Adobe Acrobat 7.0.5

3D Annotations Tutorial

July 27, 2005

3D Annotations Tutorial

Introduction

Adobe PDF 1.6 introduced the capability for three-dimensional (3D) objects, such as those used by CAD software, to be embedded in PDF files.

Such 3D content can be viewed in the PDF file by means of *3D annotations*. The 3D objects can be manipulated by users (in Acrobat viewers, version 7.0 and later) as well as programatically through JavaScript.

The underlying 3D object data in a 3D annotation must conform to the Universal 3D (U3D) format developed by the 3D Industry Forum (http://www.3dif.org). Acrobat supports a subset of U3D, as described in the document *U3D Elements Supported by Acrobat*. (See "Supporting documentation" .)

This tutorial describes how to programatically create 3D annotations in PDF files. The code shown here can be used as part of an application developed using the Adobe PDF Library or an Acrobat plug-in using the Acrobat SDK. It is assumed that you already have the resources and basic information needed to create such software and can insert the code shown here.

The code illustrates how to perform the following tasks:

- Create a 3D annotation at a particular location on a PDF page. (See "Creating the annotation" on page 2.)
- Specify a U3D file as a 3D stream to be used in the annotation. (See "Specifying the 3D stream" on page 6.)
- Specify a JavaScript file to be used for animation or other purpose. (See "Specifying JavaScript code" on page 8.)
- Specify a particular view of the 3D objects to be displayed by default. (See "Setting the default view" on page 9.)
- Provide two-dimensional content that can be displayed as a poster prior to activating the annotation. (See "Setting the annotation appearance" on page 10.)

The code requires the following inputs:

- A U3D file
- Optionally, a 3D JavaScript file
- Optionally, a PDF file containing an image to be used as a poster

Supporting documentation

Refer to the following documents for information that is important in understanding the sample:

- The *PDF Reference, version 1.6* describes the PDF language in detail. It can be found at http://partners.adobe.com/public/developer/pdf/index_reference.html. Section 9.5 describes the structures needed to represent 3D annotations.
- The *Acrobat and PDF Library API Reference* describes all the methods used in this example. It can be found at http://partners.adobe.com/public/developer/acrobat/sdk/index_doc.html#plugins.

  The Acrobat and PDF Library API contains numerous methods for working with PDF documents:
  - Some methods operate on the most common items contained in PDF documents (such as pages, bookmarks, and annotations). For example, **`PDPageAddNewAnnot`** adds a new annotation to a page.
  - In addition, there is a set of methods for performing basic operations on the primitive objects that make up PDF documents (also called *Cos* objects). Many of the tasks described in this tutorial (such as adding entries to dictionaries and setting their values) require the use of Cos-level API methods. (See "Cos-level API methods" on page 4.)
- The *Acrobat 3D JavaScript Reference* describes the JavaScript APIs that can be used to add interactivity to 3D annotations. It can be found at http://partners.adobe.com/public/developer/acrobat/sdk/index_doc.html#js.
- *U3D Elements Supported by Acrobat* describes the subset of U3D supported by Acrobat 7.0. It can be found at http://partners.adobe.com/public/developer/acrobat/index_advanced.html.

Creating the annotation

This section shows the basic steps of creating an annotation in a PDF document. These steps apply to all annotation types, not just 3D annotations. The section "Adding 3D data to the annotation" on page 4 shows the additional steps needed to specify a 3D annotation.

First, the code creates a new PDF document (a **`PDDoc`** object) using the **`PDDocCreate`** method:

```
PDDoc pdDoc = PDDocCreate();
```

NOTE: There are other ways to obtain a **`PDDoc`** object, such as by opening an existing PDF file with a method like **`PDDocOpen`**.

Next, the code adds a single page to the document using the **`PDDocCreatePage`** method, which has the following parameters:

- A **`PDDoc`**.
- A constant indicating where to add the page (in this case, as the first page).

