ADAnews

Acrobat Exchange Interapplication Communication

Last month's article described the plug-in API of Adobe™ Acrobat™ Exchange version 2.0. This month, we'll describe the interapplication communication (IAC) support present in Exchange 2.0. IAC allows your application to communicate with Exchange and some plug-ins. This provides an easy yet powerful way to integrate Acrobat into your existing application or system, or to control Exchange in virtually any way a user can. A future article will describe the developer-related features of the other Acrobat products.

The IAC support in Exchange provides access to a subset of the objects and methods contained in the plug-in API, described in last month's article. The plug-in API contains four groups of objects and methods: AcroView (the application's user interface), PDModel (a PDF document), Cos (low-level data representation used in PDF files), and Acrobat Support (support and utilities). IAC support includes a subset of the AcroView and some of the PDModel objects and methods.

IAC is implemented using each platform's native interprocess communication mechanisms: Apple events and AppleScript® on the Apple® Macintosh® computer, OLE 2 and DDE in the Microsoft® Windows™ environment. In addition, the Windows version of Exchange is an OLE server and supports the Lotus Notes/FX technology. The remainder of this article provides an overview of the support Exchange provides for each IAC mechanism.

Apple events

Exchange is viewed as a collection of objects, including the application itself, menus, menu items, toolbar buttons, documents, pages, bookmarks, and annotations. Each object has one or more properties, and responds to one or more Apple events. The Apple events get or set the value of a property, or cause the object to take some action. As an example, the properties of text annotations (referred to as notes in the user interface) include a label, a bounding rectangle, a contents string, an open/closed flag, a color, and a modification date. They respond to make (creates a new text annotation) and delete Apple events, in addition to those that get or set the value of a property.

Three classes of Apple events are supported: required, core, and Acrobat-specific. The required events are the four events the Finder sends to Exchange: open, print, quit, and run. The core events are a set of events that are usable by a wide variety of applications. Exchange supports the following core events: get/set (gets or sets an object's data or the value of one of its properties), open/close, make/delete (creates/destroys an object), count (counts objects), exists (determines whether or not an object exists), hide (hides the application), move, save, and quit. The Acrobat-specific events allow your application to: find text, select text, execute any menu item, render a PDF file into a specified window, create thumbnail images, find the next note, scroll, and perform a variety of other tasks.

Adobe

Interapplication Communication

Exchange allows almost all the Apple events it supports to be accessed using either AppleScript or a C program. Because AppleScript is much simpler to use, it is generally preferred. There are a few Apple events which are not available through AppleScript. Examples include the events provided by Acrobat Search and the Exchange event to render into a window.

OLE automation

The scope and organization of the OLE automation support in the Windows version of Exchange is very similar to the Apple event support in the Macintosh version. Exchange is viewed as a collection of objects, including the application itself, documents, pages, annotations, bookmarks, text selections, and text highlights. Methods act on these objects, either causing the object to take some action or to get/set the value of its internal data. As with Apple events, there is an OLE automation method that allows Exchange to render a PDF file into an arbitrary window. Although there is no OLE automation support in the Acrobat Search plug-in, that plug-in does respond to DDE messages, as described later in this article.

OLE automation may be used either from Visual Basic or from C. Visual Basic is especially easy to use for OLE automation.

DDE

DDE support is present in the Windows version of Exchange, although we encourage you to use OLE automation instead, if possible. DDE messages allow you to open and close files, hide or show the application, execute a menu item, print a file, navigate in a document, and perform other tasks. DDE is the IAC mechanism that must be used to communicate with Windows plug-ins.

OLE server

The Windows version of Exchange is an OLE server, allowing you to embed PDF files into documents created in any OLE container application. Among other things, this allows PDF files to be embedded in Lotus Notes forms and displayed directly in Notes.

