Adobe Developers Association

ADA news

In This Issue

Volume 4, Number 3

Adobe

New Programs for Developers of Graphic Applications Plug-ins

Adobe has added several programs to help developers create and market third party plug-ins for Adobe's graphic applications. Developers writing plug-ins for Adobe Photoshop™, Adobe Illustrator™ and Adobe Premiere™ will find specialized programs in 1995 to meet their needs of getting a product to market. These programs range from development support to third party marketing opportunities.

The Graphic Applications Plug-in Developers Program

New from the Adobe Developers Association is Adobe's Graphic Applications Plug-in Developers Program (GAP Program). This program is designed to provide plug-in developers with the most up-to-date information and tools to be successful developing products for Adobe Illustrator, Adobe Photoshop and Adobe Premiere. This program is tailored for the graphic applications developer and includes the following benefits:

Technical Support

Adobe Developer Support has added specialized support engineers for each product to assist graphic applications developers through their development cycle. Developer technical support may be provided via phone, e-mail or fax. As a member of this program, the developer is entitled to five developer technical support cases per year with additional cases available for purchase.

Discounts

Members of this program will also receive substantial discounts on Adobe application programs such as Adobe Photoshop, Adobe Illustrator and Adobe Premiere. Some restrictions may apply.

Graphic Application Plug-in Development Kits

Plug-in kits for Adobe Photoshop, Adobe Illustrator, and Adobe Premiere include documentation and sample code that provide the plug-in developer with everything needed to take advantage of the full set of functionality within the plug-in interfaces.

Regular Updates and Upgrades

All members of the GAP program will automatically receive timely updates for all of the plug-in developer kits.

Plug-in Development Kits

Adobe has plug-in development kits for its vector and raster application programs on both Macintosh® and Windows® platforms. The development kits represent the current shipping product's plug-in interface, giving developers the most current tool for their development needs.

Adobe's plug-in development kits contain documentation, header files, and sample source code to be used in creating plug-in filters for Adobe application products. The development kits are distributed in diskette form and documentation on the disk is in Acrobat format.

continued on page 2

How To Reach Us

DEVELOPERS ASSOCIATION HOTLINE:

U.S. and Canada:
(415) 961-4111
M–F, 8 a.m.–5 p.m., PDT.
If all engineers are unavailable, please leave a detailed message with your developer number, name, and telephone number, and we will get back to you within 24 hours.

Europe:
+31-20-6511-355

FAX:

U.S. and Canada:
(415) 967-9231
Attention:
Adobe Developers Association

Europe:
+31-20-6511-313
Attention:
Adobe Developers Association

EMAIL:

U.S.
devsup-person@mv.us.adobe.com

Europe:
eurosupport@adobe.com

MAIL:

U.S. and Canada:
Adobe Developers Association
Adobe Systems Incorporated
1585 Charleston Road
P.O. Box 7900
Mt. View, CA 94039-7900

Europe:
Adobe Developers Association
Adobe Systems Europe B.V.
Europlaza
Hoogoorddreef 54a
1101 BE Amsterdam Z.O.
The Netherlands

Send all inquiries, letters and address changes to the appropriate address above.

Graphic Applications Plug-ins

Documentation and sample code on how to create native PowerPC™ plug-ins are also included in the development kits.

For a further discussion on developing Adobe Illustrator plug-ins, please see Volume 3, Issue 12 of the *ADA News*. For Adobe Premiere developers, Volume 4, Issue 2 contains detailed information on developing Adobe Premiere plug-ins. The Adobe Photoshop plug-in interface is introduced in the *Developing with Adobe Photoshop* column in this issue, on page 10.

Other Developer Support Activities and Services

Developer Camps

Over the course of the year, Adobe will be sponsoring Developer Camps for each of the graphic application products and platforms. These camps are generally two to three days long and cover pertinent information for developing leading edge plug-ins for Adobe™ products. One of the key benefits to the developer in attending developer camps is that there is a high level of interaction directly with the development team. In addition to working directly with the engineering teams, a certain level of camaraderie usually occurs among the other developer attendees.

As a member of the GAP program, you will be made aware of future developer camps as they are scheduled.

