:176
F_ApiGetTextPropVal() function G:123, R:23, R:179
F_ApiGetTextRange() function G:90, G:115, R:16, R:17, R:23, R:181
F_ApiGetTextVal() function R:23, R:182
F_ApiGetUBytesByName() function G:264, G:265, R:9, R:183
F_ApiGetUniqueObject() function G:89, G:270, R:15, R:185
F_ApiGetUpdateBookDefaultParams() function G:71, G:357, R:187

properties returned by R:187
F_ApiGetVal() function R:190, R:190
F_ApiHypertextCommand() function R:9, R:192
F_ApiImport() function G:230, G:256, R:6, R:9, R:15, R:24, R:52, R:120, R:192, R:250
F_ApiInitialize() function G:7, G:10, R:1, R:48, R:198
F_ApiIsEncodingSupported() function R:8, R:20, R:199
F_ApiLoadMenuCustomizationFile() function G:169, R:3, R:200
F_ApiMakeTblSelection() function G:118, R:17, R:22, R:201
F_ApiMenuExists() function R:203
F_ApiMenuItemInMenu() function G:164, R:3, R:203
F_ApiMergeIntoFirst() function R:204
F_ApiMergeIntoLast() function R:205
F_ApiMessage() function G:7, G:262, R:1, R:9, R:204
F_ApiModelessDialog() function G:214, G:216, G:219, R:5, R:208
F_ApiNewAnchoredFormattedObject() function G:148, R:209
F_ApiNewAnchoredObject() function G:148, G:236, R:14, R:24, R:210
F_ApiNewBookComponentInHierarchy() function R:212
F_ApiNewElement() function G:144, G:150, R:213
F_ApiNewElementInHierarchy() function G:150, R:215
F_ApiNewgraphicObject() function G:101, G:146, G:150, R:9, R:14, R:212
F_ApiNewSeriesObject() function G:11, G:143, G:149, R:14, R:220
F_ApiNewSubObject() function G:152, R:221
F_ApiNewTable() function G:143, G:148, G:154, R:15, R:22, R:224
F_ApiNotification() function G:34, R:25, R:226
F_ApiObjectValid() function R:24, R:237
calling G:48
F_ApiOpenResource() function G:214, G:216, G:217, R:6, R:245
F_ApiPaste() function G:127, R:3, R:23, R:246
F_ApiPopClipboard() function G:129, R:3, R:248
F_ApiPrintFAErrno() function R:4, R:248
F_ApiPrintImportStatus() function R:4, R:250
F_ApiPrintOpenStatus() function G:52, R:4, R:250, R:251
F_ApiPrintPropVal() function R:5, R:251
F_ApiPrintPropVals() function R:5, R:252
F_ApiPrintSaveStatus() function G:65, R:4, R:253
F_ApiPrintTextItem() function R:5, R:253
F_ApiPrintTextItems() function R:5, R:254
F_ApiPrintUpdateBookStatus() function G:360, R:4, R:255
F_ApiPromoteElement() function R:256
F_ApiPromptInt() function G:18, G:223, R:5, R:256
F_ApiPromptMetric() function G:18, G:223, R:5, R:258
F_ApiPromptString() function G:7, G:18, G:223, R:5, R:260
F_ApiPushClipboard() function G:129, R:3, R:261
F_ApiQuickSelect() function G:42, R:3, R:262
F_ApiRedisplay() function G:96, R:7, R:263
F_ApiReformat() function R:97, R:7, R:25, R:264
F_ApiRehyphenate() function G:73, R:7, R:265
F_ApiResetEqnSettings() function G:73, R:7, R:266
F_ApiResetReferenceFrames() function G:73, R:7, R:266
F_ApiRestartPgfNumbering() function G:73, R:7, R:267
F_ApiReturnValue() function G:37, R:3, R:268
F_ApiRun() function R:270
F_ApiSave() function G:59, G:60, R:2, R:6, R:52, R:272
F_ApiScrollBox() function G:22, R:5, R:274
F_ApiScrollToText() function R:276
F_ApiService() function R:277
F_ApiSetAttributeDefs() function R:278
F_ApiSetAttributes() function R:280
F_ApiSetClientDir() function R:25, R:282
F_ApiSetElementRange() function R:282
F_ApiSetFdFunc() function R:284
F_ApiSetId() function G:90, G:163, G:168, R:3, R:16, R:285
F_ApiSetInt() function G:7, G:11, G:90, R:6, R:15, R:16, R:287
F_ApiSetIntByName() function G:264, R:9, R:288
F_ApiSetInts() function G:90, R:16, R:289
F_ApiSetMetric() function G:90, G:99, R:16, R:291
F_ApiSetMetricByName() function G:264, R:9, R:292
F_ApiSetMetrics() function G:90, R:16, R:293
F_ApiSetPoints() function G:90, G:100, R:16, R:295
F_ApiSetPrinterState() function R:297
F_ApiSetPropertyType() function G:7, G:89, G:126, G:213, G:263
F_ApiSetProps() function G:92, G:102, G:126, R:16, R:297
F_ApiSetPropVal() function R:16, R:300
F_ApiSetTabSize() function G:90, R:103, R:16, R:305
F_ApiSetTextLoc() function G:90, R:16, R:308
F_ApiSetTextProp() function G:125
F_ApiSetTextProps() function G:124, R:17, R:23, R:309, R:669
F_ApiSetTextPropVal() function G:124, R:23, R:311
F_ApiSetTextRange() function G:90, G:115, R:17, R:23, R:313
F_ApiSetTextVal() function R:23, R:314
F_ApiSetUBytesByName() function G:264, G:265, R:9, R:316
F_ApiShortcutExists() function R:318
F_ApiShutDown() function R:318
F_ApiSilentPrintDoc() function G:7, G:57, R:2, R:6, R:15, R:319
F_ApiSimpleGenerate() function G:71, G:356, R:2, R:321
F_ApiSimpleImportElementDefs() function G:77, R:322
F_ApiSimpleImportFormats() function G:72, R:2, R:7, R:322
F_ApiSimpleNewDoc() function G:53, G:84, R:6, R:325
F_ApiSimpleOpen() function G:7, G:45, G:46, G:84, R:2, R:6, R:326
F_ApiSimpleSave() function G:41, G:46, G:59, R:2, R:6, R:327
F_ApiSleep() function R:17, R:25, R:328
F_ApiSplitElement() function R:329
F_ApiStartUp() function R:329
F_ApiStraddleCells() function G:75, R:22, R:331
F_ApiStringLength() function R:332
F_ApiSubObject() function R:221
F_ApiSystemShell() function R:333
F_ApiTakeControl() function R:25, R:334
F_ApiUnStraddleCells() function G:75, R:22, R:336
F_ApiUnWrapElement() function R:337
Index - Reserved Names

F_ApiUpdateBook() function G:356, R:338
F_ApiUpdateTextInset() function G:236, R:7, R:24, R:341
F_ApiUpdateVariables() function G:73, R:342
F_ApiUpdateXRefs() function G:73, G:221, R:7, R:343
F_ApiUserCancel() function R:24, R:344
F_ApiUSleep() function R:17, R:25, R:344
F_ApiWinConnectSession() function
F_ApiWinDefaultMessageFilter() function R:346
F_ApiWrapElement() function
F_ApUpdateBook() function calling G:70
F_Assert() function R:347
F_AttributeDefsT structure R:678
F_Calloc() function R:12, R:349
F_ChannelAppend() function R:10, R:350
F_ChannelCloseTmp() function R:10, R:351
F_ChannelEof() function R:10, R:352
F_ChannelFlush() function R:10, R:352
F_ChannelMakeTmp() function R:10, R:353
F_ChannelPeek() function R:10, R:355
F_ChannelRead() function G:377, R:10, R:356
F_ChannelSeek() function R:10, R:357
F_ChannelSize() function R:10, R:357
F_ChannelTell() function R:10, R:358
F_ChannelWrite() function G:369, G:377, R:10, R:359
F_CharIsAlphabetic() function G:382, R:2, R:360
F_CharIsAlphaNumeric() function R:2, R:361
F_CharIsControl() function R:2, R:361
F_CharIsDoubleByteFirst() function R:362, R:363
F_CharIsDoubleByteSecond() function R:364
F_CharIsEol() function R:365
F_CharIsHexadecimal() function R:2, R:366
F_CharIsLower() function R:2, R:366
F_CharIsNumeric() function R:2, R:367
F_CharIsUpper() function R:2, R:368
F_CharToLower() function G:382, R:2, R:368
F_CharToUpper() function G:382, R:2, R:369
F_ClearHandle() function G:379, R:12, R:370
F_ClearPointer() function G:379
F_ClearPtr() function R:12, R:370
F_CombinedFontsT structure R:62, R:679
F_CombinedFontT structure R:680
F_CompareRetT structure G:68, R:680
F_CopyPtr() function R:12, R:371
F_DeleteFile() function G:377, R:8, R:372
F_DuplicateHandle() function R:12, R:372
F_DuplicatePtr() function R:12, R:373
F_ElementCatalogEntriesT structure G:90, G:271, R:683
F_ElementCatalogEntryT structure R:682
F_ElementFmtsT structure G:90
F_ElementLocT structure G:286
F_ElementRangeT structure G:286
F_Exit() function R:374
F_FdeInitFontEncs() function R:20, R:375
F_FilePathBaseName() function R:8, R:376
F_FilePathCloseDir() function R:8, R:377
F_FilePathCopy() function R:7, R:378
F_FilePathFree() function G:373, G:375, R:8, R:378
F_FilePathGetNext() function G:377, R:8, R:379
F_FilePathOpenDir() function R:8, R:380
F_FilePathParent() function R:8, R:381
F_FilePathProperty() function G:377, R:8, R:381
F_FilePathToPathName() function G:375, R:7, R:382
F_FilePathUpdate() function R:8, R:383
F_Fterg() function G:377, R:8, R:384
F_FontEncID() function R:8, R:20
F_FontEncId() function R:384
F_FontEncName() function R:8, R:20, R:385
F_FontEncT structure R:682
F_FontsT structure R:11, R:99, R:682
F_FontT structure R:682
F_FreeHandle() function G:379, R:12, R:386
F_GetFilePath() function R:8, R:387
F_GetHandleSize() function R:12, R:389
F_GetMacFilePathComponent() function R:8, R:388
F_HandleEqual() function R:12, R:389
F_HashCreate() function R:383
F_HashDestroy() function R:9, R:390
F_HashEnumerate() function R:9, R:390
F_HashGet() function R:9, R:391
F_HashRemove() function R:9, R:394
F_HashReportOnData() function R:395
F_HashUpdate() function R:12, R:395
F_HashSet() function G:383, R:9, R:396
F_LanguageNumber() function R:397
F_LanguageString() function R:398
F_LockHandle() function R:12, R:399
F_MakeDir() function R:8, R:400
F_MetricApproxEqual() function R:13, R:401
F_MetricConstrainAngle() function R:13, R:401
F_MetricDiv() function R:13, R:402
F_MetricFractMul() function G:372, G:384, R:13, R:403
F_MetricMake() function R:13, R:404
F_MetricMul() function R:13, R:405
F_MetricNormalizeAngle() function R:13, R:405
F_MetricSqrt() function R:13, R:406
F_MetricSquare() function R:13, R:406
F_MetricToFloat() function G:384, R:13, R:407
F_MifBegin() function R:13, R:408
F_MifComment() function R:13, R:408
F_MifDecimal() function R:13, R:409
F_MifEnd() function R:13, R:410
F_MifGetIndent() function R:13, R:411
F_MifIndent() function R:13, R:411
F_MifIndentDec() function R:13, R:412
F_MifIndentInc() function R:14, R:412
F_MifInteger() function R:14, R:413
F_MifNewLine() function R:14, R:414
F_MifSetIndent() function G:386, R:14, R:414
F_MifSetOutputChannel() function G:386, R:14, R:415
F_MifSpace() function R:14, R:416
F_MifTab() function R:14, R:416
F_MifText() function R:14, R:417
F_MifTimeString() function R:14, R:417
F_NewMacFilePath() function R:8, R:421
F_ObjHandleT data type G:3, G:4, G:90, G:271, R:15, R:675
F_PathNameToFilePath() function G:372, G:375, R:8, R:422
F_PathNameType() function R:8, R:424
F_PointT structure R:295, R:683
F_Printf() function G:369, R:10, R:424
F_Progress() function R:426
F_PROPIdentT structure G:272, R:684
F_PROPValsT structure G:92, G:272, R:4, R:11, R:16, R:684
F_PROPValT structure G:272, R:4, R:11, R:74, R:684
F_PtrEqual() function R:12, R:427
F_ReadBytes() function G:373, R:10, R:427
F_ReadLongs() function R:10, R:428
F_ReadShorts() function R:10, R:430
F_Realloc() function R:12, R:431
F_ReallocHandle() function R:12, R:432
F_RenameFile() function R:8, R:434
F_ResetByteOrder() function G:382, R:10, R:435
F_ResetDirHandle() function R:8, R:435
F_Scanf() function R:10, R:436
F_SetAssert() function G:378, R:437
F_SetByteOrder() function G:382, R:10, R:438
F_SetDSExit() function G:379, R:12, R:439
F_Sprintf() function G:369, R:10, R:440
F_Sscanf() function R:10, R:441
F_StrAlphaToInt() function G:381, R:20, R:441
F_StrAlphaToReal() function R:20, R:442
F_StrBrk() function R:19, R:443
F_StrCat() function G:370, R:19, R:443
F_StrCatCharN() function R:19, R:444
F_StrCatDblCharNEnc() function R:20, R:445
F_StrCatIntN() function R:19, R:446
F_StrCatN() function R:19, R:447
F_StrCatNEnc() function R:20, R:448
F_StrChr() function R:19, R:448
F_StrChrEn() function R:20
F_StrChrEnc() function R:449
F_StrCmp() function G:370, G:381, R:19, R:450
F_StrCmpN() function R:19, R:451
F_StrCmpNEnc() function R:20, R:451
F_StrCopyString() function R:18, R:452
F_StrCopy() function G:370, R:18, R:453
F_StrCpyN() function R:18, R:453
F_StrCpyNEnc() function R:20, R:454
F_StrEqual() function G:370, R:19, R:455
F_StrEqual() function R:19, R:456
F_StrICmp() function R:19, R:457
F_StrICmpEnc() function R:20, R:457
F_StrICmpN() function R:19, R:458
F_StrIEqual() function R:19, R:459
F_StrIEqualEnc() function R:20, R:460
Index - Reserved Names

F_StrIEqualN() function R:19, R:461
F_StrIEqualNEnc() function R:20, R:462
F_StrPrefix() function R:19
F_StrPrefixEnc() function R:463
F_StrIsEmpty() function R:463
F_StrISuffixEnc() function R:20, R:464
F_StrLen() function G:370, R:20, R:465
F_StrLenEnc() function R:20, R:465
F_StrListAppend() function R:18, R:466
F_StrListCat() function R:18, R:466
F_StrListCopy() function R:18, R:467
F_StrListCopyList() function R:18, R:468
F_StrListFirst() function R:18, R:469
F_StrListFree() function R:18, R:469
F_StrListGet() function R:18, R:470
F_StrListIndex() function R:18, R:471
F_StrListInsert() function G:382, R:18, R:472
F_StrListLast() function R:18, R:473
F_StrListLen() function R:18, R:474
F_StrListNew() function G:382, R:18, R:474
F_StrListRemove() function R:18, R:475
F_StrListSetString() function R:18, R:476
F_StrNew() function G:381, R:18, R:477
F_StrPrefix() function R:19, R:478
F_StrPrefixN() function R:19, R:479
F_StrRChr() function R:19, R:480
F_StrRChrEnc() function R:20, R:480
F_StrReverse() function R:20, R:481
F_StrStrip() function R:20, R:482
F_StrStripLeadingSpaces() function R:20, R:482
F_StrStripTrailingSpaces() function R:20, R:483
F_StrSubString() function R:19, R:484
F_StrSuffix() function R:19, R:484
F_StrTok() function R:19, R:485
F_StrTrunc() function R:20, R:486
F_StrTruncEnc() function R:20, R:487
R:685
F_TabT structure G:104, R:4, R:162, R:305, R:685
F_TypedValT structure G:272, R:689
F_TypedValT-u union G:272
F_UBytesT structure R:4, R:11, R:183, R:317, R:691
F_UIntsT structure G:271, R:4, R:11, R:691
F_UnlockHandle() function R:12, R:488
F_ValT structure R:4
F_Warning() function G:379, R:488
F_WriteBytes() function R:10, R:489
F_WriteLongs() function R:10, R:490
F_WriteShorts() function R:10, R:491
FA_errno global variable G:45
FA_Init_DocReport initialization flag G:14, G:15, R:198
FA_Init_First initialization flag G:14, R:198
FA_Init_Subsequent initialization flag G:14, G:15, R:48, R:198
FA_Init_TakeControl initialization flag G:14, R:198
FA_Note_Alert notification constant R:235
FA_Note_BodyPageAdded notification constant R:231
FA_Note_BodyPageDeleted notification constant R:231
FA_Note_ClientCall notification constant R:230, R:269
FA_Note_DirtyBook notification constant R:229
FA_Note_DirtyDoc notification constant R:229
FA_Note_DisplayClientTiDialog notification constant R:230, R:269
FA_Note_FilterFileToFile notification constant R:235, R:236
FA_Note_FilterIn notification constant G:35, R:227, R:235, R:236
FA_Note_FilterOut notification constant G:35, R:229, R:235, R:236
FA_Note_Num notification constant R:232
FA_Note_PostAutoSaveDoc notification constant R:229
FA_Note_PostBookComponentOpen notification constant R:228
FA_Note_PostChangeElement notification constant R:231
FA_Note_PostDistill notification constant R:229
FA_Note_ClientCall notification constant R:230, R:269
FA_Note_DirtyBook notification constant R:229
FA_Note_DirtyDoc notification constant R:229
FA_Note_DisplayClientTiDialog notification constant R:230, R:269
FA_Note_FilterFileToFile notification constant R:235, R:236
FA_Note_PostDragElement notification
constant R:232
FA_Note_PostFileType notification constant G:35, G:41, R:227, R:228
FA_Note_PostGenerate notification constant R:228, R:235
FA_Note_PostGoToXrefSrc notification constant R:231
FA_Note_PostHypertext notification constant R:231, R:235
FA_Note_PostImport notification constant R:230, R:235
FA_Note_PostImportElementDefs notification constant R:232
FA_Note_PostInlineTypeIn notification constant R:232
FA_Note_PostInsertElement notification constant R:231, R:235
FA_Note_PostMouseCommand notification constant R:230, R:236
FA_Note_PostOpenBook notification constant R:228
FA_Note_PostOpenBookMIF notification constant R:228
FA_Note_PostOpenBookt notification constant R:235
FA_Note_PostOpenDoc notification constant G:35, R:227
FA_Note_PostOpenMIF notification constant R:227
FA_Note_PostOpenSGML notification constant R:227
FA_Note_PostOpenXML notification constant R:227
FA_Note_PostPrint notification constant R:231
FA_Note_PostQuitBook notification constant R:229
FA_Note_PostQuitDoc notification constant G:35, G:41, R:227, R:228
FA_Note_PostQuitSession notification constant R:230
FA_Note_PostRevertBook notification constant R:230
FA_Note_PostRevertDoc notification constant R:229
FA_Note_PostSaveAsPDFDialog notification constant R:229, R:237
FA_Note_PostSaveAsPdfDialog notification constant R:269
FA_Note_PostSaveBook notification constant R:229
FA_Note_PostSaveBookMIF notification constant R:229
FA_Note_PostSaveDoc notification constant G:35, R:228
FA_Note_PostSaveMIF notification constant R:228
FA_Note_PostSaveSGML notification constant R:228
FA_Note_PostSaveXML notification constant R:228
FA_Note_PostSetAttrValue notification constant R:232, R:235
FA_Note_PostWrapElement notification constant R:232, R:235
FA_Note_PreAutoSaveDoc notification constant R:229
FA_Note_PreBookComponentOpen notification constant R:228
FA_Note_PreChangeElement notification constant R:231
FA_Note_PreDistill notification constant R:229, R:235
FA_Note_PreDragElement notification constant R:232
FA_Note_PreFileType notification constant G:35, G:41, R:227, R:228
FA_Note_PreFunction notification constant G:38, R:230, R:235
FA_Note_PreGenerate notification constant R:228
FA_Note_PreGenerate notification constant R:235
FA_Note_PreGoToXrefSrc notification constant R:231
FA_Note_PreHypertext notification constant R:231, R:235
FA_Note_PreImport notification constant R:230, R:235
FA_Note_PreImportElementDefs notification constant R:232
FA_Note_PreInlineTypeIn notification constant R:232
FA_Note_PreInsertElement notification constant R:231, R:235
FA_Note_PreMouseCommand notification constant R:230, R:236
FA_Note_PreOpenBook notification constant R:228, R:235
FA_Note_PreOpenBookMIF notification constant R:228
FA_Note_PreOpenDoc notification constant G:35, R:227
FA_Note_PreOpenMIF notification constant R:227
FA_Note_PreOpenSGML notification constant R:227
FA_Note_PreOpenXML notification constant R:227
FA_Note_PrePrint notification constant R:231
FA_Note_PreQuitBook notification constant R:229
FA_Note_PreQuitDoc notification constant G:35, G:44, R:229
FA_Note_PreQuitSession notification constant R:230
FA_Note_PreRevertBook notification constant R:230
FA_Note_PreRevertDoc notification constant R:229
FA_Note_PreSaveAsPDFDialog notification constant R:229, R:237
FA_Note_PreSaveAsPdfDialog notification constant R:269
FA_Note_PreSaveBook notification constant R:229
FA_Note_PreSaveBookMIF notification constant R:229
FA_Note_PreSaveDoc notification constant G:35, R:228
FA_Note_PreSaveMIF notification constant R:228
FA_Note_PreSaveSGML notification constant R:228
FA_Note_PreSaveXML notification constant R:228
FA_Note_PreSaveBook notification constant R:229
FA_Note_PreSaveBookMIF notification constant R:229
FA_Note_PreSaveDoc notification constant G:35, R:228
FA_Note_PreSaveMIF notification constant R:228
FA_Note_PreSaveSGML notification constant R:228
FA_Note_PreSaveXML notification constant R:228
FA_Note_PreSetAttrValue notification constant R:232, R:235
FA_Note_PreWrapElement notification constant R:232
FA_Note_PreWrapElementt notification constant R:235
FA_Note_UpdateAllClientTi notification constant G:237, R:230
FA_Note_UpdateClientTi notification constant G:237, R:230, R:236
FA_PreNotificationPoint notification G:37, R:269
Facets properties (FO_Inset) R:599
fclose() function G:369
FDEsqrt() function R:491
FDEstrtod() function R:492
FDEstrtol() function R:493
FdeSuccess error code G:379
FDIPath path type G:376
FDOSPath path type G:376

