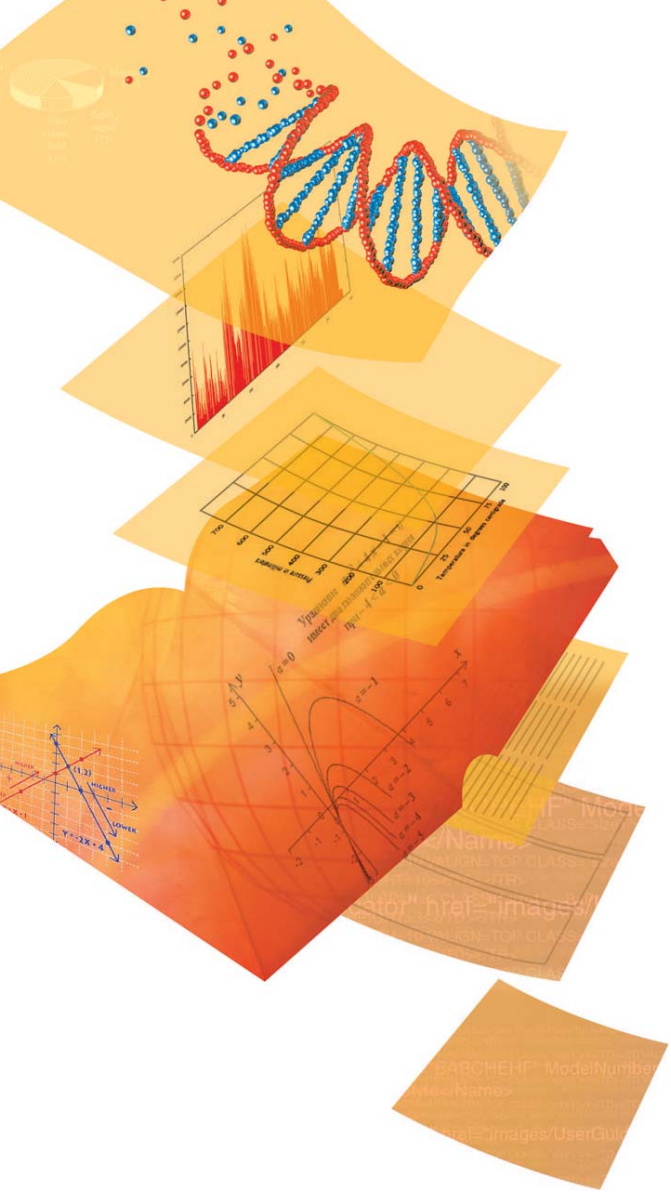

Adobe® FrameMaker® 7.2

FDK Supplement

Adobe

Adobe® FrameMaker® 7.2 FDK Supplement

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

Contents

1 What's New in the FDK

This chapter lists changes to the Frame® Developer's Kit (FDK) that are the result of changes to the Adobe® FrameMaker® product. Changes are listed for releases 7.0 and above, starting from the most recent release.

What's New for Release 7.2

With release 7.2, FrameMaker introduced the following changes that affect the FDK:

- Changes to the FDK documentation (this FDK Supplement document).
- Enhancement of undo and redo to support multiple operations.
- Enhancements to XML support, including XML Schema support and XSLT transformation.

For details and examples of these features, see Chapter 2, "New Features in the FDK: 7.2".

What's New for Release 7.1

With release 7.1, FrameMaker introduced the following changes that affect the FDK:

- Changes to the FDK documentation (an FDK Supplement document).
- Additional file types supported for import into FrameMaker.
- Enhancements to XML support, including roundtripping of conditional text and external cross-references, and the ability to open XML files as XML, locking the XML file on disk and saving back to XML by default.

For details and examples of these features, see Chapter 3, "New Features in the FDK: 7.1".

What's New for Release 7.0

With release 7.0, FrameMaker introduced the following changes that affect the FDK:

- Changes to the FDK documentation
- A single FrameMaker product, which you can run in structured or unstructured mode

- Support for importing from and exporting to XML format
- Enhanced text encoding which includes UNICODE and ICU libraries
- Support for XMP metadata for workflow management
- Support for WebDAV
- New or changed FDK API functions, properties, and constants

For details and examples of these features, see Chapter 4, "New Features in the FDK: 7.0".

