Embedding XMP Metadata in Application Files

DRAFT

June 6, 2002

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

Table of Contents

1 Embedding XMP Metadata in Application Files

1.1 Introduction

This document describes how XMP metadata can be embedded in the following file formats: TIFF, JPEG, GIF, PNG, HTML, PDF, SVG/XML, AI, PSD, and PostScript/EPS. This information is intended to help application developers understand how to embed XMP metadata in a standardized form to help ensure better document interchange.

This document is currently a stand-alone draft; when approved and finalized, it will become an appendix to the *XMP – Extensible Metadata Platform* document.

NOTE: This document assumes that the reader has a working knowledge of the referenced file formats, and does not attempt to fully describe each format or the terms used.

1.2 TIFF

In TIFF files, the XML Packet containing XMP metadata is pointed to by an entry in the Image File Directory (IFD). That entry has a Tag value of 700, as shown in Table 1.1, "TIFF IFD Directory Entry for XML Packets."

TABLE 1.1 TIFF IFD Directory Entry for XML Packets

Byte Offset	Field value	Field Tame	Comments
0, 1	700	TAG	Tag that identifies the field (decimal value).
2,3	1	Field type	The field type is "BTYE", which is represented as a value of "1".
4–7		Count	The number of values of the indicated type, or count of the indicated Type. For metadata, this is the total byte count of the XML Packet.
8–11		Value or Offset	The byte offset of the XML Packet.

References:

TIFF 6.0 Specification:

http://partners.adobe.com/asn/developer/pdfs/tn/TIFF6.pdf

1.3 JPEG

XML Packets embedded in JPEG files use an APP1 marker to designate the location of the XML packet containing the XMP metadata. Table 1.2 shows the entry format.

TABLE 1.2 Content of XMP JPEG APP1

Byte Offset	Field value	Field name	Length	Comments
0	0xFFE1	APP1	2	XMP application data marker
2	2 + length of namespace + length of XML Packet	Lp	2	Size in bytes of this count plus the following two portions
4	Null-terminated ASCII string without quotation marks.	namespace	29	XMP namespace URI, used as unique ID
	XML Packet			

The namespace used for XMP metadata in a JPEG file is:

http://ns.adobe.com/xap/1.0/

The Header plus the following data must be less than 64 KB bytes. The XML Packet cannot be split across the multiple markers, so the size of the XML Packet must be less than 64 KB minus the length of the header.

References:

(1) JPEG File Interchange Format Version 1.02

(2) ISO/IEC 10918-1 Information technology - Digital Compression and Coding of continuous-tone still images: requirements and guidelines.

(3) ISO/IEC 10918-4 Information technology — Digital compression and coding of continuous-tone still images: Registration of JPEG profiles, SPIFF profiles, SPIFF tags, SPIFF color spaces, APPn markers, SPIFF compression types and Registration Authorities (REGAUT)

NOTE: Reference (3) specifies the format of APPn markers and the file interchange format.

1.4 GIF

An XML packet in a GIF89a file should be encoded in an Application Extension block. The application identifier of that block should be ‘XMP Data’ and the Application Authenticator should be ‘XMP’. The XML Packet follows that block, encoded as UTF-8. That is followed by a 258 byte trailer, where the byte values are as shown in Table 1.3, beginning with 0x01, 0xFF, and then descending values until the final bytes with the values 0x01, 0x00, and the final byte (258) is the 0x00 Block Terminator. The XML Packet and “magic” trailer comprise what the GIF specification calls the “Application Data” portion of the Application Extension.

NOTE: The XML Packet in a GIF file must be encoded as UTF-8 for the magic trailer to work. The GIF Application Data portion actually consists of a series of GIF data sub-blocks. The first byte of each sub-block is the length of th e sub-block’s content, not counting the first byte itself. The series ends when a zero length byte is encountered. Thus to consume the Application Data portion you read a length byte. If it is zero you are done. If not zero, you consume that many bytes of data and continue.