FE_

FE_AnsianSystemRequired error code R:693
FE_BadBaseColor error code R:693
FE_BadCompare error code R:693
FE_BadDelete error code R:693
FE_BadDocId error code R:693
FE_BadElementDefId error code R:693
FE_BadElementId error code R:693
FE_BadElementSelection error code R:693
FE_BadFamilyName error code R:693
FE_BadFileType error code R:693
FE_BadInkName error code R:693
FE_BadNew error code R:694
FE_BadNewFrame error code R:694
FE_BadNewGroup error code R:694
FE_BadNewSibling error code R:694
FE_BadNotificationNum error code R:694
FE_BadObjId error code R:694
FE_BadOperation error code R:694
FE_BadPageDelete error code R:694
FE_BadParameter error code R:694
FE_BadPropNum error code R:694
FE_BadPropType error code R:694
FE_BadRange error code R:694
FE_BadSaveFileName error code R:694
FE_BadSelectionForOperation error code R:694
FE_BadShortcut error code R:694
FE_BookUnStructured error code R:694
FE_Busy error code R:694
FE_Canceled error code R:694
FE_CanceledByClient error code R:695
FE_CantSmooth error code R:695
FE_CantUpdateMacEdition error code R:695
FE_CircularReference error code R:695
FE_CompareTypes error code R:695
FE_DocModified error code R:695
FE_DupName error code R:695
FE_EmptyTextObject error code R:695
FE_FailedState error code R:695
FE_FileClosedByClient error code R:695
FE_FilterFailed error code R:695
FE_GenRuleAmbiguous error code R:695
FE_GenRuleConnectorExpected error code R:695
FE_GenRuleItemExpected error code R:695
FE_GenRuleLeftBracketExpected error code R:695
FE_GenRuleMixedConnectors error code R:695
FE_GenRuleRightBracketExpected error code R:695
FE_GenRuleSyntaxError error code R:695
FE_GroupSelect error code R:695
FE_HiddenPage error code R:695
FE_InvalidString error code R:695
FE_InvAttribute error code R:696
FE_InvAttributeDef error code R:696
FE_MissingFile error code R:696
FE_NameNotFound error code R:696
FE_NoColorFamily error code R:696
FE_NoSuchFlow error code R:696
FE_NotApiCommand error code R:696
FE_NotBodyPage error code R:696
FE_NotBookComponent error code R:696
FE_NotCommand error code R:696
FE_NotFound error code R:696
FE_NotFrame error code R:696
FE_NotGraphic error code R:696

FDK Programmer's Reference & Guide index-43
Index - Reserved Names

FE_NotGroup error code R:696
FE_NotInMenu error code R:696
FE_NotMenu error code R:696
FE_NotPgff error code R:696
FE_NotPgffOrFlow error code R:696
FE_NotTextFrame error code R:696
FE_OffsetNotFound error code R:697
FE_OutOfRange error code R:697
FE_PageFrame error code R:697
FE_PropNotSet error code R:697
FE_ReadOnly error code R:697
FE_ReservedColor error code R:697
FE_SomeUnresolved error code R:697
FE_Success error code R:697
FE_SystemError error code R:697
FE_TableInLockedTi error code R:697
FE_TintedColor error code R:697
FE_Transport error code R:697
FE_TypeUnNamed error code R:697
FE_ViewOnly error code R:697
FE_WantsCustom error code R:697
FE_WantsLandscape error code R:697
FE_WantsPortrait error code R:697
FE_WithinFrame error code R:697
FE_WrongProduct error code R:697

FF_

FF_ALERT_CANCEL_DEFAULT flag
(F_ApiAlert()) G:18, R:43
FF_ALERT_CONTINUE_NOTE flag
(F_ApiAlert()) G:18, R:43
FF_ALERT_CONTINUE_WARN flag
(F_ApiAlert()) G:11, G:18, R:43
FF_ALERT_NO_DEFAULT flag (F_ApiAlert()) G:18, R:43
FF_ALERT_OK_DEFAULT flag (F_ApiAlert()) G:18, R:43
FF_ALERT_YES_DEFAULT flag (F_ApiAlert()) G:18, R:43
FF_CLOSE_MODIFIED flag G:67
FF_CMP_CHANGE_BARS flag
(F_ApiCompare()) G:69, R:65
FF_CMP_COMPKIT flag (F_ApiCompare()) G:69, R:66
FF_CMP_SUMKIT flag (F_ApiCompare()) G:69, R:66
FF_CMP_SUMMARY_ONLY flag
(F_ApiCompare()) G:69, R:65
FF_Custom_FirstPageLeft flag
(F_ApiCustomDoc()) G:56, R:71
FF_Custom_FirstPageRight flag
(F_ApiCustomDoc()) G:56, R:71
FF_Custom_SingleSided flag
(F_ApiCustomDoc()) G:56, R:71
FF_CUT_TBL_CELLS flag
(F_ApiClipboardFunction()) G:128, R:57, R:72
FF_DONT_APPLY_ALL_ROWS flag
(F_ApiPaste()) G:128, R:246
FF_DONT_DELETE_HIDDEN_TEXT flag
(F_ApiClipboardFunction()) G:128, R:57, R:73, R:246
FF_FilePathDirectory bit flag
(F_FilePathProperty()) R:382
FF_FilePathExist bit flag (F_FilePathProperty()) R:382
FF_FilePathFile bit flag (F_FilePathProperty()) R:382
FF_FilePathReadable bit flag
(F_FilePathProperty()) R:382
FF_FilePathWritable bit flag
(F_FilePathProperty()) R:382
FF_INSERT_BELOW_RIGHT flag
(F_ApiPaste()) G:128, R:246
FF_INTERACTIVE flag
(F_ApiClipboardFunction()) G:128, R:57, R:68, R:72, R:246
FF_REPLACE_CELLS flag (F_ApiPaste()) G:128, R:246
FF_STRIP_HYPERTEXT flag
(F_ApiClipboardFunction()) R:68
FF_UFF_COLOR flag
(F_ApiSimpleImportFormats()) G:72, R:324
FF_UFF_COMBINED_FONTS flag
(F_ApiSimpleImportFormats()) G:72
FF_UFF_CombinedFont flag
(F_ApiSimpleImportFormats()) R:324
FF_UFF_COND flag
(F_ApiSimpleImportFormats()) G:72, R:324
FF_UFF_DOCUMENT_PROPS flag
(F_ApiSimpleImportFormats()) G:72, R:324
FF_UFF_FONT flag
(F_ApiSimpleImportFormats()) G:72, R:324
FF_UFF_MATH flag
(F_ApiSimpleImportFormats()) G:72, R:324
FF_UFF_PAGE flag
(F_ApiSimpleImportFormats()) G:72, R:324
FF_UFF_PGF flag
   (F_ApiSimpleImportFormats()) G:72, R:324
FF_UFF_REFPAGE flag
   (F_ApiSimpleImportFormats()) G:72, R:324
FF_UFF_REMOVE_EXCEPTIONS flag
   (F_ApiSimpleImportFormats()) G:73, R:324
FF_UFF_REMOVE_PAGE_BREAKS flag
   (F_ApiSimpleImportFormats()) G:73, R:324
FF_UFF_TABLE flag
   (F_ApiSimpleImportFormats()) G:72, R:324
FF_UFF_VAR flag
   (F_ApiSimpleImportFormats()) G:73, R:324
FF_UFF_XREF flag
   (F_ApiSimpleImportFormats()) G:73, R:324
FF_VISIBLE_ONLY flag
   (F_ApiClipboardFunction()) G:128, R:57, R:68, R:73, R:246
FF_XRUI_CLOSED_DOCS flag
   (F_ApiUpdateXRefs()) R:343
FF_XRUI_EVERYTHING flag
   (F_ApiUpdateXRefs()) R:343
FF_XRUI_FORCE_UPDATE flag
   (F_ApiUpdateXRefs()) R:343
FF_XRUI_INTERNAL flag
   (F_ApiUpdateXRefs()) R:343
FF_XRUI_OPEN_DOCS flag
   (F_ApiUpdateXRefs()) R:343
FilePathT structure G:375, R:382, R:677
FMacPath path type G:376
FO_AFrame objects G:280, G:295, R:596
   properties of R:596
FO_Arc objects G:280, R:598
   properties of R:598
   properties of R:610
FO_Book objects G:351, R:84, R:495
   properties of G:353, R:495
FO_BookComponent objects G:351, R:507
   properties of R:507
   properties of G:337, R:648
   properties of R:515
   properties of R:519
FO_CombinedFontDefn objects G:269, G:282
   properties of R:522
FO_Command objects G:161, G:269, R:75, R:81
   properties of G:162, R:523
FO_CondFmt objects G:269, G:281, G:310, R:522, R:529
   properties of R:529
FO_DialogResource objects G:211, G:214, G:221, R:531
   properties of R:531
FO_DlgBox objects G:211, R:533
FO_DlgButton objects G:211, R:533
   properties of R:533
FO_DlgCheckBox objects G:211, R:533
   properties of R:533
FO_DlgEditBox objects G:211, R:534
   properties of R:534
FO_DlgImage objects G:211, G:218, R:534
   properties of R:534
FO_DlgLabel objects G:211, R:534
   properties of R:534
FO_DlgPopUp objects G:211, G:218, R:535
   properties of R:535
FO_DlgRadioButton objects G:211, R:535
   properties of R:535
FO_DlgScrollBox objects G:211, R:536
   properties of R:536
FO_DlgScrollBar objects G:211, G:218, R:536, R:537
   properties of R:536, R:537
FO_DlgTriBox objects G:211, R:537
   properties of R:537
   properties of G:304, R:537
FO_Element objects G:281, R:638
   properties of R:638
FO_ElementDef objects G:269, G:281, G:343, R:645
   properties of R:645
FO_Ellipse objects G:280, R:598
   properties of R:598
   properties of R:579
FO_FmtChangeList objects G:106, G:282, G:346, R:582
FO_FmtRule objects G:281, G:343, G:345
Index - Reserved Names

FO_FmtRuleClause objects G:282
properties of R:581

FO_Fn objects G:281, G:303, G:327, R:581
properties of R:581

FO_Group objects G:280, G:296, R:599
properties of R:599

FO_HiddenPage objects G:280, G:289, G:292, R:84, R:611
properties of R:611

facets of G:257
properties of R:599

FO_Line objects G:280, G:295, R:602
properties of R:602

FO_Marker objects G:281, G:319, R:608
properties of R:608

FO_MarkerType objects G:269, G:281, R:609
properties of R:609

properties of R:612

FO_Math objects G:280, R:603
properties of R:603

FO_Menu objects G:161, G:270, R:77, R:82
properties of R:523, R:528

FO_MenuItemSeparator objects G:161, G:169, G:270
properties of R:523, R:528

properties of R:613

example property list for (illustrated) G:273
properties of R:621

FO_Polygon objects G:280, R:603
properties of R:603

FO_Polyline objects G:280, R:604
properties of R:604

FO_Rectangle objects G:3, G:280, R:604
properties of R:604

properties of R:612

FO_RoundRect objects G:280, R:604
properties of R:604

FO_Row objects G:281, G:332, G:333, R:83, R:651
properties of R:651

FO_Rubi objects G:282, R:630
properties of R:630

FO_RulingFmt objects G:270, G:281, G:329, R:630
properties of R:630

properties of R:631

FO_SubCol objects G:111, G:280, G:299, R:520

FO_Tbl objects G:281, G:312, G:332, R:652
properties of R:652

FO_TblFmt objects G:270, G:281, G:330, R:658
properties of R:658

properties of R:605

FO_TxtLine objects G:111, G:280, R:606
properties of R:606

FO_TiApiClient objects G:230, G:281, R:666
properties of R:666

properties of R:667

FO_TiText objects G:229, G:270, G:281, R:667
property of R:667

FO_TiTextTable objects G:229, G:281, R:668
properties of R:668

properties of R:608

FO_Var objects G:111, G:281, G:326, R:672
properties of R:672

FO_VarFmt objects G:270, G:281, G:325, R:84, R:672
properties of R:672

FO_XRef objects G:281, G:323, R:530
properties of R:530

FO_XRefFmt objects G:270, G:281, G:322, R:531
properties of R:531

FontEncIdT data type R:675

fopen() function G:369

FP_

FP_AcrobatBookmarkDisplayTags property (FO_Book) R:498

FP_AcrobatBookmarkDisplayTags property (FO_Doc) R:542

FP_AcrobatLevel property (FO_PgfFmt) R:622

FP_ActiveBook property (FO_Session) G:83, G:275, R:631

FP_ActiveDoc property (FO_Session) G:10, G:83, G:95, G:275, R:631
Index - Reserved Names

