Adobe Developers Association

ADA news

In This Issue

Volume 5, Number 3



## Adobe Acrobat—New Features

This month, we'll describe an exciting new feature of the Amber version of Acrobat Reader and Acrobat Exchange™ (generically referred to in this article as the Acrobat viewer) that enables your application to display PDF files. Using an Acrobat viewer and the PDFX library, your application can display PDF files without having the Acrobat viewer display its own window and menus. This allows your application to provide essentially seamless PDF file viewing in its windows.

In addition, we'll provide an overview of some changes to the PDF file format that optimize performance in on-line situations, and we'll describe a Web browser system that includes the PDFX library.

*Note: This article describes products that are still in development.Some changes from what is described here are likely. Nevertheless, we wanted to provide you with information about this new technology as early as possible.*

### What is Amber?

Amber is the code name for the next version of Adobe's Acrobat products. One of the features of this version is improved support for viewing PDF files in another application's window. Pre-release versions of the Amber versions of the Acrobat Reader are available in the Acrobat section of Adobe's Web site `www.adobe.com/`. Look in the Products section and point to Adobe Acrobat.

### What is PDFX?

PDFX is a library that works with the Amber version of the Acrobat viewer. It provides two main capabilities:

- It allows the Acrobat viewer to render and manage a live PDF document in another application's window (such as a Web browser's window). All of the normal Acrobat viewer capabilities such as document scrolling, hypertext links, annotations, and an optional Acrobat toolbar are available in your application's window.
- It allows another application to supply the Acrobat viewer with an interface for reading PDF files from arbitrary data sources such as Web servers.

The PDFX library is quite straightforward to use; it currently contains only eight functions (although this number will change before the final version ships).

*Note: The PDFX library itself does not provide PDF viewing; it merely provides a platform-independent interapplication communication (IAC) interface between your application and the Acrobat viewer. Whenever you wish to view PDF files in your application's window, an Acrobat viewer must be running.*

### A PDFX-based system

A system that wishes to use the PDFX library consists of your application, the PDFX library,

continued on page 2

Adobe

## 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
Europlaza
Hoogoorddreef 54a
1101 BE Amsterdam Z.O.
The Netherlands

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

## Acrobat—New Features

an Amber version of the Acrobat viewer, and the External Window Handler (EWH) plug-in for the Acrobat viewer. Figure 1 shows an overview of such a system.

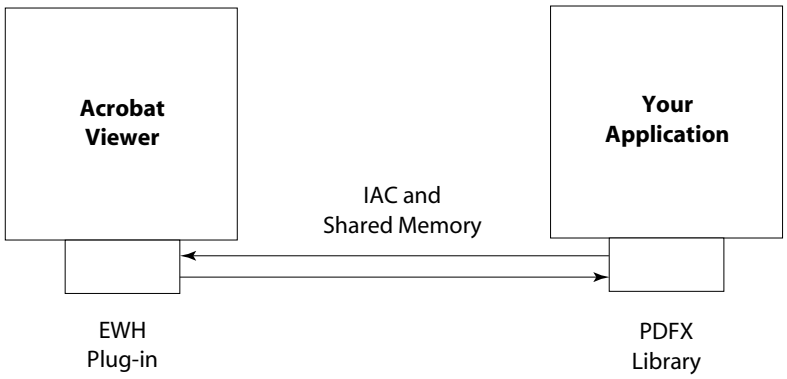


***Figure 1*** *Overview of Acrobat Amber integration*

### The PDFX library

The PDFX library provides a cross-platform C language API that allows you to draw PDF files in your application's window and support many of the Acrobat viewer's features such as displaying bookmarks, thumbnail images, and following hypertext links.

The PDFX library handles the details of the platform-dependent IAC that is used as the communication mechanism between the PDFX library and the Acrobat viewer, presenting a platform-independent API for your application to use.

#### Viewing and other functions

The PDFX library supports two modes for drawing a PDF file into your application's window:

- In full mode the file can be navigated using the Acrobat viewer's toolbar.
- In embedded mode, the PDF file is displayed as a part of another file, like a GIF can be displayed as part of an HTML page.

In addition to displaying the PDF file in your window, your application can ask the Acrobat viewer to perform a predefined set of operations on the file. This list includes:

- Printing the current PDF file.
- Finding text in the current PDF file.
- Copying the current PDF file's text or graphics selection to the clipboard.
- Handling a platform-specific event such as a mouse click.
- Deactivating the current session. This is useful, for example, if a PDF file is read from a remote site and the connection to that site is lost.