Notes/FX

The Windows version of Exchange supports the Lotus Notes/FX technology, allowing Notes to access some of the fields (such as title and subject) in an embedded PDF file. Notes can then use these fields in the same way it uses other fields, for example using the subject field to build a Notes view. Both one-way and two-way fields are supported: one-way fields can only be read into the Notes form from the PDF file, while two-way fields can be read back and forth between a Notes form and the PDF file. Changes made to one-way fields in Notes will not be reflected in the PDF file.

Interapplication Communication

There are three groups of fields available to use for integrating PDF documents into Notes forms through the Lotus Notes/FX technology. These groups are:

- Non-editable Acrobat-supplied fields—Fields that are defined as part of the structure of a PDF file and cannot be changed.

- Editable Acrobat-supplied fields—Fields that are defined as part of the structure of a PDF file and can be changed.

- User-defined fields—Fields that a developer or user has added to the structure of a PDF file. These fields can be changed.

IAC and plug-ins

Plug-ins can support their own Apple events and DDE messages. This allows you to make your plug-in's functions available to other applications, or to provide IAC access to plug-in API methods that are not normally available via IAC. On the Macintosh, it is possible to make a plug-in's Apple events available to AppleScript.

The Acrobat Search plug-in, described in the following section, supports its own Apple events and DDE messages in order to make a number of powerful cross-document search capabilities available to other applications.

Search plug-in

Mac and Windows search plug-ins support Apple events (only from C code, not AppleScript) and DDE. Using these, your application can:

- Submit a search query and display the results in the Acrobat Search plug-in's window. Options including word stemming, proximity, case sensitivity, sounds like, and the thesaurus can be specified.

- Get the list of indexes currently in use.

- Add an index to the index list.

- Remove an index from the index list.

- Get the title of an index.

- Get the pathname to the index.

Adobe PostScript™

Interapplication Communication

Conclusion

Acrobat Exchange has a powerful and flexible IAC interface that complements the plug-in API. Using IAC, you can control Exchange in virtually any way a user can. In addition, you can have Exchange render PDF files into a window in your application, providing a straight-forward way to integrate Acrobat into your existing application or system.

You are not required to choose between implementing your solution using IAC or as a plug-in. Some solutions are best implemented as a combination of IAC and plug-in, in which an application uses IAC to communicate with a plug-in, and the plug-in uses the full plug-in API to manipulate Exchange and PDF files.

The Acrobat SDKs contain detailed information on the IAC support in Exchange, as well as sample code. Contact the Adobe Developers Association for availability and pricing. §

Questions Answers

Q My application creates both PostScript language and PDF file output. I would like to add an "Embed Fonts" feature to allow users to include fonts in these documents. Is this legal?

A Adobe permits embedding certain typefaces into documents for the purposes of viewing and printing only. A list of type foundries with whom Adobe has the rights to grant the ability to embed is included below. Documents with embedded typefaces may not be edited unless those embedded typefaces are licensed to, and installed on, the computer doing the editing. The reason for this is that editing the file adds value to the file. For this, the customer is obtaining value from the typefaces and is therefore required to have a valid license for the typefaces.

In general, embedding certain typefaces is legal, but the embedded typefaces cannot be used to edit documents or accessed for use in other documents. Printing to a PostScript language file and embedding typefaces into PDF files are legal ways to embed typefaces. Sending typefaces on floppies along with print jobs is not legal, since they can be installed and used as typefaces in other documents.

If you plan to support an "Embed Fonts" feature in your application, we recommend that you document the above legal restrictions for your users in your application's Users' Guide.

Q Which typefaces may be embedded in documents if the above rules are followed?

A Typefaces in the Adobe Type Library that were created by Adobe, Linotype, ITC, Monotype, Agfa, and Fundicion Tipografica Neufville may be embedded in a document file, for the purpose of viewing and printing the document. Adobe is negotiating with all its other type foundries to obtain this permission. Until the contracts with the other type foundries are final, users must contact the company from which they licensed their typefaces to determine if embedding is legal.