FP_AddMarkerTypeToStandardMarkers property (FO_Session) R:632
FP_AdjHyphens property (FO_FmtChangeList) R:582
FP_AdjHyphens property (FO_Pgf) R:617
FP_AdjHyphens property (FO_PgfFmt) R:625
FP_AFrameIsCropped property (FO_AFrame) R:596
FP_AFrameIsFloating property (FO_AFrame) R:596
FP_Alignment property (FO_AFrame) R:596
FP_AllowAsSpecialCase property (FO_Element) R:643
FP_AlsoInsert property (FO_ElementDef) R:645
FP_AnchorType property (FO_AFrame) R:596
FP_Angle property (FO_GraphicObject) R:592
FP_ApplyFmtRules property (FO_Session) G:110
FP_ApplyFormatRules property (FO_Element) G:110, R:631
FP_ArrowBaseAngle property (FO_GraphicObject) R:592
FP_ArrowLength property (FO_GraphicObject) R:592
FP_ArrowScaleFactor property (FO_GraphicObject) R:592
FP_ArrowScaleHead property (FO_GraphicObject) R:593
FP_ArrowTipAngle property (FO_GraphicObject) R:593
FP_ArrowType property (FO_GraphicObject) R:593
FP_AttrDisplay property (FO_Element) R:638
FP_AttributeDefs property (FO_ElementDef) R:645
FP_Attributes property (FO_Element) R:638
FP_AttributeValueInvalid property (FO_Element) R:643
FP_AutoBackup property (FO_Session) R:632
FP_AutoChangeBars property (FO_Doc) R:549
FP_AutoNumChar property (FO_FmtChangeList) R:584
FP_AutoNumChar property (FO_Pgf) R:613
FP_AutoNumChar property (FO_PgfFmt) R:623
FP_AutoNumString property (FO_FmtChangeList) G:347, R:584
FP_AutoNumString property (FO_Pgf) R:303, R:613
FP_AutoNumString property (FO_PgfFmt) R:623
FP_AutoSave property (FO_Session) R:632
FP_AutoSaveSeconds property (FO_Session) R:632
FP_BaseFamily property (FO_CombinedFontDefn) R:522
FP_BaselineOffset property (FO_AFrame) R:596
FP_BasePointX property (FO_Math) R:603
FP_BasePointX property (FO_TextLine) R:606
FP_BasePointY property (FO_Math) R:603
FP_BasePointY property (FO_TextLine) R:607
FP_Black property (FO_Color) R:519
FP_BlockLines property (FO_FmtChangeList) R:588
FP_BlockLines property (FO_Pgf) R:619
FP_BlockLines property (FO_PgfFmt) R:627
FP_BookComponent property (FO_Element) R:642
FP_BookComponentIsGeneratable property (FO_BookComponent) R:508
FP_BookComponentMissing property (FO_Element) R:643
FP_BookComponentType property (FO_BookComponent) R:508
FP_BookDontUpdateReferences property (FO_Book) R:495
FP_BookIsModified property (FO_Book) R:495
FP_BookIsSelected property (FO_Book) R:495
FP_BookIsViewOnly property (FO_Book) R:507
FP_BookParent property (FO_BookComponent) R:508
FP_BookTreeRoot property (FO_Book) G:355
FP_BorderWidth property (FO_GraphicObject) R:593
FP_BottomMargin property (FO_Doc) R:559
FP_BottomRowSelection property (FO_Tbl) G:87, R:656
FP_BottomSeparator property (FO_FmtChangeList) R:582
FP_BottomSeparator property (FO_Pgf) R:620
FP_BottomSeparator property (FO_PgfFmt) R:628
FP_BottomSepAtIndent property (FO_FmtChangeList) R:583
FP_CanHaveCheckMark property (FO_Command) G:173, R:524
FP_Capitalization property (FO_CharFmt) R:515
FP_Capitalization property (FO_Doc) R:573
FP_Capitalization property (FO_FmtChangeList) R:585
FP_Capitalization property (FO_Pgf) R:614
FP_Capsulation property (FO_FmtChangeList) R:623
FP_Capitalization property (text location) R:669
FP_CellAboveInCol property (FO_Cell) R:648
FP_CellAngle property (FO_Cell) R:648
FP_CellBelowInCol property (FO_Cell) R:648
FP_CellBottomMargin property (FO_FmtChangeList) R:589
FP_CellBottomMargin property (FO_Pgf) R:616
FP_CellBottomMargin property (FO_PgfFmt) R:628
FP_CellBottomMarginChange property (FO_FmtChangeList) R:589
FP_CellBottomMarginFixed property (FO_FmtChangeList) R:589
FP_CellColNum property (FO_Cell) R:648
FP_CellDefaultBottomRuling property (FO_Cell) G:337, R:648
FP_CellDefaultLeftRuling property (FO_Cell) G:336, R:648
FP_CellDefaultRightRuling property (FO_Cell) R:648
FP_CellDefaultTopRuling property (FO_Cell) G:334, R:648
FP_CellIsShown property (FO_Cell) R:648
FP_CellIsStraddled property (FO_Cell) G:337, R:648
FP_CellLeftMargin property (FO_FmtChangeList) R:589
FP_CellLeftMargin property (FO_Pgf) R:616
FP_CellLeftMargin property (FO_PgfFmt) R:628
FP_CellLeftMarginChange property (FO_FmtChangeList) R:589
FP_CellLeftMarginFixed property (FO_FmtChangeList) R:589
FP_CellMarginsFixed property (FO_Pgf) R:616
FP_CellMarginsFixed property (FO_PgfFmt) R:628
FP_CellNumColsStraddled property (FO_Cell) G:337, R:648
FP_CellNumRowsStraddled property (FO_Cell) G:337, R:648
FP_CellOverrideBottomRuling property (FO_Cell) G:337, R:649
FP_CellOverrideFill property (FO_Cell) G:334, G:336, R:649
FP_CellOverrideLeftRuling property (FO_Cell) G:336, R:649
FP_CellOverrideRightRuling property (FO_Cell) R:649
FP_CellOverrideShading property (FO_Cell) G:334, R:650
FP_CellOverrideTopRuling property (FO_Cell) G:334, R:650
FP_CellRow property (FO_Cell) R:649
FP_CellRightMargin property (FO_FmtChangeList) R:589
FP_CellRightMargin property (FO_Pgf) R:616
FP_CellRightMargin property (FO_PgfFmt) R:628
FP_CellRightMarginChange property (FO_FmtChangeList) R:589
FP_CellRightMarginFixed property (FO_FmtChangeList) R:589
FP_CellRow property (FO_Cell) R:649
FP_CellTopMargin property (FO_FmtChangeList) R:589
FP_CellTopMargin property (FO_Pgf) R:616
FP_CellTopMargin property (FO_PgfFmt) R:628
FP_CellTopMarginChange property (FO_FmtChangeList) R:589
FP_CellTopMarginFixed property (FO_FmtChangeList) R:589
FP_CellUseOverrideBRuling property (FO_Cell) R:649
FP_CellUseOverrideFill property (FO_Cell) G:334, G:336, R:649
FP_CellUseOverrideLRuling property (FO_Cell) R:649
FP_CellUseOverrideRRuling property (FO_Cell) R:649
FP_CellUseOverrideShading property (FO_Cell) G:334, R:650
FP_CellUseOverrideTRuling property (FO_Cell) R:650
FP_CellVAlignment property (FO_FmtChangeList) R:589
FP_CellVAlignment property (FO_Pgf) R:617
FP_CellVAlignment property (FO_PgfFmt) R:629
FP_ChangeBar property (FO_CharFmt) R:515
FP_ChangeBar property (FO_Doc) R:573
FP_ChangeBar property (FO_FmtChangeList) R:585
FP_ChangeBar property (FO_Pgf) R:614
FP_ChangeBar property (FO_PgfFmt) R:623
FP_ChangeBar property (text location) R:669
FP_ChangeBarColor property (FO_Doc) R:549
FP_ChangeBarDistance property (FO_Doc) R:549
FP_ChangeBarPosition property (FO_Doc) R:549
FP_ChangeBarThickness property (FO_Doc) R:550
FP_ChapNumComputeMethod property (FO_BookComponent) R:509
FP_ChapNumComputeMethod property (FO_Doc) R:546
FP_ChapterNumber property (FO_BookComponent) R:509
FP_ChapterNumber property (FO_Doc) R:546
FP_ChapterNumStyle property (FO_BookComponent) R:509
FP_ChapterNumStyle property (FO_Doc) R:546
FP_ChapterNumText property (FO_BookComponent) R:509
FP_ChapterNumText property (FO_Doc) R:546
FP_Characteristic properties (FO_CharFmt) G:308
Index - Reserved Names

FP_CharTag property (FO_CharFmt) R:515
FP_CharTag property (FO_Doc) R:573
FP_CharTag property (text location) G:317, R:669
FP_CheckMarkIsOn property (FO_Command) G:173, R:524
FP_ColGap property (FO_Doc) R:559
FP_ColGapWidth property (FO_TextFrame) R:605
FP_Color property (FO_CharFmt) R:515
FP_Color property (FO_Doc) R:573
FP_Color property (FO_FmtChangeList) R:585
FP_Color property (FO_GraphicObject) R:593
FP_Color property (FO_Pgf) R:614
FP_Color property (FO_PgfFmt) R:623
FP_Color property (text location) R:669
FP_ColorOverprint property (FO_Color) R:519
FP_ColorPrintCtl property (FO_Color) R:519
FP_ColorTintPercent property (FO_Color) R:519
FP_ColorViewCtl property (FO_Color) R:519
FP_ContextLabel property (FO_Element) R:643
FP_ContextLabel property (FO_FmtRuleClause) G:346, R:590
FP_DefaultExclusions property (FO_Doc) R:566
FP_DefaultInclusions property (FO_Doc) R:566
FP_Dictionary property (FO_Doc) R:546
FP_DontUpdateTextInsets property (FO_Doc) R:547
FP_DontUpdateXRefs property (FO_Doc) R:547
FP_DocAcrobatColumnArticleThreads property (FO_Book) R:499
FP_DocAcrobatColumnArticleThreads property (FO_Doc) R:542
FP_DocAcrobatDefaultChanged property (FO_Book) R:499
FP_DocAcrobatDefaultChanged property (FO_Doc) R:542
FP_DocFluidFlow property (FO_Doc) R:578
FP_DocsDoubleSided property (FO_Doc) R:559
FP_DocsType property (FO_Doc) R:546
FP_DocsModified property (FO_Doc) R:547
FP_DocsViewOnly property (FO_Doc) R:547, R:578
FP_DocOpenType property (FO_Doc) R:547
FP_DocSaveType property (FO_Doc) R:547
FP_DontUpdateTextInsets property (FO_Doc) R:547
FP_DontUpdateXRefs property (FO_Doc) R:547
FP_DoubleClick property
   (FO_DialogResource) G:221, R:531
FP_DoubleClick property (FO_DlgScrollBox) R:536
FP_DTheta property (FO_Arc) R:598
FP_ElementCatalog property (FO_Book) R:503
FP_ElementCatalog property (FO_Doc) R:566
FP_ElementCatalogDisplay property
   (FO_Book) R:504
FP_ElementCatalogDisplay property (FO_Doc) R:566
FP_ElementDef property (FO_ElementDef) R:642
FP_ElementDef property (FO_FmtRule) G:345, R:590
FP_ElementDefType property
   (FO_ElementDef) R:646
FP_ElementInCatalog property
   (FO_ElementDef) G:344, R:646
FP_ElementIsCollapsed property (FO_Element) R:638
FP_ElementIsExcludedInContext property
   (FO_Element) R:644
FP_ElementIsInvalidInParent property
   (FO_Element) G:348, R:644
FP_ElementIsInvalidInPosition property
   (FO_Element) R:644
FP_ElementIsUndefined property
   (FO_Element) R:643
FP_ElementPgf property (FO_ElementDef) R:646
FP_ElementPgfFormat property
   (FO_ElementDef) G:344
FP_ElementSelection property G:287
FP_ElementSelection property (FO_Book) R:504
FP_ElementSelection property (FO_Doc) R:565
FP_ElementType property (FO_Element) R:639
FP_ElemPrefixSuffix property
   (FO_FmtRuleClause) R:591
FP_EnabledWhen property (FO_Command) G:174, R:524
table of values R:526
FP_EqnIntegralSizeLarge property (FO_Doc) R:550
FP_EqnIntegralSizeMed property (FO_Doc) R:550
FP_EqnIntegralSizeSmall property (FO_Doc) R:550
FP_EqnLevel1SizeLarge property (FO_Doc) R:550
FP_EqnLevel1SizeMed property (FO_Doc) R:550
FP_EqnLevel1SizeSmall property (FO_Doc) R:550
FP_EqnLevel2SizeLarge property (FO_Doc) R:550
FP_EqnLevel2SizeMed property (FO_Doc) R:550
FP_EqnLevel2SizeSmall property (FO_Doc) R:550
FP_EqnLevel3SizeLarge property (FO_Doc) R:551
FP_EqnLevel3SizeMed property (FO_Doc) R:551
FP_EqnLevel3SizeSmall property (FO_Doc) R:551
FP_EqnSigmaSizeMed property (FO_Doc) R:551
FP_EqnSigmaSizeSmall property (FO_Doc) R:551
FP_ErrorInBookComponent property
   (FO_Element) R:643
FP_Exclusions property (FO_ElementDef) R:646
FP_ExpandOMaticParent property
   (FO_Command) R:524
FP_ExportFilters property (FO_Session) R:633
FP_ExtractElementTags property
   (FO_BookComponent) R:514
FP_ExtractTags property
   (FO_BookComponent) R:510
FP_FamilyName property (FO_Color) R:519
FP_Fcode property (FO_Command) R:524
FP_Fcodes property (FO_Command) R:524
FP_FileExtensionOverride property (FO_Doc) R:504
FP_Fill property (FO_GraphicObject) G:11, R:593
FP_FirstAFrame property (FO_SubCol) R:521
FP_FirstAFrame property (FO_TextFrame) R:605
FP_FirstCell property (FO_SubCol) R:521
FP_FirstCell property (FO_TextFrame) R:605
FP_FirstCellInRow property (FO_Row) G:336, R:651
FP_FirstCharFmtInDoc property (FO_Doc) G:283, G:311, R:538
FP_FirstChildElement property (FO_Element) G:121, R:642
FP_FirstColorInDoc property (FO_Doc) G:282, R:538
FP_FirstCombinedFontDefinDoc property
   (FO_Doc) G:283, R:538
FP_FirstCommandInSession property
   (FO_Session) G:162, R:632
FP_FirstComponentInBook property
   (FO_Book) G:353, G:354, R:495
FP_FirstCondFmtInDoc property (FO_Doc) G:283, G:311, R:538
FP_FirstElementDefInDoc property (FO_Book) R:504
FP_FirstElementDefInDoc property (FO_Doc) R:566
FP_FirstElementDefInDocInDoc property
   (FO_Doc) G:283
FP_FirstFmtChangeListInDoc property
   (FO_Doc) R:504
FP_FirstFmtChangeListInDoc property
   (FO_Doc) R:566
FP_FirstFn property (FO_SubCol) R:521
Index - Reserved Names

FP_FirstFn property (FO_TextFrame) \textit{R:605}

FP_FirstFnInDoc property (FO_Doc) G:283, \textit{R:538}


FP_FirstGraphicInFrame property (FO_AFrame) G:85, G:290, G:295, \textit{R:597}

FP_FirstGraphicInFrame property (FO_UnanchoredFrame) \textit{R:608}

FP_FirstGraphicInGroup property (FO_Group) G:296, \textit{R:599}

FP_FirstIndent property (FO_FmtChangeList) \textit{R:584}

FP_FirstIndent property (FO_Pgf) \textit{R:619}

FP_FirstIndent property (FO_PgfFmt) \textit{R:626}

FP_FirstIndentChange property (FO_FmtChangeList) \textit{R:584}

FP_FirstIndentIsRelative property (FO_FmtChangeList) \textit{R:584}

FP_FirstMarkerInDoc property (FO_Doc) G:282, \textit{R:538}

FP_FirstMarkerTypeInDoc property (FO_Doc) G:282, \textit{R:538}


FP_FirstMenuItemInMenu property (FO_Menu) G:162, \textit{R:528}

FP_FirstMenuItemInSession property (FO_Session) G:162, \textit{R:632}

FP_FirstOpenBook property (FO_Session) G:275, \textit{R:632}

FP_FirstOpenDoc property (FO_Session) G:275, \textit{R:632}

FP_FirstPageNum property (FO_BookComponent) \textit{R:510}

FP_FirstPageNum property (FO_Doc) \textit{R:559}

FP_FirstPageVerso property (FO_Doc) \textit{R:559}

FP_FirstPgf property (FO_Cell) G:76, G:337, \textit{R:650}

FP_FirstPgf property (FO_Fn) G:328, \textit{R:581}

FP_FirstPgf property (FO_SupCol) \textit{R:521}

FP_FirstPgf property (FO_Tbl) G:33, \textit{R:657}

FP_FirstPgf property (FO_TextFrame) G:304, \textit{R:605}


FP_FirstPgfInDoc property (FO_Doc) G:282, G:304, \textit{R:539}

FP_FirstPgfRules property G:108

FP_FirstPgfRules property (FO_ElementDef) G:343, \textit{R:646}


FP_FirstRowInTbl property (FO_Tbl) G:336, \textit{R:654}

FP_FirstRubInDoc property (FO_Doc) G:283, \textit{R:539}


FP_FirstSelectedComponentInBook property (FO_Book) G:353, \textit{R:495}


FP_FirstSelectedTblInDoc property (FO_Doc) \textit{R:539}, \textit{R:565}

FP_FirstSubCol property (FO_TextFrame) G:299, \textit{R:605}

FP_FirstTblFmtInDoc property (FO_Doc) G:283, G:331, \textit{R:539}

FP_FirstTblInDoc property (FO_Doc) G:283, \textit{R:539}

FP_FirstVarFmtInDoc property (FO_Doc) G:282, \textit{R:539}

FP_FirstVarInDoc property (FO_Doc) G:282, \textit{R:539}

FP_FirstVis property (FO_DlgScrollBox) \textit{R:536}

FP_FirstXRefFmtInDoc property (FO_Doc) G:282, \textit{R:539}

FP_FirstXRefInDoc property (FO_Doc) G:282, \textit{R:540}

FP_Flow property (FO_TextFrame) G:270, \textit{R:605}

FP_FlowIsAutoConnect property (FO_Flow) \textit{R:580}

FP_FlowIsFeathered property (FO_Flow) \textit{R:580}

FP_FlowIsPostScript property (FO_Flow) \textit{R:580}

FP_FlowIsSynchronized property (FO_Flow) \textit{R:580}

FP_FlowTreeRoot property (FO_Flow) G:300

FP_FM_BinDir property (FO_Session) \textit{R:632}

FP_FM_CurrentDir property (FO_Session) \textit{R:632}

FP_FM_HelpDir property (FO_Session) \textit{R:633}

FP_FM_HomeDir property (FO_Session) \textit{R:633}

FP_FM_InitDir property (FO_Session) \textit{R:633}

FP_FM_StructureDir property (FO_Session) \textit{R:637}

FP_FM_XmlDir property (FO_Session) \textit{R:638}

FP_Fmt property (FO_VarFmt) G:325, \textit{R:672}

FP_Fmt property (FO_XRefFmt) G:323, \textit{R:531}

FP_FmtChangeList property G:107

FP_FmtChangeList property (FO_FmtRuleClause) G:346, \textit{R:591}

FP_FmtChangeListName property (FO_FmtRuleClause) \textit{R:591}

FP_FmtChangeListTag property G:107, G:153

FP_FmtRule property (FO_FmtRuleClause) G:346, \textit{R:591}

FP_FmtRuleClause property (FO_ElementDef) G:345
FP_FmtRuleClause property (FO_FmtRule)  R:590
FP_FmtRuleClauses property (FO_FmtRule)  G:345, R:590
FP_FmtRuleType property (FO_FmtRule)  G:345, R:590
FP_FnCustNumString property (FO_Doc)  R:557
FP_FnFirstNum property  (FO_BookComponent)  R:510
FP_FnFirstNum property (FO_Doc)  R:557
FP_FnFmt property (FO_Doc)  R:557
FP_FnHeightPerCol property (FO_Doc)  R:558
FP_FnInstancePosition property (FO_Doc)  R:558
FP_FnInstancePrefix property (FO_Doc)  R:558
FP_FnInstanceSuffix property (FO_Doc)  R:558
FP_FnNum property (FO_Fn)  G:328, R:581
FP_FnNumPerLine property (FO_Doc)  R:558
FP_FnNumComputeMethod property  (FO_BookComponent)  R:510
FP_FnNumStyle property  (FO_BookComponent)  R:558
FP_FnNumberingPerPage property (FO_Doc)  R:558
FP_FnNumStyle property (FO_Doc)  G:328, R:558
FP_FnRefPosition property (FO_Doc)  R:559
FP_FnRefPrefix property (FO_Doc)  R:559
FP_FnRefSuffix property (FO_Doc)  R:559
FP_FontAngle property (FO_CharFmt)  R:515
FP_FontAngle property (FO_Doc)  R:574
FP_FontAngle property (FO_FmtChangeList)  R:586
FP_FontAngle property (FO_Pgf)  R:614
FP_FontAngle property (FO_PgfFmt)  R:624
FP_FontAngle property (text location)  R:669
FP_FontAngleNames property (FO_Session)  G:277, R:633
FP_FontEncodingName property  (FO_CharFmt)  R:515
FP_FontEncodingName property  (FO_FmtChangeList)  R:586
FP_FontEncodingName property (FO_Pgf)  R:614
FP_FontEncodingName property (FO_PgfFmt)  R:624
FP_FontVariation property  (FO_FmtChangeList)  R:586
FP_FontWeight property (FO_CharFmt)  R:516
FP_FontWeight property (FO_Doc)  R:574
FP_FontWeight property (FO_FmtChangeList)  R:587
FP_FontWeight property (FO_Pgf)  R:615
FP_FontWeight property (FO_PgfFmt)  R:624
FP_FontWeight property (text location)  G:317, R:670
FP_FontWeight property (FO_Session)  G:277, R:634
FP_FontWeightNames property (FO_Session)  G:277, R:634
FP_FormatName property  G:107
FP_FormatOverride property (FO_Element)  R:639
FP_FormatTag property (FO_FmtRuleClause)  R:591
FP_FrameParent property  (FO_GraphicObject)  G:101, R:593
Index - Reserved Names