## Acrobat—New Features

**Alternate data sources**

The PDFX library optionally allows you to provide the PDF file's data instead of using a file system already registered by the Acrobat viewer. You must use this capability if your application obtains PDF files from a source other than those supported by the Acrobat viewer's registered file systems. For example, you must use this capability if your application retrieves data from the Web, via a modem connection, from a database, or generates PDF on the fly. You do not need to use this capability if your application accesses files on mounted disks.

If you choose to provide the PDF file's data, you register a procedure that the PDFX library calls each time the Acrobat viewer needs data from the PDF file. Your procedure is passed a list of one or more requests, each specifying a range of bytes that are needed from the PDF file. Your procedure must obtain at least these byte ranges and pass them to the Acrobat viewer using a PDFX library call. Your procedure is free to pass more data to the Acrobat viewer than the viewer requested. For example, you can send the entire PDF file to the Acrobat viewer if it is available.

**Using PDFX**

To use the PDFX library, you must link it into your application, then make calls to it while your application is running. The steps needed to open a PDF file and draw it in your application's window are:

- Your application makes a PDFX call to initialize the PDFX library and (optionally, as described in the previous section) specifies a procedure that the Acrobat viewer can call when it needs data from the PDF file.
- Your application makes a PDFX call to specify a PDF file that is to be displayed.
- Your application makes a PDFX call to send the Acrobat viewer a handle to the window in which the document will be displayed.
- (Only if you provided a data-reading procedure) The Acrobat viewer sends a message back to your application via the PDFX library, requesting that your application read bytes from the PDF file.
- (Only if you provided a data-reading procedure) Your application sends the requested bytes to the Acrobat viewer using a PDFX call.
- The Acrobat viewer displays the document in your application's window.

While the document is being displayed:

- Your application makes a PDFX call each time the window into which the Acrobat viewer is drawing changes size. This notifies the Acrobat viewer that it must redraw the window.
- Your application makes a PDFX call (as needed) to print the PDF file, copy its selection to the clipboard, find text, or handle an event.

continued on page 4

## Acrobat—New Features

- (Macintosh® only) Your application makes a PDFX call each time an update event or mouse click occurs in the window into which the Acrobat viewer is drawing. This call passes the event to the Acrobat viewer to handle.
- (Only if you provided a data-reading procedure) The Acrobat viewer continues to request bytes from the PDF file as it needs them and your application sends them to the Acrobat viewer using a PDFX call.

When you wish to close the file:

- Your application makes a PDFX call to stop the Acrobat viewer from displaying in your window and release any resources associated with that display.

When you are completely done using the PDFX library:

- Your application makes a PDFX call to shut down the PDFX library and release its resources.

### The External Window Handler plug-in

The Amber version of the Acrobat viewer includes a new Acrobat viewer plug-in, called the External Window Handler (EWH) plug-in, which implements the Acrobat viewer's IAC interface. EWH is the piece of the Acrobat viewer with which the PDFX library communicates. The EWH plug-in supports simultaneous rendering in multiple windows from one or more calling applications. The EWH plug-in is not compatible with version 2.1 and earlier of the Acrobat viewer.

At this point, it's useful to describe in more detail a system that uses the PDFX library; we've chosen to describe a Web browser, which is a common use for the PDFX library. Before we can do that, however, we need to describe a few enhancements that are being made to the PDF file format.

### Enhancements to PDF

The Acrobat viewer needs to be able to randomly access a PDF file's contents in order to read the resources (fonts, images, etc.) that are required to render a given page. To support this, every PDF file has an internal cross-reference table that specifies the location of every object in the file.

In order to provide good performance over relatively slow communication links, it must be possible to download and view one page of a PDF file at a time, rather than requiring the entire file (including fonts and images) to be downloaded before any of it can be viewed. Two enhancements make this possible: file linearization and hint tables.

#### File linearization

The order of objects within a PDF file can be optimized so that the bytes needed for a single page are near each other in the file. The data for the first page is placed near the beginning of the file, allowing the first page to be displayed immediately while the remainder of the file is downloaded.

