Acrobat Forms API Reference

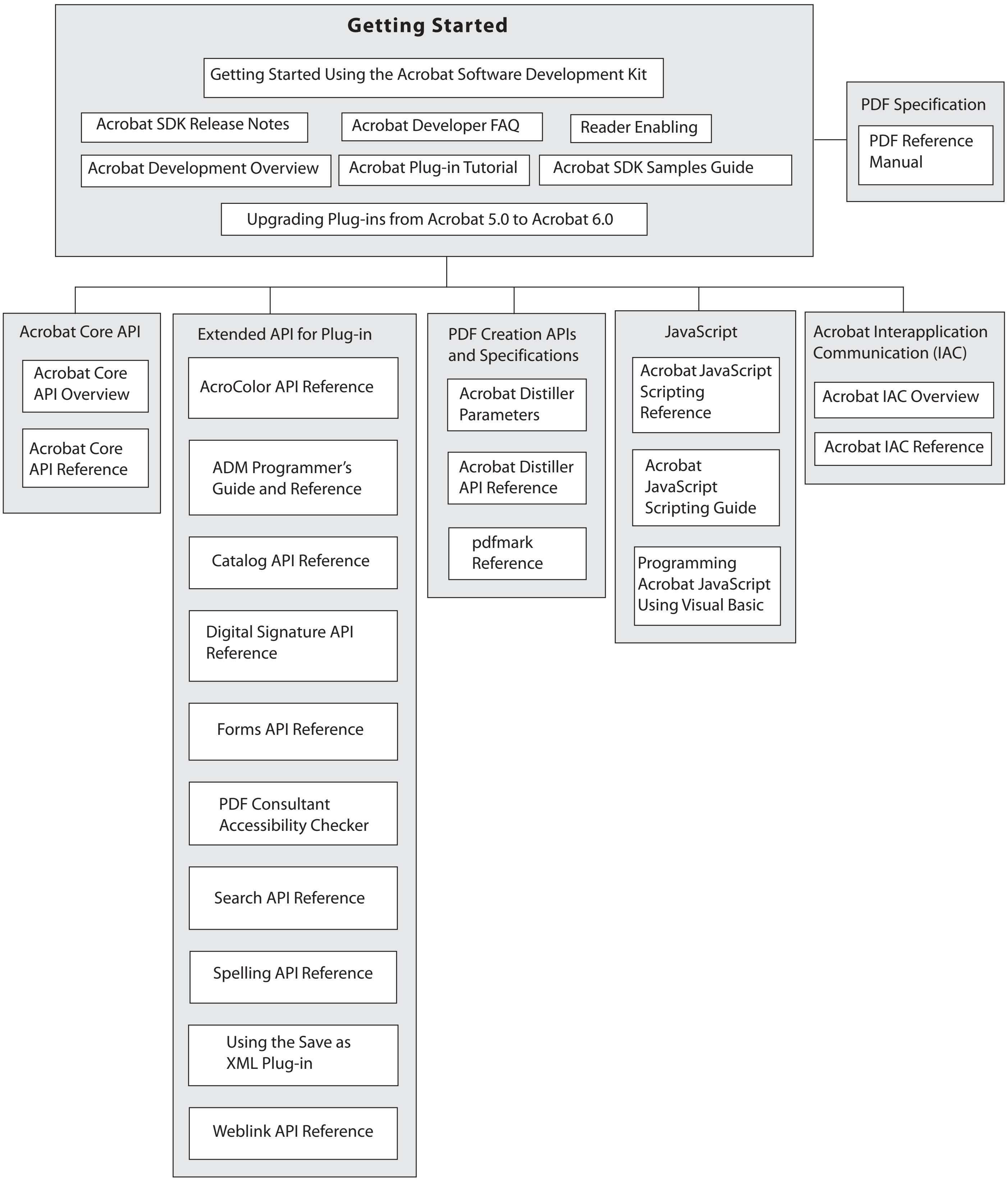
Technical Note #5181

Version : Acrobat 6.0

ADOBE SYSTEMS INCORPORATED
Corporate Headquarters
345 Park Avenue
San Jose, CA 95110-2704
(408) 536-6000
http://partners.adobe.com

May 2003

Acrobat SDK Documentation Roadmap


Getting Started
Getting Started Using the Acrobat Software Development Kit
Acrobat SDK Release Notes
Acrobat Developer FAQ
Reader Enabling
Acrobat Development Overview
Acrobat Plug-in Tutorial
Acrobat SDK Samples Guide
Upgrading Plug-ins from Acrobat 5.0 to Acrobat 6.0
PDF Specification
PDF Reference Manual
Acrobat Core API
Acrobat Core API Overview
Acrobat Core API Reference
Extended API for Plug-in
AcroColor API Reference
ADM Programmer's Guide and Reference
Catalog API Reference
Digital Signature API Reference
Forms API Reference
PDF Consultant Accessibility Checker
Search API Reference
Spelling API Reference
Using the Save as XML Plug-in
Weblink API Reference
PDF Creation APIs and Specifications
Acrobat Distiller Parameters
Acrobat Distiller API Reference
pdfmark Reference
JavaScript
Acrobat JavaScript Scripting Reference
Acrobat JavaScript Scripting Guide
Programming Acrobat JavaScript Using Visual Basic
Acrobat Interapplication Communication (IAC)
Acrobat IAC Overview
Acrobat IAC Reference

Contents

Preface

The Adobe® Acrobat® Forms plug-in allows a Portable Document Format (PDF) document to act as a form; that is, the Acrobat equivalent of a paper form with fields. It is now faster and easier to exchange information either in familiar paper or electronic forms converted to PDF files with Acrobat, or as dynamic interactive database templates.

NOTE: Forms as used here do not refer to `XObject` forms as defined in the *PDF Reference*.

The Forms plug-in for Acrobat (versions 4.0 and above) allows users to author form fields. For Acrobat Reader, the Forms plug-in does not allow form authoring, but allows users to fill in data and print Acrobat forms. The Reader Forms plug-in also does not allow users to save data to the local hard disk. Both Acrobat and Reader allow Web designers to send data from the form back to a Web server.

What Is In This Document

This document is a reference for developers who want to take advantage of the Forms API. It is divided into two sections:

- AcroForm API Reference describes the Forms API and explains how to use it. It includes the new Text-to-Speech APIs for Windows.
- AcroForm OLE Automation describes the OLE Automation methods exported by the Adobe Acrobat AcroForm plug-in.

If you received this technical note without obtaining the entire Acrobat Software Development Kit (SDK), you can get the complete SDK by visiting:

http://partners.adobe.com/asn/developer/acrosdk/main.html.Other Useful Documentation

Other Useful Documentation

You should be familiar with the Acrobat core API. The following technical notes provide this information.

Acrobat Core API Overview, technical note 5190,gives an overview of the objects and methods provided by the Acrobat core API.

Acrobat Core API Reference, technical note 5191,describes in detail the objects and methods provided by the Acrobat core API.

Acrobat Development Overview, technical note 5167, describes how to develop Acrobat plug-ins on the various platforms available.

PDF Reference, Fourth Edition, version 1.5. Provides a description of the PDF file format, as well as suggestions for producing efficient PDF files.

Conventions Used in This Book

The Acrobat documentation uses text styles according to the following conventions.

Font	Used for	Examples
monospaced	Paths and filenames	`C:\templates\mytmpl.fm`
	Code examples set off from plain text	These are variable declarations: `AVMenu commandMenu,helpMenu;`
monospaced bold	Code items within plain text	The **`GetExtensionID`** method ...
	Parameter names and literal values in reference documents	The enumeration terminates if **`proc`** returns **`false`**.
monospaced italic	Pseudocode	`ACCB1 void ACCB2 ExeProc(void) { `*`do something`*` }`
	Placeholders in code examples	`AFSimple_Calculate(`*`cFunction, cFields`*`)`
blue	Live links to Web pages	The Acrobat Solutions Network URL is: http://partners/adobe.com/asn/
	Live links to sections within this document	See Using the SDK.
	Live links to other Acrobat SDK documents	See the *Acrobat Core API Overview*.
	Live links to code items within this document	Test whether an **`ASAtom`** exists.
bold	PostScript language and PDF operators, keywords, dictionary key names	The **setpagedevice** operator
	User interface names	The **File** menu

Font	**Used for**	**Examples**
italic	Document titles that are not live links	*Acrobat Core API Overview*
	New terms	*User space* specifies coordinates for...
	PostScript variables	*filename* **deletefile**

AcroForm API Reference

Introduction

The Acrobat Forms plug-in exports its own Host Function Table (HFT), whose methods can be used by other plug-ins. To use the Acrobat Forms plug-in's HFT, a plug-in must:

- include the header file `FormsHFT.h` (which includes `AF_ExpT.h` and `AF_Sel.h`).
- import the HFT using `ASExtensionMgrGetHFT`. A convenient way to do this is to use the `Init_AcroFormHFT` macro defined in `FORMSHFT.H`.
  ```
  #define Init_AcroFormHFT
  ASExtensionMgrGetHFT(ASAtomFromString(AcroFormHFT_NAME),
  AcroFormHFT_LATEST_VERSION)
  ```
- assign the HFT returned by this call to a plug-in-defined global variable named `gAcroFormHFT`.

Data may be imported and exported into Acrobat Forms in Forms Data Format (FDF). FDF is used to submit form data to a server, as well as to receive the response and incorporate it into a form. FDF is based on PDF and uses the same syntax and set of basic object types as PDF. It also has the same file structure, except that the cross-reference table is optional. See the *PDF Reference (Version 1.5, Fourth Edition),* Section 8.6.6, for more information.

Contents

This reference contains the following:

- AcroForm Objects introduces AcroForm-specific API objects and briefly describes Core objects that are extensively used in the AcroForm API.
- AcroForm Methods describes in detail all AcroForm methods, including their parameters, return value, and related methods.
- AcroForm Callbacks describes in detail the callback functions.
- AcroForm Declarations describes in detail the data structures used by the Acrobat Forms plug-in.
- AcroForm Macros describes in detail the macros available for use by developers.
- Text-to-Speech API (Windows only) describes in detail all methods in the Text-to-Speech API.

Exceptions

All AcroForm methods may return an exception. Possible exceptions are:

Exception Name	Description
`gAFpdErrExportFdf`	Error during export of FDF document.
`gAFpdErrBadFdf`	Invalid FDF document.
`gAFfileErrSubmitFdf`	Error opening URL to submit this form.
`gAFpdErrReqdFld`	Required field found empty during export of FDF document.

AcroForm Objects

AcroForm Objects

The PDField object is introduced in the Forms API.

PDField

A PDField is an opaque object representing a field in an Acrobat form.

A PDF document that contains forms has an **AcroForm** entry in the document catalog dictionary, which contains an array of references to each of the root fields in the document.

The three most important properties of a field are its type, name, and value. Other properties specify the appearance of a field. Fields can be organized into a hierarchy, and other field properties associate it with its parent and children.

There is a field dictionary for every `PDField`. Acrobat uses annotations to represent a field's appearance and to manage user interactions. A `PDField` dictionary may also be an annotation, in which case its subtype is `Widget`. There is no ambiguity, because the keys of annotations and `PDField`s do not overlap.

`PDField`s are not created until needed. This saves memory and enhances performance. Suppose, for example, a PDF file represents a 100-page catalog, where a purchase order is on the last page. `PDField`s are not created for all the elements in the purchase order when the document is opened. When the user displays the purchase order, the annotation handler draws the field announcements and creates the `PDField`s for form fields.

When the annotation handler creates a `PDField`, it typically displays a page that contains the field.

Core Objects

These objects are defined in the Acrobat Core API and used by the Forms API. For further information about these objects, see the *Acrobat Core API Reference*.

ASAtom

A hashed token used in place of strings to optimize performance (it is much faster to compare **`ASAtoms`** than strings).

ASBool

A boolean value.

ASFile

An opaque representation of an open file.

ASPathName

A file system-specific named location for a particular type of device. Uses the **`ASFileSys`** structure pointers for callback. An **`ASPathName`** is specific to a given **`ASFileSys`**.

AVDoc

A view of a PDF document in a window. There is one **`AVDoc`** per displayed document. Unlike a **`PDDoc`**, an **`AVDoc`** has a window associated with it.

CosDoc

A Cos-level representation of an entire PDF file.

CosObj

A general object in a PDF file, which may be of any Cos object type.

PDAnnot

A structure that defines objects and methods used to manipulate annotations, as per the logical model defined in the PDF specification.

PDDoc

The underlying PDF representation of a document. There is a correspondence between a **`PDDoc`** and an **`ASFile`**; the **`PDDoc`** object is the hidden object behind every **`AVDoc.`** An **`ASFile`** may have zero or more underlying files, so a PDF file does not always correspond to a single disk file. For example, an **`ASFile`** may provide access to PDF data in a database.

Through **`PDDocs`**, your application can perform most of the **Edit -> Pages** menu items from Acrobat (delete, replace, and so on). You can create and delete thumbnails through this object. You can also set and retrieve document information fields through this object as well. The first page in a **`PDDoc`** is page 0.