FP_FrameParent property (FO_Rectangle) G:4
FP_FrameParent property (FO_SubCol) R:521
FP_Functions property (FO_Doc) R:551
FP_GeneralRule property (FO_ElementDef) G:344, R:647
FP_GeneralRuleErrorOffsets property (FO_ElementDef) R:647
FP_GenerateAcrobatInfo property (FO_Book) R:499
FP_GenerateAcrobatInfo property (FO_Doc) R:543
FP_GenerateInclude property (FO_BookComponent) R:511
FP_GraphicCantBeSelected property (FO_GraphicObject) R:593
FP_GraphicIsButton property (FO_TextFrame) R:605
FP_GraphicIsSelected property (FO_GraphicObject) R:593
FP_Gravity property (FO_Session) R:634
FP_GreekSize property (FO_Session) R:634
FP_GroupParent property (FO_GraphicObject) G:101, R:593
FP_HasShiftOrUnshiftCommand property (FO_Command) R:524
FP_HeadArrow property (FO_GraphicObject) R:593
FP_Height property (F_ApiSetMetric()) G:97, G:99
FP_Height property (FO_GraphicObject) G:3, G:263, R:593
FP_Height property (FO_Rectangle) G:3
FP_Height property (FO_Row) R:651
FP_Height property (FO_SubCol) R:521
FP_Height property (text location) R:670
FP_HelpLink property G:176
FP_HelpLink property (FO_Command) R:525
FP_HelpLink property (FO_DialogResource) R:532
FP_HiddenPage property (FO_Doc) G:283, G:292, R:540
FP_HighestLevelElement property (FO_Book) R:504
FP_HighestLevelElement property (FO_Cell) R:650
FP_HighestLevelElement property (FO_Flow) R:581
FP_HighestLevelElement property (FO_Tbl) R:657
FP_HoleBeforeElement property (FO_Element) R:644
FP_HostName property (FO_Session) R:634
FP_HypertextCommandText property (FO_Doc) R:552
FP_HypertextDoValidate property (FO_Doc) R:551
FP_HypertextParseBadParam property (FO_Doc) R:552
FP_HypertextParsedArgs property (FO_Doc) R:552
FP_HypertextParsedClientName property (FO_Doc) R:553
FP_HypertextParsedCmdCode property (FO_Doc) R:552, R:555
FP_HypertextParsedCmdDest property (FO_Doc) R:552
FP_HypertextParsedCmdDestObjID property (FO_Doc) R:552
FP_HypertextParsedCmdDestObjType property (FO_Doc) R:552
FP_HypertextParsedCmdMatrixColumns property (FO_Doc) R:552
FP_HypertextParsedCmdMatrixRows property (FO_Doc) R:552
FP_HypertextParsedFlowName property (FO_Doc) R:553
FP_HypertextParsedLinkName property (FO_Doc) R:553
FP_HypertextParsedMessage property (FO_Doc) R:553
FP_HypertextParsedPageName property (FO_Doc) R:553
FP_HypertextParsedTitle property (FO_Doc) R:553
FP_HypertextParseErr property (FO_Doc) R:552
FP_HypertextParseErrMsg property (FO_Doc) R:552
FP_HypertextValidateErr property (FO_Doc) R:552
FP_Hyphenate property (FO_FMTChangeList) R:583
FP_Hyphenate property (FO_Pgf) R:617
FP_Hyphenate property (FO_PgfFmt) R:625
FP_HyphMinPrefix property (FO_FMTChangeList) R:583
FP_HyphMinPrefix property (FO_Pgf) R:617
FP_HyphMinPrefix property (FO_PgfFmt) R:625
FP_HyphMinSuffix property (FO_FMTChangeList) R:583
FP_HyphMinSuffix property (FO_Pgf) R:617
FP_HyphMinSuffix property (FO_PgfFmt) R:625
FP_HyphMinWord property (FO_FMTChangeList) R:583
FP_HyphMinWord property (FO_Pgf) R:617
FP_HyphMinWord property (FO_PgfFmt) R:625
FP_HiconBarOn property (FO_Session) R:634
FP_Hiconified property (FO_Doc) G:97
FP_ImportFilters property (FO_Session) R:634
FP_ImportFmtInclude property (FO_BookComponent) R:511
FP_ImportHint property (FO_Inset) R:599
FP_ImportHint property (FO_BookComponent) R:511
syntax of FP_ImportHint strings G:247, R:600
FP_ImportHint property (FO_TextInset) R:662
syntax of FP_ImportHint strings  R:663
FP_Inclusions property (FO_ElementDef) R:647
FP_InCond property (FO_Doc) R:574
FP_InCond property (FO_Row) G:333, R:651
FP_InCond property (text location) G:126, G:317, R:670
FP_IncrVal property (FO_DlgScrollBar) R:536
FP_InitStructurePattern property (FO_ElementDef) R:646
FP_InsertLinks property (FO_BookComponent) R:511
FP_InsetDpi property (FO_Inset) G:256, G:259, G:263, R:599
FP_InsetEditor property (FO_Inset) G:256, G:262, G:263, R:599
FP_InsetIsFixedSize property (FO_Inset) G:257, R:599
FP_InsetIsFlippedSideways property (FO_Inset) G:257, R:599
FP_InsetUpdater property (FO_Inset) R:600
FP_InTextFrame property (FO_AFrame) R:597
FP_InTextFrame property (FO_Cell) R:650
FP_InTextFrame property (FO_Fn) R:581
FP_InTextFrame property (FO_Pgf) G:303, R:620
FP_InTextFrame property (text location) R:670
FP_InTextObj property (FO_AFrame) R:597
FP_InTextObj property (FO_Cell) R:650
FP_InTextObj property (FO_Fn) G:328, R:581
FP_InTextObj property (FO_Pgf) G:303, R:620
FP_InTextObj property (text location) R:670
FP_InvalidHighestLevel property (FO_Element) R:644
FP_InvariantName property (FO_Marker) R:609
FP_IsIconified property (FO_Book) R:495
FP_IsIconified property (FO_Doc) R:577
FP_IsIconified property (FO_Session) R:634
FP_IsInFront property (FO_Book) G:97, R:496
FP_IsInFront property (FO_Doc) R:577
FP_IsInFront property (FO_Session) R:634
FP_IsOnScreen property (FO_Book) R:496
FP_IsOnScreen property (FO_Doc) G:98, R:548, R:577
FP_IsOnScreen property (FO_Session) R:634
FP_IsTextRange property (FO_FMTRuleClause) R:591
FP_KeepWithNext property (FO_FMTChangeList) R:588
FP_KeepWithNext property (FO_Pgf) R:619
FP_KeepWithNext property (FO_PgfFmt) G:274, R:627
FP_KeepWithPrev property (FO_FMTChangeList) R:588
FP_KeepWithPrev property (FO_Pgf) R:619
FP_KeepWithPrev property (FO_PgfFmt) R:627
FP_KernX property (FO_CharFmt) R:516
FP_KernX property (FO_Doc) R:574
FP_KernX property (FO_FMTChangeList) R:587
FP_KernX property (FO_Pgf) R:615
FP_KernX property (FO_PgfFmt) R:624
FP_KernX property (text location) R:670
FP_KernY property (FO_CharFmt) R:517
FP_KernY property (FO_Doc) R:574
FP_KernY property (FO_FMTChangeList) R:587
FP_KernY property (FO_Pgf) R:615
FP_KernY property (FO_PgfFmt) R:624
FP_KernY property (text location) R:670
FP_KeypadShortcutLabel property (FO_Command) R:525
FP_KeypadShortcuts property (FO_Command) R:525
FP_Label property (FO_Book) G:98, R:496
FP_Label property (FO_Command) G:98, R:523
FP_Label property (FO_DialogResource) G:212, R:532
FP_Label property (FO_DlgButton) G:212, R:533
FP_Label property (FO_DlgCheckBox) G:212, R:533
FP_Label property (FO_DlgLabel) G:212, R:534
FP_Label property (FO_DlgRadioButton) G:212, R:535
FP_Label property (FO_DlgTriBox) G:212, R:537
FP_Label property (FO_Menu) G:98, R:577
FP_Label property (FO_Menu) R:523
FP_Label property (FO_MenuItemSeparator) R:523
FP_Label property (FO_Session) R:634
FP_Labels property (FO_Command) G:170, R:525
FP_Labels property (FO_DlgImage) G:212, R:534
FP_Labels property (FO_DlgPopUp) G:212, R:535
FP_Labels property (FO_DlgScrollBar) G:212, R:537
FP_Language property (FO_CharFmt) R:516
FP_Language property (FO_FMTChangeList) R:586
FP_Language property (FO_Pgf) R:618
FP_Language property (FO_PgfFmt) R:626
FP_Language property (FO_Session) R:635
FP_Language property (FO_TextLine) R:607
FP_LastAFrame property (FO_SubCol) R:521
FP_LastAFrame property (FO_TextFrame) R:605
Index - Reserved Names

FP_LastCell property (FO_SubCol) R:521
FP_LastCell property (FO_TextFrame) R:605
FP_LastChildElement property (FO_Element) R:642
FP_LastFn property (FO_SubCol) R:521
FP_LastFn property (FO_TextFrame) R:605
FP_LastGraphicInFrame property (FO_AFrame) G:85, G:290, G:295, R:597
FP_LastGraphicInFrame property (FO_UnanchoredFrame) G:85, G:290, G:295, R:597
FP_LastGraphicInGroup property (FO_Group) G:296, R:599
FP_LastMasterPageInDoc property (FO_Doc) G:283, G:291, R:540
FP_LastPgff property (FO_Cell) G:76, G:337, R:650
FP_LastPgff property (FO_Fn) G:328, R:581
FP_LastPgff property (FO_SubCol) R:521
FP_LastPgff property (FO_Tbl) G:333, R:658
FP_LastPgff property (FO_TextFrame) G:304, R:605
FP_LastPgffRules property G:108
FP_LastPgffRules property (FO_ElementDef) G:343, R:647
FP_LastRefPageInDoc property (FO_Doc) G:283, G:291, R:540
FP_LastRowInTbl property (FO_Tbl) G:336, R:654
FP_LastSubCol property (FO_TextFrame) G:299, R:605
FP_LastTextFrameInFlow property (FO_Flow) G:298, R:580
FP_LastUpdate property
  (F_ApiUpdateTextInset()) G:236, R:341
FP_LastUpdate property (FO_TextInset) R:663
FP_Leading property (FOFmtChangeList) R:584
FP_Leading property (FO_Pgf) G:3, R:619
FP_Leading property (FO_PgfFmt) R:627
FP_LeadingChange property
  (FOFmtChangeList) R:584
FP_LeftColNum property (FO_Tbl) G:87, R:656
FP_LeftIndent property (FOFmtChangeList) R:584
FP_LeftIndent property (FO_Pgf) G:3, G:303, R:619
FP_LeftIndent property (FO_PgfFmt) G:274, R:626
FP_LeftIndentChange property
  (FOFmtChangeList) R:584
FP_LeftMargin property (FO_Doc) R:559
FP_LeftMasterPage property (FO_Doc) R:559
FP_LetterSpace property (FOFmtChangeList) R:583
FP_LetterSpace property (FO_Pgf) R:621
FP_LetterSpace property (FO_PgfFmt) R:629
FP_LineAscent property (text location) R:670
FP_LineBaseLine property (text location) R:670
FP_LineBreakAfter property (FO_Doc) R:576
FP_LineCap property (FO_GraphicObject) R:593
FP_LineDescent property (text location) R:670
FP_LineSpacing property (FO_Pgf) R:619
FP_LineSpacing property (FO_PgfFmt) R:627
FP_LineSpacingFixed property
  (FOFmtChangeList) R:584
FP_Locked property (FO_Pgf) R:547, R:617
FP_Locked property (FO_Tbl) R:653
FP_Locked property (FO_Var) R:672
FP_Locked property (FO_XRef) R:530
FP_Locked property (text location) R:670
FP_LocX property (F_ApiSetMetric()) G:100
FP_LocX property (FO_Doc) G:97
FP_LocX property (FO_GraphicObject) R:594
FP_LocX property (FO_Row) R:651
FP_LocX property (FO_SubCol) R:521
FP_LocX property (text location) R:670
FP_LocY property (FO_ApiSetMetric()) G:100
FP_LocY property (FO_Doc) G:97
FP_LocY property (FO_GraphicObject) R:594
FP_LocY property (FO_Row) R:651
FP_LocY property (FO_SubCol) R:521
FP_LocY property (text location) R:670
FP_Magenta property (FO_Color) R:519
FP_MainFlowInDoc property (FO_Doc) G:283, R:540
FP_MakeVisible property (F_ApiOpen()) G:98
FP_MarkerNames property (FO_Session) R:635
FP_MarkerText property (FO_Marker) G:320, R:608
FP_MarkerType property (FO_Marker) G:320, R:609
FP_MarkerTypeId property (FO_Marker) R:609
FP_MarkerTypeNames property (FO_Doc) G:320, R:540
FP_MasterPage property (FO_BodyPage) R:610
FP_MatchingContextInUserString property (FO_Element) R:641
FP_MatchingFirstPgffClauses property (FO_Element) R:639
FP_MatchingLastPgffClauses property (FO_Element) R:639
FP_MatchingObjectClauses property G:108
FP_MatchingObjectClauses property (FO_Element) R:639
FP_MatchingPrefixClauses property (FO_Element) R:640
Index - Reserved Names

FP_MatchingSuffixClauses property  
(FO_Element) R:640
FP_MatchingTextClauses property  
G:108
FP_MatchingTextClauses property  
(FO_Element) R:640
FP_MathFullForm property  
(FO_Math) R:603
FP_MathSize property  
(FO_Math) R:603
FP_MaxBottomMargin property  
(FO_Doc) R:567
FP_MaxFirstIndent property  
(FO_Doc) R:567
FP_MaxFontSize property  
(FO_Doc) R:567
FP_MaxInterlinePadding property  
(FO_Flow) R:559, R:580
FP_MaxInterPgfPadding property  
(FO_Flow) R:559, R:580
FP_MaxJLetSpace property  
(FO_FmtChangeList) R:583
FP_MaxJLetSpace property  
(FO_Pgf) R:613
FP_MaxJLetSpace property  
(FO_PgfFmt) R:622
FP_MaxJRomSpace property  
(FO_FmtChangeList) R:583
FP_MaxJRomSpace property  
(FO_Pgf) R:613
FP_MaxJRomSpace property  
(FO_PgfFmt) R:622
FP_MaxLeading property  
(FO_Doc) R:567
FP_MinBottomMargin property  
(FO_Doc) R:567
FP_MinFirstIndent property  
(FO_Doc) R:567
FP_MinFontSize property  
(FO_Doc) R:568
FP_MinHang property  
(FO_Flow) R:580
FP_MinJLetSpace property  
(FO_FmtChangeList) R:583
FP_MinJLetSpace property  
(FO_Pgf) R:613
FP_MinJLetSpace property  
(FO_PgfFmt) R:622
FP_MinJRomSpace property  
(FO_FmtChangeList) R:583
FP_MinJRomSpace property  
(FO_Pgf) R:613
FP_MinJRomSpace property  
(FO_PgfFmt) R:622
FP_MinLeading property  
(FO_Doc) R:568
FP_MinLeftIndent property  
(FO_Doc) R:568
FP_MinLeftMargin property  
(FO_Doc) R:568
FP_MinRightIndent property  
(FO_Doc) R:568
FP_MinRightMargin property  
(FO_Doc) R:568
FP_MinSpace property  
(FO_FmtChangeList) R:583
FP_MinSpace property  
(FO_Pgf) R:621
FP_MinSpace property  
(FO_PgfFmt) R:629
FP_MinSpaceAbove property  
(FO_Doc) R:568
FP_MinSpaceBelow property  
(FO_Doc) R:568
FP_MinSpread property  
(FO_Doc) R:568
FP_MinStretch property  
(FO_Doc) R:568
FP_MinTabPosition property  
(FO_Doc) R:568
FP_MinTopMargin property  
(FO_Doc) R:568
FP_Mintracking property  
(FO_Doc) R:568
FP_MinVal property  
(FO_DlgScrollBar) R:536
FP_Mode property  
(FO_Command) R:526
FP_MoveTabs property  
(FO_FmtChangeList) R:585
FP_Name property  
(FO_Book) G:353, R:496
FP_Name property  
(FO_BookComponent) R:511
FP_Name property  
(FO_CharFmt) R:517
FP_Name property  
(FO_Color) R:519
FP_Name property  
(FO_CombinedFontDefn) R:522
FP_Name property  
(FO_Command) R:523
FP_Name property  
(FO_CondFmt) R:529
FP_Name property  
(FO_Doc) G:97, R:548
FP_Name property  
(FO_ElementDef) G:344, R:647
FP_Name property  
(FO_Flow) R:580
FP_Name property  
(FO_FmtChangeList) G:347, R:582, R:584
FP_Name property  
(FO_HiddenPage) R:611
FP_Name property  
(FO_Marker) R:609
FP_Name property  
(FO_MasterPage) R:612
FP_Name property  
(FO_Menu) R:523
FP_Name property  
(FO_MenuItemSeparator) R:523
FP_Name property  
(FO_Pgf) G:217, G:303, R:621
Index - Reserved Names