2 New Features in the FDK: 7.2

This chapter provides details of new features that were introduced in Release 7.2. These include:

- XML Schema support: FrameMaker 7.2 added validation of structure against XML Schema for XML import and export, and the ability to create or modify an EDD from a Schema. The FDK has been extended to support this.
- XSL transformation support: FrameMaker 7.2 added XSL transformation support for XML import and export. Transformations are applied before read rules on import, and after write rules on export.
- Multiple Undo and Redo: FrameMaker 7.2 added multiple undo and redo. The FDK added functions that allow a client to control multiple undo and redo operations programmatically.

NOTE: FDK XML clients compiled with FDK 7.1 must be recompiled with FDK 7.2. If you do not recompile, FDK 7.2 reports this error:

```
This application has failed to start because xerces-c_1_5_1.dll
was not found. Re-installing the application may fix this problem.
```

The recommended compiler is Visual Studio.NET.

XML Schema support

An *XML schema language* is a method for specifying a grammar or rules for a class of XML documents. This release supports W3C's XML Schema by mapping it to DTD, from which the definitions are mapped into EDD.

- You can validate an XML document with respect to an associated Schema while opening it or while saving it to XML.

  For importing an XML document, include the path of the schema file in the XML using attributes—`noNamespaceSchemaLocation` or `schemaLocation` depending on whether your schema includes a target namespace or not. For export, specify the Schema in the `XmlApplication` element of the `structapps.fm` file.

- You can import XML Schema documents into DTD and then EDD in order to create structured applications for Schema, and author documents based on Schema.

A DTD is generated automatically when you import XML that references Schema, and the EDD is generated from the DTD. Types are converted to DTD-equivalent types. For complete details of how Schema is mapped to DTD, see the *Structure Application Developer's Guide*.

This release offers support for Schema that is equivalent to what was previously available for DTD. That is, EDD has not been extended to accommodate features in Schema that are not available in DTD. For this reason, Schemas are read-only, and you cannot export the EDD back

out to Schema. Upon load, a general warning is shown for elements in the Schema that cannot be converted to DTD.

Schema support in structure applications

The new element `Schema`, a child of the `XmlApplication` element, specifies Schema information in the `structapps.fm` file. This element contains the path of a Schema file to be used for validation when exporting a FrameMaker document to XML. For details, see the *Structure Application Developer's Guide*.

In order for a structure application to be selectable in the Use Structured Application list while importing a document that is associated with a Schema, the Schema's root element must be included in the application's `DOCTYPE` in the `XmlApplication` element. The property `Namespace` in `XmlApplication` must be set to `true` if instance documents use namespaces.

Schema support in the FDK

To convert a Schema definition to DTD programmatically, use the function `F_Api_callClient` with a new option, `StructuredSchemaToDTD`, specifying the path to the Schema file. For example:

```
F_ApiCallClient("FmDispatcher", "StructuredSchemaToDTD pathname");
```

The function resolves the path in the same way as for the option `StructuredReadDtd`. It writes the resulting DTD file to the output directory and with the suffix that you have specified using the options `StructuredOutputDir` and `StructuredOutputSuffix`. For example:

```
F_ApiCallClient("FmDispatcher", "StructuredOutputDir pathname");
F_ApiCallClient("FmDispatcher", "StructuredOutputSuffix suffix");
```

The return value for the new option can be one of these constants:

- `FE_BadSaveFileName`: The DTD file could not be written out due to permission problems.
- `FE_BadParameter`: The Schema to DTD conversion generated errors.
- `FE_Success`: The Schema to DTD conversion was successful.

For additional information on the function `F_Api_callClient`, see Appendix A of the *FDK Programmer's Reference*.

Schema support in the Import/Export API

A new function, `Srw_GetExportSchemaFilePath()`, has been added to the Import/Export API. The function retrieves the platform-independent file path of the Schema to use when exporting FrameMaker documents to XML, as specified in the structure application.

For details of this change, see the *Structure Import/Export Programmer's Guide*.

XSL transformation support

XSLT (XSL Transformation language), a specialized programming language written in XML, is the means by which transformations defined in XSL (Extensible Stylesheet Language) are applied to XML documents. FrameMaker 7.2 includes an XSLT processor that allows you to associate an XSL file with an XML structure application or XML document, and apply the transformations defined in that document when importing from or exporting to XML. The XSLT processor supports the W3C XSLT 1.0 standard.

New elements in the structure application allow you to specify an XSL file as part of your XML structure application, to be used for both import and export. To specify an XSL file for either import or export in the structure application, include the element `XSLTPreferences` in the `Stylesheets` element which is a child of the `XMLApplication` element. The `XSLTPreferences` element can have one `PreProcessing` and/or one `PostProcessing` child element, each of which in turn contains a `Stylesheet` element. The `Stylesheet` element references the XSL file that is to be used for pre- or postprocessing. Each can also contain a `StylesheetParameters` element, which allows you to set parameters in the XSL at run time, before the transformation is applied.