PDPage

A single page in the PDF representation of a document. Just as PDF files are partially composed of their pages, **`PDDocs`** are composed of **`PDPages`**. A page contains a series of objects representing the objects drawn on the page (**`PDGraphic`**), a list of resources used in drawing the page, annotations (**`PDAnnot`**), an optional thumbnail image of the page, and the beads used in any articles that occur on the page. The first page in a **`PDDoc`** is page 0.

AcroForm Methods

AFExecuteThisScript

```
ASBool AFExecuteThisScript (PDDoc pdd, const char* cScript,
char** pRetValue);
```

Description

Executes a JavaScript script.

Parameters

`pdd`	The `PDDoc` in which the script is to be executed.
`cScript`	A string containing the text of the script to be executed. If Unicode, the string must begin with 0xFEFF and end with 2 null bytes. If this is not the case, it is assumed to be in the application's language encoding, as returned by `AVAppGetLanguageEncoding`.
`pRetValue`	To get a return value from the execution of the script, pass a non-`NULL` value for this parameter. If on return `*pRetVal` is non-`NULL`, the caller should dispose of the string by calling `ASFree`. If present, the value will be in host encoding. **NOTE:** The script sets this value by assigning it to `event.value`. See the *Acrobat JavaScript Object Specification* for more information.

Return Value

The JavaScript value of `event.rc`. This function pre-initializes it to `true`; a script may set it to `false` if desired.

Header File

`AF_Sel.h`

Example

```
AVDoc avDoc = AVAppGetActiveDoc();
PDDoc pdDoc = AVDocGetPDDoc (avDoc);
char * buf = "console.println(\"myName\");";
AFExecuteThisScript(pdDoc, buf, NULL);
```

AFImportAppearance

```
ASBool AFImportAppearance(CosDoc cd, CosObj *coIcon,
AVDoc avd, char *cTitle);
```

Description

Opens the dialog box that allows the user to select a PDF to use as the icon for a button.

Parameters

`cd`	The `CosDoc` that contains the appearance you are trying to import.
`coIcon`	If `AFImportAppearance` is successful, `coIcon` is a pointer to a `CosObj` that will contain the Cos representation of the appearance.
`avd`	The `AVDoc` that you want the window to be the child of. You can pass `NULL` if you do not have an `AVDoc`.
`cTitle`	The window title of the dialog box when it appears. You can pass `NULL` if you want the title of the dialog to be the same as it is when brought up through the field properties dialog.

Return Value

`true` if appearance was imported properly, `false` otherwise.

Header File

`AF_Sel.h`

Related Methods

None

Example

```
AVDoc avDoc = AVAppGetActiveDoc();
PDDoc pdDoc = AVDocGetPDDoc (avDoc);
char ReturnValue[128] = "";
char* pReturnValue = ReturnValue;
char* buf = "var rc = app.response({cQuestion: 'What is the company ?',
     cDefault: 'Adobe'} ); if(rc!=null) event.value=rc;";
ASBool bRc = AFExecuteThisScript (pdDoc, buf, &pReturnValue);
if(pReturnValue) {
     AVAlertNote(pReturnValue)
     ASfree(pReturnValue);
     }
```

AFLayoutBorder

```
void AFLayoutBorder (void* vlayout,
AFPDWidgetBorderRec border, PDColorValue pdcvBrdr,
PDColorValue pdcvBg, ASBool bDown);
```

Description

Draws a border into the layout context.

Parameters

`vlayout`	The layout of the annotation. Use `AFLayoutNew` to create a new layout before calling this method.
`border`	A pointer to a structure containing information about the appearance a border.
`pdcvBrdr`	A `PDColorValue` structure representing the color of the annotations border.
`pdcvBg`	A `PDColorValue` structure representing the color of the annotations background.
`bDown`	A boolean specifying whether the background should be drawn as it is drawn in forms while being "pressed" (clicked by the mouse) or not. If `true`, draws as if it is a field that is being pressed.

Return Value

None

Header File

`AF_Sel.h`

Related Methods

`AFLayoutNew`

Example

```
AFLayoutBorder (layout, &border, &pdcvBorder, &pdcvBackground, false);
```

AFLayoutCreateStream

```
CosObj AFLayoutCreateStream (void* vlayout);
```

Description

Creates a layout stream that can be used as an annotation appearance.

Parameters

`vlayout`	The layout of the annotation. Use `AFLayoutNew` to create a new layout before calling this method.

Return Value

A stream `CosObj`.

Header File

`AF_Sel.h`

Related Methods

AFLayoutNew
AFLayoutDelete

Example

```
// Write the whole thing to a CosStream.
CosObj coStream = AFLayoutCreateStream(layout);
AFLayoutDelete(layout);
```

AFLayoutDelete

```
void AFLayoutDelete (void* vlayout);
```

Description

Frees the layout context.

Parameters

`vlayout`	The layout of the annotation to remove.

Return Value

None

Header File

`AF_Sel.h`

Related Methods

`AFLayoutNew`
`AFLayoutCreateStream`

Example

```
CosObj coStream = AFLayoutCreateStream(layout);
AFLayoutDelete(layout);
```

AFLayoutNew

```
void* AFLayoutNew (ASFixedRectP frBbox,
PDRotate annotRotation, CosDoc cd);
```

Description

Create a new layout context for annotations. Use **PDAnnotGetRect** to get the annotation’s bounding box, then use this method to define new layout context.

Parameters

`frBbox`	The bounding box of the area for text and border data to flow into.
`annotRotation`	The rotation of the annotation.
`cd`	The `CosDoc`.

Return Value

A new layout.

Header File

`AF_Sel.h`

Related Methods

`AFLayoutBorder`
`AFLayoutText`
`AFLayoutCreateStream`

Example

```
PDAnnotGetRect (theAnnot, &frBBox);
void *layout = AFLayoutNew (&frBBox, PDPageGetRotate (pdPage),
theCosDoc);
```

AFLayoutText

```
void AFLayoutText (void* vlayout, ASBool bMultline,
ASBool bWrap, AFPDWidgetBorderRec border, TextAppearanceP ta,
char* cBytes);
```

Description

Sets the text layout for the annotation.

NOTE: Before calling this method, you should call `AFLayoutNew` to create a new layout, as well as `AFLayoutBorder`.

Parameters

`vlayout`	The layout of the annotation.
`bMultline`	If `true`, the text can use multiple lines in a text field. Otherwise, text is single line.
`bWrap`	If `true`, the text will wrap.
`border`	The border appearance that defines the width and line style of a border. The border of the annotation should be the same as in your call to `AFLayoutBorder`.
`ta`	A pointer to structure containing font, point size, and color information. You should initialize the structure with the `SetDefaultTextAppearanceP` macro which defaults to Helvetica.
`cBytes`	The text string for the layout.

Return Value

None

Exceptions

Raises an exception if the field is a radio box or button.

Header File

`AF_Sel.h`

Related Methods

`AFLayoutNew`
`AFLayoutBorder`

Example

```
PDAnnotGetBBox(theAnnot, &frBBox);
void *layout = AFLayoutNew(&frBBox, PDPageGetRotate(pdPage), theCosDoc);
AFLayoutBorder(layout, &border, &pdcvBorder, &pdcvBackground, false);
AFLayoutText(layout, true, true, &border, &textAppearance, "Welcome to
AcroForm");
```

AFPDDocEnumPDFields

```
void AFPDDocEnumPDFields (PDDoc doc, ASBool terminals,
ASBool no_repeat, AFPDFieldEnumProc proc, void *clientData);
```

Description

Enumerates all `PDField`s that exist in a `PDDoc`.

Parameters

`doc`	The `PDDoc` whose `PDField`s are enumerated.
`terminals`	If `true`, only `PDField`s without children are enumerated.
`no_repeat`	If `true`, then only one instance of a `PDField` for each name is enumerated.
`proc`	A user-supplied callback that is called for each `PDField`. The enumeration terminates if `proc` returns `false`.
`clientData`	Pointer to user-specified data that is passed to `proc` each time it is called.

Return Value

None

Header File

`AF_Sel.h`

Example

```
AVDoc currentAVDoc;
PDDoc currentPDDoc;
AFPDFieldEnumProc enumProc;

currentAVDoc = AVAppGetActiveDoc ();
currentPDDoc = AVDocGetPDDoc (currentAVDoc);

/* Set up enumeration */
enumProc = ASCallbackCreateProto (AFPDFieldEnumProc, &FieldEnumProc);
AFPDDocEnumPDFields (currentPDDoc, false, false, enumProc, NULL);
```

AFPDDocGetPDFieldFromName

```
PDField AFPDDocGetPDFieldFromName (PDDoc doc, char* name);
```

Description

Retrieves a **PDField** with a given name from a **PDDoc**. (If multiple fields have the same name, a change to any of them affects all of them.)

Parameters

doc	The **PDDoc** containing the field.
name	Name of the field to retrieve.

Return Value

The **PDField** specified by **name**. Returns **NULL** if there is no **PDField** with the given name in the **PDDoc**.

Header File

AF_Sel.h

Related Methods

AFPDFieldGetName

AFPDDocLoadPDFields

```
void AFPDDocLoadPDFields (PDDoc doc);
```

Description

Makes sure every **`PDField`** in the given **`PDDoc`** exists.

NOTE: It is no longer necessary to call this method. It still exists for backwards compatibility, but its purpose is now automatically taken care of internally.

Parameters

`doc`	The **`PDDoc`** for the form whose **`PDField`**s are loaded.

Return Value

None

Header File

`AF_Sel.h`

AFPDFieldFromCosObj

```
PDField AFPDFieldFromCosObj (CosObj dict);
```

Description

Retrieves the **`PDField`** object for which a Cos object is the dictionary.

Parameters

`dict`	The Cos object for which to retrieve the corresponding **`PDField`** object.

Return Value

The **`PDField`** corresponding to **`dict`**. Returns **`NULL`** if the Cos object is not a **`PDField`**.

Header File

`AF_Sel.h`

Related Methods

`AFPDFieldGetCosObj`

Example

```
PDAnnot anAnnotation;
CosObj annotCosObj;
PDField annotField;

/* Get PDField for Cos object */
annotCosObj = PDAnnotGetCosObj (anAnnotation);
annotField = AFPDFieldFromCosObj (annotCosObj);
```

AFPDFieldGetCosObj

```
CosObj AFPDFieldGetCosObj (PDField fldP);
```

Description

Retrieves the Cos object which is the field object of a **`PDField`** object.

Parameters

`fldP`	The **`PDField`** object for which to retrieve the corresponding Cos object.

Return Value

Cos object corresponding to the **`PDField`** object **`fldP`**.

Header File

`AF_Sel.h`

Related Methods

`AFPDFieldGetCosObj`
`AFPDFieldGetDefaultTextAppearance`
`AFPDFieldGetName`
`AFPDFieldGetValue`

Example

```
PDField fldP;
CosObj fieldCosObject;

if (AFPDFieldIsValid (fldP)) {
/* Get Cos Object for field */
fieldCosObject = AFPDFieldGetCosObj (fldP);
```

AFPDFieldGetDefaultTextAppearance

```
void AFPDFieldGetDefaultTextAppearance (PDField fldP,
TextAppearanceP aP);
```

Description

Gets the default text appearance of a field. Use this method to get the font, size, color, and so forth—values that were set through the field properties dialog box.

Parameters

`fldP`	The **`PDField`** object for which to retrieve the text appearance.
`aP`	A pointer to a structure describing the text appearance. Returns text matrix, quadding, text color, and so forth.

Return Value

None

Header File

`AF_Sel.h`

Related Methods

`AFPDWidgetSetAreaColors`

AFPDFieldGetFlags

```
ASUns32 AFPDFieldGetFlags (PDField fldP, AF_Flags_t flagType);
```

Description

Retrieves the flags of a **PDField** for a given flag type.

Parameters

fldP	The **PDField** whose flags are obtained.
flagType	Type of flags to obtain.

Return Value

The value of flags of **flagType**.

Header File

AF_Sel.h

Related Methods

AFPDFieldGetName
AFPDFieldGetValue
AFPDFieldSetFlags

AFPDFieldGetName

```
char* AFPDFieldGetName (PDField fldP);
```

Description

Gets the name of a **PDField** object.

Parameters

fldP	The **PDField** whose name is retrieved.

Return Value

A C program **NULL**-terminated string.

NOTE: Do not modify or free this string. To use this string indefinitely, copy it to a private buffer.

Header File

AF_Sel.h

Related Methods

AFPDFieldGetDefaultTextAppearance
AFPDFieldGetValue

Example

```
PDField fldP;
char *fieldName;

fieldName = AFPDFieldGetName (fldP);
```

AFPDFieldGetValue

```
CosObj AFPDFieldGetValue (PDField fldP);
```

Description

Retrieves the value from a **`PDField`**. The value is stored in the field entry with the **V** key.

Values can be inherited. If a **`PDField`** Cos object does not have a **V** key, its value is inherited from a parent's **V** key (the parent must have the same name as the **`PDField`**). For example, radio buttons inherit values this way.