FP_Name property (FO_PgfFmt) G:274, R:629
FP_Name property (FO_RefPage) R:612
FP_Name property (FO_RulingFmt) G:329, R:630
FP_Name property (FO_Tbl) G:336
FP_Name property (FO_TblFmt) G:331, R:659
FP_Name property (FO_TextInset) G:236, R:663
FP_Name property (FO_UnanchoredFrame) R:608
FP_Name property (FO_VarFmt) R:672
FP_Name property (FO_XRefFmt) G:323, R:531
FP_Namespace property (FO_Element) R:643
FP_Namespace property (FO_Element) R:643
FP_NarrowRubiSpaceForKanji property (FO_Doc) R:564
FP_NarrowRubiSpaceForOther property (FO_Doc) R:564
FP_NewElemAttrDisplay property (FO_Book) R:504
FP_NewElemAttrDisplay property (FO_Doc) R:568
FP_NewElemAttrEditing property (FO_Book) R:504
FP_NewElemAttrEditing property (FO_Doc) R:569
FP_NextAFrame property (FO_AFrame) R:597
FP_NextCell property (FO_Cell) R:650
FP_NextCellInRow property (FO_Cell) R:650
FP_NextCellInTbl property (FO_Cell) R:650
FP_NextCharFmtInDoc property (FO_CharFmt) G:310, R:517
FP_NextColorInDoc property (FO_Color) R:519
FP_NextCombinedFontDefnInDoc property (FO_CombinedFontDefn) R:522
FP_NextCommandAndSession property (FO_Command) G:162, R:526
FP_NextComponentInBook property (FO_BookComponent) G:105, G:354, R:511
FP_NextCondFmtInDoc property (FO_CondFmt) G:311, R:529
FP_NextElementDefInDoc property (FO_ElementDef) G:344, R:647
FP_NextFlowInDoc property (FO_Flow) G:298, R:580
FP_NextFmtChangeListInDoc property (FO_FmtChangeList) G:347, R:582
FP_NextFn property (FO_Fn) G:328, R:581
FP_NextFnInDoc property (FO_Fn) G:328, R:581
FP_NextGraphicInDoc property (FO_GraphicObject) G:295, R:594
FP_NextGraphicInFrame property (FO_GraphicObject) G:85, G:100, G:295, R:217, R:594
FP_NextGraphicInGroup property (FO_GraphicObject) G:296, R:594
FP_NextInvalidElement property (FO_Element) R:644
FP_NextMarkerInDoc property (FO_Marker) G:320, R:609
FP_NextMarkerTypeInDoc property (FO_Marker) R:609
FP_NextMenuItemInMenu property (FO_Command) G:163, R:35, R:523
FP_NextMenuItemInMenu property (FO_MenuItemSeparator) G:163, G:165, G:168, R:523
FP_NextMenuItemInSession property (FO_Command) R:523
FP_NextMenuItemInSession property (FO_Menu) R:523
FP_NextMenuItemInSession property (FO_MenuItemSeparator) R:523
FP_NextMenuItemInSession property (FO_Session) G:162
FP_NextOpenBookInSession property (FO_Book) G:83, G:275, G:353, R:496
FP_NextOpenDocInSession property (FO_Doc) G:83, G:275, R:540
FP_NextPgfFmtInDoc property (FO_PgfFmt) G:282, R:627
FP_NextPgfInDoc property (FO_Pgf) G:304, R:620
FP_NextPgfInFlow property (FO_Pgf) G:304, G:328, R:620
FP_NextRowInTbl property (FO_Row) G:336, R:651
FP_NextRubInDoc property (FO_Rubi) R:630
FP_NextRulingFmtInDoc property (FO_RulingFmt) G:329, R:630
FP_NextSelectedComponentInBook property (FO_BookComponent) R:511
FP_NextSelectedGraphicInDoc property (FO_GraphicObject) G:296, R:594
FP_NextSiblingElement property G:105
FP_NextSiblingElement property (FO_Element) G:121, R:642
FP_NextSubCol property (FO_SubCol) G:299, R:521
FP_NextTag property (FO_Pgf) R:621
FP_NextTag property (FO_PgfFmt) R:629
FP_NextTblFmtInDoc property (FO_TblFmt) G:331, R:659
FP_NextTblInDoc property (FO_Tbl) R:654
FP_NextTextFrameInFlow property (FO_TextFrame) G:298, R:605
Index - Reserved Names

FP_NextTiInDoc property (FO_TextInset) R:663
FP_NextVarFmtInDoc property (FO_VarFmt) G:326, R:673
FP_NextVarInDoc property (FO_Var) G:327, R:672
FP_NextXRefFmtInDoc property (FO_XRefFmt) G:323, R:531
FP_NextXRefInDoc property (FO_XRef) G:324, R:530
FP_NumAtEnd property (FO_FmtChangeList) R:584
FP_NumAtEnd property (FO_Pgf) R:613
FP_NumAtEnd property (FO_PgfFmt) R:623
FP_Numbers property (FO.Doc) R:551
FP_NumCols property (FO.Doc) R:559
FP_NumColumns property (FO_TextFrame) R:606
FP_NumItems property (FO_DialogResource) G:532
FP_NumLines property (FO_DlgScrollBox) R:537
FP_NumNamespaces property (FO_Element) R:643
FP_NumPoints property (FO_Line) R:602
FP_NumPoints property (FO_Polygon) R:604
FP_NumPoints property (FO_Polyline) R:604
FP_NumTabs property (FO_FmtChangeList) R:585
FP_NumTabs property (FO_Pgf) R:621
FP_NumTabs property (FO_PgfFmt) R:629
FP_Object property G:109
FP_Object property (FO_Element) R:642
FP_ObjectAttributes property (FO_GraphicObject) R:594
FP_ObjectFmtRules property (FO_ElementDef) G:343, R:647
FP_ObjectType property (FO_ElementDef) G:344, R:647
FP_OldTypeNum property (FO_Marker) R:609, R:609
FP_OpenDir property (FO_Session) G:94, R:635
FP_OpJLetSpace property (FO_FmtChangeList) R:583
FP_OpJLetSpace property (FO_Pgf) R:613
FP_OpJLetSpace property (FO_PgfFmt) R:622
FP_OpJRomSpace property (FO_FmtChangeList) R:583
FP_OpJRomSpace property (FO_Pgf) R:613
FP_OpJRomSpace property (FO_PgfFmt) R:622
FP_OptSpace property (FO_FmtChangeList) R:583
FP_OptSpace property (FO_Pgf) R:621
FP_OptSpace property (FO_PgfFmt) R:629
FP_OrphanRows property (FO_Tbl) R:653
FP_OrphanRows property (FO_TblFmt) R:658
FP_Outline property (FO_CharFmt) R:517
FP_Outline property (FO_Doc) R:575
FP_Outline property (FO_FmtChangeList) R:587
FP_Outline property (FO_Pgf) R:615
FP_Outline property (FO_PgfFmt) R:624
FP_Outline property (text location) R:671
FP_Overflowed property (FO_Cell) R:650
FP_Overflowed property (FO_Fn) R:581
FP_Overflowed property (FO_SubCol) R:521
FP_Overflowed property (FO_Tbl) R:653
FP_Overline property (FO_CharFmt) R:517
FP_Overline property (FO_Doc) R:575
FP_Overline property (FO_FmtChangeList) R:587
FP_Overline property (FO_FmtChangeList) R:587
FP_Overline property (FO_Pgf) R:615
FP_Overline property (FO_PgfFmt) R:624
FP_Overline property (text location) R:671
FP_Overprint property (FO_GraphicObject) R:594
FP_OyamojiTextRange property (FO_Rubi) R:630
FP_PageBackground property (FO_BodyPage) R:611
FP_PageFrame property (FO_BodyPage) G:290, R:611
FP_PageFrame property (FO_HiddenPage) G:290, R:611
FP_PageFrame property (FO_MasterPage) G:290, R:612
FP_PageFrame property (FO_RefPage) G:290, R:612
FP_PageFrame property (FO_UncachedFrame) R:608
FP_PageHeight property (FO_BodyPage) R:611
FP_PageHeight property (FO_Doc) R:559
FP_PageHeight property (FO_HiddenPage) R:611
FP_PageHeight property (FO_MasterPage) R:612
FP_PageHeight property (FO_RefPage) R:612
FP_PageIsRecto property (FO_BodyPage) R:611
FP_PageNext property (FO_BodyPage) G:291, R:611
FP_PageNext property (FO_MasterPage) G:291, R:612
FP_PageNext property (FO_RefPage) G:291, R:612
FP_PageNum property (FO_BodyPage) R:611
FP_PageNum property (FO_Inset) R:600
FP_PageNum property (FO_MasterPage) R:612
FP_PageNum property (FO_RefPage) R:612
FP_PageNumComputeMethod property (FO_BookComponent) R:511
FP_PageNumComputeMethod property (FO_Doc) R:548, R:560
FP_PageNumString property (FO_BodyPage) R:611
Index - Reserved Names

FP_PageNumStyle property (FO_BookComponent) R:512
FP_PageNumStyle property (FO_Doc) R:560
FP_PagePrefix property (FO_BookComponent) R:512
FP_PagePrev property (FO_BodyPage) G:291
FP_PagePrev property (FO_HiddenPage) R:611
FP_PagePrev property (FO_MasterPage) G:291, R:612
FP_PagePrev property (FO_RefPage) G:291, R:612
FP_PageRounding property (FO_Doc) R:560
FP_PageSide property (FO_BookComponent) R:512
FP_PageSuffix property (FO_BookComponent) R:512
FP_PageWidth property (FO_BodyPage) R:611
FP_PageWidth property (FO_Doc) R:560
FP_PageWidth property (FO_HiddenPage) R:612
FP_PageWidth property (FO_MasterPage) G:291, R:612
FP_PageWidth property (FO_RefPage) G:291, R:612
FP_PairKern property (FO_CharFmt) R:517
FP_PairKern property (FO_Doc) R:575
FP_PairKern property (FO_FmtChangeList) R:587
FP_PairKern property (FO_Pgf) R:615
FP_PairKern property (FO_PgfFmt) R:624
FP_PairKern property (text location) R:671
FP_ParentElement property (FO_Element) R:642
FP_ParentTextFrame property (FO_SubCol) G:299, R:521
FP_Path property (FO_Session) R:635
FP_PDFBookmark property (FO_Book) R:499
FP_PDFBookmark property (FO_Doc) R:543
FP_PDFBookmarksOpenLevel property (FO_Book) R:500
FP_PDFBookmarksOpenLevel property (FO_Doc) R:543
FP_PDFConvertCMYKtoRGB property (FO_Book) R:500
FP_PDFConvertCMYKtoRGB property (FO_Doc) R:544
FP_PDFDistillerAbsent property (FO_Book) R:500
FP_PDFDistillerAbsent property (FO_Doc) R:544
FP_PDFEndPage property (FO_Book) R:500
FP_PDFEndPage property (FO_Doc) R:544
FP_PDFJobOption property (FO_Book) R:501
FP_PDFJobOption property (FO_Doc) R:544
FP_PDFJobOptionsAbsent property (FO_Book) R:501
FP_PDFJobOptionsAbsent property (FO_Doc) R:544
FP_PDFPdfLevel property (FO_PgfFmt) R:623
FP_PDFPageHeight property (FO_Book) R:544
FP_PDFPageHeight property (FO_Doc) R:544
FP_PDFPageWidth property (FO_Book) R:501
FP_PDFPageWidth property (FO_Doc) R:544
FP_PDFPrintPageRange property (FO_Book) R:501
FP_PDFPrintPageRange property (FO_Doc) R:545
FP_PDFRegistrationMarks property (FO_Book) R:501
FP_PDFRegistrationMarks property (FO_Doc) R:545
FP_PDFSeparateFiles property (FO_Book) R:501
FP_PDFSeparateFiles property (FO_Doc) R:545
FP_PDFStartPage property (FO_Book) R:501
FP_PDFStartPage property (FO_Doc) R:545
FP_PDFStructureLevel property (FO_PgfFmt) R:622
FP_PDFZoomFactor property (FO_Book) R:501
FP_PDFZoomFactor property (FO_Doc) R:545
FP_PDFZoomType property (FO_Book) R:501
FP_PDFZoomType property (FO_Doc) R:545
FP_Pen property (FO_GraphicObject) R:594
FP_Pen property (FO_RulingFmt) R:631
FP_PgfAlignment property (FO_FmtChangeList) R:585
FP_PgfAlignment property (FO_Pgf) R:619
FP_PgfAlignment property (FO_PgfFmt) R:627
FP_PgfCatalogReference property (FO_FmtChangeList) G:347, R:582
FP_PgfIsAutoNum property (FO_Pgf) R:618
FP_PgfIsAutoNum property (FO_PgfFmt) R:613
FP_PgfIsAutoNum property (FO_PgfFormat) R:623
FP_PgfNumber property (FO_Pgf) G:303, R:614
FP_PgfNumbering property (FO_BookComponent) R:512
FP_PgfNumComputeMethod property (FO_BookComponent) R:512
FP_PgfNumComputeMethod property (FO_Doc) R:548, R:558
FP_PgfSpellChecked property (FO_Pgf) R:618
FP_Placement property (FO_FmtChangeList) R:588
FP_Placement property (FO_Pgf) R:620
FP_Placement property (FO_PgfFmt) R:627
FP_Placement property (FO_Session) R:635
FP_PointPageNum property (FO_BodyPage) R:611
FP_PointPageNum property (FO_Doc) R:546
FP_PointPageNum property (FO_Doc) R:558
FP_Points property (FO_Axis) G:100
FP_Points property (FO_Line) R:602
FP_Points property (FO_Polygon) R:604
FP_Points property (FO_Polyline) R:604
FP_PolyIsBezier property (FO_Polygon) R:604
FP_PolyIsBezier property (FO_Polyline) R:604

FDK Programmer’s Reference & Guide
Index - Reserved Names

FP_Position property (FO_CharFmt) R:517
FP_Position property (FO_Doc) R:575
FP_Position property (FO_FmtChangeList) R:587
FP_Position property (FO_Pgf) R:615
FP_Position property (FO_PgfFmt) R:624
FP_Position property (text location) R:671
FP_PrefixRules property G:108
FP_PrefixRules property (FO_ElementDef) G:343, R:647
FP_PrevAFrame property (FO_AFrame) R:597
FP_PrevCell property (FO_Cell) R:651
FP_PrevCellInRow property (FO_Cell) R:651
FP_PrevComponentInBook property (FO_BookComponent) G:105, G:354, R:513
FP_PrevFn property (FO_Fn) G:328, R:581
FP_PrevGraphicInGroup property (FO_GraphicObject) G:296, R:595
FP_PrevMenuInMenu property (FO_Command) G:163, R:523
FP_PrevPgFrmFlow property (FO_Pgf) G:304, G:328, R:620
FP_PrevRowInTbl property (FO_Row) G:336, R:651
FP_PrevSiblingElement property (FO_Element) R:642
FP_PrevSubCol property (FO_SubCol) G:299, R:521
FP_PrevTextFrameInFlow property (F_ApiNewGraphicObject()) G:150
FP_PrevTextFrameInFlow property (FO_TextFrame) G:298, R:606
FP_PrintBlankPages property (FO_Book) R:502
FP_PrintBlankPages property (FO_Doc) R:561
FP_PrintCollated property (FO_Book) R:502
FP_PrintCollated property (FO_Doc) R:562
FP_PrintCols property (FO_Doc) R:562
FP_PrintDownloadAsianFonts property (FO_Book) R:502
FP_PrintDownloadAsianFonts property (FO_Doc) R:562
FP_PrintEmulsion property (FO_Book) R:502
FP_PrintEmulsion property (FO_Doc) R:562
FP_PrintEndPoint property (FO_Doc) R:562
FP_PrintFileName property (FO_Book) R:503
FP_PrintFileName property (FO_Doc) R:562
FP_PrintImaging property (FO_Book) R:503
FP_PrintImaging property (FO_Doc) R:562
FP_PrintInclude property (FO_BookComponent) R:513
FP_PrintLastSheetFirst property (FO_Book) R:503
FP_PrintLastSheetFirst property (FO_Doc) R:562
FP_PrintLowRes property (FO_Book) R:503
FP_PrintLowRes property (FO_Doc) R:562
FP_PrintManualFeed property (FO_Doc) R:562
FP_PrintNumCopies property (FO_Book) R:503
FP_PrintNumCopies property (FO_Doc) R:562
FP_PrintOddPages property (FO_Book) R:503
FP_PrintOddPages property (FO_Doc) R:562
FP_PrintPaperHeight property (FO_Book) R:503
FP_PrintPaperHeight property (FO_Doc) R:562
FP_PrintPaperWidth property (FO_Book) R:503
FP_PrintPaperWidth property (FO_Doc) R:563
FP_PrintRegistrationMarks property (FO_Book) R:503
FP_PrintRegistrationMarks property (FO_Doc) R:563
FP_PrintRrows property (FO_Doc) R:563
FP_PrintScale property (FO_Book) R:503
FP_PrintScale property (FO_Doc) R:563
FP_PrintScope property (FO_Doc) R:563
FP_PrintSeps property (FO_Book) R:503
FP_PrintSeps property (FO_Doc) R:563
FP_PrintStartPage property (FO_Doc) R:319, R:563
FP_PrintStartPageName property (FO_Doc) R:563
FP_PrintStartPoint property (FO_Doc) R:563
FP_PrintThumbnails property (FO_Doc) R:563
FP_PrintToFile property (FO_Book) R:503
FP_PrintToFile property (FO_Doc) R:563
FP_ProcessNumber property (FO_Session) G:97, R:264, R:636
FP_RememberMissingFontNames property
  (FO_Session) R:636
FP_Required property (FO_Marker) R:610
FPReservedColor property (FO_Color) R:520
FP_RightColNum property (FO_Tbl) G:87, R:656
FP_RightIndent property (FO_FmtChangeList) R:585
FP_RightIndent property (FO_Pgf) R:619
FP_RightIndent property (FO_PgfFmt) R:626
FP_RightIndent property (FO_Pgf) R:619
FP_RightIndent property (FO_PgfFmt) R:626
FP_RightIndentChange property
  (FO_FmtChangeList) R:585
FP_RightMargin property (FO_Doc) R:561
FP_RightMasterPage property (FO_Doc) R:540
FP_RowIsShown property (FO_Row) R:651
FP_RowKeepWithNext property (FO_Row) G:336, R:652
FP_RowKeepWithPrev property (FO_Row) R:652
FP_RowTbl property (FO_Row) R:652
FP_RowType property (FO_Row) G:336, R:652
FP_RuleClauseType property
  (FO_FmtRuleClause) G:346, R:591
FP RulingGap property (FO_RulingFmt) G:329, R:631
FP RulingLines property (FO_RulingFmt) G:329, R:631
FP RulingPenWidth property (FO_RulingFmt) G:329, R:631
FP RulingSep property (FO_RulingFmt) R:631
FP Runaround property
  (FO_GraphicObject) R:595
FP RunaroundGap property
  (FO_GraphicObject) R:595
FP RunInSeparator property
  (FO_FmtChangeList) R:588
FP RunInSeparator property (FO_Pgf) R:620
FP RunInSeparator property (FO_PgfFmt) R:627
FP ScreenHeight property (FO_Book) R:496
FP ScreenHeight property (FO_Doc) R:577
FP ScreenHeight property (FO_Session) R:636
FP ScreenWidth property (FO_Book) R:496
FP ScreenWidth property
  (FO_DialogResource) R:532
FP ScreenWidth property (FO_Doc) R:577
FP ScreenWidth property (FO_Session) R:636
FP ScreenX property (FO_Book) R:496
FP ScreenX property (FO_DialogResource) R:532
FP ScreenX property (FO_Doc) R:577
FP ScreenX property (FO_Session) R:636, R:637
FP SelectedTbl property (FO_Doc) G:283, G:285, R:540, R:656
FP Sensitivity property (FO_DlgButton) G:212, G:219, R:533
FP Sensitivity property (FO_DlgCheckBox) G:212, G:219, R:533
FP Sensitivity property (FO_DlgEditBox) G:212, G:219, R:534
FP Sensitivity property (FO_DlgImage) G:212, G:219, R:534
FP Sensitivity property (FO_DlgLabel) G:212, G:219, R:534
FP Sensitivity property (FO_DlgPopUp) G:212, G:219, R:535
FP Sensitivity property (FO_DlgScrollBar) G:212, G:219, R:536
FP Sensitivity property (FO_DlgScrollBox) G:212, G:219, R:537
FP Sensitivity property (FO_DlgTriBox) G:212, G:219, R:537
FP SeparateInclusions property (FO_Book) R:504
FP SeparateInclusions property (FO_Doc) R:569
FP SepOverride property (FO_CondFmt) R:529
FP SepOverride property (FO_Doc) R:575
FP SepOverride property (FO_Row) R:652
FP SepOverride property (text location) R:671
FP SgmApplication property (FO_Doc) R:505
FP SgmApplication property (FO_Doc) R:569
FP Shadow property (FO_CharFmt) R:517
FP Shadow property (FO_Doc) R:575
FP Shadow property (FO_FmtChangeList) R:585
FP Shadow property (FO_Pgf) R:615
FP Shadow property (FO_PgfFmt) R:625
FP Shadow property (text location) R:671
FP ShiftOrUnshiftCommand property
  (FO_Command) G:174, R:526
Index - Reserved Names