## Acrobat—New Features

*Note: File linearization is not a change to the PDF file format. The PDF file format allows objects to be in almost any order.*

**Hint tables**

In order to support page-at-a-time viewing of PDF files, the PDF file format is being enhanced by the addition of hint tables that indicate which portions of a file are needed to view each page. Information about the structure of the hint tables will be published in the PDF file format documentation.

**Amber Viewer and optimized/enhanced PDF files**

*Note: This section assumes your application provides a procedure to read data from a PDF file.*

To take advantage of files that have been linearized and to which hint tables have been added, the Amber version of the Acrobat viewer initially requests the first 1kB of data when a PDF file is opened. This contains enough information for the viewer to determine which byte ranges are needed to draw the first page of the file.

The viewer then makes a single call that requests all byte ranges required to view the first page, and no more. The viewer begins drawing the page as soon as enough data has been downloaded.

Each time the user goes to a new page, the viewer requests the byte ranges needed to draw that page, unless they have already been downloaded and cached. The bytes needed may have already been downloaded either because the user previously visited the page, or because your application downloaded them in the background and passed them to the viewer while the user was viewing another page.

**Web browsers integration**

One common use of the Amber version of the Acrobat viewer is to provide PDF viewing in Web browsers. This section describes the components of a Web browser system that uses the Amber version of the Acrobat viewer, and lists several integration issues for improving the user experience in such a system.

Figure 2 shows a Web browser system. In addition to the components and requirements mentioned previously, the Web system contains a Web server. Also, both the Web browser and the Web server must have certain features in order to take full advantage of the Acrobat viewer's capabilities.

continued on page 6

## Acrobat—New Features

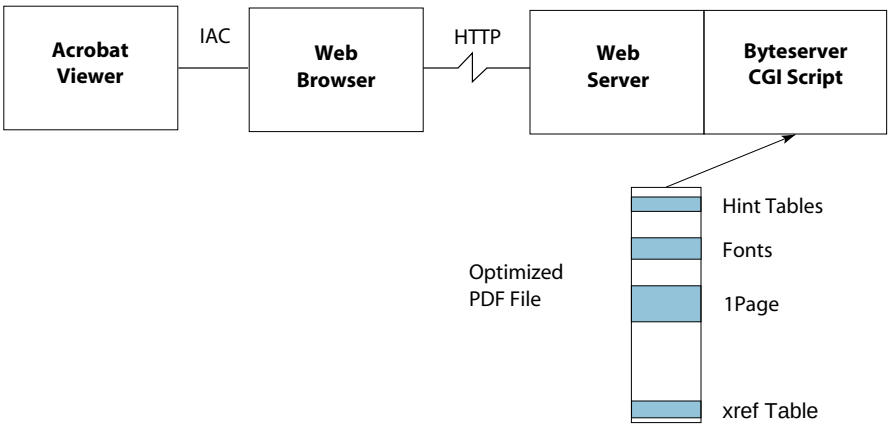


*Figure 2 Acrobat Amber viewer Web integration*

### Page at a time downloading

As described previously, optimizations and enhancements have been made to the PDF file format to improve performance over slow communication lines. These allow the Acrobat viewer to specify, in a single request, all the byte ranges needed to view a page.

Supporting page at a time downloading requires several components:

- Optimized PDF files containing hint tables.
- A Web browser that supports a proposed HTTP extension for specifying byte ranges in a file to be downloaded.
- A Web server that supports byte range downloading using a CGI script or other mechanism.

### Byte range support

The byte range support is based on an IETF Internet-Draft proposal: Byte Range Extension To HTTP.

*Note: This proposal is available from* `http://ds0.internic.net/internet-drafts/`, *as* `draft-luotonen-http-url-byterange-02.txt`

### Web server support

To support page at a time downloading, Web servers must be able to service byte range requests using a CGI script or some other mechanism. Because the Acrobat viewer contains the intelligence to determine which bytes must be downloaded for each page, the byteserver CGI script is very simple (less than 100 lines of PERL). Its only responsibility is to parse the HTTP request, extract the specified bytes from the file and return a properly formed HTTP response.

## Acrobat—New Features

Adobe intends to freely distribute an implementation of the byteserver CGI script. Netscape and some other server vendors have announced that they are planning to build byteserving support directly into their Web servers.

If PDF files are not optimized, and the Web server does not handle byte range requests, PDF files will be downloaded entirely before any pages may be viewed in the calling application's window. If the PDF files are not optimized, but the Web server handles byte range requests, the Acrobat viewer will attempt to approximate page at a time downloading, but without optimal efficiency.