Graphic Applications Plug-in Registry

The Adobe Developers Association will also be implementing a plug-in registry for graphic applications plug-ins. The purpose of the registry is several fold. Primarily it will function as a way to efficiently track current shipping plug-ins. This could directly impact the sales and exposure of a given plug-in. If, for example, an Adobe customer were considering developing a specific plug-in and found through the registry that a plug-in with the desired functionality already existed, the customer may choose to purchase the plug-in from the original developer in lieu of going through the development cycle in-house. A further role the registry would play is to keep developers abreast of already existing plug-ins. This would give the developer a better idea of where other opportunities may lie. The contents of the registry may also be made available to end users through special promotions and mailings, offering an additional channel for the plug-in developer.

Third Party Marketing Opportunities

The Adobe Plug-in Connection

Adobe has launched the Adobe Plug-in Connection (formerly the Aldus Developers Cooperative) as a means of getting the developer's message and plug-in product to potential customers. The Connection is a cooperative association, owned and operated by member developers. Its purpose is to provide high quality, cost efficient marketing, sales and fulfillment

support to developers who engineer and manufacture plug-ins or other products that add unique and specific value to Adobe's product line.

Membership in the Adobe Plug-in Connection gives the developer access to the people most likely to buy plug-ins—Adobe customers who own Adobe Photoshop, Adobe Illustrator or Adobe Premiere. A few of the vehicles that the Connection provides are: a bi-monthly catalog with a distribution rate that will reach over two million Adobe users in 1995; trade show participation where developers can show their product and distribute literature at major industry venues; mailings and other product line specific marketing activities.

The Connection also provides a sales arm for the developer. Its staff can coordinate the order and fulfillment process as well as manage the electronic mailbox, the funds received, and the customer database.

On the whole, the Connection provides marketing and sales for the busy developer that cannot afford the time or manpower to get a product to market quickly and efficiently.

For more information on the Adobe Plug-in Connection, please contact David Appel at (206) 343-4221.

Public Relations

At appropriate trade shows or industry conferences, Adobe will be providing third party press kits that hold developer press releases. Although Adobe will assist the developer by providing simple press release templates, the developer would still be required to create the final press release. We hope that this will be an excellent vehicle to give our developers strong visibility in the press. Adobe's own internal public relations department will also be including success stories of third party developers in its press releases as it relates to a given product line.

Conclusion

In 1995 the Adobe Developers Association will be strongly poised to help third party developers achieve a better level of success. By providing programs and services such as the Graphics Applications Plug-in Developers Program, the Adobe Connection and other marketing opportunities, the third party developer will find a complete solution to developing and shipping plug-ins for Adobe applications. Regular updates on programs, classes and other opportunities will be made available to GAP members. If you would like to join the GAP program, please contact the Adobe Developers Association.

The article above outlines 3rd party marketing programs for plug-in developers for the Adobe Photoshop, Adobe Illustrator, and Adobe Premiere software programs. If you develop plug-ins for other Adobe products, such as Adobe™ Acrobat™ or PageMaker™, please stay tuned to future ADA mailings for more information about 3rd party marketing programs in your technology areas. §

PostScript Language Technologies

If you've read any technical notes relating to Level 2 PostScript™ language features, then you've no doubt encountered emulation code and self-configuring code. Emulation code is typically Level 1 code that emulates a specific Level 2-only feature. Self-configuring code generally evaluates the environment in which the code is running and then defines different code to be used for different implementations. For example, in the case of Level 2 forms, the self-configuring code would determine whether the environment is Level 1 or Level 2. Then, if running in a Level 2 environment, the built-in forms mechanism and operators would be used; if running in a Level 1 environment, the forms emulation code would be used.