- A *media box* specifying the page size. In this case, the page is 8.5 x 11 inches. The dimensions of the media box must be of type **`ASFixed`** in units of *default user space* (72 units per inch). The **`fixedZero`** constant represents the number **`(ASFixed) 0x00000000L`**.

  ```
  PDPage pdPage;
  ASFixedRect mediaBox;
  mediaBox.left   = fixedZero;
  mediaBox.top    = Int16ToFixed(11*72);
  mediaBox.right  = Int16ToFixed(8.5*72);
  mediaBox.bottom = fixedZero;

  pdPage = PDDocCreatePage(pdDoc, PDBeforeFirstPage, mediaBox);
  ```

The next step is to create the annotation. In PDF, annotations are represented by an annotation dictionary. A *dictionary* is a data structure with one or more entries, which are *key-value pairs:*

- The key is a *name object*. (See section 3.2.4 in the *PDF Reference*.)
- The value is some type of PDF object. Section 3.2 in the *PDF Reference* describes all the PDF object types. If the value is a dictionary, that dictionary has its own key-value pairs. Therefore, dictionaries can be nested within other dictionaries, as you will see with the 3D structures.

General annotation dictionary entries are described in Table 8.11 in the *PDF Reference*. Those specific to 3D annotations are described in Table 9.33.

The following code creates a 3D annotation with corners (1, 9.5) and (7,4) using the **`PDPageAddNewAnnot`** method:

```
ASFixedRect annotRect;
annotRect.left   = Int16ToFixed(1*72); ;
annotRect.top    = Int16ToFixed(9.5*72);
annotRect.right  = Int16ToFixed(7.0*72);
annotRect.bottom = Int16ToFixed(4*72);

PDAnnot newAnnot = PDPageAddNewAnnot(pdPage, -2, ASAtomFromString("3D"),
&annotRect);
```

The parameters to this method are the following:

- The **`PDPage`** on which to place the annotation.
- An **`ASInt32`** indicating where to add the annotation in the page's annotation array. The value -2 is normally used and means the annotation is added to the end of the page's annotation array.
- The type of the annotation, which in this case is **3D**. This sets the value of the annotation dictionary's **Subtype** entry. It is important to note that in PDF this is a *name object*, not a string. In the API, the **`ASAtom`** type is frequently used to represent names; the **`ASAtomFromString`** method converts a string to a name.
- The rectangle in which the annotation appears on the page. This sets the value of the annotation dictionary's **Rect** entry.

After the annotation has been created, you must complete the other entries in the annotation dictionary. The **F** (flags) entry is set here:

```
PDAnnotSetFlags(newAnnot, pdAnnotPrint | pdAnnotReadOnly);
```

The annotation's initial appearance (the **AP** entry) can be set as described in "Setting the annotation appearance" on page 10.

Other entries are set as described in "Adding 3D data to the annotation" using the `EmbedDataIn3Dannot` function.

Adding 3D data to the annotation

The `EmbedDataIn3dAnnot` function embeds the 3D data and other information in the 3D annotation dictionary.

This function returns an error code and takes the following parameters:

```
bool EmbedDataIn3DAnnot(PDPage pdPage,
             PDAnnot theAnnot,         // 3D annot
             char* u3dFileName,        // Path to the U3D file
             char* JsFileName,         // Path to the JavaScript file
             bool  bCreate3DV)         // whether to create 3DV dictionary
```

This function requires Cos-level API methods, which are described in the next section.

Cos-level API methods

The Acrobat and PDF Library API does not contain methods for working specifically with 3D annotations. To add entries to a 3D annotation dictionary, you must use Cos-level API methods. These methods enable you to create PDF objects of any type (see section 3.2 in the *PDF Reference*) and set their values.

The Cos API uses two primary object types:

- `CosDoc` represents an entire PDF document.
- `CosObj` represents all other PDF objects: simple types like numbers and booleans as well as complex types such as dictionaries, arrays, and streams.

The API provides methods for creating and working with different object types. The creation methods are of the form `CosNew`*`Object`*, where *`Object`* is the object being created: for example, `CosNewDict`, which creates a dictionary object, `CosNewBoolean`, and `CosNewStream`.

The first two parameters for the `CosNew`*`Object`* methods (with the exception of `CosNewNull`, which takes no parameters) are these:

- The `CosDoc` object that represents the current document.
- A boolean that specifies whether the new object should be indirect. Indirect objects (see section 3.2.9 in the *PDF Reference*) are given object numbers and can be referred to

from more than one place in a PDF file. Direct objects are specified only where they are used and cannot be referred to from anywhere else.

The remaining parameters vary depending on the object type:

- For simple types, the third parameter specifies its value. For arrays and dictionaries, the third parameter specifies the number of elements.
- For streams, there are several additional parameters. See the *API Reference* for details on each method.