When encoded as UTF-8 there will be no zero bytes in the midst of the XML Packet. Software that is unaware of XMP will view packet data bytes as sub-block lengths, skipping through the XML Packet. Eventually this will move into the magic trailer. The byte encountered there will casuse a skip to the Block Terminator at the end.,

References:

The GIF 89a specification is available at http://members.aol.com/royalef/gif89a.txt

TABLE 1.3 XMP in GIF File Format

	7 6 5 4 3 2 1 0	Field Name	Type
0	0x21	Extension Introducer	Byte
1	0xFF	Extension Label	Byte
0	0x0B	Block Size	Byte
1	'X' 0x58	Application Identifier	8 Bytes
2	'M' 0x4D		
3	'P' 0x50		
4	' ' 0x20		
5	'd' 0x64		
6	'a' 0x61		
7	't' 0x74		
8	'a' 0x61		
9	'X' 0x58	Application Authentication Code	3 Bytes
10	'M' 0x4D		
11	'P' 0x50		
	[XML packet]	XML Packet, UTF-8 encoded	Byte
	0x01	"Magic trailer"	258 Bytes
	0xFF		
	0xFE		
	⋮		
	0x01		
	0x00		
	0x00	Block Terminator	Byte

1.5 PNG

An XML packet can be embedded in a PNG graphic file by adding a chunk of type "iTXt". This chunk is semantically equivalent to the tEXt and zTXt chunks, but the textual data is in the UTF-8 encoding of the Unicode character set, instead of Latin-1.

The Chunk Data portion is the XML packet. The packet must be marked as read-only.

NOTE: XMP software that is not aware of the file format must not be allowed to change the content of the XML packet because of the CRC checksum following the chunk data. This is why the packet must be marked as read-only.

There should be no more than one iTXt chunk present in each PNG file. Encoders are encouraged to place the chunk at the beginning of the file, but this is not a requirement.

References:

The specification for the PNG file format can be found at:

http://www.w3.org/TR/REC-png.html

TABLE 1.4 PNG data format

Field	Length	Comments
Length	4	An unsigned integer giving the number of bytes in the chunk's data field. The length counts only the data field, not itself, the chunk type code, or the CRC.
Chunk Type	4	"iTXt"
Chunk Data:		Standard iTXt chunk header plus the XML packet.
Keyword	17	"XML:com.adobe.xmp"
Null separator	1	value = 0x00
Compression flag	1	value = 0x00, specifies uncompressed data.
Compression method	1	value = 0x00
Language tag	0	Not used for XMP metadata.
Null separator	1	value = 0x00
Translated keyword	0	Not used for XMP metadata.
Null separator	1	value = 0x00
Text	length of packet	XML packet
CRC	4	The CRC (Cyclic Redundancy Check) is calculated on the preceding bytes in the chunk, including the chunk type code and chunk data fields, but not including the length field.

1.6 HTML

Adobe recommends the use of W3C recommendations for embedding XML in HTML. This was the subject of a meeting held by W3C in May 1998; see *References*, below, for the meeting report.

Briefly, there are three approaches to embedding XML in HTML. Two use the SCRIPT element, which is probably recognized by older software. The third uses the XML element, which might not be recognized by older software. Recognition of the SCRIPT or XML element is important so that text representing the value of RDF properties is not displayed as page content. Use of the XML element is probably to be preferred, unless there are known incompatibilities with older software.

The SCRIPT or XML element may be placed in any legal location, it is suggested that it be at the end of the HEAD element. The content of the SCRIPT or XML element is the XMP packet.

NOTE: Adobe has noticed problems with using the SCRIPT element and LANGUAGE attribute in Microsoft Word 2000 running under Microsoft Windows XP, the body content may not be displayed. Use of the other two approaches seem to work fine in Microsoft Word 2000 running under Microsoft Windows XP. All three approaches work fine in Microsoft Word 2000 running under Mac OS X.

References:

The meeting report for the May 1998 W3C meeting may be found at:

http://www.w3.org/TR/NOTE-xh

Examples

A partial example of each approach is shown below:

EXAMPLE 1.1 Example using the SCRIPT element and LANGUAGE attribute

```
<html>
 <head>
  <SCRIPT LANGUAGE="XML">
  <?xpacket begin='' id='W5M0MpCehiHzreSzNTczkc9d'?>
  <!-- The serialized RDF goes here. It is removed for brevity. -->
  <?xpacket end='w'?>
  </SCRIPT>
 </head>
 <body>
 </body>
</html>
```

EXAMPLE 1.2 Example using the SCRIPT element and TYPE attribute

```
<html>
 <head>
  <SCRIPT TYPE="text/xml">
  <?xpacket begin='' id='W5M0MpCehiHzreSzNTczkc9d'?>
  <!-- The serialized RDF goes here. It is removed for brevity. -->
```

```
  <?xpacket end='w'?>
  </SCRIPT>
 </head>
 <body>
 </body>
</html>
```