The `xml-stylesheet` processing instruction (PI) now allows you to specify an XSL file in an XML markup document. By default FrameMaker does not use this PI for XSL tranformation. To use it, include the `ProcessStylesheetPI` element in the `XSLTPreferences` element in the structure application. If this element is set to `enable`, an XSL file specified by the PI supercedes any XSL specified in the structure application when importing that document, and, if `Retain Stylesheet Information` is enabled in the structure application, when exporting it.

When you import an XML document into FrameMaker, the specified XSL file is used to preprocess the XML before read rules are applied. That is, the result of applying an XSL transformation on import is a new file, which (if it is an XML file) is passed to the read/write rules. The XSL can be embedded in the XML document, or it can be an external file.

Upon export, XSL tranformations are applied after the default or explicit write rules. The result of applying read/write rules on export is a new XML file, which, if it is valid, is passed to the XSLT processor.

For additional information, see the *Structured Application Developer's Guide*.

Multiple Undo and Redo

FrameMaker 7.2 has enhanced undo and redo capability in a number of ways. In this release, undo capability is available for more commands, such as Text Option changes and Global Replace. You can undo and redo multiple operations. The Edit > Undo/Redo menu shows the most recent command, but a complete command history is available in the new Edit > History palette, so that the user can select a specific action to undo or redo.

The command history, (undo and redo stacks) is kept separately for each document, so undoing or redoing an operation in one document does not change the undo capability of another open document. For details of which commands are undoable and which clear the undo history, see the *FrameMaker 7.2 Release Notes*.

In the FDK, undo capability has been added to API commands, and new functions have been added to the FDK to support the programmatic control of this feature (see Controlling Undo/Redo in the FDK API below).

FDK programmers should note the following:

- The UIDs (unique identifiers) of document objects, which are generally persistent across sessions, can change after undo and redo operations.
- Commands from `fmbatch` clients in UNIX are *not* recorded in the undo history.
- Commands from asynchronous clients can be recorded, according to the status of the flag and property described below.

Controlling Undo/Redo in the FDK API

New initialization and session properties control whether undo information is recorded for the FDK, and new API function allow you to explicitly control the undo and redo stacks.

New initialization flag: EnableUndoInFDK

A new flag, `EnableUndoInFDK`, in the initialization file (`maker.ini` in Windows, or `xresources/Maker` in UNIX) allows you to explicitly enable or disable undo/redo functionality for API commands, and its associated overhead. It is `false` (off) by default, which means that the undo behavior is the same as in previous releases; that is, calls to API commands clear the undo and redo stacks in the selected document, and API commands cannot be undone. To enable the new undo behavior for API commands, set the flag to `true`. (This flag does not affect the FrameMaker user interface or interactive behavior.)

When `EnableUndoInFDK` is `true`, all API commands that modify document contents can be undone (see Undoable API commands). Commands that do not modify content, such as saving a document, copying text, or manipulating windows, cannot be undone and are not recorded in the command history (undo stack). Calls originating from a UNIX `fmbatch` client are not undoable.

New session properties

FP_UndoFDKRecording. A new session property, `FP_UndoFDKRecording`, can override the default value specified in the initialization flag `EnableUndoInFDK`. Use `F_ApiSetInt` to set this property value, and `F_ApiGetInt` to retrieve it. Set the property to zero to disable FDK Undo recording for a session, or to a non-zero value to enable Undo recording.

FP_StackWarningLevel. A new session property, `FP_StackWarningLevel`, determines how warnings are displayed when history-clearing operations occur. It corresponds to an option set in the Preferences dialog, and to the preference-file flag `hpWarning`. Use `F_ApiSetInt` to set this property value, and `F_ApiGetInt` to retrieve it. Allowed values are:

- `FvWarnNever`: Disables warnings for history-clearing operations for the session.
- `FvWarnOnce`: Displays a warning when a particular history-clearing command is issued, but does not warn on subsequent uses of that command.
- `FvWarnAlways`: Displays warnings every time a history-clearing command is issued.

New API functions

In previous releases, API commands cleared the undo stack. In this release, they no longer do so. However, a new command, `F_ApiUndoCancel`, explicitly clears both the undo and redo stacks in a specified document.

Many API commands call two or more other API functions. By default, each API call is recorded as a separate undo action in the undo stack of the selected document. To treat a series of API calls as one command, call `F_ApiUndoStartCheckpoint` before the first call and `F_ApiUndoEndCheckpoint` after the last call in the group.