NOTE: Retrieving the value of a radio button or combo box requires Cos-level programming. See the *PDF Reference* for details.

Parameters

`fldP`	The **`PDField`** whose value is retrieved.

Return Value

The value of the field as a **`CosObj`**. If the field has no value, a NULL **`CosObj`** is returned.

Header File

`AF_Sel.h`

Related Methods

`AFPDFieldGetCosObj`
`AFPDFieldGetDefaultTextAppearance`
`AFPDFieldGetName`
`AFPDFieldSetValue`

Example

This example shows how to obtain the string value of an encoded string from the **`CosObj`**.

```
PDField fldP;
CosObj fieldCosValue = AFPDFieldGetValue(fldP);
char *fieldValue = CosCopyStringValue(fieldCosValue, NULL);
ASText asText = ASTextFromPDText(fieldValue);
char *realField = ASTextGetEncodedCopy(asText,
     (ASHostEncoding)PDGetHostEncoding());
```

AFPDFieldIsAnnot

```
ASBool AFPDFieldIsAnnot (PDField fldP);
```

Description

Determines whether or not a `PDField` is an annotation (`PDAnnot`), that is, whether the field dictionary is also an annotation. If this is the case, the value of **Subtype** is **Widget**.

Parameters

`fldP`	The field in question.

Return Value

`true` if the `PDField` is an Acrobat annotation, `false` otherwise.

Header File

`CosField.h`

Related Methods

`AFPDFieldIsTerminal`
`AFPDFieldIsValid`

Example

```
PDField fldP;

if (AFPDFieldIsAnnot (fldP)) {
/* Process annotation */
```

AFPDFieldIsTerminal

```
ASBool AFPDFieldIsTerminal (PDField fldP);
```

Description

Determines whether a field is terminal: the field has no children or it has the same name as its children.

Parameters

`fldP`	The `PDField` to test.

Return Value

If `true`, `fldP` is valid and all its children (if any) have the same name as `fldP`. Otherwise all have unique names.

Header File

`AF_Sel.h`

Related Methods

`AFPDFieldIsAnnot`

`AFPDFieldIsValid`

Example

```
PDField fldP;

if (AFPDFieldIsTerminal (fldP)) {
/* Process field */
```

AFPDFieldIsValid

```
ASBool AFPDFieldIsValid (PDField fldP);
```

Description

Determines whether a field is valid or not.

NOTE: This method is intended only to ensure that the field has not been deleted, not to ensure that all necessary information is present and the field has valid values.

Parameters

`fldP`	The field to test.

Return Value

`true` if `fldP` is a valid `PDField`; `false` otherwise.

Header File

`AF_Sel.h`

Related Methods

`AFPDFieldIsAnnot`

`AFPDFieldIsTerminal`

Example

```
PDField fldP;

if (AFPDFieldIsValid (fldP)) {
/* Process field */
}
```

AFPDFieldReset

```
void AFPDFieldReset (PDField fldP);
```

Description

Sets a **`PDField`**'s value to its default state. This is the value associated with the **DV** key for the field. If there is no **DV** key for the field, set the field's value to the **`NULL`** Cos object; or, if the field has options, the default **DV** is the first element of the **Opt** array.

A **DV** entry's value can be inherited from a parent (just like **V**).

The **`PDField`**'s value is set only if it is terminal; see **`AFPDFieldIsTerminal`**.

Parameters

`fldP`	The **`PDField`** to reset.

Return Value

None

Header File

`AF_Sel.h`

Related Methods

`ResetForm`

Example

```
AVDoc currentAVDoc;
PDField annotField;

/* Remove current field selection, so selected field is operated on */
currentAVDoc = AVAppGetActiveDoc ();
AFPDFieldReset (annotField);
```

AFPDFieldSetDefaultTextAppearance

```
void AFPDFieldSetDefaultTextAppearance(PDField fldP,
TextAppearanceP aP);
```

Description

Sets the default text appearance of a field. Use this method to set the font, size, color, and so forth.

Parameters

`fldP`	The `PDField` object for which to set the text appearance.
`aP`	A pointer to a structure describing the text appearance to set.

Return Value

None

Header File

`AF_Sel.h`

Related Methods

`AFPDWidgetSetAreaColors`

AFPDFieldSetFlags

```
void AFPDFieldSetFlags (PDField fldP, AF_Flags_t flagType,
AFPDFieldFlags_t flags);
```

Description

Sets the flags of type `flagType` for a `PDField`.

The flags of a field's children (if any) are also set.

Parameters

`fldP`	The `PDField` for which to set flags and children's flags.
`flagType`	The type of the flags to set. If `flagType` is `Flags_Annot` and the `PDField` is not an annotation, no flags are changed.
`flags`	The value of the flags.

Return Value

None

Header File

`AF_Sel.h`

Related Methods

`AFPDFieldGetDefaultTextAppearance`
`AFPDFieldSetOptions`
`AFPDFieldSetValue`

AFPDFieldSetOptions

```
RetCode AFPDFieldSetOptions (PDField fldP, CosObj array);
```

Description

Sets the options entry for a field. The options entry has the key **Opt** and represents a list of options for a choice field.

If the field is not valid, remove the options entry.

Parameters

`fldP`	The `PDField` whose options are set.
`array`	The value to set in the options entry.

Return Value

`Good` if the operation succeeded, `Bad` otherwise.

Header File

`AF_Sel.h`

Related Methods

`AFPDFieldSetFlags`
`AFPDFieldSetValue`

Example

```
const char cOption1[] = "Blue";
const char cOption2[] = "Red";
CosObj coOpts = CosNewArray(cd, false, 2);
CosObj coOpt = CosNewString(cd, false, cOption1, strlen(cOption1));
CosArrayPut(coOpts, 0, coOpt);
coOpt = CosNewString(cd, false, cOption2, strlen(cOption2));
CosArrayPut(coOpts, 1, coOpt);
AFPDFieldSetOptions(coOpts);
```

AFPDFieldSetValue

```
void AFPDFieldSetValue (PDField fldP, CosObj val);
```

Description

Sets the value for a `PDField`. The value is stored in the field entry with the **V** key. Updates the display of the field and its namesakes; that is, fields with the same fully qualified names, if any. Changing the field dictionary's value for the **V** entry directly does not change the appearance of the field, which is represented by the **AP** key.

NOTE: Setting the value of a radio button or combo box requires Cos-level programming. See the *PDF Reference* for details.

Parameters

`fldP`	The field whose value is set.
`val`	The field's new value. If it is different than the previous value, all previous instances of value keys (**V**) in the field's dictionary and those of its namesakes (fields with the same fully qualified name) are removed.

Return Value

None

Header File

`AF_Sel.h`

Related Methods

`AFPDFieldGetValue`
`AFPDFieldSetFlags`
`AFPDFieldSetOptions`

Example

```
PDField fldP;
CosDoc currentCosDoc;
CosObj cosObject, newString;
char *fieldName;
ASInt32 fieldNameLength;

cosObject = AFPDFieldGetCosObj (fldP);
currentCosDoc = CosObjGetDoc (cosObject);
fieldName = AFPDFieldGetName (fldP);
fieldNameLength = strlen (fieldName);
newString = CosNewString (currentCosDoc, false, fieldName,
fieldNameLength);
AFPDFieldSetValue (fldP, newString);
```

AFPDFormFromPage

```
CosObj AFPDFormFromPage (CosDoc cd, PDPage pdp);
```

Description

Creates an `XObject` form from a PDF page. A form XObject is a content stream that can be treated as a single graphics object. Use this method for importing PDF graphics into documents.

Parameters

`cd`	The `CosDoc` that the `XObject` will be created in.
`pdp`	The `PDPage` that you want to create the `XObject` from.

Return Value

A Cos object pointing to the `XObject` on the PDF page.

Header File

`AF_Sel.h`

Related Methods

None

AFPDWidgetGetAreaColors

```
void AFPDWidgetGetAreaColors(PDAnnot pdan,
PDColorValue borderP, PDColorValue bkgndP);
```

Description

Gets the border and background colors of an annotation.

Parameters

`pdan`	The annotation.
`borderP`	*(Filled by the method)* A pointer to a structure representing the border color of the annotation.
`bkgndP`	*(Filled by the method)* A pointer to a structure representing the background color of the annotation.

Return Value

None

Header File

`AF_Sel.h`

Related Methods

`AFPDWidgetSetAreaColors`

AFPDWidgetGetBorder

```
ASBool AFPDWidgetGetBorder(PDAnnot pdan,
AFPDWidgetBorderRec pdwb);
```

Description

Gets the border of an annotation.

Parameters

`pdan`	The annotation.
`pdwb`	*(Filled by the method)* A pointer to a structure describing the form field appearance definitions for the outside border of an annotation.

Return Value

`true` if successful getting border, `false` otherwise.

Header File

`AF_Sel.h`

Related Methods

`AFPDWidgetSetAreaColors`

AFPDWidgetGetRotation

```
ASAtom AFPDWidgetGetRotation (PDAnnot pdan);
```

Description

Gets the rotation of the annotation.

Parameters

`pdan`	The annotation widget.

Return Value

Returns a `ASAtom` object representing the annotation's rotation with respect to the page.

Header File

`AF_Sel.h`

Related Methods

None

AFPDWidgetSetAreaColors

```
void AFPDWidgetSetAreaColors(PDAnnot pdan,
PDColorValue borderP, PDColorValue bkgndP);
```

Description

Sets the border and background color of the annotation.

Parameters

`pdan`	The annotation.
`borderP`	A pointer to a structure representing the border color of the annotation.
`bkgndP`	A pointer to a structure representing the background color of the annotation.

Return Value

None

Header File

`AF_Sel.h`

Related Methods

`AFPDWidgetGetAreaColors`

AFPDWidgetSetBorder

```
void AFPDWidgetSetBorder(PDAnnot pdan,
AFPDWidgetBorderRec pdwb);
```

Description

Sets the border of an annotation.

Parameters

`pdan`	The annotation whose change border appearance will be set.
`pdwb`	A pointer to a structure describing the form field appearance definitions for the outside border of an annotation. Possible border types are solid, dashed, beveled, inset, underline.

Return Value

None

Header File

`AF_Sel.h`

Related Methods

`AFPDWidgetSetAreaColors`

AssembleFormAndImportFDF

```
PDDoc AssembleFormAndImportFDF (PDDoc pdCurrForm,
CosDoc cdFDF, ASBool bAddToCurr);
```

Description

Constructs an Acrobat form from templates and imports an FDF file.

Parameters

`pdCurrform`	The current form being viewed, if any, at the time **`cdFDF`** is being imported. This parameter can be **`NULL`**; if not **`NULL`**, then **`cdFDF`** can refer to templates in the current form, by omitting the **F** key in the **TRef** dictionary. Even if the **F** key is not **`NULL`**, it can be a relative path (as opposed to an absolute path), as long as **`pdCurrForm`** is not **`NULL`**.
`cdFDF`	The FDF file being imported.
`bAddToCurr`	If **`true`** (and **`pdCurrForm`** is not **`NULL`**), then instead of creating a new form, the templates spawn pages that are appended at the end of **`pdCurrForm`** (and the function returns **`pdCurrForm`**).

Return Value

The **`PDDoc`** for the newly-created form (or **`pdCurrForm`**, if **`bAddToCurr`** is **`true`**).

Header File

`AF_Sel.h`

Related Methods

`ExportAsFDF`
`ExportAsFDFEx`
`ExportAsFDFWithParams`
`ImportAnFDF`

Example

```
CosObj dictFDF = CosDictGet(CosDocGetRoot(cdFDF), FDF_K);
if (CosObjGetType(CosDictGet(dictFDF, Pages_K)) == CosArray)
  AssembleFormAndImportFDF(0/*no current doc is open*/, cdFDF, false);
```

ExportAsFDF

```
CosDoc ExportAsFDF (PDDoc pdForm, CosObj rgIncExcFlds,
ASBool bIncl, ASBool bEmpty, ASBool bMenu, ASBool bLoadFields,
ASPathName fdfPath);
```

Description

Exports form data to a **CosDoc**, which can be written to an FDF file. See the *PDF Reference* for a description of this format.

To create an FDF file from this CosDoc, call the Core API method **CosDocSaveToFile**.

NOTE: **ExportAsFDFWithParams** provides the same functionality with additional options.

Parameters

pdForm	The **PDDoc** for the form for which we want to export the data.
rgIncExcFlds	If **rgIncExcFlds** is **CosNewNull**, then all fields are exported, and **bIncl** is ignored. If **rgIncExcFlds** is a **CosArray**, then the array elements may be: a) names of fields (the names may be of non-terminal fields, which is a fast and easy way to cause all their children to be included in the FDF file). b) arrays, whose first element is a field name, and the rest of the elements are field dictionary key names whose values should be exported in the FDF file. For example: **[(My listbox) /AP /Opt]**. This variety of **rgIncExcFlds** array element can only be used if **bIncl** is **true** (see below). c) If **rgIncExcFlds** contains a single element, which is itself an array as described in b) above, and its first element, which corresponds to the field name, is **NULL**, then the FDF file will include the requested field properties of all fields. For example: **[null /F /Ff]**.
bIncl	If **true**, **rgIncExcFlds** is an array of the fields to export. Otherwise, **rgIncExcFlds** is an array of the fields to *exclude* from exporting.
bEmpty	If **true**, all fields selected per the above criteria are exported. If **false**, exclude fields that have no value.
bMenu	If **true**, suppresses saving text fields that have the "password" flag set, and does not force filling-in required fields when creating an FDF file.