Emulation code for working with Level 2 forms and patterns is provided in the PostScript SDK, specifically in technical notes #5113, "Emulation of the execform Operator", and #5112, "Emulation of the makepattern and setpattern Operators". However, emulation code for storing and retrieving these resources is not provided there. Below we offer some sample code for the **defineresource** and **findresource** operators. This code can be used for resources such as forms, patterns, and procedure sets, in a mixed Level 1/Level 2 environment.

```
%%BeginResource: file Resource_Emulation 1.5 0
 /defineresource where {
     pop /DR /defineresource load def
     /FR /findresource load def
 }{
     userdict begin
     /DR {                       % /ResName dict /Category DR dict'
            userdict /Resources  % ... /Category userdict /Resources
            2 copy known {
               get begin         % ... /Category
            }{
               15 dict dup       % ... /Category userdict /Resources dict15 dict15
               begin put         % ... /Category
            } ifelse             % /ResName dict /Category

            exch readonly exch   % /ResName dict' /Category
            currentdict 1 index known not {
               dup 30 dict def
            } if                 % /ResName dict' /Category
            load                 % /ResName dict' CategoryDict
            3 -1 roll            % dict' CategoryDict /ResName
            2 index put          % dict'
            end                  % dict'; Resources now off dict stack
     } bind readonly def

     /FR {  % /ResName /Category FR dict
            % Note: either of the two lines following could raise undefined.
            % This is in accordance with the Red Book

            userdict /Resources get         % /ResName /Category dict
            exch get                        % /ResName CategoryDict
            % If resource not defined, following line incorrectly raises
            % 'undefined instead of 'undefinedresource.
            exch get                        % dict

     } bind readonly def
 } ifelse
%%EndResource
```

To use the above code, just put the code sample in your prolog and then use 'DR' instead of **defineresource** in your script to define a resource. For example to define a form, you would make the following call, with a form dictionary on the stack:

```
/formname exch /Form DR pop
```

Then, use 'FR' later, instead of **findresource**, to retrieve the form resource, as follows:

```
formname /Form FR
```

Note that the above code should *not* be used for defining and retrieving font resources; the **definefont** and **findfont** operators can be used for this purpose in any environment. Named resources are documented in section 3.9 of the *PostScript Language Reference Manual, 2nd Edition.* §

Developing With

Adobe PageMaker

As mentioned in last month's column, PageMaker Additions are now referred to as plug-ins. In the source code excerpts below, we have left the term 'Addition' for now, since that is what you will see in the code until the next release of the SDK.

Last month we discussed plug-in registration in Adobe PageMaker on the Macintosh platform. As promised, this month's column addresses plug-in registration on the PC platform. If you are unfamiliar with the registration process, please see last month's column for some basic information on the subject.

The way that registration is done on the PC will change somewhat with the next release of the PageMaker Plug-ins SDK. The basic concept is maintained, but less files are used, which will hopefully simplify the development process.

The Old Way

The header file, *cqaddi.h* contains the data definitions and declarations used to register a plug-in:

```
/*
*     data definitions
*/
#define RI_REGINFO    101                  // resource id for registration info
#define ADNI          301                  // resource type for binary table

/*
*     data declarations
*/
// Addition registration resource consists of a header (ADDIHEAD)
// followed by an array of ADDINFO (one for each Addition in the DLL)

typedef struct
{
       LONG   fill;                        // reserved for future use
       WORD   versInterface;               // Interface version
       SHORT  count;                       // Number of Additions
} ADDHEAD;

typedef struct
{
       DWORD  dwStr;                       // string id# for menu name
       unsigned long int bAppear:1;        // TRUE if Addition
                                           // appears in menu.
       WORD   versAddition;                // Addition version
       WORD   reserved1;
       LONG   lFunction;                   // Function ID
       LONG   reserve2;
} ADDINFO;
```

The PageMaker Plug-ins SDK contains a couple of **addi.c* files that we suggest you copy and reuse in your code. The plug-in registration information is compiled from this C file rather than resource statements because a more complex structure can be used. At some point in the MAKE process this file needs to be compiled into an *.exe* and then converted to a *.bin* by using EXE2BIN and then linked into the resource file (*.rc*). This is how the plug-in's registration information becomes a part of the DLL resource information and that's where the Adobe PageMaker application reads the information from, in order to register and load the plug-in.

The New Way

With the next release of the PageMaker Plug-ins SDK the structures used for a plug-in's registration information will be in the *.rc* file itself. This was done primarily because of the fact that in the past the *.mak* file could be modified to add instructions that compiled the **addi.c* files, convert them to *.bin* files and compile them into the resource (*.rc*) file. Now, with most development environments using projects, you can't edit them to add these instructions. In order to make it easier to build the plug-in, we added the information to the *.rc* file directly. This does have one drawback.