FP_ShowAll property (FO_Doc)  R:550
FP_ShowCondIndicators property (FO_Doc)  R:550
FP_SideHeadGap property (FO_TextFrame)  R:606
FP_SideHeadPlacement property (FO_TextFrame)  R:606
FP_SideHeadRoomInFlow property (FO_Flow)  R:580
FP_SideOffset property (FO_AFrame)  R:597
FP_Size property (FO_DlgScrollBar)  R:536
FP_SkipBlankSeps property (FO_Book)  R:503
FP_SkipBlankSeps property (FO_Doc)  R:563
FP_SmallCapsSize property (FO_Doc)  R:576
FP_SmallCapStretch property (FO_Doc)  R:576
FP_SmartQuotes property (FO_Doc)  R:561
FP_SmartSpaces property (FO_Doc)  R:561
FP_Snap property (FO_Session)  R:637
FP_SnapAngle property (FO_Doc)  R:577
FP_SnapGridUnits property (FO_Doc)  R:577
FP_SpaceAbove property (FO_FmtChangeList)  R:585
FP_SpaceAbove property (FO_Pgf)  R:620
FP_SpaceAbove property (FO_PgfFmt)  R:628
FP_SpaceAboveChange property (FO_FmtChangeList)  R:585
FP_SpaceBelow property (FO_FmtChangeList)  R:585
FP_SpaceBelow property (FO_Pgf)  R:620
FP_SpaceBelow property (FO_PgfFmt)  R:628
FP_SpaceBelowChange property (FO_FmtChangeList)  R:585
FP_Spacing property (FO_Flow)  R:580
FP_Specification property (FO_FmtRuleClause)  R:591
FP_SpecificationForCSS property (FO_Element)  R:592
FP_SpotColorView property (FO_Doc)  R:577
FP_Spread property (FO_CharFmt)  R:517
FP_Spread property (FO_Doc)  R:575
FP_Spread property (FO_FmtChangeList)  R:587
FP_Spread property (FO_Pgf)  R:615
FP_Spread property (FO_PgfFmt)  R:625
FP_Spread property (text location)  R:671
FP_SpreadChange property (FO_FmtChangeList)  R:587
FP_Start property (FO_FmtChangeList)  R:588
FP_Start property (FO_Pgf)  R:620
FP_Start property (FO_PgfFmt)  R:628
FP_State property (FO_DlgButton)  G:212,  R:533
FP_State property (FO_DlgCheckBox)  G:212,  R:533
FP_State property (FO_DlgImage)  G:212,  R:534
FP_State property (FO_DlgPopUp)  G:212,  R:535
FP_State property (FO_DlgRadioButton)  G:212,  R:535
FP_State property (FO_DlgScrollBar)  G:212,  R:536
FP_State property (FO_DlgScrollBox)  G:212,  R:537
FP_State property (FO_DlgTriBox)  G:212,  R:537
FP_StatusLine property (FO_Book)  G:353,  R:496
FP_StatusLine property (FO_Doc)  G:98,  R:548
FP_StopCountingAt property (FO_FmtRule)  R:590
FP_Stretch property (FO_CharFmt)  R:517
FP_Stretch property (FO_Doc)  R:575
FP_Stretch property (FO_FmtChangeList)  R:587
FP_Stretch property (FO_Pgf)  R:615
FP_Stretch property (FO_PgfFmt)  R:625
FP_Stretch property (text location)  R:671
FP_StretchChange property (FO_FmtChangeList)  R:587
FP_Strikethrough property (FO_CharFmt)  R:517
FP_Strikethrough property (FO_Doc)  R:575
FP_Strikethrough property (FO_FmtChangeList)  R:587
FP_Strikethrough property (FO_Pgf)  R:615
FP_Strikethrough property (FO_PgfFmt)  R:625
FP_Strikethrough property (text location)  R:671
FP_FmtRule property  G:107
FP_FmtRule property (FO_FmtRuleClause)  R:592
FP_FmtRule property (FO_FmtRuleClause)  R:592
FP_FmtRule property (FO_FmtRuleClause)  R:592
FP_ScriptShift property (FO_Doc)  R:576
FP_ScriptShift property (FO_Doc)  R:576
FP_ScriptSize property (FO_Doc)  R:576
FP_ScriptStretch property (FO_Doc)  R:576
FP_SuffixRules property  G:108
FP_SuffixRules property (FO_Doc)  G:343,  R:647
FP_ScriptShift property (FO_Doc)  R:576
FP_ScriptSize property (FO_Doc)  R:576
FP_ScriptRound property (FO_Doc)  R:576
FP_Symbols property (FO_Doc)  G:212,  R:533
FP_Symbols property (FO_Doc)  R:551
FP_Symbols property (FO_Doc)  R:551
FP_SystemVar property (FO_VarFmt)  G:325,  R:673
FP_TableTagging property (FO_ElementDef)  R:647
FP_Tabs property (FO_FmtChangeList)  R:585
FP_Tabs property (FO_Pgf)  R:621
FP_Tabs property (FO_PgfFmt) R:629
FP_TailArrow property
  (FO_GraphicObject) R:595
FP_TblAlignment property (FO_Tbl) R:653
FP_TblAlignment property (FO_TblFmt) R:658
FP_TblBodyFirstColor property (FO_Tbl) R:656
FP_TblBodyFirstColor property (FO_TblFmt) R:661
FP_TblBodyFirstFill property (FO_Tbl) G:336, R:656
FP_TblBodyFirstFill property (FO_TblFmt) R:661
FP_TblBodyFirstPeriod property (FO_Tbl) R:656
FP_TblBodyFirstPeriod property (FO_TblFmt) R:661
FP_TblBodyNextColor property (FO_Tbl) R:656
FP_TblBodyNextColor property (FO_TblFmt) R:661
FP_TblBodyNextFill property (FO_Tbl) G:336, R:656
FP_TblBodyNextFill property (FO_TblFmt) R:661
FP_TblBodyNextPeriod property (FO_Tbl) R:657
FP_TblBodyNextPeriod property (FO_TblFmt) R:661
FP_TblBodyRowRuling property (FO_Tbl) G:336, R:655
FP_TblBodyRowRuling property (FO_TblFmt) G:331, R:660
FP_TblCatalogEntry property (FO_Tbl) R:654
FP_TblCatalogEntry property (FO_TblFmt) R:659
FP_TblColRuling property (FO_Tbl) R:655
FP_TblColRuling property (FO_TblFmt) G:331, R:660
FP_TblColRuling property (FO_Tbl) R:655
FP_TblColRuling property (FO_TblFmt) G:331, R:660
FP_TblColWidths property (FO_Tbl) R:654
FP_TblFnCellPosition property (FO_Doc) R:572
FP_TblFnCellPrefix property (FO_Doc) R:572
FP_TblFnCellSuffix property (FO_Doc) R:572
FP_TblFnCustNumString property (FO_Doc) R:572
FP_TblFnFirstNum property
  (FO_BookComponent) R:513
FP_TblFnFmt property (FO_Doc) R:572
FP_TblFnNumComputeMethod property
  (FO_BookComponent) R:513
FP_TblFnNumStyle property
  (FO_BookComponent) R:513
FP_TblFnNumStyle property (FO_Doc) R:573
FP_TblFnPosition property (FO_Doc) R:573
FP_TblFnPrefix property (FO_Doc) R:573
FP_TblFnSuffix property (FO_Doc) R:573
FP_TblHFColor property (FO_Tbl) R:657
FP_TblHFColor property (FO_TblFmt) R:662
FP_TblHFill property (FO_Tbl) R:657
FP_TblHFill property (FO_TblFmt) R:662
FP_TblHFRowRuling property (FO_Tbl) R:655
FP_TblHFRowRuling property (FO_TblFmt) R:660
FP_TblHSeparatorRuling property (FO_Tbl) G:336, R:655
FP_TblHSeparatorRuling property (FO_TblFmt) G:331, R:660
FP_TblInitNumBodyRows property
  (FO_TblFmt) R:660
FP_TblInitNumCols property (FO_TblFmt) G:331, R:660
FP_TblInitNumFRows property (FO_TblFmt) R:660
FP_TblInitNumHRows property (FO_TblFmt) G:331, R:660
FP_TblInLockedTi property (FO_Tbl) R:653
FP_TblLastBodyRuling property (FO_Tbl) R:655
FP_TblLastBodyRuling property (FO_TblFmt) R:661
FP_TblLeftIndent property (FO_Tbl) R:653, R:659
FP_TblLeftIndent property (FO_TblFmt) R:659
FP_TblLeftRuling property (FO_Tbl) R:655
FP_TblLeftRuling property (FO_TblFmt) R:659
FP_TblNumbering property (FO_Tbl) R:654
FP_TblNumbering property (FO_TblFmt) R:659
FP_TblNumCols property (FO_Tbl) G:156, R:654
FP_TblNumCols property (FO_TblFmt) R:659
FP_TblNumRows property (FO_Tbl) G:156, R:654
FP_TblOtherBodyRowRuling property
  (FO_Tbl) R:655
FP_TblOtherBodyRowRuling property
  (FO_TblFmt) R:661
FP_TblOtherColRuling property (FO_Tbl) R:655
FP_TblOtherColRuling property (FO_TblFmt) R:661
FP_TblPlacement property (FO_Tbl) R:653
FP_TblPlacement property (FO_TblFmt) R:659
FP_TblRightIndent property (FO_Tbl) R:654
FP_TblRightIndent property (FO_TblFmt) R:659
FP_TblRightRuling property (FO_Tbl) R:655
FP_TblRightRuling property (FO_TblFmt) R:661
FP_TblSpaceAbove property (FO_Tbl) R:654
FP_TblSpaceAbove property (FO_TblFmt) R:659
FP_TblSpaceBelow property (FO_Tbl) R:654
FP_TblSpaceBelow property (FO_TblFmt) R:659
FP_TblTag property (FO_Tbl) R:654
Index - Reserved Names

FP_TblTag property (FO_TblFmt)  R:659
FP_TblTitleGap property (FO_Tbl)  R:658
FP_TblTitleGap property (FO_TblFmt)  R:659
FP_TblTitlePosition property (FO_Tbl)  R:658
FP_TblTitlePosition property (FO_TblFmt)  R:659
FP_TblTitleSelected property (FO_Tbl)  R:656, R:658
FP_TblTopRuling property (FO_Tbl)  G:336, R:656
FP_TblTopRuling property (FO_TblFmt)  R:661
FP_TblWidth property (FO_Tbl)  R:654
FP_Text property (FO_DlgEditBox)  R:534
FP_TextFmtRules property (FO_ElementDef)  G:343, G:344, R:647
FP_TextIsInvalidInElement property (FO_Element)  R:644
FP_TextLineType property (FO_Math)  R:603
FP_TextLineType property (FO_TextLine)  R:607
FP_TextLoc property (FO_AFrame)  R:597
FP_TextLoc property (FO_Fn)  R:581
FP_TextLoc property (FO_Marker)  R:609
FP_TextRange property (FO_Element)  R:640
FP_TextRange property (FO_TextInset)  R:663
FP_TextRange property (FO_Var)  R:672
FP_TextRange property (FO_XRef)  R:530
FP_Theta property (FO_Arc)  R:598
FP_TiAutomaticUpdate property (FO_TextInset)  R:663
FP_TiByRows property (FO_TiTextTable)  R:668
FP_TiClientData property (FO_TiApiClient)  R:666
FP_TiClientName property (FO_Ti_ApiClient)  G:236, R:666
FP_TiClientSource property (FO_TiApiClient)  G:236, R:666
FP_TiClientType property (FO_TiApiClient)  G:236, R:666
FP_TiEOELsEOP property (FO_TiText)  R:668
FP_TiFile property (FO_TextInset)  R:663
FP_TiFlowName property (FO_TiFlow)  R:667
FP_TiFlowPageSpace property (FO_TiFlow)  R:667
FP_TiFormat property (FO_TiFlow)  R:667
FP_TiHeadersEmpty property (FO_TiTextTable)  R:668
FP_TisUnresolved property (FO_TiApiClient)  R:666
FP_TiLocked property (FO_TextInset)  G:235, R:663
FP_TiMacEdition property (FO_TextInset)  R:663
FP_TiMainFlow property (FO_TiFlow)  R:667
FP_Tint property (FO_GraphicObject)  R:595
FP_TintBaseColor property (FO_Color)  R:520
FP_TiNumCols property (FO_TiTextTable)  R:668
FP_TiNumHeaderRows property (FO_TiTextTable)  R:668
FP_TiNumSeparators property (FO_TiTextTable)  R:668
FP_TiRemoveOverrides property (FO_TiFlow)  R:667
FP_TiRemovePageBreaks property (FO_TiFlow)  R:667
FP_TiSeparator property (FO_TiTextTable)  R:668
FP_TiTextEncoding property (FO_TiText)  R:668
FP_TiTextEncoding property (FO_TiTextTable)  R:668
FP_TmpDir property (FO_Session)  R:637
FP_TopMargin property (FO_Doc)  R:561
FP_TopRowSelection property (FO_Tbl)  G:87, R:656
FP_TopSeparator property (FO_FMTChangeList)  R:583
FP_TopSeparator property (FO_Pgf)  R:620
FP_TopSeparator property (FO_PgfFmt)  R:628
FP_TopSepAtIndent property (FO_FMTChangeList)  R:583
FP_Tracking property (FO_CharFmt)  R:517
FP_Tracking property (FO_Doc)  R:575
FP_Tracking property (FO_FMTChangeList)  R:587
FP_Tracking property (FO_Pgf)  R:615
FP_Tracking property (FO_PgfFmt)  R:625
FP_Tracking property (text location)  R:671
FP_TrackingChange property (FO_FMTChangeList)  R:587
FP_Transient property (FO_Marker)  R:610
FP_TrapswiseCompatibility property (FO_Doc)  R:563
FP_TypeOfDisplayText property (FO_Book)  G:353, R:497
FP_Underlining property (FO_CharFmt)  R:517
FP_Underlining property (FO_Doc)  R:575
FP_Underlining property (FO_FMTChangeList)  R:588
FP_Underlining property (FO_Pgf)  R:616
FP_Underlining property (FO_PgfFmt)  R:625
FP_Underlining property (text location)  R:671
FP_Unique property (FO_BookComponent)  R:513
FP_Unique property (FO_Cell)  R:651
FP_Unique property (FO_Element)  R:640
FP_Unique property (FO_Fn)  R:581
FP_Unique property (FO_GraphicObject)  R:595
FP_Unique property (FO_Marker)  R:609
FP_Unique property (FO_Pgf) R:618
FP_Unique property (FO_Rubi) R:630
FP_Unique property (FO_SubCol) R:521
FP_Unique property (FO_Tbl) R:654
FP_Unique property (FO_TextInset) R:663
FP_Unique property (FO_Var) R:672
FP_Untouchable property (FO_Doc) R:548
FP_UseCapitalization property (FO_CharFmt) R:517
FP_UseChangeBar property (FO_CharFmt) R:517
FP_UseCharacteristic properties (FO_CharFmt) G:308
FP_UseColor property (FO_CharFmt) R:517
FP_UseFontAngle property (FO_CharFmt) R:518
FP_UseFontFamily property (FO_CharFmt) R:518
FP_UseFontSize property (FO_CharFmt) R:518
FP_UseFontVariation property (FO_CharFmt) R:518
FP_UseKernX property (FO_CharFmt) R:518
FP_UseKernY property (FO_CharFmt) R:518
FP_UseNextTag property (FO_Pgf) R:621
FP_UseNextTag property (FO_PgfFmt) R:629
FP_UseOutline property (FO_CharFmt) R:518
FP_UseOverline property (FO_CharFmt) R:518
FP_UsePairKern property (FO_CharFmt) R:518
FP_UsePosition property (FO_CharFmt) R:518
FP_UserHomeDir property (FO_Session) R:637
FP_UserLogin property (FO_Session) R:637
FP_UserName property (FO_Session) R:637
FP_UserString property (FO_Session) G:110
FP_UserString property (FO_CombinedFontDefn) R:522
FP_UserString property (FO_Element) R:640
FP_UseSepOverride property (FO_CondFmt) R:529
FP_UseSepOverride property (FO_Doc) R:575
FP_UseSepOverride property (FO_Row) R:652
FP_UseSepOverride property (text location) R:671
FP_UseShadow property (FO_CharFmt) R:518
FP_UseSpread property (FO_CharFmt) R:518
FP_UseStretch property (FO_CharFmt) R:518
FP_UseStrikethrough property (FO_CharFmt) R:518
FP_UseTracking property (FO_CharFmt) R:518
FP_UseUnderlining property (FO_CharFmt) R:518
FP_Validating property G:110
FP_Validating property (FO_Session) G:110, R:637
FP_ValidationFlags property (FO_Element) R:645
FP_ValidateHighestLevel property (FO_ElementDef) R:647
FP_VarFmt property (FO_Var) G:327, R:672
FP_Variables property (FO_Doc) R:551
FP_VersionMajor property (FO_Session) R:637
FP_VersionMinor property (FO_Session) R:637
FP_VerticalTrackingLarge property (FO_Doc) R:551
FP_VerticalTrackingMedium property (FO_Doc) R:551
FP_VerticalTrackingSmall property (FO_Doc) R:551
FP_ViewBorders property (FO_Doc) R:577
FP_ViewDisplayUnits property (FO_Doc) R:577
FP_ViewFontSizeUnits property (FO_Doc) R:578
FP_ViewFormattingBar property (FO_Session) R:637
FP_ViewGrid property (FO_Doc) R:578
FP_ViewGridUnits property (FO_Doc) R:578
FP_ViewNoGraphics property (FO_Doc) R:578
FP_ViewOnlyDeadCodes property (FO_Book) R:507
FP_ViewOnlyDeadCodes property (FO_Doc) R:578
FP_ViewOnlyMenuBar property (FO_Doc) R:557, R:578
FP_ViewOnlySelect property (FO_Doc) R:579
FP_ViewOnlyWinBorders property (FO_Book) R:507
FP_ViewOnlyWinBorders property (FO_Doc) R:579
FP_ViewOnlyWinMenuBar property (FO_Book) R:507
FP_ViewOnlyWinMenuBar property (FO_Doc) R:579
FP_ViewOnlyWinPalette property (FO_Doc) R:548, R:579
FP_ViewOnlyWinPopup property (FO_Book) R:507
FP_ViewOnlyWinPopup property (FO_Doc) R:579
FP_ViewPageScrolling property (FO_Doc) R:578
FP_ViewQuickAccessBar property (FO_Session) R:637
FP_ViewRulers property (FO_Session) R:637
FP_ViewRulerUnits property (FO_Doc) R:578
FP_ViewTextSymbols property (FO_Doc) R:578
FP_Visibility property (FO_DialogBox) R:533
FP_Visibility property (FO_DlgButton) R:533
FP_Visibility property (FO_DlgCheckBox) R:533
FP_Visibility property (FO_DlgEditBox) R:534
FP_Visibility property (FO_DlgImage) R:534
FP_Visibility property (FO_DlgLabel) R:534
FP_Visibility property (FO_DlgPopup) R:535
FP_Visibility property (FO_DlgRadioButton) R:535
FP_Visibility property (FO_DlgTriBox) R:537
FP_Visibility property (FO_ScrollBar) R:536
FP_Visibility property (FO_ScrollBox) R:537
FP_VolrNumComputeMethod property (FO_BookComponent) R:513
FP_VolrNumComputeMethod property (FO_Doc) R:548
FP_VolumeNumber property (FO_BookComponent) R:513
FP_VolumeNumber property (FO_Doc) R:548
FP_VolumeNumStyle property (FO_BookComponent) R:514
FP_VolumeNumStyle property (FO_Doc) R:549
FP_VolumeNumText property (FO_BookComponent) R:514
FP_VolumeNumText property (FO_Doc) R:549
FP_WesternFamily property (FO_CombinedFontDefn) R:522
FP_WesternFontPlatformName property (FO_CharFmt) R:516
FP_WesternFontPostScriptName property (FO_CharFmt) R:516
FP_WesternShift property (FO_CombinedFontDefn) R:522
FP_WesternSize property (FO_CombinedFontDefn) R:522
FP_WideRubiSpaceForKanji property (FO_Doc) R:564
FP_WideRubiSpaceForOther property (FO_Doc) R:564
FP_Width property (F_ApiSetMetric()) G:99
FP_Width property (FO_Doc) G:97
FP_Width property (FO_GraphicObject) G:3, G:263, R:595
FP_Width property (FO_Rectangle) G:3
FP_Width property (FO_Row) R:652
FP_Width property (FO_SubCol) R:521
FP_Width property (text location) R:671
FP_WindowSystem property (FO_Session) R:637
FP_XmlEncoding property (FO_Doc) R:569
FP_XmlEncoding property (FO_XRef) R:505, R:569
FP_XmlEncoding property (FO_XRef) R:505, R:570
FP_XmlStandalone property (FO_Book) R:506
FP_XmlStandalone property (FO_Doc) R:570
FP_XmlStyleSheet property (FO_Book) R:506
FP_XmlStyleSheet property (FO_Doc) R:571
FP_XmlStyleSheetList property (FO_Book) R:506, R:571
FP_XmlStyleSheetList property (FO_Doc) R:506, R:571
Index - Reserved Names