`bLoadFields`	Not used.
`fdfPath`	The path where the FDF file will be saved (by the client of **`ExportAsFDF`**) after it is produced. You need this in order to create an FDF file with an **F** key that gives the relative path to the form, from the location where the FDF file will be saved. Pass **`NULL`** if an absolute pathname is desired.

Return Value

The FDF **`CosDoc`** containing the exported data.

Exceptions

Raises **`gAFpdErrExportFdf`** if it cannot export field data. The viewer may raise other exceptions.

Header File

`AF_Sel.h`

Related Methods

`ExportAsFDFEx`
`ExportAsFDFWithParams`
`ImportAnFDF`
`AssembleFormAndImportFDF`

Example

```
/* Export all fields to CosDoc */
cosDoc = ExportAsFDF (currentPDDoc, rgFld, true, true, false, false,
NULL);
```

ExportAsFDFEx

```
CosDoc ExportAsFDFEx (PDDoc pdForm, CosObj rgIncExcFlds,
ASBool bIncl, ASBool bEmpty, ASBool bMenu, ASBool bLoadFields,
ASPathName fdfPath, const char* submitBtnName);
```

Description

Exports form data to a **CosDoc**, which can be written to an FDF file. See the *PDF Reference* for a description of this format.

To create an FDF file from this CosDoc, call the Acrobat viewer plug-in API method **CosDocSaveToFile**.

NOTE: This method is the same as **ExportAsFDF**, with the exception of the additional parameter **submitBtnName**. **ExportAsFDFWithParams** provides the same functionality with additional options.

Parameters

pdForm	A **PDDoc** for the form whose data is exported.
rgIncExcFlds	If **rgIncExcFlds** is **CosNewNull**, then all fields are exported, and **bIncl** is ignored. If **rgIncExcFlds** is a **CosArray**, then the array elements may be: a) names of fields (the names may be of non-terminal fields, which is a fast and easy way to cause all their children to be included in the FDF file). b) arrays, whose first element is a field name, and the rest of the elements are field dictionary key names whose values should be exported in the FDF file. For example: **[(My listbox) /AP /Opt]**. This variety of **rgIncExcFlds** array element can only be used if **bIncl** is **true** (see below). c) If **rgIncExcFlds** contains a single element, which is itself an array as described in b) above, and its first element, which corresponds to the field name, is **NULL**, then the FDF file will include the requested field properties of all fields. For example: **[null /F /Ff]**.
bIncl	If **true**, **rgIncExcFlds** is an array of the fields to export. Otherwise, **rgIncExcFlds** is an array of the fields to *exclude* from exporting.
bEmpty	If **true**, all fields selected per the above criteria are exported. If **false**, exclude fields that have no value.

bMenu	If **true**, suppresses saving text fields that have the "password" flag set, and doesn't force filling-in required fields when creating an FDF file.
bLoadFields	Not used.
fdfPath	The path where the FDF file will be saved after it is produced. You need this in order to create an FDF file with an **F** key that gives the relative path to the form, from the location where the FDF file will be saved. Pass **NULL** if an absolute pathname is desired.
submitBtnName	A **NULL**-terminated string containing the name of the button used to submit. If the value passed is not **NULL**, then the FDF file will include a field dictionary corresponding to the submit button, which will only contain one key, namely **T**. NOTE: This dictionary is no different than the one you get when an AcroForm has an empty text field (that is, no value), and parameter **bEmpty** is true.

Return Value

The FDF **CosDoc** containing the exported data.

Exceptions

Raises **gAFpdErrExportFdf** if it cannot export field data. The viewer may raise other exceptions.

Header File

AF_Sel.h

Related Methods

ExportAsFDF
ExportAsFDFWithParams
ImportAnFDF
AssembleFormAndImportFDF

Example

```
/* Export all fields to CosDoc */
cosDoc = ExportAsFDFEx (currentPDDoc, rgFld, true, true, false, false,
NULL, NULL);
```

ExportAsFDFWithParams

```
CosDoc ExportAsFDFWithParams (ExportAsFDFParamsRec params);
```

Description

Exports form data to a **CosDoc**, which can be written to an FDF file. See the *PDF Reference* for a description of this format.

Parameters

`params`	An **ExportAsFDFParamsRec** structure.

Return Value

The FDF **CosDoc** containing the exported data.

NOTE: Call **CosDocSaveToFile** to create an FDF file from this CosDoc.

Exceptions

Raises **gAFpdErrExportFdf** if cannot export field data. The viewer may raise other exceptions.

Header File

`AF_Sel.h`

Related Methods

ExportAsFDF
ExportAsFDFEx
ImportAnFDF
AssembleFormAndImportFDF

ExportAsHtml

```
void ExportAsHtml (PDDoc formPD, CosObj rgIncExcFlds,
ASBool bIncl, ASBool bEmpty, ASFile Hfile);
```

Description

Exports data from a form to a file in HTML format.

Parameters

`formPD`	The `PDDoc` for the form whose data is exported.
`rgIncExcFlds`	If `rgIncExcFlds` is `CosNewNull`, then all fields are exported, and `bIncl` is ignored. If it is a `CosArray`, then the array elements must be indirect references to field dictionaries.
`bIncl`	If `true`, `rgIncExcFlds` is an array of the fields to export. Otherwise, `rgIncExcFlds` is an array of the fields to *exclude* from exporting.
`bEmpty`	If `true`, all fields selected per the above criteria are exported. If `false`, exclude fields that have no value.
`Hfile`	File to which the HTML data is written.

Return Value

None

Header File

`AF_Sel.h`

Related Methods

`ExportAsHtmlEx`

ExportAsHtmlEx

```
void ExportAsHtmlEx (PDDoc pdForm, CosObj rgIncExcFlds,
ASBool bIncl, ASBool bEmpty, ASFile Hfile,
const char* submitBtnName);
```

Description

Exports data from a form to a file in HTML format.

NOTE: This method is the same as `ExportAsHtml`, with the exception of the additional parameter `submitBtnName`.

Parameters

`pdForm`	The `PDDoc` for the form whose data is exported.
`rgIncExcFlds`	If `rgIncExcFlds` is `CosNewNull`, then all fields are exported, and `bIncl` is ignored. If it is a `CosArray`, then the array elements must be names of fields. The names may be of non-terminal fields, which is a fast and easy way to cause all their children to be exported.
`bIncl`	If `true`, `rgIncExcFlds` is an array of the fields to export. Otherwise, `rgIncExcFlds` is an array of the fields to *exclude* from exporting.
`bEmpty`	If `true`, all fields selected per the above criteria are exported. If `false`, exclude fields that have no value.
`Hfile`	File to which the HTML data is written.
`submitBtnName`	A `NULL`-terminated string containing the name of the button used to submit. If the value passed is not `NULL`, then include `"...&submitBtnName=&..."` as part of the generated `x-www-form-urlencoded` output. **NOTE:** This type of output is the same one you get when an AcroForm has an empty text field (that is, no Value), and parameter `bEmpty` is true.

Return Value

None

Header File

`AF_Sel.h`

Related Methods

`ExportAsHtml`
`ImportAnFDF`

ImportAnFDF

```
ASBool ImportAnFDF (PDDoc pdForm, CosDoc cdFDF);
```

Description

Imports data from an FDF file into a `PDDoc`'s form. See the *PDF Reference* for a description of this format.

Parameters

`pdForm`	The `PDDoc` for the form into which we want to import the data.
`cdFDF`	The `CosDoc` for the FDF file containing the data.

Return Value

`true` if the fields in the FDF file only contained values or flags (that is, **V**, **F**, **Ff**, **ClrF**, **ClrFf**, **SetF**, **SetFf**), `false` if any field contained other attributes, such as appearances (that is, **AP**), actions (that is, **A**), and so forth.

Exceptions

Raises `gAFpdErrBadFdf` if the FDF file is invalid.

Raises an exception if memory cannot be allocated for FDF file data.

Header File

`AF_Sel.h`

Related Methods

`AssembleFormAndImportFDF`
`ExportAsFDF`
`ExportAsFDFEx`
`ExportAsFDFWithParams`

IsPDDocAcroForm

```
ASBool IsPDDocAcroForm (PDDoc doc);
```

Description

Indicates whether a **PDDoc** contains an Acrobat form.

Parameters

doc	The **PDDoc** to test.

Return Value

true if **doc** contains a form, **false** otherwise.

Header File

AcroForm.h

Related Methods

AFPDFieldIsValid

Example

```
/* Return true if active doc has form */
if ((avDoc = AVAppGetActiveDoc ()) != NULL)
return IsPDDocAcroForm (AVDocGetPDDoc (avDoc));
```

ResetForm

```
void ResetForm (PDDoc formPD, CosObj rgIncExcFlds,
ASBool bIncl);
```

Description

Resets the indicated fields of a `PDDoc`'s form to their default values.

A `PDField`'s value is reset only if it is terminal; see `AFPDFieldIsTerminal`.

Parameters

`formPD`	The `PDDoc` for the form whose fields are reset.
`rgIncExcFlds`	If `rgIncExcFlds` is `CosNewNull`, then all fields are reset, and `bIncl` is ignored. If it is a `CosArray`, then the array elements must be names of fields. The names may be of non-terminal fields, which is a fast and easy way to cause all their children to be reset.
`bIncl`	If `true`, `rgIncExcFlds` is an array of the fields to reset. Otherwise, `rgIncExcFlds` is an array of the fields to *exclude* from resetting.

Return Value

None

Header File

`AF_Sel.h`

Related Methods

`AFPDFieldSetDefaultTextAppearance`

Example

```
AVDoc currentAVDoc;
PDDoc currentPDDoc;
CosObj rgFld = CosNewNull ();

ResetForm (currentPDDoc, rgFld, (ASBool)true);
```

AcroForm Callbacks

AFPDFieldEnumProc

```
ACCB1 ASBool ACCB2 AFPDFieldEnumProc (PDField fldP,
void *clientData);
```

Description

Callback used by `AFPDDocEnumPDFields`. It is called once for each `PDField` in a form.

Parameters

`fldP`	The `PDField` currently being enumerated.
`clientData`	User-supplied data passed in the call to `AFPDDocEnumPDFields`.

Return Value

`true` to continue the enumeration, `false` to halt enumeration.

Header File

`AF_ExpT.h`

Related Methods

`AFPDDocEnumPDFields`

AcroForm Declarations

This chapter describes the data types and declarations used in the AcroForms API.

- AcroForm Data Types
- Core Data Types

AcroForm Data Types

The following are data type and values defined in the AcroForms API.

AF_Flags_t

```
typedef enum {
    Flags_Annot,
    Flags_Field,
    SetFlagsAnnot,
    ClrFlagsAnnot,
    SetFlagsField,
    ClrFlagsField,
/* new in Acrobat 6.0 */
    FlagsIgnore
} AF_Flags_t;
```

Description

The type of flag to be set in **`AFPDFieldSetFlags`**.

If **`Flags_Annot`**, **`SetFlagsAnnot`**, or **`ClrFlagsAnnot`** are specified and the **`PDField`** is not an annotation, no flags are changed. See the *PDF Reference* for more information on flags used for annotations.

Values

`Flags_Annot`	Sets the flags defined in the **F** key of the annotation to the values specified in the **`flags`** parameter of **`AFPDFieldSetFlags`**.
`Flags_Field`	Sets the flags defined in the **Ff** key of the form field to the values specified in the **`flags`** parameter of **`AFPDFieldSetFlags`**.

`SetFlagsAnnot`	Sets the bits in the flags to those that are 1s in the **`flags`** parameter of **`AFPDFieldSetFlags`**.
`ClrFlagsAnnot`	Clears the bits in the flags to those that are 1s in the **`flags`** parameter of **`AFPDFieldSetFlags`**.
`SetFlagsField`	Sets the bits in the flags to those that are 1s in the **`flags`** parameter of **`AFPDFieldSetFlags`**.
`ClrFlagsField`	Clears the bits in the flags to those that are 1s in the **`flags`** parameter of **`AFPDFieldSetFlags`**.
`FlagsIgnore`	Does not change any flags for the field or annotation.

Header File

`AF_ExpT.h`

Related Methods

`AFPDFieldSetFlags`

AFPDFieldBorder

```
typedef AFPDWidgetBorderRec AFPDFieldBorder;
```

Description

Field Border data type.

Header File

`AF_ExpT.h`

Related Methods

`AFLayoutBorder`
`AFLayoutText`
`AFPDWidgetGetBorder`
`AFPDWidgetSetBorder`

AFPDFieldFlags_t

```
typedef ASUns32 AFPDFieldFlags_t;
```

Description

Used for the value of flags set by `AFPDFieldSetFlags`.