**Web browser support**

A Web browser must supply a procedure for reading data from a PDF file. In addition, to support page at a time downloading, a Web browser must be able to issue byte range requests in response to a request for data from the Acrobat viewer. This allows the Web browser to retrieve the bytes for an entire page in a single HTTP connection with the Web server. As each byte range is returned in a multi-part MIME, the bytes should be sent to the Acrobat viewer so that it can begin incrementally drawing the page as bytes arrive.

**Integration considerations**

A goal of Acrobat Amber viewer's integration capabilities is to make PDF files a first class citizen in the Web browsing experience. To assist this, browsers should integrate PDF files into their goback stack, so that viewing a PDF file has the same go back semantics as viewing an HTML file.

Installation is another integration consideration. Ideally, the Amber version of the Acrobat Reader will be bundled with the Web browser when it is distributed on disk, or the Web browser will at least prompt the user to download and install the Acrobat Reader when the first PDF file is encountered on the Web.

### Conclusion

The upcoming Amber version of the Acrobat viewer provides exciting new opportunities for integrating PDF file viewing into your application. Developers who have purchased an Acrobat SDK can apply for pre-release copies of the PDFX library and documentation in order to get a head start on developing products that use the library. Contact the Adobe Developers Association for more information or to purchase the Acrobat SDK. §

# Developing With Adobe Photoshop

## Cross-application plug-in development

For many developers, expanding their plug-in to work in other applications across different markets seems cumbersome. Accounting for different code resources and requirements of multiple applications is a real burden for small, taxed research and development departments.

It doesn't have to be that way. Currently, a Filter plug-in module written to the Adobe Photoshop 3.0.5 Application Plug-in Interface (API) specification works with five different Adobe applications, expanding your opportunities beyond Adobe Photoshop software's primary "imaging and design" market.

Cross-application plug-in development is a win-win situation. We estimate that with 10-20% more development time for tasks such as adding specific resources and providing alternate code for older host-versions, a plug-in written for the Adobe Photoshop application can be optimized for the After Effects™, Adobe Illustrator®, PageMaker®, PhotoDeluxe™ and Adobe Premiere® products. That's an excellent return on your R&D investment—it's a great win for customers, who can use your plug-ins with all their graphics applications.

This column begins with our recommended cross-application plug-in development approach, and charts the current status of which API structures are supported by Adobe graphics applications. Future columns will provide more detailed instructions for adapting a plug-in to a specific application. If you just can't wait that long, all this information is available in the Adobe Photoshop Software Development Kit (SDK), and the individual SDKs for other Adobe products.

The different SDKs are available from Adobe's web site: `www.adobe.com`. Look in the Support & Services section and point to Developer Relations.

We recommend you follow the process for cross-application plug-in development outlined in Table 1.

**Table 1: Process for cross-application plug-in development**

| Step | Process | Example |
|---|---|---|
| 1 | Assess and determine the problem your plug-in will solve. | We need a Filter plug-in that makes every Nth pixel a specified color. |
| 2 | Acquire the primary SDK for your base development. | Download the Photoshop SDK from `www.adobe.com`. Look in the Support & Services section and point to Developer Relations. |
| 3 | Examine the examples and read the API specification in the primary SDK. | Read the initial chapters and the chapter on Filters in the Photoshop SDK. Browse through *PIFilters.h, PIGeneral.h* header files; read through example code for the Dissolve filter. |
| 4 | Determine your development strategy for your base application. | We'll produce a plug-in for both PPC and 680x0 machines, and we need it to work with version 2.5. |
| 5 | Read the information in the *Plug-in Resource Guide.pdf*, the Cross-Application Plug-in Development Resource Guide, with the needs of your plug-in in mind. | Originally, we were only going to parse filterRect and apply our coloring function. We're interested in dynamic resources, and are considering other features. |
| 6 | Reassess your development strategy for your base application. | Our programming goals have changed to include extensibility for changing saturation levels over a period, such as in a QuickTime® movie. We will need to modify our initial approach to include variables for saturation, instead of static values. |