FS_DisallowBookDoc property
  (F_ApiGetOpenDefaultParams()) R:139
FS_DisallowBookMIF property
  (F_ApiGetOpenDefaultParams()) R:139
FS_DisallowDoc property
  (F_ApiGetImportDefaultParams()) R:121
FS_DisallowDoc property
  (F_ApiGetOpenDefaultParams()) R:139
FS_DisallowFilterTypes property
  (F_ApiGetImportDefaultParams()) R:121
FS_DisallowFilterTypes property
  (F_ApiGetOpenDefaultParams()) R:121
FS_DisallowGraphicTypes property
  (F_ApiGetImportDefaultParams()) R:121
FS_DisallowMacEditions property
  (F_ApiGetImportDefaultParams()) R:121
FS_DisallowMIF property
  (F_ApiGetImportDefaultParams()) R:121
FS_DisallowMIF property
  (F_ApiGetOpenDefaultParams()) R:140
FS_DisallowPlainText property
  (F_ApiGetImportDefaultParams()) R:121
FS_DisallowPlainText property
  (F_ApiGetOpenDefaultParams()) R:140
FS_DisallowSgml property
  (F_ApiGetImportDefaultParams()) R:122
FS_DisallowSgml property
  (F_ApiGetOpenDefaultParams()) R:140
FS_DisallowXml property
  (F_ApiGetImportDefaultParams()) R:122
FS_DisallowXml property
  (F_ApiGetOpenDefaultParams()) R:140
FS_DontNotifyAPIClients property
  (F_ApiGetOpenDefaultParams()) R:122
FS_DontNotifyAPIClients property
  (F_ApiGetImportDefaultParams()) R:122
FS_DontNotifyAPIClients property
  (F_ApiGetOpenDefaultParams()) R:140
FS_FileIsGraphic property
  (F_ApiGetImportDefaultParams()) R:125
FS_FileIsGraphic property
  (F_ApiGetOpenDefaultParams()) R:141
FS_FileIsInUse property
  (F_ApiGetOpenDefaultParams()) R:140
FS_FileIsInUse property
  (F_ApiGetImportDefaultParams()) R:125
FS_FileIsInUse property
  (F_ApiGetSaveDefaultParams()) R:153
FS_FileIsMakerDoc property
  (F_ApiGetImportDefaultParams()) R:124
FS_FileIsOldVersion property
  (F_ApiGetOpenDefaultParams()) R:140
FS_FileIsSgmlDoc property
  (F_ApiGetImportDefaultParams()) R:122
FS_FileIsStructured property
  (F_ApiGetOpenDefaultParams()) R:141
FS_FileIsText property
  (F_ApiGetImportDefaultParams()) R:126
FS_FileIsText property
  (F_ApiGetOpenDefaultParams()) R:141
FS_FileXmlDoc property
  (F_ApiGetImportDefaultParams()) R:122
FS_FileType property
  (F_ApiGetSaveDefaultParams()) G:62, R:154
FS_FileTypeHint property
  (F_ApiGetImportDefaultParams()) R:122
FS_FileTypeHint property
  (F_ApiGetOpenDefaultParams()) R:141
FS_FitGraphicInSelectedRect property
  (F_ApiGetImportDefaultParams()) R:125
FS_FontChangedMetric property
  (F_ApiGetOpenDefaultParams()) R:141
FS_FontNotFoundInCatalog property
  (F_ApiGetOpenDefaultParams()) R:141
FS_FontNotFoundInDoc property
  (F_ApiGetOpenDefaultParams()) R:142
FS_ForceImportAsText property
  (F_ApiGetOpenDefaultParams()) R:122
FS_ForceOpenAsText property
  (F_ApiGetOpenDefaultParams()) R:142
FS_FormatImportedText property
  (F_ApiGetImportDefaultParams()) R:124
FS_GraphicDpi property
  (F_ApiGetImportDefaultParams()) R:125
FS_HowToImport property
  (F_ApiGetImportDefaultParams()) R:140
FS_ImportAsType property
  (F_ApiGetImportDefaultParams()) R:123
FS_ImportAsType property (F_ApiImport) R:121
FS_ImportedFileName property (F_ApiImport) R:194
FS_ImportedTblTag property
  (F_ApiGetImportDefaultParams()) R:126
FS_ImportFlowPageSpace property
  (F_ApiGetImportDefaultParams()) R:124
FS_ImportFlowTag property
  (F_ApiGetImportDefaultParams()) R:124
FS_ImportFileType property
  (F_ApiGetImportDefaultParams()) R:124
FS_ImportNativeError property (F_ApiImport) R:194
FS_ImportStatus property (F_ApiImport) R:194
possible status flags (table) R:195
FS_LanguageNotAvailable property
  (F_ApiGetOpenDefaultParams()) R:142
FS_LeaveHeadingRowsEmpty property
  (F_ApiGetImportDefaultParams()) R:126
FS_LockCantBeReset property
(F_ApiGetOpenDefaultParams()) R:142
FS_MakeIconic property
(F_ApiGetOpenDefaultParams()) R:142
FS_MakePageCount property
(F_ApiGetSaveDefaultParams()) R:155
FS_MakeVisible property
(F_ApiGetOpenDefaultParams()) R:142
FS_MakeVisible property
(F_ApiGetUpdateBookDefaultParams()) G:358, R:188
FS_ManualUpdate property
(F_ApiGetImportDefaultParams()) R:123
FS_NameStripe property
(F_ApiGetOpenDefaultParams()) G:49, R:142
FS_NewDoc property
(F_ApiGetOpenDefaultParams()) G:49, R:143, R:239
FS_NumCellSeparators property
(F_ApiGetImportDefaultParams()) R:126
FS_NumColumns property
(F_ApiGetImportDefaultParams()) R:126
FS_OpenDocViewOnly property
(F_ApiGetOpenDefaultParams()) G:49, R:143
FS_OpenedFileName property (F_ApiOpen()) G:51, R:239
FS_OpenFileNotWritable property
(F_ApiGetOpenDefaultParams()) R:143
FS_OpenId property
(F_ApiGetOpenDefaultParams()) R:143
FS_OpenInNewWindow property
(F_ApiGetOpenDefaultParams()) R:144
FS_OpenNativeError property (F_ApiOpen()) G:51, R:239
values of and associated status flags R:240
FS_OpenStatus property (F_ApiOpen()) G:51, R:240, R:243
status flags of R:240
FS_RasterDpi property
(F_ApiGetImportDefaultParams()) R:125
FS_RasterImageWidth property
(F_ApiGetImportDefaultParams()) R:125
FS_RefFileNotFound property
(F_ApiGetOpenDefaultParams()) R:144
FS_RemoveManualPageBreaks property
(F_ApiGetImportDefaultParams()) R:124
FS_RemoveOverrides property
(F_ApiGetImportDefaultParams()) R:124
FS_RetainNameStripe property
(F_ApiGetSaveDefaultParams()) R:155
FS_SaveAsModeName property
(F_ApiGetSaveDefaultParams()) R:156
FS_SavedFileName property (F_ApiSave()) G:65, R:272
FS_SaveFileNotWritable property
(F_ApiGetSaveDefaultParams()) R:155
FS_SaveFileTypeHint property
(F_ApiGetSaveDefaultParams()) R:155
FS_SaveLockCantBeReset property
(F_ApiGetSaveDefaultParams()) R:154
FS_SaveModDateChanged property
(F_ApiGetSaveDefaultParams()) R:155
FS_SaveMode property
(F_ApiGetSaveDefaultParams()) G:62, R:156
FS_SaveNativeError property (F_ApiSave()) G:65, R:272
values of and associated status flags R:273
FS_SaveStatus property (F_ApiSave()) G:65, R:273
status flags of R:273
FS_SaveTextExtraBlankLineAtEOP property
(F_ApiGetSaveDefaultParams()) R:156
FS_SaveTextTblSetting property
(F_ApiGetSaveDefaultParams()) R:156
FS_SgmlImportApplication property
(F_ApiGetImportDefaultParams()) R:123
FS_SgmlOpenApplication property
(F_ApiGetOpenDefaultParams()) R:144
FS_SgmlSaveApplication property
(F_ApiGetOpenDefaultParams()) R:156
FS_ShowBookErrorLog property
(F_ApiGetUpdateBookDefaultParams()) G:358, R:188
FS_ShowBrowser property
(F_ApiGetImportDefaultParams()) R:123
FS_ShowBrowser property
(F_ApiGetOpenDefaultParams()) G:49, R:144
FS_ShowRasterDpiDialog property
(F_ApiGetImportDefaultParams()) R:125
FS_ShowSaveTextDialog property
(F_ApiGetSaveDefaultParams()) R:157
FS_StructuredImportApplication property
(F_ApiGetImportDefaultParams()) R:123
FS_StructuredOpenApplication property
(F_ApiGetOpenDefaultParams()) R:144
FS_StructuredSaveApplication property
(F_ApiGetOpenDefaultParams()) R:157
FS_TblNumHeadingRows property
(F_ApiGetOpenDefaultParams()) R:126
FS_TextInsetName property
(F_ApiGetImportDefaultParams()) R:123
Index - Reserved Names

FS_TreatParaAsRow property
(F_ApiGetImportDefaultParams()) R:127
FS_UNIXpermissions property
(F_ApiGetSaveDefaultParams()) R:157
FS_UpdateBookGeneratedFiles property
(F_ApiGetUpdateBookDefaultParams()) R:188
FS_UpdateBookMasterPages property
(F_ApiGetUpdateBookDefaultParams()) R:188
FS_UpdateBookNumbering property
(F_ApiGetUpdateBookDefaultParams()) R:189
FS_UpdateBookOleLinks property
(F_ApiGetUpdateBookDefaultParams()) R:189
FS_UpdateBookStatus property
(F_ApiUpdateBook()) R:339
FS_UpdateBookTextReferences property
(F_ApiGetUpdateBookDefaultParams()) R:189
FS_UpdateBookXRefs property
(F_ApiGetUpdateBookDefaultParams()) R:189
FS_UpdateBrowserDirectory property
(F_ApiGetOpenDefaultParams()) R:144
FS_UpdateFRVList property
(F_ApiGetSaveDefaultParams()) R:157
FS_UpdateTextReferences property
(F_ApiGetOpenDefaultParams()) R:144
FS_UpdateXRefs property
(F_ApiGetOpenDefaultParams()) R:145
FS_UseAutoSaveFile property
(F_ApiGetOpenDefaultParams()) R:145
FS_UseDefaultUNIXpermissions property
(F_ApiGetSaveDefaultParams()) R:157
FS_UseMainFlow property
(F_ApiGetImportDefaultParams()) R:124
FS_UseRecoverFile property
(F_ApiGetOpenDefaultParams()) R:145

FT__
FT_ElementCatalog property value type G:273, R:690
FT_Id property value type G:273, R:690
FT_Integer property value type G:273, R:690
FT_Ints property value type G:273, R:690
FT_Metric property value type G:273, R:690
FT_Metrics property value type G:273, R:690
FT_Points property value type G:273, R:690
FT_String property value type G:273, R:690
FT.Strings property value type G:273, R:690
FT_Tabs property value type G:273, R:690
FT_TextLoc property value type G:273, R:690
FT_TextRange property value type G:273, R:690
FT_UBytes property value type G:273, R:690
FT_ULints property value type G:273, R:690

FTF__
FTF_ALL bit flag (FTI_CharPropsChange) G:316, R:689
FTF_ANGLE bit flag (FTI_CharPropsChange) G:315, R:167, R:688
FTF_CAPITALIZATION bit flag
(FTI_CharPropsChange) G:315, R:167, R:688
FTF_CHANGEBAR bit flag
(FTI_CharPropsChange) G:315, R:167, R:688
FTF_CHARTAG bit flag
(FTI_CharPropsChange) G:315, R:167, R:688
FTF_COLOR bit flag (FTI_CharPropsChange) G:315, R:167, R:688
FTF_CONDITIONTAG bit flag
(FTI_CharPropsChange) G:315, R:167, R:688
FTF_ENCODING bit flag
(FTI_CharPropsChange) G:315, R:167, R:688
FTF_FAMILY bit flag (FTI_CharPropsChange) G:315, R:167, R:688
FTF_KERNX bit flag (FTI_CharPropsChange) G:315, R:168, R:688
FTF_KERNY bit flag (FTI_CharPropsChange) G:315, R:168, R:688
FTF_LANGUAGE bit flag
(FTI_CharPropsChange) G:315, R:168, R:688
FTF_OVERLINE bit flag
(FTI_CharPropsChange) G:315, R:168, R:688
FTF_PAIRKERN bit flag
(FTI_CharPropsChange) G:315, R:168, R:688
FTF_POSITION bit flag
(FTI_CharPropsChange) G:315, R:168, R:688
FTF_SHADOW bit flag
(FTI_CharPropsChange) G:315, R:168, R:688
FTF_SIZE bit flag (FTI_CharPropsChange) G:315, R:168, R:688
FTF_SPREAD bit flag
(FTI_CharPropsChange) G:315, R:168, R:688
FTF_STRETCH bit flag
(FTI_CharPropsChange) G:315, R:168, R:688
FTF_STRIKETHROUGH bit flag
(FTI_CharPropsChange) G:315, R:168, R:688
Index - Reserved Names

FTF_TSUME bit flag (FTI_CharPropsChange) G:315, R:168, R:689
FTF_UNDERLINING bit flag (FTI_CharPropsChange) G:315, R:168, R:688
FTF_VARIATION bit flag (FTI_CharPropsChange) G:315, R:168, R:688
FTF_WEIGHT bit flag (FTI_CharPropsChange) G:315, R:168, R:688
FTI_ElementBegin text item type G:314, R:166, R:687
FTI_ElementEnd text item type G:314, R:166, R:687
FTI_ElementPrefixBegin text item type R:166
FTI_ElementPrefixEnd text item type R:166
FTI_ElementSuffixBegin text item type R:166
FTI_ElementSuffixEnd text item type R:166
FTI_FlowBegin text item type G:313, R:166, R:686
FTI_FlowEnd text item type G:313, R:166, R:686
FTI_FnAnchor text item type G:313, G:327, R:166, R:687
FTI_FrameAnchor text item type G:313, R:166, R:687
FTI_LineBegin text item type G:313, R:166, R:686
FTI_LineEnd text item type G:313, R:166, R:686
FTI_MarkerAnchor text item type G:314, G:319, R:166, R:687
FTI_PageBegin text item type G:313, R:166, R:686
FTI_PageEnd text item type G:313, R:166, R:686
FTI_PgfBegin text item type G:313, R:166, R:686
FTI_PgfEnd text item type G:313, R:166, R:686
FTI_RubiCompositeBegin text item type G:314, R:172, R:687
FTI_RubiCompositeEnd text item type G:314, R:687
FTI_RubiTextBegin text item type G:314, R:171, R:688
FTI_RubiTextEnd text item type G:314, R:171, R:688
FTI_String text item type G:313, G:324, G:326, R:166, R:686
FTI_SubColBegin text item type G:313, R:166, R:687
FTI_SubColEnd text item type G:313, R:167, R:687
FTI_TblAnchor text item type G:312, G:313, G:332, R:167, R:687
FTI_TextFrameBegin text item type G:313, R:167, R:687
FTI_TextFrameEnd text item type G:313, R:167, R:687
FTI_TextInsetBegin text item type G:313, R:167, R:686
FTI_TextInsetEnd text item type G:314, R:167
FTI_TextObjId text item type G:313, R:167, R:686
FTI_VarEnd text item type G:314, G:326, G:327, R:167, R:687
FTI_XRefBegin text item type G:314, G:323, R:167, R:687
FTI_XRefEnd text item type G:314, G:323, R:167, R:687
FunctionT data type G:367
FUnixPath path type G:376