Header File

`AF_ExpT.h`

Related Methods

`AFPDFieldSetFlags`

AFPDWidgetBorder

AFPDWidgetBorderRec

```
typedef struct _AFPDWidgetBorderRec {
    AFPDWidgetBorderStyle nStyle;
    ASInt32 nWidth;
} AFPDWidgetBorderRec; *AFPDWidgetBorder
```

Description

Form field appearance definitions for the outside border of an annotation.

Members

`nStyle`	Style of the border. Possible types are solid, dashed, beveled, inset, underline. See `AFPDWidgetBorderStyle`.
`nWidth`	Width of the border.

Header File

`AF_ExpT.h`

Related Methods

`AFLayoutBorder`
`AFLayoutText`
`AFPDWidgetGetBorder`
`AFPDWidgetSetBorder`

AFPDWidgetBorderStyle

```
typedef enum {
    AFPDWBSolid,
    AFPDWBDashed,
    AFPDWBBeveled,
    AFPDWBInset,
    AFPDWBUnderline,
    AFPDWPInvalid
} AFPDWidgetBorderStyle;
```

Description

Form field appearance definitions for the outside border of an annotation.

Values

AFPDWBSolid	Strokes the entire perimeter of the widget with a solid line.
AFPDWBDashed	Strokes the entire perimeter of the widget with a dashed line.
AFPDWBBeveled	Equivalent to **AFPDWBSolid** style with an additional beveled (pushed-out appearance) border applied inside the solid border.
AFPDWBInset	Equivalent to **AFPDWBBeveled** style with an additional beveled (pushed-in appearance) border applied inside the solid border.
AFPDWBUnderline	Strokes the bottom portion of the widget with a underline styled line.
AFPDWBInvalid	Do not change border.

Header File

AF_ExpT.h

Related Methods

AFLayoutBorder

ExportAsFDFParams

ExportAsFDFParamsRec

```
typedef struct _t_ExportAsFDFParams {
    ASSize_t size;
    AVDoc avForm;
    PDDoc pdForm;
    CosObj rgIncExcFlds;
    ASBool bIncl;
    ASBool bEmpty;
    ASBool bExcludePasswd;
    ASBool bCheckReqd;
    ASAtom asaEncoding;
    ASPathName fdfPath;
    char* submitBtnName;
} ExportAsFDFParamsRec;
```

Description

A parameter block used in the call to `ExportAsFDFWithParams`.

Members

`size`	Size of the data structure. Must be set to `sizeof (ExportAsFDFParamsRec)`.
`avForm`	The `AVDoc` for the form from which you want to export the data. You can pass `NULL` if the document does not have an `AVDoc` (that is, it is open only at the PD level).
`pdForm`	The `PDDoc` for the form from which you want to export the data.

`rgIncExcFlds`	If `rgIncExcFlds` is `CosNewNull`, then all fields are exported, and `bIncl` is ignored. If it is a `CosArray`, then the array elements may be: ● Names of fields (the names may be of non-terminal fields, which is a fast and easy way to cause all their children to be included in the FDF file). ● Arrays, whose first element is a field name, and the rest of the elements are field dictionary key names whose values should be exported in the FDF file. For example, **[(My listbox) /AP /Opt]**. ● If `rgIncExcFlds` contains a single element, which is itself an array as described in b) above, and its first element, which corresponds to the field name, is `NULL`, then the FDF will include the requested field properties of all fields. For example: **[null /F /Ff]**. This variety of `rgIncExcFlds` array element can only be used if `bIncl` is `true`.
`bIncl`	If `true`, then `rgIncExcFlds` is an array of the fields to export. Otherwise, it is an array of the fields to exclude from exporting.
`bEmpty`	If `true`, then all fields selected per the criteria above are exported. Otherwise, exclude those that have no value.
`bExcludePasswd`	If `true`, will not include in the FDF text fields that have the `"`password" flag set.
`bCheckReqd`	If `true`, an alert pops up if any required field is empty.
`asaEncoding`	Encoding to use in the produced FDF file for the value of the **V** entry when it is a string. If `ASAtomNull` gets passed, then `ExportAsFDF` will determine which encoding to use. If `PDFDocEncoding` is passed, then do not do any encoding conversions and simply send **V** as is (which means, as `PDFDocEncoding` or Unicode). Other allowed values are `Shift_JIS`, `BigFive`, `GBK`, and `UHC`.
`fdfPath`	Path where the FDF file will be saved (by the client) after it is produced (by `ExportAsFDFWithParams`). You need this in order to create an FDF file with an **/F** key that gives the relative path to the form, from the location where the FDF file will be saved. Pass `NULL` if you want an absolute pathname.

submitBtnName	A **NULL**-terminated string containing the name of the button used to submit. If the value passed is not **NULL**, then the FDF file will include a field dictionary corresponding to the submit button, which will only contain one key, namely **/T**. This dictionary is no different than the one you get when an AcroForm has an empty text field (that is, no Value), and **bEmpty** is **true**.

Header File

AF_ExpT.h

Related Methods

ExportAsFDFWithParams

RetCode

```
typedef enum {
    Good,
    Bad,
    Failed
} RetCode;
```

Description

A return code from functions.

Values

`Good`	Indicates success.
`Bad`	Indicates possible parameter problem or other exception.
`Failed`	Indicates a failure.

Header File

`AF_ExpT.h`

Related Methods

`AFPDFieldSetOptions`

TextAppearance

TextAppearanceP

```
typedef struct TextAppearance_t {
    ASFixed charSpacing;
    ASFixed wordSpacing;
    ASFixed horizontalScale;
    ASFixed leading;
    ASFixed textRise;
    ASFixed textSize;
    ASFixedMatrix textMatrix;
    PDColorValueRec strokeColor;
    PDColorValueRec fillColor;
    ASUns16 renderMode;
    ASUns16 quadding;
    ASAtom baseFont;
    ASAtom nameFont;
} *TextAppearanceP;
```

Description

A structure containing information about the text appearance of a field.

Members

`charSpacing`	The spacing between characters.
`wordSpacing`	The spacing between words.
`horizontalScale`	Horizontal scale.
`leading`	Leading.
`textRise`	Text rise.
`textSize`	Point size of text. When **`fixedZero`**, enables text autosizing.
`textMatrix`	Font matrix.
`strokeColor`	Pen color.
`fillColor`	Fill color.
`renderMode`	Mode.
`quadding`	Text justification.

`baseFont`	Default font name.
`nameFont`	Current font name.

Header File

`AF_ExpT.h`

Related Methods

`AFLayoutText`

Related Macros

`Init_TextAppearanceP`

Core Data Types

The following are data types from the Core API that are used in the AcroForms API.

PDColorSpace

Constants that specify the color space in which a color value is specified (for example, RGB or grayscale).

Members

`PDDeviceGray`	Grayscale. Requires 1 **`value`** entry to specify the color.
`PDDeviceRGB`	Red–Green–Blue color specification. Requires 3 **`value`** entries to specify the color.
`PDDeviceCMYK`	Cyan–Magenta–Yellow–Black color specification. Requires 4 **`value`** entries to specify the color.

Related Structures

`PDColorValue`

PDColorValue

PDColorValueRec

Data structure representing a color.

```
typedef struct _t_PDColorValueRec {
    PDColorSpace space;
    ASFixed value[4];
} PDColorValueRec, *PDColorValue;
```

Members

`space`	The color space.
`value`	The color value. The number of elements needed in the **`value`** field depends on the color space type specified in the **`space`** field.

Related Structures

`TextAppearanceP`

Related Methods

`AFLayoutBorder`
`AFPDWidgetGetAreaColors`
`AFPDWidgetSetAreaColors`

PDRotate

Constant values that specify page rotation, in degrees. Used for routines that set and get the value of a page's **Rotate** key.

```
typedef ASEnum16 PDRotate;
enum {
    pdRotate0 = 0,
    pdRotate90 = 90,
    pdRotate180 = 180,
    pdRotate270 = 270
};
```

Related Methods

`AFLayoutNew`

AcroForm Macros

Init_TextAppearanceP

```
#define Init_TextAppearanceP(aP) {\
memset(aP, 0, sizeof(values));\
(aP)->baseFont = ASAtomNull;\
(aP)->nameFont = ASAtomNull;\
}
```

Description

Macro for setting text appearance.

Header File

`AF_ExpT.h`

SetDefaultTextAppearanceP

```
#define Init_TextAppearanceP(aP) {\
memset(aP, 0, sizeof(values));\
(aP)->baseFont = ASAtomFromString("Helvetica”);\
(aP)->textSize = AutoSize;\
(aP)->horizontalScale = fixedHundred;\
(aP)->renderMode = Fill_text;\
(aP)->fillColor.space = PDDeviceGray;\
(aP)->textMatrix.a = fixedOne;\
(aP)->textMatrix.d = fixedOne;\
(aP)->quadding = LeftQ;\
}
```

Description

Macro for setting text appearance. This macro uses Helvetica as the default font.

Header File

`AF_ExpT.h`

TextApperanceIsValid

```
#define TextApperanceIsValid(aP) (\
(aP)->quadding <= RightQ && \
(aP)->renderMode <= Invisible_text && \
(aP)->baseFont != ASAtomNull && \
(aP)->textSize >= Autosize)
```

Description

Macro for checking whether a text appearance is valid.

Header File

`AF_ExpT.h`

Text-to-Speech API (Windows only)

The Acrobat Forms plug-in provides a speech server API implemented over the Windows Speech API (SAPI). It allows document vocalization via text-to-speech conversion to enable accessibility for people with visual and reading disabilities.

To implement this API, the Forms plug-in exports another Host Function Table (AcroTTS) in addition to its common AcroForms HFT. To use the AcroTTS HFT, a plug-in must do the following:

- Include the header file **`TTSHFT.H`**, which in turn includes **`AFTTS_SEL.H`**.
- Import the HFT using **`ASExtensionMgrGetHFT`** and assign the HFT returned by this call to a plug-in-defined global variable named **`gAcroTTSHFT`**. The easiest way to do this is to use the **`Init_AcroTTSHFT`** macro defined in **`TTSHFT.h`**.

NOTE: This API is also available throught Acrobat's JavaScript interface.

AFTTSGetNumberOfVoices

```
ASUns32 AFTTSGetNumberOfVoices();
```

Description

Gets the number of different speakers available to the current TTS engine.

Return Value

The number of speakers available.

Header File

`AFTTS_Sel.h`

Related Methods

`AFTTSGetVoiceName`

AFTTSGetPitch

```
ASUns32 AFTTSGetPitch();
```

Description

Gets the baseline pitch for the voice of a speaker.

Return Value

The baseline pitch. The valid range is from 0 to 10, with 5 being the default for the speaker.

Header File

`AFTTS_Sel.h`

Related Methods

`AFTTSSetPitch`

AFTTSGetSpeaker

```
char* AFTTSGetSpeaker();
```

Description

Gets the name of the current speaker.

Return Value

The name of the current speaker.

Header File

`AFTTS_Sel.h`

Related Methods

`AFTTSSetSpeaker`

AFTTSGetSpeechRate

```
ASUns32 AFTTSGetSpeechRate();
```

Description

Gets the speed at which text is being spoken by the TTS engine.

Return Value

The speed, in number of words per minute, at which text is being spoken.

Header File

`AFTTS_Sel.h`

Related Methods

`AFTTSSetSpeechRate`

AFTTSGetVoiceName

```
char* AFTTSGetVoiceName (ASInt32 index);
```

Description

Gets the voice name of any of the available speakers in the installed TTS engine.

Parameters

`index`	The index of the speaker.

Return Value

The name of the voice.

Header File

`AFTTS_Sel.h`

Related Methods

`AFTTSSetSpeaker`

AFTTSGetVolume

```
ASUns32 AFTTSGetVolume();
```

Description

Gets the volume for the speaker.

Return Value

The volume. Valid values are from 0 (mute) to 10 (loudest). The default is 5.

Header File

`AFTTS_Sel.h`

Related Methods

`AFTTSSetVolume`

AFTTSIsAvailable

```
ASBool AFTTSIsAvailable();
```

Description

Determines whether the TTS object is available and the Text-to-Speech engine can be used.

NOTE: Calling any method in this API causes the Forms plug-in to initialize TTS automatically, if an SAPI engine is installed on the system.

Return Value

`true` if the Text-to-Speech engine can be used, `false` otherwise.

Header File

`AFTTS_Sel.h`

AFTTSPause

```
ASBool AFTTSPause();
```

Description

Immediately pauses TTS output on a TTS object. Playback of the remaining queued text can be resumed via **`AFTTSResume`**.

Return Value

`true` if the speech engine is available, **`false`** otherwise.

Header File

`AFTTS_Sel.h`

Related Methods

`AFTTSResume`

AFTTSQSilence

```
ASBool AFTTSQSilence (ASUns32 duration);
```

Description

Queues a period of silence into the text.

Parameters

`duration`	The amount of silence in milliseconds.