EXAMPLE 1.3 Example using the XML element

```
<html>
 <head>
  <XML>
  <?xpacket begin='' id='W5M0MpCehiHzreSzNTczkc9d'?>
  <!-- The serialized RDF goes here. It is removed for brevity. -->
  <?xpacket end='w'?>
  </XML>
 </head>
 <body>
 </body>
</html>
```


1.7 PDF

For PDF files, the XML packet is embedded in a metadata stream (PDF 1.4) contained in a PDF object. Full documentation on metadata streams in PDF files is available in the *PDF Reference*, Version 1.3/1.4. This is available at:

http://partners.adobe.com/asn/developer/technotes/acrobatpdf.html

The following is a partial example of XMP metadata embedded as an XML packet, stored as a metadata stream:

EXAMPLE 1.4 Example of XMP Metadata in a PDF File

```
1152 0 obj
<< /Type /Metadata /Subtype /XML /Length 1706 >>
stream
<?xpacket begin='' id='W5M0MpCehiHzreSzNTczkc9d'?>
<!-- The serialized RDF goes here. It has been removed for brevity. -->
<?xpacket end='w'?>
endstream
endobj
```

NOTE: XMP software scanning for XML Packets without being aware of the file format should know that PDF files may have multiple packets that all look like the “main” XMP metadata. PDF files have an “edit by append” model. When modified a new

packet is appended without removing the old. Top level PDF dictionaries are also rewritten, so that only the new packet is seen by PDF savvy software.

1.8 SVG/XML

Since XMP uses the standard XML representation of RDF, the serialized XMP metadata can be directly embedded within an XML document. This can be with or without the XML packet wrapper, which is a pair of XML processing instructions. Using the XML packet wrapper is recommended, doing so will allow the metadata to be located by other software in a uniform manner.

The XMP metadata might have an outermost <x:xapmeta> or <x:xmpmeta> element, where x is the namespace “adobe:ns:meta/”. This is a serialization option. Next there will be an <rdf:RDF> element, inside of which are the <rdf:Description> elements containing the properties.

There is no XML declaration in the XML Packet, the XML Packet is not intended to be a complete standalone XML document. The XMP may be placed anywhere within the XML document. If the toolkit is asked to parse the entire XML document it is capable of finding the <rdf:RDF> element.

If DTD or XML Schema validation is required, be aware that RDF provides many equivalent ways to express the same model. Also, the open nature of XMP means that it is in general not possible to predict or desirable to constrain the allowable set of XML elements and attributes. There appears to be no way to write a DTD that allows arbitrary elements and attributes. Even use of the “ANY” requires declared child elements, see validity constraint #4 in section 3 of the XML specification.

The recommended approach to placing XMP in XML using DTD validation is to wrap the XML Packet in a CDATA section. This will require escaping any use of “]]>” in the packet.

It is recommended that the file be encoded as Unicode using UTF-8 or UTF-16. This provides compatibility for software that scans for XML packets and parses just their content.

References:

The XML specification is available at http://www.w3.org/TR/2000/REC-xml-20001006

1.9 Adobe Illustrator .ai Format

A “.ai” file generated by Adobe Illustrator® is in the Portable Document Format (PDF). Hence, the format for embedding XMP metadata is the same as for PDF files. See Section 1.7, “PDF” for information on PDF metadata embedding.

1.10 Adobe Photoshop .psd Format

Adobe Photoshop® .psd files contain image resource blocks, which are used to store non-pixel data associated with an image. The format of an image resource block is shown in Table 1.5

TABLE 1.5 Image Resource Block for Photoshop .psd File

Type	Name	Description
OSType	Type	Photoshop always uses its signature, 8BIM.
2 bytes	ID	ID = 1060 for XMP metadata.
PString	Name	A Pascal string, padded to make size even (a null name consists of two bytes of 0). For Photoshop 7, XMP metadata uses a Name value of “XMP”.
4 bytes	Size	Actual size of resource data. This does not include the Type, ID, Name, or Size fields.
Variable	Data	Resource data, padded to make size even.

NOTE: For the Name and Data fields in the above table, “padded to make size even” means that an extra zero byte is appended to the "raw" field value, if necessary.

1.11 PostScript and EPS

Metadata can be placed in a PostScript or EPS file for use in either PostScript or PDF workflows. This section focuses on its use for PDF workflows, and assumes that the primary goal is for the PostScript file to be converted to PDF by Acrobat Distiller.