As the *.rc* file is delivered with the PageMaker Plug-ins SDK it is a text file and the registration information reads as follows:

```
// These are the codes for Addition Resources:
#define RI_REGINFO 101
#define ADNI 301
```

continued on page 6

Colophon

This newsletter was produced entirely with Adobe PostScript software on Macintosh and IBM® PC compatible computers. Typefaces used are from the Minion™ and Myriad™ families, all of which are from the Adobe Type Library.

Managing Editors:
Nicole Frees, Debi Hamrick

Technical Editor:
Jim DeLaHunt

Art Director:
Karla Wong

Contributors:
Nicole Frees, David Hackel, Jeff Matulich, Mike Mitchell, Michelle Sellars, Dave Wise

Part Number ADA0056 2/95

Adobe PostScript

Pagemaker column

```
// This is the declaration of the RI_REGINFO ADNI resource.
RI_REGINFO ADNI MOVEABLE //101 301
// Header:
BEGIN
0L          // reserved (LONG)
0x0100      // Interface Version 1.0 (WORD)
1           // number of Additions in DLL Addition library (SHORT)

// definition of Addition number 1:
1L          // string id# for menu name (DWORD)
1L          // 1 if Addition should appear in menu, 0 otherwise (unsigned long)
0x0203      // Addition Version 2.3 (WORD)
0           // reserved (WORD)
1L          // Addition number. (The first Addition get number 1) (LONG)
0L          // reserved (LONG)
// more Additions in the same Addition library are defined here:
END

// For each Addition in the Addition library, include a string resource
// for its menu name. The numbers of the string resource should correspond
// with the Additions' string id# numbers in the ADNI resource above.
STRINGTABLE
BEGIN
      1,            "Addition Framework..."
      // If you have more than one Addition in your Addition Library,
      // add more numbers and names here:
END
```

In this form it looks pretty much just like it always did. But if you want to use the resource editor in say Microsoft® Visual C++™ v1.5, when you saved the resource it will convert the above structure into its hex equivalent:

```
//////////////////////////////////////////////////////////////////////////////
//
// 301
//

RI_REGINFO 301 MOVEABLE PURE
BEGIN
    0x0000, 0x0000, 0x0100, 0x0001, 0x0001, 0x0000, 0x0001, 0x0000, 0x0203,
    0x0000, 0x0001, 0x0000, 0x0000, 0x0000
END
```

As you can see it's a bit more complicated to edit the registration information, however you can reference this data against the text version and identify the location of any data that you needed to change or modify. We recommend that you edit the resource file using the File-Open menu with the Open As: parameter set to Text. This will not convert the registration structure into its hex equivalent, making it easier to work with. §

Developing With Adobe Fetch

To increase the compatibility between Adobe™ Fetch™ software and your product, your application should support the direct import of data from Fetch. Fetch has three commands that allow users to send data to another application.

Command #1 The Copy command

For images, the Copy command generally copies a PICT preview of a single selected item to the Clipboard. Your user can paste the resulting PICT into a document using the standard Paste menu item. For EPSF files, Fetch includes the original PostScript language file data as a PICT comment within the PICT. MooVs are copied in MooV format. Sounds are copied in SND format. For all file types except PICS, Fetch provides the user with an option of selecting a portion of the file for copying. For PICS files, the Adobe Fetch application copies a single frame.

Command #2 The Copy Reference command

The Copy Reference command supports multiple selections. It copies file location references for selected catalog items to the clipboard.

The format in which information is copied to the Clipboard is the FTCH desk scrap type. The FTCH desk scrap type contains an array of records of type FetchRec—one record for each item copied. Each FetchRec record includes the full file path and file type for each original file. For a full explanation of the FTCH desk scrap type, refer to the Fetch Compatibility Toolkit.

In addition to the FTCH desk scrap, the Adobe Fetch application includes text or visual information, depending on the option the user selects from the Copy Reference submenu.

With the Include Text option, Fetch includes a TEXT string composed of each item's pathname. This string helps the user identify the file.