Return Value

`true` if the speech engine is available, `false` otherwise.

Header File

`AFTTS_Sel.h`

Related Methods

`AFTTSQSound`

`AFTTSQueueTextData`

AFTTSQSound

```
ASBool AFTTSQSound (const char* soundName);
```

Description

Puts a sound into the queue. The sound can then be performed by **AFTTSTalk**.

Parameters

soundName	The sound name. Names are: **ActionCopy**, **ActionCut**, **ActionDelete**, **ActionPaste**, **DocActive**, **DocClose**, **DocOpen**, **DocPrint**, **DocSave**, **KeyEnd**, **KeyHome**, and **PageTurn**. This list can be augmented by adding sound files to the **SoundCues** folder in Acrobat's installation, in 22kHz 16-bit PCM **.wav** format.

Return Value

true if the speech engine is available, **false** otherwise.

Header File

AFTTS_Sel.h

Related Methods

AFTTSQSilence

AFTTSQueueTextData

AFTTSQueueTextData

```
ASBool AFTTSQueueTextData (const char* textdata,
ASBool UseDefaultSpeaker);
```

Description

Puts text into the queue to be performed by **`AFTTSTalk`**.

Parameters

`textdata`	The text that will be put into the queue.
`UseDefaultSpeaker`	Whether to use the default speaker.

Return Value

`true` if the speech engine is available, `false` otherwise.

Header File

`AFTTS_Sel.h`

Related Methods

`AFTTSQSilence`

`AFTTSQSound`

AFTTSReset

```
ASBool AFTTSReset();
```

Description

Stops playback of the currently queued text, and flushes the queue. Resets all the properties of the TTS object to their default values.

NOTE: Text playback cannot be resumed via `AFTTSResume`.

Return Value

`true` if the speech engine is available, `false` otherwise.

Header File

`AFTTS_Sel.h`

Related Methods

`AFTTSStop`

AFTTSResume

```
ASBool AFTTSResume();
```

Description

Resumes playback of text on a paused TTS object.

Return Value

`true` if the speech engine is available, `false` otherwise.

Header File

`AFTTS_Sel.h`

Related Methods

`AFTTSPause`

AFTTSSetPitch

```
ASBool AFTTSSetPitch (ASUns32 pitch);
```

Description

Sets the baseline pitch for the voice of a speaker.

Parameters

`pitch`	The baseline pitch. The valid range is from 0 to 10, with 5 being the default for the speaker.

Return Value

`true` if the speech engine is available, **`false`** otherwise.

Header File

`AFTTS_Sel.h`

Related Methods

`AFTTSGetPitch`

AFTTSSetSpeaker

```
ASBool AFTTSSetSpeaker (const char* voiceName);
```

Description

Sets the current voice. Valid values are any of those enumerated via `AFTTSGetVoiceName` and `AFTTSGetNumberOfVoices`.

Parameters

`voiceName`	The speaker name.

Return Value

`true` if the speech engine is available, `false` otherwise.

Header File

`AFTTS_Sel.h`

Related Methods

`AFTTSGetSpeaker`

AFTTSSetSpeechRate

```
ASBool AFTTSSetSpeechRate (ASUns32 speed);
```

Description

Sets the speed at which text is being spoken by the TTS engine.

Parameters

`speed`	The speed in number of words per minute.

Return Value

`true` if the speech engine is available, `false` otherwise.

Header File

`AFTTS_Sel.h`

Related Methods

`AFTTSGetSpeechRate`

AFTTSSetVolume

```
ASBool AFTTSSetVolume (ASUns32 volume);
```

Description

Sets the speech volume.

Parameters

`volume`	The volume. Valid values are from 0 (mute) to 10 (loudest).

Return Value

`true` if the speech engine is available, `false` otherwise.

Header File

`AFTTS_Sel.h`

Related Methods

`AFTTSGetVolume`

AFTTSStop

```
ASBool AFTTSStop();
```

Description

Stops playback of currently queued text, and flushes the queue. Playback of queued text cannot be resumed.

Return Value

`true` if the speech engine is available, `false` otherwise.

Header File

`AFTTS_Sel.h`

Related Methods

`AFTTSReset`

AFTTSTalk

```
ASBool AFTTSTalk();
```

Description

Sends whatever is in the queue to be spoken by the SAPI TTS engine. If the text output had been paused, resumes playback of the queued text.

Return Value

`true` if the speech engine is available, `false` otherwise.

Header File

`AFTTS_Sel.h`

Related Methods

`AFTTSQueueTextData`

AcroForm OLE Automation

Description

The Acrobat Forms plug-in has been enhanced to work as an Automation server in the Win32 environment. Since the automation capabilities have been added to a plug-in, rather than an executable that can be directly launched, the following steps are necessary to access them from an Automation controller:

First instantiate the Acrobat application by using the VB **`CreateObject`** method. For example:

```
CreateObject("AcroExch.App")
```

This causes the Acrobat Forms plug-in to run, at which time it registers its class object with OLE. Then its main exposed object can be instantiated, that is:

```
CreateObject("AFormAut.App")
```

Presently, registration in the Windows registry (which is different from the class object registration described above) happens every time Acrobat loads the plug-in. Therefore, you have to run Acrobat at least once with the **`AForm32.api`** file in the **`plug-ins`** folder before its type library can be found for object browsing from within the Visual Basic environment. This is also necessary in order to allow early binding. Dim the program variables as the corresponding classes in **`AFORMAUTLib`**, and not simply **`As Object`**.

The object model was designed in accordance with the applicable standards and guidelines for document-centric applications from the *OLE Programmer's Reference*. That manual uses the term *document* to describe whatever an application uses as a file or an individual entity of data (in our case a field).

NOTE: Neither Acrobat, nor the Acrobat Forms plug-in, are thread-safe, and therefore Acrobat Forms OLE Automation uses the single-threading model.

Contents

This reference contains the following sections:

- OLE Automation Objects. This section describes the Acrobat objects represented as OLE objects.
- OLE Automation Methods. This section describes each OLE method, including its parameters, return value, and related methods.
- OLE Automation Properties. This section details the properties that can be set in the various objects. Each property describes the key, the property type (for example, read-only), and the semantics.

Conventions

The syntax used in this reference follows that used in Microsoft Visual Basic references.

Exceptions

All methods and properties may return an exception. Some of the possible ones are standard OLE exceptions, such as:

- **`E_OUTOFMEMORY`** (0x8007000E)
- **`E_INVALIDARG`** (0x80070057)

These exceptions are not specifically called out in the descriptions of the methods and properties that appear below. Others are AcroForm-specific, and are listed in the following table.

The actual numeric value of the returned exception is assembled as an **`HRESULT`**, uses the **`FACILITY_ITF`**, and starts with decimal 512 (hex 0x0200), as recommended by Microsoft. For example, the numeric value of the exception **`AutErcNoForm`** is hex 0x80040201. The important part is the right-most hex 201, which is the first error in the enumeration below.

Exception Name	Numeric Value	Description
`AutErcNoDoc`	1	No document is currently open in the Acrobat Viewer.
`AutErcNotTerminal`	2	This property or method applies to terminal fields, or their annotations.
`AutErcNotToThisFieldType`	3	This property or method is not applicable to this type of field.

OLE Automation Objects

AFormApp

`AFormApp` is the only object the controller can externally instantiate (that is, using `CreateObject`). All other objects must be created by navigating down the hierarchy with the methods and properties described in this document.

Field

A field in the document that is currently active in Acrobat.

Fields

A collection of all the fields in the document that are currently active in Acrobat at the time `Fields` is instantiated.

The Fields collection includes both *terminal* and *non-terminal* fields. A terminal field is one that either does not have children, or if it does, they are simply multiple appearances (that is, child annotations) of the field in question.

NOTE: If you instantiate a `Fields` object, and subsequently fields are manually added or removed using the Forms tool in Acrobat, the `Fields` object will no longer be in sync with the document. You must re-instantiate the `Fields` object.

OLE Automation Methods

Field

PopulateListOrComboBox

```
void PopulateListOrComboBox (const VARIANT& arrItems,
const VARIANT& arrExportVal);
```

Description

Specifies the item names and optionally exports values for a field of type listbox or combobox.

Parameters

`arrItems`	An array of strings, with each element representing an item name. **NOTE:** There is a limit of 64K for string data in a combo or list box control on Windows platforms. For Mac OS systems, the limit is 200 entries for the combo or list box control. Using more than these limits degrades performance and makes the control unusable.
`arrExportVal`	*(Optional)* An array of strings, the same size as the first parameter, with each element representing an export value. **NOTE:** Some of the elements in `exportString` may be empty strings.

Return Value

None

Exceptions

Raises `AutErcNotToThisFieldType` if the field is not of type listbox or combobox.

Related Methods

`Add`

SetBackgroundColor

```
void SetBackgroundColor (LPCTSTR bstrColorSpace,
float GorRorC, float GorM, float BorY, float K);
```

Description

Specifies the background color for a field. The background color is used to fill the field's rectangle.

Parameters

`bstrColorSpace`	Values are defined by using transparent, gray, RGB or CMYK color. Valid strings include (and must be spelled exactly as shown): ● T ● G ● RGB ● CMYK
`GorRorC`	Used if **`bstrColorSpace`** is set to **T**, **G**, or **RGB**. A float range between zero and one inclusive.
`GorM`	Used if **`bstrColorSpace`** is set to **G**. A float range between zero and one inclusive.
`BorY`	Used if **`bstrColorSpace`** is set to **RGB**. A float range between zero and one inclusive.
`K`	Used if **`bstrColorSpace`** is set to **CMYK**. A float range between zero and one inclusive.

Return Value

None

Related Methods

`SetBorderColor`
`SetForegroundColor`

Example

```
Field.SetBackgroundColor "RGB", 0.7, 0.3, 0.6, 0
```

SetBorderColor

```
void SetBorderColor (LPCTSTR bstrColorSpace, float GorRorC,
float GorM, float  BorY, float K);
```

Description

Specifies the border color for a field. The border color is used to stroke the field’s rectangle with a line as large as the border width. The new border color is propagated to any child annotations underneath, so the field may be non-terminal.

Parameters

`bstrColorSpace`	A string specifying the color space. Valid strings include (and must be spelled exactly as shown): ● `T`: transparent ● `G`: gray ● `RGB` ● `CMYK`
`GorRorC`	Used if `bstrColorSpace` is set to `T`, `G`, or `RGB`. A float range between zero and one inclusive.
`GorM`	Used if `bstrColorSpace` is set to `G`. A float range between zero and one inclusive.
`BorY`	Used if `bstrColorSpace` is set to `RGB`. A float range between zero and one inclusive.
`K`	Used if `bstrColorSpace` is set to `CMYK`. A float range between zero and one inclusive.

Return Value

None

Related Methods

`SetBackgroundColor`
`SetForegroundColor`

Example

```
Field.SetBorderColor "RGB", 0.7, 0.3, 0.6, 0
```

SetButtonCaption

```
void SetButtonCaption (LPCTSTR bstrFace, LPCTSTR bstrCaption);
```

Description

The caption to be used for the appearance of a field of type button.

Parameters

`bstrFace`	A string that specifies the face for which the caption will be used. Valid strings include. ● `N` is the Normal appearance ● `D` is the Down appearance ● `R` is the appearance for Rollover
`bstrCaption`	The caption for the button. **NOTE:** If a button's layout is of type icon only, the caption is not used in generating its appearance. In addition, only the Normal face is displayed, unless the Highlight is of type push.

Return Value

None

Exceptions

Raises `AutErcNotToThisFieldType` if the field is not of type button. The new appearance is propagated to any child annotations underneath; the field may be non-terminal.

Related Methods

`SetButtonIcon`

Example

```
Field.SetButtonCaption "D", "Submit Form"
```

SetButtonIcon

```
void SetButtonIcon (LPCTSTR bstrFace, LPCTSTR bstrFullPath,
short pageNum);
```

Description

Specifies the icon to be used for the appearance of a field of type button.

Parameters

`bstrFace`	A string that specifies the face for which the icon will be used. Valid strings include: ● `N` is the Normal appearance ● `D` is the Down appearance ● `R` is the appearance for Rollover
`bstrFullPath`	The full pathname of the PDF file to be used as the source of the appearance.
`pageNum`	Used to select the page inside that PDF file (zero-based). **NOTE:** If a button's layout is of type icon only, the caption is not used in generating its appearance. In addition, only the Normal face is displayed, unless the Highlight is of type push.

Return Value

None

Exceptions

Raises `AutErcNotToThisFieldType` if the field is not of type button. The new appearance is propagated to any child annotations underneath, so it is OK if the field is non-terminal.

Related Methods

`SetButtonCaption`

Example

```
Field.SetButtonIcon "N", "c:\Clipart.pdf", 0
```

SetExportValues

```
void SetExportValues (const VARIANT& arrExportVal);
```

Description

Sets the export values for each of the annotations of a field of type radio button and checkbox.

For radio button fields, this is necessary to make the field work properly as a group, with the one button checked at any given time giving its value to the field as a whole.