XMP metadata can be embedded in a PostScript file in two ways, although only the first is supported by Distiller 5.0:

- The PostScript language `pdfmark` command can be used to set values that Distiller will put into the PDF document information dictionary (**Info**). How to use the PostScript `pdfmark` command for this purpose is explained below in Section 1.11.1, “Setting PDF DocInfo Metadata for PostScript Files.” The `pdfmark` command can also be used to attach an object-level XML Packet to a specific image or form in a PostScript file (for Distiller 4.0 or higher). See Section 1.11.3, “Object Level Metadata.” for more information.

- An XML Packet can be embedded into the PostScript file to describe document level metadata, but that use is not supported by Distiller 5.0. That means that you can embed the document-level XML Packet for use in non-PDF workflows, or possibly with future versions of Distiller. Section 1.11.2, "Saving XMP packets in PostScript/EPS Files" explains how to embed the XML Packet in a PostScript language file.

1.11.1 Setting PDF DocInfo Metadata for PostScript Files

There are two methods for putting metadata in a PostScript file so Distiller will put it in the PDF document **Info** dictionary and also create and embed an XML Packet for that data in the PDF document. You can use:

- DSC (Document Structuring Conventions) comments. The DSC comments are processed only if DSC parsing is enabled, that is, only if the job file contains the following line:

  ```
  /ParseDSCCommentsForDocInfo true
  ```

  Using DSC comments to create PDF DocInfo entries is discussed in "EPS files" on page 18. For more information on DSC comments, see the DSC specification:
  http://partners.adobe.com/asn/developer/pdfs/tn/5001.DSC_Spec.pdf

- `DOCINFO pdfmark` command. Information on `pdfmark` is available from the Distiller application Help menu, under "pdfmark Guide."

Because the `pdfmark` command is more reliable than DSC comments, many applications use it to set DocInfo properties for a PDF document. The following is an example of PostScript code, created by FrameMaker, which illustrates the use of the `DOCINFO pdfmark` operator:

```
/Creator (FrameMaker 6.0)
/CreationDate (D:20020214144924)
/ModDate (D:20020215142701)
/Author(John Doe)
/Title (Processing XMP Data in EPS Files)
/Subject (XMP)
/Keywords (XMP, pdfmark)
/DOCINFO pdfmark
```

Distiller will place these seven key words – plus one additional key "Producer" – into the resulting PDF file in two places: DocInfo and document level Metadata as an XML Packet. The Producer is the product name, e.g. "Acrobat Distiller 5.0 (Windows)." It is possible to add other Key/Value pairs to PDF DocInfo, but they are not added to the document level Metadata in Distiller 5.0.

Care must be taken if the file might be sent to a PostScript interpreter instead of to Distiller. Some PostScript interpreters may not recognize the `pdfmark` command, e.g. those in older

printers. One way to avoid problems is to conditionally define the `pdfmark` operator to the "`cleartomark`" operator:

```
/pdfmark where { pop true } { false } ifelse
/currentdistillerparams where {pop currentdistillerparams
/CoreDistVersion get 5000 ge } { false } ifelse
and not {userdict /pdfmark /cleartomark load put} if
```


1.11.2 Saving XMP packets in PostScript/EPS Files

XMP metadata must be embedded in a PostScript file in a way that it will be recognized by software that scans files for metadata, which means embedding the complete XML Packet. However, if that file were sent to a PostScript output device, the packet data would cause PostScript errors, and the job would fail.

To be able to handle arbitrary data, we need a procedure to read the XMP data from the current file, and discard the data if it is not intended to be interpreted. How you do that depends on whether both PostScript LanguageLevel 1 and 2 must be supported. The following sections explain how to handle these two cases.

With PostScript LanguageLevel 1, we can use the `readline` operator to read in the data. If PostScript LanguageLevel 1 support is not needed, the solution is simpler: the PostScript LanguageLevel 2 `SubFileDecode` filter can be used to read in the data.

Once the XMP data is read in using `readline` or the filter, we can save the data in a pdfmark named stream and the stream can later be attached to marked content for an EPS, or to an Image XObject, or a Form XObject.

NOTE: In the following sections, we define some procedures in a private dictionary like this:

```
privatedict /metafile_pdfmark {flushfile cleartomark} bind put
```

The name `privatedict` is for illustration purpose only. In the real product code, these procedures should be defined in a unique dictionary so that several EPS files can be used in one document and slightly different versions of these procedures can co-exist.

PostScript LanguageLevel 1

With PostScript LanguageLevel 1, there are at least two approaches: using `readstring` to read in the whole XMP packet, or `readline` to read in the XMP packet data line by line until an end marker is found.