F_ApiUndoCancel

```
VoidT F_ApiUndoCancel(F_ObjHandleT docId);
```

Clears both the undo and redo stacks in the document specified by *docId*. If an undo checkpoint has been started and not ended in this document, this call cancels the grouping operation.

On success, sets `FA_errno` to `FE_Success`. Otherwise, sets `FA_errno` to `FE_BadDocId` (invalid document ID).

Clear Undo Stack Example. This example demonstrates how to clear the undo stack when undoing an action could corrupt an external file. In this case, a string is deleted from an external database permanently when selected text is deleted in FrameMaker. If you cannot restore the deleted string in the database, undoing the delete action from a FrameMaker

document would result in database corruption. To protect against this, the example uses F_ApiUndoCancel to clear all user actions saved in the undo stack.

```
F_TextRangeT tr;
F_TextItemsT textItems;
StringT string = NULL;

tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
textItems = F_ApiGetTextForRange(docId, &tr, FTI_String);
string = CreateStringFromTextItems(textItems);

F_ApiDeleteText(docId, &tr);
DeleteTextFromDatabasePermanently(string);
F_ApiUndoCancel(docId);
```

F_ApiUndoStartCheckpoint

```
VoidT F_ApiUndoStartCheckpoint(F_ObjHandleT docId,
                               ConStringT description);
```

Records the starting point of a series of API calls that are to be treated as a single undoable operation in the document specified by *docId*. The *description* string appears in the Undo and Redo menus and the Command History palette.

If there is no corresponding call to F_APIUndoEndCheckpoint, all subsequent API calls are grouped into a single undoable operation.

You cannot nest checkpoints. A second call to this function that appears before the corresponding call to F_APIUndoEndCheckpoint is ignored.

On success, sets FA_errno to FE_Success. Otherwise, sets FA_errno to FE_BadDocId (invalid document ID).

F_ApiUndoEndCheckpoint

```
VoidT F_ApiUndoEndCheckpoint(F_ObjHandleT docId);
```

Marks the ending point of a series of API calls that are to be treated as a single undoable operation. The *docid* must specify the same document as the corresponding call to F_ApiUndoStartCheckpoint.

If any API call in the series clears the undo stack, the stack is cleared after the end checkpoint is reached.

On success, sets FA_errno to FE_Success. Otherwise, sets FA_errno to FE_BadDocId (invalid document ID).

Checkpoint Example. This example combines two API calls (F_ApiSetTextProps and F_ApiAddText) into one undoable action. The specified command name, “Add Big Red Text”

appears in the Undo menu and the Command History palette, rather than the two command names “Set Text Property” and “Add Text”.

```
#define pts (MetricT)65536
F_TextRangeT tr;
F_ObjHandleT colorId;
F_PropValsT props;
IntT i;

tr = F_ApiGetTextRange(FV_SessionId, docId, FP_TextSelection);
colorId = F_ApiGetNamedObject(docId, FO_Color, (StringT)"Red");
props = F_ApiGetTextProps(docId, &tr.beg);
i = F_ApiGetPropIndex(&props, FP_Color);
props.val[i].propVal.u.ival = colorId;
i = F_ApiGetPropIndex(&props, FP_FontSize);
props.val[i].propVal.u.ival = 100 * pts;

F_ApiUndoStartCheckpoint(docId, "Add Big Red Text");
F_ApiSetTextProps(docId, &tr, &props);
F_ApiDeallocatePropVals(&props);
F_ApiAddText(docId, &tr.beg, (StringT)"Big Red Text!");
F_ApiUndoEndCheckpoint(docId);
```

Undoable API commands

The following API commands are undoable.

F_ApiAddCols	F_ApiAddRows	F_ApiAddText
F_ApiApplyPageLayout	F_ApiClear	F_ApiClearAllChangebars
F_ApiCut	F_ApiDelete	F_ApiDeleteCols
F_ApiDeletePropByName	F_ApiDeleteRows	F_ApiDeleteText
F_ApiDeleteTextInsetContents	F_ApiDeleteUndefinedAttributes	F_ApiDemoteElement
F_ApiImport	F_ApiMergeIntoFirst	F_ApiMergeIntoLast
F_ApiNewAnchoredFormattedObject	F_ApiNewAnchoredObject	F_ApiNewBookComponentInHierarchy
F_ApiNewElement	F_ApiNewElementInHierarchy	F_ApiNewGraphicObject
F_ApiNewNamedObject	F_ApiNewSeriesObject	F_ApiNewSubObject
F_ApiNewTable	F_ApiPaste	F_ApiPromoteElement
F_ApiReformat	F_ApiUnStraddleCells	F_ApiResetEqnSettings
F_ApiResetReferenceFrames	F_ApiRestartPgfNumbering	F_ApiSetAttributeDefs
F_ApiSetAttributes	F_ApiSetElementRange	F_ApiSetId
F_ApiSetInt	F_ApiSetIntByName	F_ApiSetInts
F_ApiSetMetric	F_ApiSetMetricByName	F_ApiSetMetrics