For checkbox fields, unless an export value is specified, the default is used when the field is checked is Yes. When it is unchecked, its value is Off (this is also true for a radio button field when none of its buttons are checked).

Parameters

`arrExportVal`	An array of strings, which is expected to have as many elements as there are annotations in the field. The elements of the array are distributed among the individual annotations comprising the field using their tab order.

Return Value

None

Exceptions

Raises `AutErcNotToThisFieldType` if the field is not of type radio button or checkbox.

Related Methods

`Add`

Example

```
Dim arrExp(1) As String
arrExp(0) = "Visa"
arrExp(1) = "Mastercard"
Field.SetExportValues arrExp
```

SetForegroundColor

```
void SetForegroundColor (LPCTSTR bstrColorSpace,
float GorRorC, float GorM, float BorY, float K);
```

Description

Specifies the foreground color for a field. It represents the text color for text, button, combobox, or listbox fields and the check color for checkbox or radio button fields.

NOTE: Parameters are similar to SetBorderColor and SetBackgroundColor except that transparent color space is not allowed.

Parameters

`bstrColorSpace`	Values are defined by using transparent, gray, RGB or CMYK color. Valid strings include (and must be spelled exactly as shown): ● `T`: transparent ● `G`: gray ● `RGB` ● `CMYK`
`GorRorC`	Used if **`bstrColorSpace`** is set to `T`, `G`, or `RGB`. A float range between zero and one inclusive.
`GorM`	Used if **`bstrColorSpace`** is set to `G`. A float range between zero and one inclusive.
`BorY`	Used if **`bstrColorSpace`** is set to `RGB`. A float range between zero and one inclusive.
`K`	Used if **`bstrColorSpace`** is set to `CMYK`. A float range between zero and one inclusive.

Return Value

None

Related Methods

`SetBackgroundColor`
`SetBorderColor`

Example

```
Field.SetForegroundColor "CMYK", 0.25, 0.25, 0.25, 0.1
```

SetJavaScriptAction

```
void SetJavaScriptAction (LPCTSTR bstrTrigger,
LPCTSTR bstrTheScript);
```

Description

Sets the action of the field to be of type JavaScript. When using **`SetJavaScriptAction`**, within Visual Basic, you may use Chr(13) to add a <CR>, and Chr(9) for tabs, so that the function is nicely formatted.

Parameters

`bstrTrigger`	A string that specifies the trigger for the action. Valid strings include:
	`up`
	`down`
	`enter`
	`exit`
	`calculate`
	`validate`
	`format`
	`keystroke`

`bstrTheScript`	The script itself. **NOTE:** If the trigger is calculate, an entry is added at the end of the calculation order array (see the `CalcOrderIndex` property).

Return Value

None

Related Methods

None

Calculate Notes

There is a simple calculate script supplied with Acrobat.

`AFSimple_Calculate(`*`cFunction, cFields`*`)`

– *cFunction* is one of "AVG", "SUM", "PRD", "MIN", "MAX"

– *cFields* is the list of the fields to use in the calculation.

Formatting Notes

The following scripts and formats can be used for the **`format`** and **`keystroke`** triggers:

`AFDate_KeystrokeEx(`*`cFormat`*`)`
`AFDate_Format(`*`cFormat`*`)`

– *cFormat* is one of:

  "m/d", "m/d/yy", "mm/dd/yy", "mm/yy", "d-mmm", "d-mmm-yy", "dd-mmm-yy", "yy-mm-dd", "mmm-yy", "mmmm-yy", "mmm d, yyyy", "mmmm d, yyyy", "m/d/yy h:MM tt", "m/d/yy HH:MM"

`AFTime_Keystroke(`*`ptf`*`)`
`AFTime_Format(`*`ptf`*`)`

– *ptf* is the time format:

  0 = 24HR_MM [14:30]
  1 = 12HR_MM [2:30 PM]
  2 = 24HR_MM_SS [14:30:15]
  3 = 12HR_MM_SS [2:30:15 PM]

`AFPercent_Keystroke(`*`nDec, sepStyle`*`)`
`AFPercent_Format(`*`nDec, sepStyle`*`)`

– *nDec* is the number of places after the decimal point;
– *sepStyle* is an integer denoting whether to use a separator or not. If *sepStyle*=0, use commas. If *sepStyle*=1, do not separate.

`AFSpecial_Keystroke(`*`psf`*`)`
`AFSpecial_Format(`*`psf`*`)`

– *psf* is the type of formatting to use:

  0 = zip code
  1 = zip + 4
  2 = phone
  3 = SSN

`AFNumber_Format(`*`nDec, sepStyle, negStyle, currStyle, strCurrency, bCurrencyPrepend`*`)`
`AFNumber_Keystroke(`*`nDec, sepStyle, negStyle, currStyle, strCurrency, bCurrencyPrepend`*`)`

– *nDec* is the number of places after the decimal point;
– *sepStyle* is an integer denoting whether to use a separator or not. If *sepStyle*=0, use commas. If *sepStyle*=1, do not separate.

– *negStyle* is the formatting used for negative numbers:

  0 = MinusBlack
  1 = Red
  2 = ParensBlack
  3 = ParensRed

– *currStyle* is the currency style - not used

– *strCurrency* is the currency symbol

– *bCurrencyPrepend* is true to prepend the currency symbol; false to display on the end of the number

SetResetFormAction

```
void SetResetFormAction (LPCTSTR bstrTrigger, long theFlags,
const VARIANT& arrFields);
```

Description

Sets the action of the field to be of type ResetForm.

Parameters

`bstrTrigger`	A string that specifies which trigger is used for the action. Valid strings include (and must be spelled exactly as shown): ● **`up`** (for Mouse Up) ● **`down`** (for Mouse Down) ● **`enter`** (for Mouse Enter) ● **`exit`** (for Mouse Exit)
`theFlags`	When 0 (Include), **`arrFields`** specifies which fields to *include* in the reset operation. When non-zero (Exclude), **`arrFields`** specifies which fields to *exclude* from the reset operation.
`arrFields`	(*Optional*) An array of strings for the fully-qualified names of the fields. Depending on the value of **`theFlags`**, these fields are included in or excluded from the reset operation. When the fields are included, the set can include the names of non-terminal fields, which is a fast and easy way to cause all their children to be included in the action. When not supplied, all fields are reset.

Return Value

None

Related Methods

None

SetSubmitFormAction

```
void SetSubmitFormAction (LPCTSTR bstrTrigger,
LPCTSTR bstrTheURL, long theFlags, const VARIANT& arrFields);
```

Description

Sets the action of the field to be of type SubmitForm.

Parameters

`bstrTrigger`	A string that specifies which trigger is used for the action. Valid strings include (and must be spelled exactly as shown): ● `up` (for Mouse Up) ● `down` (for Mouse Down) ● `enter` (for Mouse Enter) ● `exit` (for Mouse Exit)
`bstrTheURL`	A string containing the URL.
`theFlags`	A collection of flags that define various characteristics of the action. **NOTE:** See Section 8.6 in the *PDF Reference* to learn how the binary value of this long is interpreted.
`arrFields`	(*Optional*) If passed, represents an array of strings for the fully-qualified names of the fields to submit when the action gets executed. If the array is interpreted as fields to submit (as opposed to exclude from submitting, depending on the least-significant bit in the flags), then it may include the names of non-terminal fields, which is a fast and easy way to cause all their children to be included in the submission. If not passed, then the created action will not include a **/Fields** key.

Return Value

None

Related Methods

None

Fields

Add

```
LPDISPATCH Add (LPCTSTR bstrFieldName, LPCTSTR bstrFieldType,
short pageNum, float left, float top, float right,
float bottom);
```

Description

Adds a new field "on-the-fly" to the Acrobat form and to the **`Fields`** collection.

Returns the newly-created **`Field`** object. You can pass as parameter the name of an existing field, as long as that field is the same type as the one being created.

This is useful in the following circumstances:

- For radio buttons to use the **`SetExportValues`** method to make the radio buttons mutually exclusive.
- For fields that should have multiple appearances (that is, child annotations) in the document.

Parameters

`bstrFieldName`	The fully-qualified name of the field.
`bstrFieldType`	Field type for the newly created field. Valid types are: ● **`"text"`** ● **`"button"`** ● **`"combobox"`** ● **`"listbox"`** ● **`"checkbox"`** ● **`"radio button"`** ● **`"signature"`** You must use the quotation marks. See the sample code below. **NOTE:** When creating list or combo boxes, there is a limit of 64K for string data on Windows platforms. MacOS systems have a limit of 200 entries for the list or combo boxes. Using more than the limit degrades performance. You populate the fields of the list and combo boxes using the **`PopulateListOrComboBox`** method.
`pageNum`	The page number (zero-based).

`left, top, right, bottom`	These parameters are floats representing the left, top, right, and bottom coordinates of the field rectangle, measured in *rotated page space*; that is, [0,0] is always at the left bottom corner regardless of page rotation.

Return Value

The newly-created **`Field`** object.

Related Methods

`PopulateListOrComboBox`
`Remove`

Example

```
Set Field = Fields.Add("payment”,_ "radiobutton”, 0, 100, 600, 130, 570)
```

AddDocJavascript

```
void AddDocJavascript (LPCTSTR bstrScriptName,
LPCTSTR bstrTheScript);
```

Description

Adds a document-level JavaScript function to the PDF file. When using **`AddDocJavascript`**, within Visual Basic, you can use **`Chr(13)`** to add a <CR>, and **`Chr(9)`** for tabs, so that the function is nicely formatted.

Parameters

`bstrScriptName`	The name of the function being added to the document.
`bstrTheScript`	The definition being added to the document.

Return Value

None

Related Methods

`ExecuteThisJavascript`

Example

```
'Adding a document-level JavaScript
'function, to compute factorials:
Fields.AddDocJavaScript "Fact", _
"function Fact(n)" & Chr(13) & _
"{" & Chr(13) & _
Chr(9) & "if (n <= 0)" & Chr(13) & _
Chr(9) & Chr(9) & "return 1;" & Chr(13) & _
Chr(9) & "else" & Chr(13) & _
Chr(9) & Chr(9) & "return n * Fact(n - 1);" & Chr(13) & _
"}"
```

ExecuteThisJavascript

```
CString ExecuteThisJavascript (LPCTSTR bstrTheScript);
```

Description

Executes the specified JavaScript script.

Parameters

`bstrTheScript`	A string containing a JavaScript script, which is executed by Acrobat in the context of the currently active document. **NOTE:** See the *Acrobat JavaScript Scripting Reference* for information on event level values.

Return Value

Returns a result by assigning it to event value.

Related Methods

`AddDocJavascript`

Example

```
Fields.ExecuteThisJavaScript "var f =_ this.getField(""myButton"");
f.delay =_ false;"
```

To get the return value in Visual Basic:

```
Dim cSubmitName As String
cSubmitName = Fields.ExecuteThisJavaScript
     "event.value = this.getField(""myField"").submitName;"
```

ExportAsFDF

```
void ExportAsFDF (LPCTSTR bstrFullPath,
LPCTSTR bstrSubmitButton, BOOL bEmptyFields,
const VARIANT& arrFields);
```

Description

Exports the data as FDF from an AcroForm.

Parameters

`bstrFullPath`	A full pathname of the file to which the produced FDF file will be saved.
`bstrSubmitButton`	The name of an existing form field of type Button (in case you want it included in the FDF file, as if it had been used to trigger a `SubmitForm` action). You may pass an empty string.
`bEmptyFields`	A boolean to indicate whether fields with no value should be included in the produced FDF file.
`arrFields`	(*Optional*) An array of strings representing the fully-qualified names of the fields to include in the FDF file. This array may include the names of non-terminal fields, which is a fast and easy way to cause all their children to be included in the FDF file.

Return Value

None

Related Methods

`ImportAnFDF`
`ExportAsHtml`

Example

```
Dim arrFields(1) As String
arrFields(0) = "name”
arrFields(1) = "address”
‘This will create an FDF that includes
‘name.last, name.first, address.street,
‘etc., but only if they have a value
‘(since we are passing False for the
‘ "bEmptyFields” parameter.
Fields.ExportAsFDF "C:\Temp\out.fdf”, "”, False, arrFields
```

ExportAsHtml

```
void ExportAsHtml (LPCTSTR bstrFullPath,
LPCTSTR bstrSubmitButton, BOOL bEmptyFields,
const VARIANT& arrFields);
```

Description

Exports the data as HTML from an AcroForm. This method is similar to `ExportAsFDF`. The only difference is that the form data is exported in URL-encoded format.

Parameters

`bstrFullPath`	A full pathname of the file to which the produced FDF file will be saved.
`bstrSubmitButton`	The name of an existing form field of type Button (in case you want it included in the FDF file, as if it had been used to trigger a `SubmitForm` action). You may pass an empty string.
`bEmptyFields`	A boolean to indicate whether fields with no value should be included in the produced FDF file.
`arrFields`	(Optional) An array of strings representing the fully-qualified names of the fields to include in the FDF file. This array may include the names of non-terminal fields, which is a fast and easy way to cause all their children to be included in the FDF file.