With the Include Thumbnail option, Adobe Fetch includes a PICT composed of grouped thumbnails—one thumbnail for each of the items selected in the catalog. Your application should either allow users to simply paste the entire PICT into

Fetch Operations		Your Application's Suggested Functionality			
Command	**Result**	**Paste Standard**	**Paste Reference**	**Paste Import File**	**Paste Thumbnails**
Standard Copy, (single selection)	Copies a PICT to the Clipboard	Paste a PICT	N/A	N/A	N/A
Copy Reference, Include Text option, (multiple selection)	Copies file references and TEXT strings to Clipboard	Paste TEXT of all selected file pathnames	Paste multiple references	Paste multiple files, importing each file	N/A
Copy Reference, Include Thumbnails option (multiple selection)	Copies file references and a single PICT containing grouped thumbnails to Clipboard	Paste a single PICT, which includes all selected thumbnails	Paste multiple references	Paste multiple files, importing each file	Paste multiple thumbnails
Send Reference (multiple selection)	Sends a Place event with a list of file aliases	N/A	Paste multiple references	Paste multiple files, importing each file	N/A

a document using a standard Paste command, or it should ungroup the PICT into individual thumbnails.

Command #3 The Send Reference command

The Send Reference command sends the Place Apple event from Adobe Fetch to your application. The event passes a list of file aliases of the original files for selected catalog items to your application. Note that this event does not write anything to the Clipboard. Like Copy Reference, this command also supports multiple selection.

Command action summary

The table on page 7 lists the Fetch import commands, the resulting Clipboard contents or Apple event, and suggested functionality you can implement to support each command.

Things to consider

For all the listed commands (other than standard Copy command), your application must be able to render the original file to the output device. To obtain the original file, use the file reference that is pasted in the Clipboard or passed via the Place event. Your application must also be able to handle a case in which the original file format is not supported by your application.

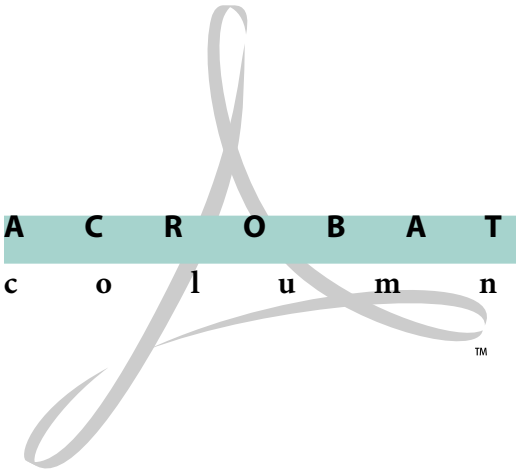
Your application can either import the file contents into a document or maintain a reference to the original file and use the file reference to generate a temporary screen display. By storing a file reference only, you will maintain a smaller file size. Storing a reference also allows the file in your document to be automatically updated when the original file is modified.

If you intend to support any Fetch commands, beyond the standard Copy command, you will need to decide how multiple items should be handled. These items include file references to original files, as well as the individual thumbnails that make up a grouped thumbnail PICT in the Clipboard.

Support for the Place event involves resolving all of the issues mentioned above. In addition, there are timing concerns to consider. Because the Place event is generated asynchronously with respect to the application, your application should maintain the received file references in a holding area. You can make the references available to the user via a pasteboard or palette interface, thus allowing the users to place items in the document individually at their discretion.

In Summary

The Fetch Compatibility Toolkit provides more information about these commands and gives suggestions on how to implement the user interface. The toolkit also comes with sample code which demonstrates how to parse the resources that the Adobe Fetch software copies to the Clipboard when the user executes the Copy Reference command. To order the toolkit, contact the Adobe Developers Association. §



Acrobat Exchange 2.0 Apple Event Support

Adobe Acrobat™ Exchange 2.0 for the Macintosh gives developers access to an extensive library of Apple event calls and objects so that applications can integrate more closely with the Acrobat viewer than ever before.

Acrobat Exchange provides 10 Apple event objects which support a total of over 40 Apple events. Acrobat Exchange is Scriptable, which is one of three levels of Apple event support described in the AppleScript™ Language Guide, Chapter 1. Acrobat Exchange is not Recordable or Attachable.

Developers can communicate with Acrobat Exchange to determine the current state of the viewer or a PDF file. They can also set preferences and add or remove menu items in order to customize the viewer. Acrobat Exchange can also display a PDF file into another application's window so that developers can create a custom interface that includes PDF information within their own application. A similar display-into-window interface is available in Microsoft Windows using OLE 2.0.