We present the `readline` approach here. The `readline` approach solves two problems that exist for `readstring`:

- We don't have to know the exact size of the whole packet, just need to know the maximum length of the lines.

- The exact length of an XMP packet may change if the PostScript/EPS file is re-saved by a text editor with different line ending convention (`\n`, or `\r\n`).

Here is a sample showing how to save an XMP packet in a PostScript stream using the `readline` operator:

```
% Define pdfmark to cleartomark, so the data is discarded when consumed
% by a PostScript printer or by Distiller 4.0 or earlier.
%
/currentdistillerparams where
{pop currentdistillerparams /CoreDistVersion get 5000 lt} {true} ifelse
%
% All following references to “privatedict” should be changed to a
% unique name to avoid potential conflicts:
%
{privatedict /pdfmark /cleartomark load put
privatedict /metafile_pdfmark {flushfile cleartomark } bind put}
{privatedict /metafile_pdfmark {/PUT pdfmark} bind put} ifelse
%
% Define another procedure to read line by line from current file until
% marker line is found.
% On stack: [ {name} maxLineLength MarkerString
%
privatedict /metastring_pdfmark
{ 2 dict begin
/makerString exch def string /tmpString exch def
{ currentfile tmpString readline pop
makerString anchorsearch
{pop pop cleartomark exit}
{3 copy /PUT pdfmark pop 2 copy (\n) /PUT pdfmark} ifelse
} loop
end
}bind put
%
% Create a pdfmark named stream object in PDF to hold the data. It is
% recommended to use an application name and time stamp to generate a
% unique name {myApp_time}. The name has no significance in PDF - it is
% just a handle used to refer to the object in the PostScript program.
% We use {my_metadata_stream_123} as an example. A better way is to use
% NamespacePush and NamespacePop to bracket all pdfmarks.
%
[/_objdef {my_metadata_stream_123} /type /stream /OBJ pdfmark
%
% Read xpacket from the current file and put it as the contents of the
% stream just created. It will be discarded on real printers. Define a
% string with the exact size to hold the XMP packet: <LineLength> must
% be larger than Maximal length of the lines in the XMP packet.
%
```

```
[{my_metadata_stream_123} <LineLength> (% &&end XMP packet marker&&)
metastring_pdfmark
... XMP packet goes here, each line should be shorter than <LineLength>
bytes...
% &&end XMP packet marker&&
%
% At this point, the XMP data is saved in a stream object that can be
% referenced using the name {my_metadata_stream_123}. Set the stream's
% Type and subtype, both are required if it is used as a Metadata
% stream in PDF by some objects:
%
[{my_metadata_stream_123}
2 dict begin /Type /Metadata def /Subtype /XML def currentdict end
/PUT pdfmark
```

PostScript LanguageLevel 2

With the PostScript LanguageLevel 2 SubFileDecode filter, we don't have to know the length of the XMP packet before hand. We can just put a marker string after the packet and ask the filter to read until we reach the marker. Here is a sample that shows how to save an XMP packet in a PostScript stream using the SubFileDecode filter:

```
% Define pdfmark to cleartomark, so the data is discarded when consumed
% by a PostScript printer or by Distiller 4.0 or earlier.
%
/currentdistillerparams where
{pop currentdistillerparams /CoreDistVersion get 5000 lt} {true} ifelse
%
% All following references to "privatedict" should be changed to a
% unique name to avoid potential conflicts:
%
{ privatedict /pdfmark /cleartomark load put
privatedict /metafile_pdfmark {flushfile cleartomark } bind put}
{ privatedict /metafile_pdfmark {/PUT pdfmark} bind put} ifelse
%
% Create a pdfmark named stream object in PDF to hold the data. It is
% recommended to use an application name and time stamp to generate a
% unique name {myApp_time}. The name has no significance in PDF - it is
% just a handle used to refer to an object in a PostScript program.
% We use {my_metadata_stream_123} as an example. A better way is to use
% NamespacePush and NamespacePop to bracket all pdfmarks.
%
[/_objdef {my_metadata_stream_123} /type /stream /OBJ pdfmark
%
% Read xpacket from the current file and put it as the contents of
% the stream just created. It will be discarded on real printers.
%
[{my_metadata_stream_123}
```

```
currentfile 0 (% &&end XMP packet marker&&)
/SubFileDecode filter metafile_pdfmark
... XMP packet goes here ...
% &&end XMP packet marker&&
%
% At this point, the XMP data is saved in a stream object that can be
% referenced using the name {my_metadata_stream_123}. Set the stream's
% Type and subtype, both are required if it is used as a Metadata
% stream in PDF by some objects:
%
[{my_metadata_stream_123}
2 dict begin /Type /Metadata def /Subtype /XML currentdict end
/PUT pdfmark
```