Q Our printer has two paper trays. We use a custom application in-house to print out mailings to our customers on company letterhead. I have placed plain letter size, white paper in the upper tray, and replaced the ledger size paper in the lower tray with letter size letterhead stationery. I sent the following code to set up the printer with this information:

```
true 0 startjob
<< /InputAttributes
    <<
        0 << /PageSize [612 792] /MediaType (plain) >>
        1 << /PageSize [612 792] /MediaType (letterhead) >>
        /Priority [1 0]
    >>
>> setpagedevice
```

Then I indicated that the printer should print on letterhead paper by putting the following setpagedevice call at the beginning of my PostScript language print job:

```
<</PageSize [612 792] /MediaType (letterhead) >> setpagedevice
```

But, the printer prints on the plain paper in the upper tray. What's wrong with this code? Why doesn't it work?

A Your code would work as you expected it to on a printer with two letter size paper trays. The problem is that the lower paper tray on your printer is a ledger paper tray. Most printer's paper trays have a mechanism whereby the print engine can sense what size paper is in the trays. Your printer thinks that you have ledger paper in your lower tray (tray 1), because you are using a ledger size paper tray. So, there is a conflict between the page size indicated by your **setpagedevice** call and the page size values that the printer retrieved from the paper trays. In this case, if the tray has sensors that tell the print engine what size paper is present, there is no way to override the InputAttributes values with a call to **setpagedevice**. Since you have requested letter paper, your job prints from paper tray 0 which is a letter paper tray. Some printers come with *universal* paper trays, which support a variety of paper sizes. You may want to check with your printer manufacturer to find out if you can purchase an additional letter tray or a universal tray for your printer.

Questions & Answers

Q **To follow up on my previous question, I verified that the printer's attributes indicate that the lower tray contains ledger size paper. As a matter of fact, I found that the printer is completely ignoring my setpagedevice call to set new InputAttributes. I sent the following code to the printer:**

```
/GetAtt {
    currentpagedevice /InputAttributes get
    {
        exch ==
        dup type /dicttype eq
        { { exch == ==} forall
        }{
                        ==
        } ifelse
    } forall

} def
```

Then I called setpagedevice to set InputAttributes as I described earlier. After that, I call my GetAtt procedure and verified that the printer's attributes are unchanged. Even the Priority value is unchanged. Why does the printer completely ignore my request, without even giving me an error message?

A Your code tries to set the InputAttributes in tray 1 to an illegal value. The tray only accepts ledger paper and you are requesting letter size paper. When we request an unavailable feature in a PostScript language program, we usually expect to receive an error message. But, for the **setpagedevice** operator, the Policies dictionary determines what should occur if a request cannot be satisfied. To find out the Policy for InputAttributes, send down the code:

```
currentpagedevice /Policies get
/InputAttributes get ==
```

If you get an **undefined** error on **get**, then no policy has been defined for InputAttributes. In that event, the general policy for all features that have no designated policies is used; this is the PolicyNotFound policy. To find out what this is for your printer, you may send down the following code:

```
currentpagedevice /Policies get
/PolicyNotFound get ==
```

A returned value of '1' means "Ignore the feature request." This is the usual default policy in most products. If you received the value '1' from the printer, that explains why your request was ignored and you received no error message. If you would prefer to receive a configuration error message when an InputAttributes request cannot be satisfied, you can set the InputAttributes policy on your printer with the following code:

```
true 0 startjob
<< /Policies
    <<
        /InputAttributes 0
    >>
>> setpagedevice
```

Note that most of the above code samples assume that you have bidirectional communication with the printer, such as over AppleTalk. If you are not working in such an environment, you need to modify the code samples, so that instead of sending information up the backchannel via the **==** operator, you can print out the values from the stack onto a page on the printer.

The material discussed in these two Q&As is covered extensively in section 4.11 of the *PostScript Language Reference Manual, Second Edition.* §