The following is an AppleScript example demonstrating several Acrobat Exchange 2.0 Apple events and objects. This example script queries the user for a rotation factor, asks for a folder containing PDF files, and then rotates each page of each valid PDF file found in the folder. It then saves and closes each file and finally exits the viewer.

```
tell application "Acrobat™ Exchange 2.0"
   -- turns off splash screen and open dialog
   set show splash at startup to false
   set open dialog at startup to false
   activate
   -- determines rotation factor to be applied
   display dialog ¬
      "Rotate Left 90, 180 degrees, or Right 90" buttons ¬
      {"Left", "180", "Right"} default button "Right"
   if (button returned of result = "Right") then
      set rot to 90
   else if (button returned of result = "Left") then
      set rot to -90
   else
      set rot to 180
   end if
   -- prompts user to enter a folder of PDF files
   set thefolder to choose folder with prompt "Choose folder of PDF files"
   set thelist to (list folder thefolder)
   repeat with fname in thelist
      set fname to ((thefolder as string) & fname)
      -- checks if creator type of PDF file is CARO
      if (file creator of (info for file fname) = "CARO") then
         open file fname
         set active doc to document 1
         -- repeats rotation factor for each page of the document
         repeat with i from 1 to (count document 1 each PDPage)
            set number of some AVPageView of document 1 to i
            set x to get rotation of PDPage (number of some AVPageView ¬
               of document 1)
            -- check that new rotation factor is between 0 - 270; reset if otherwise
            if (x + rot) > 360 then
               set rotat to (x + rot) mod 360
            else if (x + rot) < 0 then
               set rotat to ((x + rot) mod 360 + 360)
            else
               set rotat to (x + rot)
            end if
            set rotation of PDPage (number of some AVPageView of document 1) ¬
               to rotat
         end repeat
         -- closes all docs and saves them
         close all docs with saving
      else
        -- displays dialog if not a valid PDF file
         display dialog fname & " is not a valid PDF file." buttons {"OK"} default ¬
            button "OK"
      end if
   end repeat
   quit
end tell
```

The Adobe Acrobat Software Development Kit (SDK) includes the documentation and code samples needed to develop software using Apple events on the Macintosh, as well as DDE and OLE under Windows. This SDK contains several example AppleScripts that cause the viewer to perform various tasks, as well as a sample Apple events viewer written using THINK C™

continued on page 10

Acrobat Column

that demonstrates how to display a PDF page into the window of a custom application. Plug-ins can intercept the Apple event stream going to Acrobat Exchange to extend the viewer's Apple event suite or modify the viewer's current Apple event support. Information on creating plug-ins is contained in the Adobe Acrobat Plug-ins SDK. This SDK has everything found in the Adobe Acrobat SDK, plus the specifications and sample code for creating Acrobat viewer plug-ins using the Acrobat Exchange Plug-in API. Contact the Adobe Developers Association to obtain an Acrobat Developer Information Kit describing both the Acrobat SDK and Acrobat Plug-ins SDK.

Note to Developers: *If you write Acrobat plug-ins, you must register with Adobe for a unique identifier. Please contact the Adobe Developers Association by sending a fax to 415-967-9231. Please send the fax to the attention of the "Acrobat Plug-in Registry" and include the following information: company name, contact name, e-mail address, phone and fax numbers, as well as a brief description of your plug-in.* §

Developing With

Adobe Photoshop

Adobe Photoshop 3.0 is a high-end photo retouching, image editing, and color painting application. Its Application Programming Interface (API) enables developers to vastly extend its base functionality. Adobe Photoshop plug-ins allow third party vendors to read and write non-native file formats, acquire images directly from scanners or video frame devices, export images directly to hardware devices or save them in proprietary formats, and filter or modify selected portions of images. Plug-ins are installed by a user simply by placing them in the folder or directory designated in the user's preferences file.

The current versions of the plug-in development kits are supplied with example code and makefiles for MPW on Macintosh and Visual C++ on the Windows platform. However, it is possible to develop plug-ins using other development systems including those supplied by Symantec, Metrowerks and Borland. Future versions of the development kits will provide project files for the above environments.

The Adobe Photoshop APIs only support a modal interface. At anytime you can use standard operating system calls to do any interaction with the user. Documentation and sample code on how to create native PowerPC plug-ins for Adobe Photoshop is also included in the development kit.

For more information on developing Adobe Photoshop plug-ins please contact the Adobe Developers Association. §