F_ApiSetPoints
F_ApiSetProps
F_ApiSetPropVal
F_ApiSetString
F_ApiSetStrings
F_ApiSetTabs
F_ApiSetTextLoc
F_ApiSetTextProps
F_ApiSetTextPropVal
F_ApiSetTextRange
F_ApiSetTextVal
F_ApiSetUBytesByName
F_ApiUnWrapElement
F_ApiSimpleImportElementDefs
F_ApiSimpleImportFormats
F_ApiSplitElement
F_ApiStraddleCells
F_ApiUnWrapElement
F_ApiUpdateTextInset
F_ApiSave
F_ApiSimpleSave

3 New Features in the FDK: 7.1

This chapter provides details of new features that were introduced in Release 7.1. These include:

- Importing PDF Pages: FrameMaker 7.1 added the ability to import a page of a PDF document into a FrameMaker document. The FDK added a new property that enables a client to specify this operation with `F_ApiImport`.
- New File Types Available for Import: FrameMaker 7.1 made new file types available for import into FrameMaker. The FDK added arguments to the `FP_ImportHint` function to support these file type.
- New File Types Available for Import
  - Conditions and Cross-References Maintained in XML: FrameMaker 7.1 enhanced XML roundtripping by adding conditional text and external crossreferences support. This requires FDK users to link to a new version of the structure library.
  - Open as XML: FrameMaker 7.1 provided the ability to open XML files as XML, locking the XML file on disk and saving back to XML by default. This changed the behavior of the FDK functions `F_ApiSave` and `F_ApiSimpleSave`.

Importing PDF Pages

An FDK client can use the API scriptable property `FS_PDFPageNum` to import a particular page of a PDF document. The following example illustrates the use of this property.

```
IntT index;
IntT pageNum;
F_PropValsTparams;
/* Get the default params list for import */
params = F_ApiGetImportDefaultParams();
/* Get the index of the FS_PDFPageNum property in this list */
index = F_ApiGetPropIndex(&params, FS_PDFPageNum);
/* Specify the page number to be imported, here page 3 of the PDF doc*/
pageNum = 3;
params.val[index].propVal.u.ival = pageNum;
/* call F_ApiImport : See F_ApiImport documentation for details*/
...
F_ApiImport(...);
...
```

New File Types Available for Import

The following table shows the hint strings that can be used with `FP_ImportHint` to import additional file types added in FrameMaker 7.1. Note that the filter name is a four-byte code. If the name string contains fewer than four alphanumeric characters, the remaining bytes are filled out with spaces.

File type	Hint string
PDF	`"0001FRAMPDF "`
Photoshop (PSD)	`"0001ADBIPSD "`
JPEG2000 with extension:	
JP2	`"0001ADBIJP2 "`
JPX	`"0001ADBIJPX "`
J2C	`"0001ADBIJ2C "`
J2K	`"0001ADBIJ2K "`
JPC	`"0001ADBIJPC "`
JPF	`"0001ADBIJPF "`
Quark	`"0001ADBIQXD "`
PageMaker:	
7.0 document	`"0001ADBIPM D "`
7.0 template	`"0001ADBIPM T "`
7.0 document	`"0001ADBIP65 "`
7.0 document	`"0001ADBIT65 "`

XML Enhancement: Release 7.1

Release 7.1 enhanced XML support in the following ways:

- Conditions and Cross-References Maintained in XML
- Open as XML

Conditions and Cross-References Maintained in XML

FrameMaker 7.1 enhanced XML roundtripping by adding conditional text and external cross-references support.

The updated `struct.lib` in FrameMaker 7.1 makes these features available to structured clients. When clients link to the `struct.lib` supplied with FDK 7.1, these new features become part of the client.

Open as XML

Framemaker 7.1 added support for opening XML files as XML. When you open an XML file in FrameMaker, it retains the `.xml` extension, and is saved by default to XML. FrameMaker locks the XML file while it is open.

NOTE: Simple Save Incompatible Change

If an XML client opens an XML file, edits the file and then saves the same file using `F_ApiSimpleSave`, the function either fails to save the file or saves it as as FASL.