**Table 1: Process for cross-application plug-in development (cont.)**

| Step | Process | Example |
|---|---|---|
| 7 | Determine any host requirements for the other target applications. | Because we want to be compatible with version 2.5, we will need to include alternate code that does our job without using the newer callback suites. We also need to make our saturation variable accessible to the dynamic resources that are used by the After Effects and Adobe Premiere applications. |
| 8 | Create and program your plug-in. | "NthPixelChange" by MySoftware, Inc. |
| 9 | Test under your base application. | Test "NthPixelChange" with the Adobe Photoshop 3.0.5 software. |
| 10 | Program and optimize based on your testing results. | We found that we can use the Image Services suite more extensively to help us calculate our changed pixels resulting in increased efficiency. |
| 11 | Test under the other target applications. | We added the 'ANIM' and 'FltD' resources and tested in the After Effects and Adobe Premiere software. |
| 12 | Modify and optimize based on those results. | Adobe Premiere doesn't support the Image Services suite, so we created a check and a branch to implement our original code, if the suite is unavailable. |
| 13 | Implement your beta-testing program. | A handful of associated development partners were given our plug-in to test. |
| 14 | Reassess and modify as needed. | Our beta testers reported bugs and we fixed the errors. |
| 15 | Package and release your product. | Initial product roll-out. |

Table 2 is a list of Adobe applications that support the API outlined in the Adobe Photoshop 3.0.5 SDK. Refer to the *Plug-in Resource Guide.pdf* or the individual product SDKs for specific implementation issues and emulation caveats.

**Table 2: Adobe Photoshop API host emulators**

| Software | Photoshop API version supported | Photoshop modules supported |
|---|---|---|
| After Effects | 3.0 | Filter, Format |
| Adobe Illustrator | 3.0.4 subset; Mac® only. | Filter, Format |
| PageMaker | 3.0.4 | Filter |
| Adobe PhotoDeluxe | 3.0.4 LE | Acquire, Export, Filter, Format |
| Adobe Premiere | 2.5 | Filter |

In future columns, we'll discuss specific implementation issues with the different major Adobe applications and offer detailed guidelines for making a plug-in forward- and backward-compatible. §

# PostScript Language Technologies

## Creating Color Separations in the RIP

A number of PostScript output devices, including all Level 2 imagesetters, are capable of color separating composite PostScript language files in-RIP (raster image processor). In this month's column, we give step-by-step instructions and sample code for separating a PostScript language file into CMYK process color plates on these types of devices.

### Determining that in-RIP separations are available

First, to determine whether the device you are printing to is capable of in-RIP separations, you may send the following code to the device:

```
/setpagedevice where { % on a Level 2 device
    pop
    2 dict begin
        % set things up to avoid a configuration error
      /Policies 1 dict dup /Separations 1 put def
        % and request separations
      /Separations true def
    currentdict end
    setpagedevice
        % check for Separations entry in the
        % page device dictionary
    currentpagedevice /Separations known

}{ % on a Level 1 device
    false
} ifelse
== % send value to back-channel
```

A value of true, returned by the above code, indicates that the output device is capable of creating separations in the RIP.

If your application or driver does not have bi-directional communication with the printer, you may parse its PPD file to check for the separations functionality. A *Separations keyword value of true indicates that in-RIP separations are supported; false indicates that in-RIP separations are not supported. You may also check the value of the *LanguageLevel keyword; if it is '1', then the device is not capable of in-RIP separations. If the *Separations keyword is absent from the PPD file, and the *LanguageLevel value is greater than '1', it may be very difficult to reliably determine whether or not a device supports in-RIP separations. For this reason, you may want to add a user-interface to your application to allow users to choose between RIP-based and host-based separations.

### Steps to produce color separations

After you've determined that the destination output device supports in-RIP separations, you may print the color separations by adding a few lines of set-up code to the composite PostScript language file, and sending it to the printer. The code must do the following:

1. Set the page device Separations key to true.

2. Set the page device ProcessColorModel key to /DeviceCMYK.

3. (optional) Set the page device SeparationColorNames array to the list of all process or custom (spot) color inks that your document contains. The names of the colorants of the native color space are included implicitly, regardless of the contents of the array; thus, the empty array [ ] is equivalent to [/Cyan /Magenta /Yellow /Black], after you have performed step 2, above.

4. (optional) Specify which separations to produce and the order that the separations should be output in, using the SeparationOrder page device key. Legal values are the names of the colorants of the native color space, as well as any additional names in the SeparationColorNames array. An empty array [ ] requests that separations for all colors of the native color space, as well as all additional colors in the SeparationColorNames array, be produced in some unspecified order. In our example below, we request that only the cyan and black process color separations be produced, in that order.