FV_

FV_AllElements flag (modifiers) R:91
FV_ALTERNATIVE constant (F_ElementCatalogEntryT) R:113, R:683
FV_AT_CHOICES constant (F_AttributeDefT) R:109
FV_AT_INTEGER constant (F_AttributeDefT) R:109
FV_AT_INTEGERS constant (F_AttributeDefT) R:109
FV_AT_REAL constant (F_AttributeDefT) R:109
FV_AT_REALS constant (F_AttributeDefT) R:109
FV_AT_STRING constant (F_AttributeDefT) R:109
FV_AT_UNIQUE_ID constant (F_AttributeDefT) R:109
FV_AT_UNIQUE_IDREF constant (F_AttributeDefT) R:109
FV_AT_UNIQUE_IDREFS constant (F_AttributeDefT) R:109
FV_BadEnclosingDocId status flag (FS_ImportStatus) R:195
FV_BadImportFileName status flag (FS_ImportStatus) R:195
FV_BadImportFileId status flag (FS_ImportStatus) R:195
FV_BadImportFileType status flag (FS_ImportStatus) R:195
FV_BadImportScriptValue status flag (FS_ImportStatus) R:195
FV_CancelFileDoc status flag (FS_ImportStatus) R:196
FV_CancelFileFilterable status flag (FS_ImportStatus) R:196
FV_CancelFileMacEdition status flag (FS_ImportStatus) R:196
FV_CancelFileMIF status flag (FS_ImportStatus) R:196
FV_CancelFileGraphic status flag (FS_ImportStatus) R:196
FV_CancelFileMacEdition status flag (FS_ImportStatus) R:196
FV_CancelFileMIF status flag (FS_ImportStatus) R:196
FV_CancelFileText status flag (FS_ImportStatus) R:196
FV_CancelFileText status flag (FS_ImportStatus) R:196
FV_CancelOpen mode constant R:54
FV_CancelOpenDir mode constant R:54
Index - Reserved Names

FV_ChooseSave mode constant R:54
FV_ChooseSelect mode constant R:54
FV_CmdAlert constant (FP_HypertextParsedCmdCode ) R:555
FV_CmdAlertTitle constant (FP_HypertextParsedCmdCode ) R:555
FV_CmdExit constant (FP_HypertextParsedCmdCode ) R:555
FV_CmdGoToLink constant (FP_HypertextParsedCmdCode ) R:555
FV_CmdGoToLinkFitWin constant (FP_HypertextParsedCmdCode ) R:555
FV_CmdGoToNew constant (FP_HypertextParsedCmdCode ) R:555
FV_CmdGoToObjectId constant (FP_HypertextParsedCmdCode ) R:555
FV_CmdGoToObjectIdFitWin constant (FP_HypertextParsedCmdCode ) R:555
FV_CmdGoToPage constant (FP_HypertextParsedCmdCode ) R:555
FV_CmdMatrix constant (FP_HypertextParsedCmdCode ) R:555
FV_CmdMessage constant (FP_HypertextParsedCmdCode ) R:555
FV_CmdNewLink constant (FP_HypertextParsedCmdCode ) R:555
FV_CmdNextPage constant (FP_HypertextParsedCmdCode ) R:555
FV_CmdQuit constant (FP_HypertextParsedCmdCode ) R:556
FV_CmdQuitAll constant (FP_HypertextParsedCmdCode ) R:556
FV_CmdUnknown constant (FP_HypertextParsedCmdCode ) R:555
FV_DestFirstPage constant (FP_HypertextParsedCmdDest ) R:556
FV_DestFluidFlow constant (FP_HypertextParsedCmdDest ) R:556
FV_DestLastPage constant (FP_HypertextParsedCmdDest ) R:556
FV_DestMarker constant (FP_HypertextParsedCmdDest ) R:556
FV_DestMarkerNewLink constant (FP_HypertextParsedCmdDest ) R:556
FV_DestNowhere constant (FP_HypertextParsedCmdDest ) R:556
FV_DestObjectId constant (FP_HypertextParsedCmdDest ) R:556
FV_DestPageNum constant (FP_HypertextParsedCmdDest ) R:556
FV_DestXRef constant (FP_HypertextParsedCmdDest ) R:556
FV_DisallowedImportType status flag (FS_ImportStatus) R:195
FV_DlgClose constant (itemNum) G:217, G:226, R:93
FV_DlgEnter constant (itemNum) G:226, R:93
FV_DlgNoChange constant (itemNum) G:226, R:93
FV_DlgOptDontCare constant (itemNum) G:227
FV_DlgReset constant (itemNum) G:226, R:93
FV_DlgUndo constant (itemNum) G:226, R:94
FV_Element flag (modifiers) R:91
FV_ElmentsOfType flag (modifiers) R:91
FV_ENABLE_ALWAYS_DISABLE constant (FP_EnabledWhen) G:174, R:526
FV_ENABLE_ALWAYS_ENABLE constant (FP_EnabledWhen) G:174, R:526
FV_ENABLE_CAN_PASTE constant (FP_EnabledWhen) R:527
FV_ENABLE_COPY constant (FP_EnabledWhen) R:527
FV_ENABLE_COPY_FONT constant (FP_EnabledWhen) R:527
FV_ENABLE_IN_CELL_TEXT constant (FP_EnabledWhen) R:527
FV_ENABLE_IN_FLOW constant (FP_EnabledWhen) R:527
FV_ENABLE_IN_MATH constant
(FP_EnabledWhen) R:527
FV_ENABLE_IN_PARA_TEXT constant
(FP_EnabledWhen) R:527
FV_ENABLE_IN_TABLE constant
(FP_EnabledWhen) R:527
FV_ENABLE_IN_TABLE_TITLE constant
(FP_EnabledWhen) R:527
FV_ENABLE_IN_TEXT constant
(FP_EnabledWhen) R:527
FV_ENABLE_IN_TEXT_LINE constant
(FP_EnabledWhen) G:174, R:527
FV_ENABLE_IS_AFRAME constant
(FP_EnabledWhen) R:527
FV_ENABLE_IS_CELL constant
(FP_EnabledWhen) R:527
FV_ENABLE_IS_CELLS constant
(FP_EnabledWhen) R:527
FV_ENABLE_IS_GRAPHIC_INSET constant
(FP_EnabledWhen) R:527
FV_ENABLE_IS_OBJ constant
(FP_EnabledWhen) R:527
FV_ENABLE_IS_OR_IN_FRAME constant
(FP_EnabledWhen) R:527
FV_ENABLE_IS_TABLE constant
(FP_EnabledWhen) R:527
FV_ENABLE_IS_TABLE_FRAME constant
(FP_EnabledWhen) R:527
FV_ENABLE_IS_TEXT_FRAME constant
(FP_EnabledWhen) R:527
FV_ENABLE_IS_TEXT_INSET constant
(FP_EnabledWhen) R:527
FV_ENABLE_IS_TEXT_SEL constant
(FP_EnabledWhen) G:174, R:527
FV_ENABLE_IS_VIEW_ONLY constant
(FP_EnabledWhen) R:527
FV_ENABLE_NEEDS_BOOKP_ONLY constant
(FP_EnabledWhen) R:528
FV_ENABLE_NEEDS_DOCP_ONLY constant
(FP_EnabledWhen) R:526, R:527, R:528
FV_ENABLE_OBJ_PROPS constant
(FP_EnabledWhen) R:527
FV_EvCaps flag (modifiers) R:94
FV_EvCommand flag (modifiers) R:94
FV_EvControl flag (modifiers) R:94
FV_EvMetaflag (modifiers) R:94
FV_EvOption flag (modifiers) R:94
FV_EvShift flag (modifiers) R:94
FV_FO_AFRAME constant (F_TextLocT) R:214, R:215
FV_FO_EQN constant (F_TextLocT) R:214, R:215
FV_FO_MARKER constant (F_TextLocT) R:214, R:215
FV_FO_TBL constant (F_TextLocT) R:214, R:215
FV_FO_VAR constant (F_TextLocT) R:214, R:215
FV_FO_XREF constant (F_TextLocT) R:214, R:215
FV_HypertextBadMatrixSize constant
(FP_HypertextValidateErr) R:554
FV_HypertextBadSyntaxPathSpec constant
(FP_HypertextParseErr) R:553
FV_HypertextCanOpenDestFile constant
(FP_HypertextValidateErr) R:554
FV_HypertextCommandIllegalWithinPopup constant
(FP_HypertextValidateErr) R:555
FV_HypertextDestinationLinkNotFound constant
(FP_HypertextValidateErr) R:554
FV_HypertextEmptyCommand constant
(FP_HypertextParseErr) R:553
FV_HypertextEmptyLineInMiffleOfPopup constant
(FP_HypertextValidateErr) R:555
FV_HypertextExpectedNumberParam constant
(FP_HypertextParseErr) R:554
FV_HypertextExtraArguments constant
(FP_HypertextParseErr) R:553
FV_HypertextFcodeInvalid constant
(FP_HypertextValidateErr) R:555
FV_HypertextFileNotMakerDoc constant
(FP_HypertextValidateErr) R:554
FV_HypertextFileNotRegular constant
(FP_HypertextValidateErr) R:554
FV_HypertextFlowMissingLines constant
(FP_HypertextValidateErr) R:554
FV_HypertextHelpDirNotFound constant
(FP_HypertextParseErr) R:554
FV_HypertextMatrixCommandInvalid constant
(FP_HypertextValidateErr) R:554
FV_HypertextMissingArguments constant
(FP_HypertextParseErr) R:553
FV_HypertextMissingPopupLabelItem constant
(FP_HypertextValidateErr) R:555
FV_HypertextMissingPopupMarker constant
(FP_HypertextValidateErr) R:555
FV_HypertextNoNamedFlow constant
(FP_HypertextValidateErr) R:554
FV_HypertextObjectIDNotFound constant
(FP_HypertextValidateErr) R:554
FV_HypertextPlaceholderNotFound constant
(FP_HypertextValidateErr) R:554
FV_HypertextRecursiveFlow constant
(FP_HypertextValidateErr) R:555
FV_HypertextSyntaxOK constant
(FP_HypertextParseErr) R:553
Index - Reserved Names

FV_HypertextUnanchoredPartialPath constant
(F_HypertextParseErr) R:554
FV_HypertextUnrecognizedCommand constant
(F_HypertextParseErr) R:553
FV_HypertextUnrecognizedObjectType constant
(F_HypertextValidateErr) R:554
FV_HypertextUsesDefaultText constant
(F_HypertextValidateErr) R:554
FV_HypertextValid constant
(F_HypertextValidateErr) R:554
FV_ImportedByCopy status flag
(FS_ImportStatus) R:195
FV_ImportedFilteredFile status flag
(FS_ImportStatus) R:196
FV_ImportedGraphicFile status flag
(FS_ImportStatus) R:196
FV_ImportedMakerDoc status flag
(FS_ImportStatus) R:196
FV_ImportedMIF status flag (FS_ImportStatus) R:195
FV_ImportedSgmlDoc status flag
(FS_ImportStatus) R:196
FV_ImportedXmlDoc status flag
(FS_ImportStatus) R:196
FV_ImportFileNotReadable status flag
(FS_ImportStatus) R:195
FV_ImportTextTable status flag
(FS_ImportStatus) R:195
FV_INCLUSION constant
(F_ElementCatalogEntryT) R:113
FV_InsertionPointInFootnote status flag
(FS_ImportStatus) R:196
FV_InsertionPointInTableCell status flag
(FS_ImportStatus) R:196
FV_InsertionPointNotInText status flag
(FS_ImportStatus) R:195
FV_insufficientMemory status flag
(FS_ImportStatus) R:195
FV_LANG_BRAZILIAN API language constant
(F_LanguageNumber()) R:397
FV_LANG_BRITISH API language constant
(F_LanguageNumber()) R:397
FV_LANG_CANADIAN_FRENCH API language constant
(F_LanguageNumber()) R:397
FV_LANG_CATALAN API language constant
(F_LanguageNumber()) R:397
FV_LANG_DANISH API language constant
(F_LanguageNumber()) R:397
FV_LANG_DUTCH API language constant
(F_LanguageNumber()) R:397
FV_LANG_ENGLISH API language constant
(F_LanguageNumber()) R:397
FV_LANG_FINNISH API language constant
(F_LanguageNumber()) R:398
FV_LANG_FRENCH API language constant
(F_LanguageNumber()) R:397
FV_LANG_GERMAN API language constant
(F_LanguageNumber()) R:397
FV_LANG_ITALIAN API language constant
(F_LanguageNumber()) R:397
FV_LANG_JAPANESE API language constant
(F_LanguageNumber()) R:398
FV_LANG_KOREAN API language constant
(F_LanguageNumber()) R:398
FV_LANG_NORWEGIAN API language constant
(F_LanguageNumber()) R:398
FV_LANG_NYNORSK API language constant
(F_LanguageNumber()) R:398
FV_LANG_PORTUGESE API language constant
(F_LanguageNumber()) R:397
FV_LANG_SIMPLIFIED_CHINESE API language constant
(F_LanguageNumber()) R:398
FV_LANG_SPANISH API language constant
(F_LanguageNumber()) R:397
FV_LANG_SWEDISH API language constant
(F_LanguageNumber()) R:398
FV_LANG_SWISS_GERMAN API language constant
(F_LanguageNumber()) R:397
FV_LANG_TRADITIONAL_CHINESE API language constant
(F_LanguageNumber()) R:398
FV_LOOSELY_VALID constant
(F_ElementCatalogEntryT) R:113, R:683
FV_MissingScript status flag (FS_ImportStatus) R:195
FV_NoFlowWithSpecifiedName status flag
(FS_ImportStatus) R:195
FV_NoMainFlow status flag (FS_ImportStatus) R:195
FV_OBJ_END_OFFSET constant G:117, R:689
FV_ObjectDataLink constant
(F_ParsedCmdDestObjType) R:557
FV_ObjectElement constant
(F_ParsedCmdDestObjType) R:557
FV_ObjectGraphic constant
(F_ParsedCmdDestObjType) R:557
FV_ObjectMarker constant
(F_ParsedCmdDestObjType) R:557
FV_ObjectPgf constant
(F_ParsedCmdDestObjType) R:557
FV_ObjectTextInset constant
(F_ParsedCmdDestObjType) R:557
FV_ObjectUnknown constant
(F_ParsedCmdDestObjType) R:557
Index - Reserved Names

FV_ObjectXref constant
  (FP_HypertextParsedCmdDestObjType) R:557
FV_SaveFmtBinary for
  (F_ApiGetSaveDefaultParams()) G:62, R:154
FV_SaveFmtBinary60 for
  (F_ApiGetSaveDefaultParams()) G:62, R:154
FV_SaveFmtFilter for
  (F_ApiGetSaveDefaultParams()) R:154
FV_SaveFmtInterchange for
  (F_ApiGetSaveDefaultParams()) G:62, R:154
FV_SaveFmtPdf for
  (F_ApiGetSaveDefaultParams()) G:62, R:154
FV_SaveFmtSgml for
  (F_ApiGetSaveDefaultParams()) G:62, R:154
FV_SaveFmtStationery for
  (F_ApiGetSaveDefaultParams()) G:62, R:154
FV_SaveFmtText for
  (F_ApiGetSaveDefaultParams()) G:62, R:154
FV_SaveFmtViewOnly for
  (F_ApiGetSaveDefaultParams()) G:62, R:154
FV_SaveFmtXml for
  (F_ApiGetSaveDefaultParams()) G:62, R:154
F_V_STRINGLY_VALID constant
  (F_ElementCatalogEntryT) R:113, R:683
FV_T_STRINGS constant (F_AttributeDefT) R:109
FV_UserCanceledImport status flag
  (FS_ImportStatus) R:196
FV_UserCanceledImportBrowser status flag
  (FS_ImportStatus) R:196
fwrite() function G:369

G, H

GenericType data type G:367, R:675
HandleT structure R:677
HashTable structure R:678

I, J, K, L

importing files
  PDF G:230
  SGML G:230
  XML G:230
in (metric constant for inches) G:271
KBD_ALIGN f-code G:39
KBD_INPUT f-code G:39
KBD_OPEN f-code G:39

M

Makes R:468
markup file, importing G:230
message apiclient hypertext command G:7, G:31
  responding to G:32
    function for R:1, R:206
  syntax for G:32
MfUnitT data type G:386
  values of (table) G:386

N, O

NativeCharT data type G:368
NativeDoubleT data type G:367
NativeIntT data type G:368
NativeLongT data type G:368
NativeULongT data type G:368
NO_DSE flag (FDE functions) G:372, G:379

P, Q, R

PByteT data type G:368
PCharT data type G:368
PDF files, importing G:230
PGF_APPLY_TAG f-code G:39
PRealT data type G:368
printf() function G:369
ProcedureT data type G:368
PShortT data type G:368
PtrT data type G:368
pts (metric constant for points) G:271
PUByteT data type G:368
PUCharT data type G:368
PUShortT data type G:368
QSort() C library function R:27
RealT data type G:368

S, T

SGML files, importing G:230
ShortT data type G:368, R:675
sleep() C library function R:328
sprintf() function G:369
sqrt() C library function R:491
StrBuffT data type G:368
strcat() C library function G:370
strcmp() C library function G:370
strcpy() C library function G:370
StringListT structure G:382, R:678
strlen() C library function G:370
strtod() C library function R:492
strtol() C library function R:493
Text properties (FO_Cell) R:651
Text properties (FO_Flow) R:580
Text properties (FO_SubCol) R:521
Text properties (FO_TextFrame) R:606
Text properties (FO_TextLine) R:607
TXT_10 f-code G:39
TXT_FAMILY_AND_VARIATION f-code G:39
UByteT data type G:368, R:675
UCharT data type G:368, R:675
UILanguage G:25
UIntT data type G:368, R:675
UShortT data type G:368, R:675

U, V, W, X, Y, Z

valType constants, tables of G:273, R:690
VoidT data type G:368, R:675
XML files, importing G:230