In this respect, FrameMaker 7.1 behaves differently from FrameMaker 7.0.

- FrameMaker 7.0 opens `file1.xml` as `file1.fm`, and `F_ApiSimpleSave` saves the file `file1.fm`.
- FrameMaker 7.1, opens the file as `file1.xml` and locks the file. If you use `F_ApiSimpleSave` to save the file, giving the original file name, you get an error because the file is locked and cannot be saved. If locking is turned off, the file is saved to FASL format.

You can use `F_ApiSimpleSave` to save to a filename that is different from the name of the open file. However, if you wish to save to XML, it is best to use `F_ApiSave` instead.

The following samples illustrate how to open and save an XML file.

Open an XML File

This opens an XML file. In FrameMaker 7.0, the XML file opens as an FM file, but in FrameMaker 7.1 and above, is opens as XML and the file is locked on disk.

```
VoidT openXML () {
     F_PropValsT params, *returnParamsp = NULL;
     IntT i;
/* Get default open properties. Return if it can't be allocated. */
     params = F_ApiGetOpenDefaultParams();
     if (params.len == 0) {
```

```
        F_Printf(NULL, (StringT)"File not opened.Unable to allocate
                  memory.\n");
     return;
     }
/* Set properties to open an XML document*/
/* Specify XML as file type to open*/
     i = F_ApiGetPropIndex(&params, FS_OpenAsType);
     params.val[i].propVal.u.ival = FV_TYPE_XML;
/*Allow user to select a file to open*/
     i = F_ApiGetPropIndex(&params, FS_ShowBrowser);
     params.val[i].propVal.u.ival = True;
/* Specify the XML application to be used when opening the document.*/
     i = F_ApiGetPropIndex(&params, FS_StructuredOpenApplication);
     params.val[i].propVal.u.sval = F_StrCopyString((StringT) "XDocBook");
     if (!(F_ApiOpen("", &params, &returnParamsp))) {
        F_ApiAlert("Couldn't open file.\n", FF_ALERT_CONTINUE_NOTE);
/* Deallocate property lists*/
        F_ApiDeallocatePropVals(&params);
        F_ApiDeallocatePropVals(returnParamsp);
        return;
        }
     else {
/* Deallocate property lists*/
        F_ApiDeallocatePropVals(&params);
        F_ApiDeallocatePropVals(returnParamsp);
        return;
     }
}
```

Save an XML File

This saves the active document to the file `sample.xml`. In FrameMaker 7.0, the active FM document is saved to disk as `sample.xml`, keeping the FM document active. In FrameMaker 7.1 and above, the open XML document is saved to `sample.xml`, making `sample.xml` the active document.

```
VoidT saveXML () {
     F_PropValsT params, *returnParamsp = NULL;
     IntT i;
     F_ObjHandleT docId = F_ApiGetId(0,0,FP_ActiveDoc);
/* Get default save properties. Return if it can't be allocated. */
     params = F_ApiGetSaveDefaultParams();
     if (params.len == 0) {
        F_Printf(NULL, (StringT)"Couldn't get save parameters!\n");
        return;
     }
/*Specify XML as file type to save*/
     i = F_ApiGetPropIndex(&params, FS_FileType);
     params.val[i].propVal.u.ival = FV_SaveFmtXml;
/* Specify the XML application to be used when saving the document.*/
     i = F_ApiGetPropIndex(&params, FS_StructuredSaveApplication);
     params.val[i].propVal.u.sval = F_StrCopyString((StringT) "XDocBook");
```

```
     if (!(F_ApiSave(docId, "sample.xml", &params, &returnParamsp))) {
        F_ApiAlert("Couldn't save file.\n", FF_ALERT_CONTINUE_NOTE);
/* Deallocate property lists*/
        F_ApiDeallocatePropVals(&params);
        F_ApiDeallocatePropVals(returnParamsp);
        return;
     }
     else {
/*Deallocate property lists*/
        F_ApiDeallocatePropVals(&params);
        F_ApiDeallocatePropVals(returnParamsp);
        return;
     }
}
```

4 New Features in the FDK: 7.0

This chapter provides details of new features that were introduced in Release 7.0 which affect the FDK. These include:

Changes to the FDK documentation

Aside from updates for the new release, there were slight changes to the documentation.