The **setpagedevice** requests in steps 1 through 4 should all be performed in one call, as shown in the example below. All of the **setpagedevice** keys referenced above are described in detail in the "PostScript Language Reference Manual Supplement" for versions 2014 and later, available from our web site:

www.adobe.com.

Look in the Support & Services section and point to Developer Relations.

### Benefits of in-RIP separations

Two benefits of using in-RIP separations are:

- Less data to send to the printer than with host-based separations.

  Regardless of the number of desired separations, the original composite job need only be sent to the printer one time. For instance, Example 1 describes a page containing two process colors: cyan and black. The composite page description is sent only once, and the output device produces two separations from this data, as shown in Figure 1.

- Flexibility in the types of documents that can be separated.

  Our Example 1 document contains objects painted with process colors; however, the technique described in this column may be used to separate any PostScript language file, regardless of the color space used for painting. Documents containing objects painted in RGB, a device-independent CIE-based color space, or any other color space supported by the PostScript language, can all be separated into CMYK process color separations by using the set-up code shown below.

### Sample Code and Output

**Example 1: Level 2 in-RIP separation example**

```
%!
%%BoundingBox: 100 100 172 172
%%DocumentNeededResources: font Times-Roman
%%+ font Helvetica
%%DocumentProcessColors: Cyan Black
%%EndComments
%%BeginProlog
%%EndProlog
%%BeginSetup
%%EndSetup
%%Page: 1 1
%%BeginPageSetup
% The following setpagedevice call performs steps 1, 2
% and 4. Step 3 is unnecessary for our example.
<<
   /Separations true
   /ProcessColorModel /DeviceCMYK
   /SeparationOrder [/Cyan /Black]
>> setpagedevice
%%EndPageSetup
%%BeginDocument: (testfiles/file1)
%!PS-Adobe-3.0
%%BoundingBox: 100 100 172 172
%%DocumentProcessColors:  Cyan Black
%%DocumentNeededResources: font Times-Roman
%%+ font Helvetica
%%EndComments
%%BeginProlog
%%EndProlog
%%Page: 1 1
%%BeginPageSetup
/pgsave save def
%%EndPageSetup
% The remainder of the code describes a black
% box with 4 rotated lines on it. The lines are
% painted with shades of cyan, ranging from
% 0% to 90% ink coverage, as shown in Figure 1.

100 100 translate
gsave
    72 120 div dup scale
    60 60 translate
    newpath -60 -60 moveto 120 0 rlineto
    0 120 rlineto -120 0 rlineto closepath
    0 setgray gsave fill grestore
    1 setlinejoin 1 setlinecap
    0 1 3 { % for
        0.3 mul 0 0 0 setcmykcolor
        5 setlinewidth
        newpath 50 -10 moveto
        -40 40 rlineto stroke
        90 rotate
    } for
grestore
%%PageTrailer
pgsave restore showpage
%%Trailer
%%EOF
%%EndDocument
%%Trailer
%%EOF
```

continued on page 12

## 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 from the Adobe Type Library.

Managing Editor:
**Jennifer Cohan**

Technical Editor:
**Nicole Frees**

Art Director/Designer:
**Karla Wong**

Contributors:
**Tim Bienz, Andrew Coven, Nicole Frees**




Part Number ADA00643 4/96

Adobe PostScript



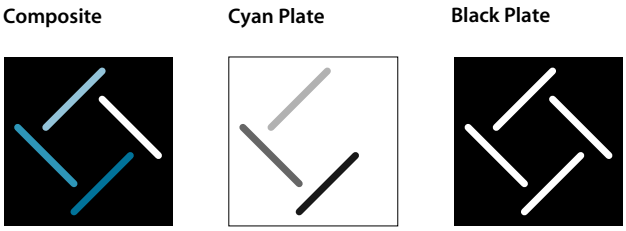


***Figure 1*** *Composite and Separation output produced from Example 1*

### Beyond CMYK separations

Example 1 shows how to create CMYK process color separations in-RIP from an arbitrary composite file. For issues concerning files that contain custom colors or overprinted inks, as well as guidelines for producing PostScript language files that are compatible with Level 1 host-based separation programs, please refer to technical note #5044, "Color Separation Conventions for PostScript Language Programs" available on our web site at: `www.adobe.com`. Look in the Support & Services section and point to Developer Relations. §