The return value of these methods is the newly created object. In the API, this object is always a `CosObj`. That is, even though you call `CosNewDict` to create a dictionary object, and you use methods like `CosDictPut` to work with dictionary objects, there is not a formal object type called `CosDict`.

If necessary, you can determine the type of a `CosObj` by calling the `CosObjGetType` method, which returns a constant (`CosNull`, `CosInteger`, `CosFixed`, `CosReal`, `CosBoolean`, `CosName`, `CosString`, `CosDict`, `CosArray`, or `CosStream`).

Most of the code described here involves setting the entries of dictionaries (`CosObj`s of type `CosDict`). Dictionaries contain a number of key-value pairs, where the key is a *name object* and the value is any type of `CosObj`.

There are several common methods that can be used, which differ only in how the keys are specified. The `CosDictPutKeyString` method (available in Acrobat 7.0 or later) allows the key to be specified as a string and is the recommended method, as in the following example:

```
CosDictPutKeyString(theDict,    // the dictionary
     "TheKey",                  // the key: a string
     theCosValue);              // the value: a CosObj
```

`CosDictPut` requires the key to be specified as an `ASAtom`. `CosDictPutKey` requires the key to be a name object (a `CosObj` of type `CosName`).

Creating the 3D annotation dictionary entries

To work with annotations at the Cos level, you must get the Cos object corresponding to the `PDAnnot`, as follows:

```
CosObj cosAnnot = PDAnnotGetCosObj(theAnnot);
```

and the `CosDoc` corresponding to the document:

```
CosDoc cosDoc = CosObjGetDoc(cosAnnot);
```

Two additional dictionary entries (which are not specific to 3D annotations)—the **P** (page) and **Contents** entries—can be set as follows:

```
CosDictPutKeyString(cosAnnot, "P", PDPageGetCosObj(pdPage));
CosDictPutKeyString(cosAnnot, "Contents",
              CosNewString(cosDoc, false, "3D Model", strlen("3D Model")));
```

The following sections show how to set the dictionary entries that are specific to 3D annotations (see Table 9.33 in the *PDF Reference)*:

- **3DD**: A 3D stream specifying the 3D content. (See "Specifying the 3D stream" on page 6.)
- **3DV**: The initial view of the 3D content. (See "Setting the default view" on page 9.)
- **3DA**: The activation dictionary. (See "Setting the activation dictionary" on page 14.)

Two additional entries, **3DI** and **3DB**, are not set by the sample and retain their default values, as described in the *PDF Reference*.

Specifying the 3D stream

The **3DD** entry of the annotation dictionary specifies a *stream* containing the U3D data. Streams are PDF objects that can be thought of as having two parts, the stream data and an associated dictionary (see section 3.2.7 in the *PDF Reference*):

- In this example, the stream data is the U3D data representing the 3D content.
- The associated dictionary (sometimes called the *attributes dictionary*) contains entries that specify information about the stream. Some entries are common to all stream dictionaries (see Table 3.4 in the *PDF Reference)*. They include:
  - **Length** (required): The length of the stream data
  - **Filter** (optional): A compression filter that is applied to the data to reduce its size (there are also filters that do not compress data)

  Other entries are unique to 3D streams (see Table 9.35 in the *PDF Reference)*. They include:
  - **Type** (optional): Must be **3D** if present.
  - **Subtype** (required): Must be **U3D**.
  - **OnInstantiate** (optional): A JavaScript script to be executed when the 3D stream is read. (See "Specifying JavaScript code" on page 8.)

Creating the stream object

To create the stream object, the first step is to open the file containing the U3D data. You must provide this file, created with 3D authoring software, so that it can be embedded in the stream.

The path name of the file is passed to the current function as the parameter **`u3dFileName`**. **`AS`**-level (Acrobat support) methods are used to open the file. The **`ASFileSysCreatePathName`** method creates an **`ASPathName`** object.