- Most of the FDK documentation got a new layout, which is more consistent with other Adobe Systems SDK documentation.
- An appendix was added to the FDK Programmer's Reference that describes how to use `F_ApiCallClient()` with the clients that ship with the FrameMaker product. This information was previously in the discussion for the function, `F_ApiCallClient()`.
- Because there is only one FrameMaker product, function descriptions no longer include a list of FrameMaker products that work with the function. Instead, the functions that require structured FrameMaker are identified at the beginning of the function description.
- The third chapter of the *Structure Import/Export API Programmer's Guide* presents the code for the XHTML Starter Kit's import/export client, and provides details about the code. In previous versions this chapter used a minimal example of the DocBook starter kit client.
- The FDK documentation was updated to include *FDK Sample Clients*, a list of sample clients in the file, `samplelist.pdf`.

Single FrameMaker Product

With release 7.0, Adobe Systems merged FrameMaker and FrameMaker+SGML into the same product. The user specifies which product interface to run with FrameMaker via user preferences. Also, there is no 7.0 release of FrameViewer or FrameReader.

Registering FDK clients

In earlier releases you could specify which FrameMaker products could load your FDK client. In FrameMaker 7.0 you can still specify which product interface is required for a session in order for FrameMaker to load your clients. For UNIX and Windows, you specify this via client registration. For the Macintosh, you can query the `FP_ProductIsStructured` session property, and cancel client initialization if necessary. For more information, see the sections on registering clients in the FDK platform guides.

Determining the product interface for a session

To determine the current product interface, get the `FP_Platform` session property. The value for this property can be one of:

```
FrameMaker
FrameMaker+SGML
FrameViewer
DemoMaker
DemoMaker+SGML
```

These are the same values as for the 6.0 release of FrameMaker. `FrameViewer` is saved for backward compatibility, and `FrameMaker+SGML` or `DemoMaker+SGML` indicate that Framemaker is running in structured mode.

Import and Export of XML

Release 7.0 of FrameMaker added import and export of XML to the product when running in structured mode. Many things changed as a result.

Name changes in the documentation

- The SGML API was renamed the Structure Import/Export API
- SGML applications were renamed structure applications
- The *SGML Application Developer's Guide* was renamed the *Structure Application Developer's Guide*
- The SGML application file is now called the structure application file—its filename has changed from `sgmlapps.fm` to `structapps.fm`

Naming changes in the FDK and Structure Import/Export API

In release 6.0 many functions, properties, and values included "SGML" in their names. Because the API now processes XML as well as SGML, these naming conventions were not strictly correct. Naming within the Structure Import/Export API changed occurences of "SGML" to "Structured".

For backward compatibility, all the old names are retained. However, these names are deprecated in preference to the new names. The documentation for the Structure Import/Export API shows the deprecated names for functions, but not for properties and values. For legacy code, you should not have to rewrite the code using new names. However, for new code you are encouraged to use the new naming.

For a list of new and changed FDK properties, see "New and changed FDK properties" on page 25.

Changes to link settings

To use the Structure Import/Export API in a client, you must link in the Xerces library for XML parsing, and the ICU library for processing text encodings. These libraries ship with the FrameMaker product—you must link to these same libararies. For information on the proper link settings, see the FDK platform guides.

Enhanced Text Encoding to Include UNICODE and ICU Libraries

FrameMaker now supports UNICODE text for import and export of XML. Upper-range double-byte characters can occur in XML element content and in XML tokens such as element GIs and attribute names.

FrameMaker ships with support for the following encodings, listed by their IANA names:

Big5	KSC_5601
EUC-CN	macintosh
EUC-JP	Shift_JIS
EUC-KR	US-ASCII
EUC-TW	UTF-16
GB2312	UTF-8
ISO-8859-1	windows-1252

These encodings are created in the ICU (International Components for Unicode) format, and stored as `.cnv` files. The supplied encodings are stored in the following locations:

- Windows: `$installdir\fminit\icu_data`
- Unix: `$installdir/fminit/icu_data`
- Macintosh: `$installdir:Modules:Icu_Data`

Users can add other ICU encodings to the FrameMaker installation—to do this they create ICU compliant mappings and save them as `.cnv` files, then store them in the icu_data directory. You can refer to Codings installed by a user the same as you refer to the encodings that are shipped with FrameMaker..

For more information about how FrameMaker handles UNICODE and ICU libraries, see Chapters 5 and 7 in the *Structure Application Developer's Guide*. For more information about ICU libraries, see http://www-124.ibm.com/icu/.

XMP Metadata Suppot

The eXtensible Metadata Platform (XMP) provides a common XML framework that standardizes the creation, processing, and interchange of document metadata across publishing workflows. For more informatin about XMP and access to the XMP SDK, see http://www.adobe.com/products/xmp/main.html.