Avoiding Name Conflicts

In the samples, we used the name *{my_metadata_stream_123}* and suggested that this actually be the application name plus a time stamp to avoid name conflicts when several EPS files are inserted in one document. A better solution is to use NamespacePush and NamespacePop pdfmarks. This is also the recommended solution in the Pdfmark Reference Manual (it is accessible from Distiller's Help menu.)

It is important to put all pdfmarks using the named objects in the same block bracketed by NamespacePush and NamespacePop pair, for example, the following PostScript code is bad:

```
[/NamespacePush pdfmark
[/_objdef {my_metadata_stream_123} /type /stream /OBJ pdfmark
[{my_metadata_stream_123}
currentfile 0 (% &&end XMP packet marker&&)
/SubFileDecode filter metafile_pdfmark
... XMP packet goes here ...
% &&end XMP packet marker&&
[/NamespacePop pdfmark
% At this point, the name {my_metadata_stream_123} is no longer usable.
% next line will cause "undefined" error:
[{my_metadata_stream_123} <</Type /Metadata /Subtype /XML>> /PUT pdfmark
```

1.11.3 Object Level Metadata

Metadata streams can be attached to specific objects in a PostScript file, such as images and forms, using the pdfmark operator. We'll show how to do that for Image and Form XObjects.

EPS files

If DSC parsing is enabled to embed EPS file information (/PreserveEPSInfo true) and we can detect an embedded EPS, marked content will be created surrounding the EPS and a Metadata

stream is attached to the marked content property dictionary. These five entries are parsed from the EPS DSC header and used in the Metadata: “Author”, “Creator”, “Title”, “Subject”, and “Keywords”.

Please note that DSC parsing can be unreliable. For example, the AdobePS5.2 printer driver puts the `%%BeginDocument` too far away from the actual EPS. The user can also disable DSC parsing. For example, the standard job option files *eBook* and *Screen* have `/PreserveEPSInfo` disabled, while *Print* and *Press* have it enabled.

Images

There is no direct support for creating Metadata for images, but a PostScript producer can attach a metadata stream to an image XObject using `pdfmark`. The idea is to name the next image in the PostScript stream using `/NI pdfmark`, and then attach a Metadata stream – as shown in Section 1.11.2, “Saving XMP packets in PostScript/EPS Files” – to the named image. We check for Distiller 4.0 in our definition of pdfmark because `/NI pdfmark` is available in Distiller 4.0 or later.

```
% Redefine pdfmark and create an XMP stream as shown earlier. Use
% pdfmark to name the next image as {myImage_123}. Again, a time stamp
% can be used here as the name.
%
[/_objdef {myImage_123} /NI pdfmark
%<Image goes here> - uses the Redbook turkey as our example
72 400 translate % Locate lower-left corner of square
120 120 scale % Scale 1 unit to 120 points
0 0 moveto % Define square
0 1 lineto
1 1 lineto
1 0 lineto
closepath
.9 setgray % Set current color to gray
fill % Fill with gray background
0 setgray % Set current color to black
24 23 % Specify dimensions of source mask
true % Set polarity to paint the 1 bits
[24 0 0 -23 0 23]
{< 003B00 002700 002480 0E4940
114920 14B220 3CB650 75FE88
17FF8C 175F14 1C07E2 3803C4
703182 F8EDFC B2BBC2 BB6F84
31BFC2 18EA3C 0E3E00 07FC00
03F800 1E1800 1FF800 >}
imagemask
% Here is the real work: Attach the stream as Metadata to the image.
[{myImage_123} <</Metadata {my_metadata_stream_123}>> /PUT pdfmark
```

Note that Distiller 5.0 always flate-compresses a stream created in this manner, making the XML packet invisible to normal text scanning.

Forms

There is, currently, no direct support in PDF for Forms created by the PostScript `execform` operator, but a metadata stream can be attached to Forms using "[/_objdef {myForm_123} ... /BP pdfmark". The same PostScript code used for an Image can be used here:

```
[{myForm_123} <</Metadata {my_metadata_stream_123}>> /PUT pdfmark
```