Return Value

None

Related Methods

`ExportAsFDF`

ImportAnFDF

```
void ImportAnFDF (LPCTSTR bstrFullPath);
```

Description

Imports the FDF file to an AcroForm.

Parameters

`bstrFullPath`	The full pathname of the file containing the FDF file to import.

Return Value

None

Related Methods

`ExportAsFDF`

Remove

```
void Remove (LPCTSTR bstrFieldName);
```

Description

Removes a field from the Acrobat Form and from the **`Fields`** collection.

Parameters

`bstrFieldName`	The fully-qualified name of the field to remove from the Acrobat form. If the field has multiple child annotations, all of them are removed. If multiple fields have the same name, all are removed.

Return Value

None

Related Methods

`Add`

Example

```
'Remove fields you no longer used.
Fields.Remove("MyOldField")
```

OLE Automation Properties

Field

Alignment

```
[get/set] String
```

Description

The text alignment of a text field. Valid alignments are (must be spelled exactly as shown):

```
left
center
right
```

Return Value

If the field is terminal, and has multiple child annotations, a get returns the alignment for the first child, whichever annotation that happens to be.

On a set, the property is propagated to any child annotations underneath, so the field may be non-terminal.

Exceptions

If the field is not of type text, an exception `AutErcNotToThisFieldType` is returned.

On a get, if the field is non-terminal, an exception `AutErcNotTerminal` is returned.

Example

```
Field.Alignment = left
```

BorderStyle

```
[get/set] String
```

Description

The border style for a field. Valid border styles include solid, dashed, beveled, inset, and underline (and must be spelled exactly as shown).

Return Value

If it is terminal, and has multiple child annotations, a get returns the value of the border style for the first child, whichever annotation that happens to be.

On a set, the property is propagated to any child annotations underneath, so the field may be non-terminal.

Exceptions

On a get, raises `AutErcNotTerminal` if the field is non-terminal, an exception is returned.

Example

```
Field.BorderStyle = "beveled"
```

BorderWidth

```
[get/set] short
```

Description

The thickness of the border when stroking the perimeter of a field's rectangle. If the border color is transparent, this property has no effect except in the case of a beveled border. The value 0 represents no border, and the value 3 is a thick border.

Return Value

If it is terminal, and has multiple child annotations, a get returns the value of the border width for the first child, whichever annotation that happens to be.

On a set, the property is propagated to any child annotations underneath, so the field may be non-terminal.

Exceptions

On a get, if the field is non-terminal, an exception **`AutErcNotTerminal`** is returned.

Example

```
Field.BorderWidth = 1
```

ButtonLayout

```
[get/set] short
```

Description

The layout appearance of a button. Valid values include:

- 0 - text only; the button has a caption but no icon.
- 1 - icon only; the button has an icon but no caption.
- 2 - icon over text; the icon should appear on top of the caption.
- 3 - text over icon; the text should appear on top of the icon.
- 4 - icon then text; the icon should appear to the left of the caption.
- 5 - text then icon; the icon should appear to the right of the caption.
- 6 - text over icon; the text should be overlaid on top of the icon.

If it is terminal, and has multiple child annotations, a get returns the layout for the first child, whichever annotation that happens to be.

On a set, the property is propagated to any child annotations underneath, so it is OK if the field is non-terminal.

Exceptions

If the field is not of type button, an exception `AutErcNotToThisFieldType` is returned.

On a get, if the field is non-terminal, an exception `AutErcNotTerminal` is returned.

Example

```
Field.ButtonLayout = 2
```

CalcOrderIndex

```
[get/set] short
```

Description

The calculation order of fields in the document. It is zero-based. If you want the calculation for a field f2 to be performed after that for field f1, you need only set the `CalcOrderIndex` for f2 to f1's `CalcOrderIndex` + 1. The elements in the calculation order array get shifted as necessary to make room for the insertion (but the first calculation is still at index 0).

See also the section "The calculation order array" in the document `AcroJS.PDF` (under the menu **Help -> Forms -> JavaScript**).

Example

```
Set F1 = Fields("SubTotal")
Set F2 = Fields("Total")
F2.CalcOrderIndex = F1.CalcOrderIndex + 1
```

CharLimit

```
[get/set] short
```

Description

The limit on the number of characters that a user can type into a text field.

On a set, the property is propagated to any child annotations underneath, if any.

Exceptions

If the field is not of type text, an exception `AutErcNotToThisFieldType` is returned.

DefaultValue

```
[get/set] String
```

Description

The default value of the field. It returns the empty string if the field has no default value. If the field is non-terminal, an exception **`AutErcNotTerminal`** is returned.

Setting a **`DefaultValue`** of **`ON`** for fields of type checkbox and radio button is equivalent to selecting "Default is checked" when you edit the properties of the field using the Forms tool in Acrobat. A **`DefaultValue`** of **`OFF`** is a default of unchecked.

See also **`Value`**.

Editable

```
[get/set] Boolean
```

Description

Determines whether the user can type in a selection or must choose one of the provided selections. Comboboxes can be editable; that is, the user can type in a selection.

On a set, the property is propagated to any child annotations underneath, if any.

Exceptions

Returns an exception of `AutErcNotToThisFieldType` if the field is not of type combobox.

Example

```
Field.Editable = False
```

Highlight

```
[get/set] String
```

Description

Defines how a button reacts when a user clicks it. The four highlight modes supported are:

```
none
invert
push
outline
```

If it is terminal, and has multiple child annotations, a get returns the highlight for the first child, whichever annotation that happens to be.

On a set, the property is propagated to any child annotations underneath, so the field may be non-terminal.

Exceptions

If the field is not of type button, an exception `AutErcNotToThisFieldType` is returned.

On a get, if the field is non-terminal, an exception `AutErcNotTerminal` is returned.

Example

```
Field.Highlight = "invert"
```

IsHidden

```
[get/set] Boolean
```

Description

Controls whether the field is hidden or visible to the user. If the value is **`true`** the field is invisible, **`false`** the field is visible.

During get operations, if the field is non-terminal, an exception **`AutErcNotTerminal`** is returned. If it is terminal, and has multiple child annotations, a get returns the value of the hidden flag for the first child, whichever annotation that happens to be.

During set operations, the property is propagated to any child annotations underneath, so it is OK if the field is non-terminal.

Example

```
'Hide "name.last"
Set Field = Fields("name.last")
Field.IsHidden = True
```

IsMultiline

```
[get/set] Boolean
```

Description

Determines whether the text field is multi-line or single-line.

On a set, the property is propagated to any child annotations underneath, if any.

Exceptions

If the field is not of type text, an exception `AutErcNotToThisFieldType` is returned.

Example

```
Field.IsMultiline = True
```

IsPassword

```
[get/set] Boolean
```

Description

The property for the field to display asterisks for the data entered into the field. Upon submission, the actual data entered is sent. Fields that have the password attribute set will not have the data in the field saved when the document is saved to disk.

On a set, the property is propagated to any child annotations underneath, if any.

Exceptions

If the field is not of type text, an exception **`AutErcNotToThisFieldType`** is returned.

Example

```
Field.IsPassword = True
```

IsReadOnly

`[get/set] Boolean`

Description

The read-only characteristic of a field. When a field is read-only, the user can see the field but cannot change it. If a button is read-only, the user cannot press it to execute an action.

Because this is a field flag and not an annotation flag, both a get and a set of this property are allowed regardless of whether the field is terminal or non-terminal.

- A get on a non-terminal field retrieves that field’s flag.
- A set changes the flag on all its terminal children.

IsRequired

```
[get/set] Boolean
```

Description

The required characteristic of a field. When a field is required, its value must be non-**`NULL`** when the user clicks a submit button that causes the value of the field to be sent to the web. If the field value is **`NULL`**, the user receives a warning message and the submit does not occur.

Since this is a field flag and not an annotation flag, both a get and a set of this property are allowed, regardless of whether the field is terminal or non-terminal.

A get on a non-terminal field retrieves that field's flag.

A set changes the flag on all its terminal children.

IsTerminal

```
[read-only] Boolean
```

Description

`true` if the field is terminal, otherwise `false`.

Example

```
Dim Field As AFORMAUTLib.Field
Dim bTerminal As Boolean

'bTerminal should be True
bTerminal = Field.IsTerminal
```

Name

```
[read-only] String
```

Description

The fully qualified name of the field. It is the default member of the Field interface.

NoViewFlag

```
[get/set] Boolean
```

Description

Determines whether a given field prints but doesn't display on the screen.

Set the **`NoViewFlag`** property to **`true`** to allow the field to appear when the user prints the document but not when it displays on the screen; set it to **`false`** to allow both printing and displaying.

On a get, if the field is non-terminal, an exception **`AutErcNotTerminal`** is returned. If it is terminal, and has multiple child annotations, a get returns the value of the no-view flag for the first child, whichever annotation that happens to be.

On a set, the property is propagated to any child annotations underneath, so the field may be non-terminal.

PrintFlag

```
[get/set] Boolean
```

Description

Determines whether a field prints or not. Set the **`PrintFlag`** property to **`true`** to allow the field to appear when the user prints the document, set it to **`false`** to prevent printing.

On a get, if the field is non-terminal, an exception **`AutErcNotTerminal`** is returned. If it is terminal, and has multiple child annotations, a get returns the value of the print flag for the first child, whichever annotation that happens to be.

On a set, the property is propagated to any child annotations underneath, so the field may be non-terminal.

Style

```
[get/set] String
```

Description

The style of a checkbox or a radio button, that is, the glyph used to indicate that the check box or radio button has been selected.

Valid styles include (must be spelled exactly as shown):

```
check
cross
diamond
circle
star
square
```

If it is terminal, and has multiple child annotations, a get returns the style for the first child, whichever annotation that happens to be.

On a set, the property is propagated to any child annotations underneath, so it is OK if the field is non-terminal.

Exceptions

During set, if the field is not of type checkbox or radio button, an exception `AutErcNotToThisFieldType` is returned.

On a get, if the field is non-terminal, an exception `AutErcNotTerminal` is returned.

Example

```
Field.Style = "star"
```

TextFont

```
[get/set] String
```

Description

The text font used when laying out the field. Valid fonts include (and must be spelled exactly as shown):

```
Courier
Courier-Bold
Courier-Oblique
Courier-BoldOblique
Helvetica
Helvetica-Bold
Helvetica-Oblique
Helvetica-BoldOblique
Symbol
Times-Roman
Times-Bold
Times-Italic
Times-BoldItalic
ZapfDingbats
```

On a set, the property is propagated to any child annotations underneath, if any.

Example

```
Field.TextFont = "Times-BoldItalic"
```

TextSize

```
[get/set] short
```

Description

The text points size used in the field. In combobox and radio button fields, the text size determines the size of the check. Valid text sizes include zero and the range from 4 to 144 inclusive.

A text size of zero means that the largest point size that can still fit in the field's rectangle should be used (in multi-line text fields and buttons this is always 12 point).

On a set, the property is propagated to any child annotations underneath, if any.

Example

```
Field.TextSize = 18
```

Type

`[read-only] String`

Description

The type of the field as a string. Valid types that are returned:

```
text
button
combobox
listbox
checkbox
radiobutton
signature
```

Example

```
Set Field = Fields("name.last")
'Should print "name.last"
print Field
' Should print the type of field. Example,
' "text"
print Field.Type
```

Value

```
[get/set] String
```

Description

A string that represents the value of the field. Returns the empty string if the field has no value. If the field is non-terminal, an exception **AutErcNotTerminal** is returned.

For fields of type checkbox, the value **Off** represents the unchecked state. The checked state is represented using the export value. This is also true for radio buttons (where each individual button in a group should have a different export value; see **SetExportValues**.) For fields of type listbox or combobox, if an export value is defined, then that represents the value, otherwise the item name is used.

These remarks apply also to **DefaultValue**.

Example

```
Dim arrExp(1) As String
arrExp(0) = "Visa"
arrExp(1) = "Mastercard"
Field.SetExportValues arrExp
Field.Value = arrExp(0)
```

Fields

Count

```
[read-only] long
```

Description

The number of items in the collection.

Example

```
Dim Field As AFORMAUTLib.Field
Dim nFields As Long

nFields = Fields.Count

For Each Field In Fields
If Field.IsTerminal Then
print Field.Value
End If
Next Field
```

Item

```
[read-only] IDispatch*
```

Description

Takes the fully qualified name of the field (for example, "name.last") as a parameter, and returns the Field object for it. It is the default member of the Fields interface (that is, the property invoked if the object name is specified by itself without a property or a method in the controller script).

Example

```
Dim Field As AFORMAUTLib.Field
Dim nFields As Long

Set Field = Fields.Item("name.last")
'Since Item is the default_ property:
Set Field = Fields("name.last")
```

_NewEnum

```
[read-only] IUnknown*
```

Description

The IEnumVariant enumerator for the collection.

NOTE: You do not need to call this property directly. Visual Basic calls it in the background whenever the code contains a "For Each Field In Fields" loop. For example:

```
For Each Field in Fields
If Field.IsTerminal
print Field.Value
End If
Next Field
```