For a FrameMaker document or book, XMP data is stored in the `FP_FileInfoPacket` property—you can use the FDK get and set the value of this `StringT`. For more information, see the discussions for XMP Metadata in the *FDK Programmer's Reference* for `FO_Document` and `FO_Book` objects.

WebDAV Support

The FrameMaker product includes support for WebDAV (World Wide Web Authoring and Versioning), which includes connecting to WebDAV servers, checking files in and out, and updating files. This support is provided by an FDK client named WebDAV. For information about calling this client from your own clients, see Appendix A of the *FDK Programmer's Guide*.

The FDK includes various properties that are used by the WebDAV client, but your clients should not use thes properties to manage WebDAV interactions. Instead, you should use the WebDAV client.

Changes to the FDK API

The following functions, constants, and properties in the FDK have new or changed behavior, syntax, or types.

Changed FDK functions

Opening, importing, and saving files

The following functions use or get property lists to script function behavior when performing file I/O with FrameMaker documents:

- `F_ApiGetImportDefaultParams()`
- `F_ApiGetOpenDefaultParams()`
- `F_ApiGetSaveDefaultParams()`

- F_ApiImport()
- F_ApiOpen()
- F_ApiSave()

These functions use the following new properties and values to manage file I/O with XML:

- FS_DisallowXml
- FS_FileIsXmlDoc
- FS_FileType
  - FV_SaveFmtBinary60
  - FV_SaveFmtXml
- FS_ImportAsType and FS_OpenAsType
  - FV_TYPE_XML
- FS_OpenNativeError and FA_errno
  - FV_CancelFileIsXml
  - FV_CancelFileXml
  - FV_ImportedXmlDoc

Notification constants

The following constants have been added for use with F_ApiNotification() and F_ApiNotify().

- FA_Note_PreOpenXML
- FA_Note_PostOpenXML

New and changed FDK properties

The following sections describe new or modified properties and property values for existing objects. Properties and values that have not changed from earlier releases are not listed here.

Session properties

- FP_ProductIsStructured
- FP_FM_StructureDir
- FP_FM_XmlDir

Book and document properties: PDF

- FP_FileExtensionOverride
- FP_FileInfoPacket
- FP_PDFJobOption
- FP_PDFOpenPage
- FP_PDFZoomType

  - FV_PDFZoomNone
  - FV_PDFZoomDefault
  - FV_PDFZoomPage
  - FV_PDFZoomWidth
  - FV_PDFZoomHeight
  - FV_PDFZoomMaxValue
- FP_PDFRegistrationMarks
  - FV_PDFRegistrationMarksNone
  - FV_PDFRegistrationMarksWestern
  - FV_PDFRegistrationMarksTombo
  - FV_PDFRegistrationMarksMax
- FP_PDFZoomFactor
- FP_PDFSeparateFiles
- FP_PDFPageWidth
- FP_PDFPageHeight
- FP_PDFPrintPageRange
- FP_PDFStartPage
- FP_PDFEndPage
- FP_PDFConvertCMYKtoRGB
- FP_PDFBookmarksOpenLevel
  - FV_PDFBookmarksOpenDefaultLevel
  - FV_PDFBookmarksOpenAllLevels
  - FV_PDFBookmarksOpenNoneLevel
- FP_PDFDistillerAbsent

Book and document properties: XML

- FP_XmlVersion
- FP_XmlEncoding
- FP_XmlStandAlone
  - FV_XML_STANDALONE_YES
  - FV_XML_STANDALONE_NO
  - FV_XML_STANDALONE_NA
- FP_XmlStyleSheet
- FP_XmlStyleSheetList
- FP_XmlUseBOM
  - FV_XML_USEBOM_YES
  - FV_XML_USEBOM_NO

  - FV_XML_USEBOM_NA
- FP_XmlWellFormed
  - FV_XML_WELLFORMED_YES
  - FV_XML_WELLFORMED_NO
  - FV_XML_WELLFORMED_NA
- FP_XmlFileEncoding
- FP_XmlDocType
- FP_XmlPublicId
- FP_XmlSystemId

Book and document properties: Header/Footer variables

- FP_SystemVar
  - FV_VAR_HEADER_FOOTER_5
  - FV_VAR_HEADER_FOOTER_6
  - FV_VAR_HEADER_FOOTER_7
  - FV_VAR_HEADER_FOOTER_8
  - FV_VAR_HEADER_FOOTER_9
  - FV_VAR_HEADER_FOOTER_10
  - FV_VAR_HEADER_FOOTER_11
  - FV_VAR_HEADER_FOOTER_12

Table properties

- FP_TblInLockedTi