The second parameter specifies the type of data by which the path is specified (**`Cstring`** for Windows and **`POSIXPath`** for Mac).

```
char sPathFlag1[16] = "Cstring";
#ifdef MAC_PLATFORM
     if(!strchr(u3dFileName, ':'))
            strcpy (sPathFlag1, "POSIXPath");
#endif

ASPathName asPathName1 = ASFileSysCreatePathName (ASGetDefaultFileSys(),
                               ASAtomFromString(sPathFlag1),
                               u3dFileName, 0);
```

The **`ASFileSysOpenFile`** method opens the file specified by the **`ASPathName`**:

```
ASFile asFile1 = NULL;
ASInt32 err = ASFileSysOpenFile(ASGetDefaultFileSys(),
     asPathName1, ASFILE_READ, &asFile1);
ASFileSysReleasePath (ASGetDefaultFileSys(), asPathName1);

//...error code
```

Then the file is read into an **`ASStm`** (stream) object:

```
ASStm fileStm = ASFileStmRdOpen(asFile1, 0);
//...error code
```

Next, the **`CosNewStream`** method is called to create a Cos stream containing the data from the **`ASStm`**. This Cos stream will become the value of the **3DD** entry of the 3D annotation.

```
CosObj stm3D = CosNewStream(cosDoc, true, fileStm, 0, false,
     CosNewNull(),       //attributes dictionary
     CosNewNull(), -1);
```

As with the other **`CosNew`*`Object`*** methods, the first two parameters to **`CosNewStream`** specify the **`CosDoc`** and whether the object is indirect. The additional parameters are:

- The source stream containing the data.
- The byte offset in the source stream to begin copying the stream.
- Whether the data should be encoded before being written to the Cos stream.
- A dictionary to be used as the attributes dictionary. This example specifies a null Cos object (created using the **`CosNewNull`** method) for this dictionary. In this case, a stream dictionary is automatically created with the required values (such as the length of the stream). The other values needed for the 3D stream dictionary will be filled in later in the example.
- Parameters to be used if the data is encoded.
- The amount of source data to be read. A value of -1 indicates that all the data should be read.

NOTE: See the *API Reference* for more details on **`CosNewStream`**.

Next, this stream object is added as the value of the **3DD** entry of the annotation dictionary:

```
CosDictPutKeyString(cosAnnot,  // the annotation dictionary
     "3DD",                    // the key
     stm3D);                   // the value
ASStmClose(fileStm);
```

Creating the attributes dictionary

The next step is to complete the entries in the 3D stream dictionary. The **`CosStreamDict`** method obtains the Cos dictionary associated with the stream:

```
CosObj attrObj = CosStreamDict(stm3D);
```

Next, entries can be added to the dictionary. The **Type** and **Subtype** entries both take name objects as values. Therefore, strings specified in the code must be explicitly converted to names:

```
CosDictPutKeyString(attrObj,          //the stream dictionary
     "Subtype",
     CosNewNameFromString(cosDoc, false, "U3D"));

CosDictPutKeyString(attrObj, "Type",
     CosNewNameFromString(cosDoc, false, "3D"));
```

Specifying JavaScript code

The following optional step specifies a JavaScript script as the value of the **OnInstantiate** entry of the 3D stream dictionary. It assumes that there is a JavaScript file that has been specified as the `JsFileName` parameter to this function.

Note that this stream is being referenced from the dictionary of another stream. The new stream also contains a stream dictionary whose entries are set as described here.

First, the JavaScript file is opened:

```
char sPathFlag2[16] = "Cstring";
#ifdef MAC_PLATFORM
     if(!strchr(JsFileName, ':'))
            strcpy (sPathFlag2, "POSIXPath");
#endif
ASPathName asPathName2 = ASFileSysCreatePathName (ASGetDefaultFileSys(),
                  ASAtomFromString(sPathFlag2), JsFileName, 0);

ASFile asFile2 = NULL;
ASInt32 err2 = ASFileSysOpenFile(ASGetDefaultFileSys(),
                                 asPathName2, ASFILE_READ, &asFile2);
ASFileSysReleasePath (ASGetDefaultFileSys(), asPathName2);
...error checking code
```

Next, the data from the file is read into a stream:

```
ASStm JsFileStm = ASFileStmRdOpen(asFile2, 0);
...error checking code
```

In the following code, an entry is added to the stream dictionary in the process of creating the stream, rather than afterwards as in the previous code. First, the `CosNewDict` method is used to create a new dictionary:

```
CosObj dictJsStm = CosNewDict(cosDoc, false, 1);
```

This method takes 3 parameters:

- The document in which the dictionary is used.
- A boolean that specifies whether the dictionary should be an indirect object. All stream dictionaries must be direct; hence the value of this parameter is `false`.
- A hint for the number of entries in the dictionary (however, dictionaries grow dynamically as needed).

Next, the value of the **Filter** entry is set to **FlateDecode** using the `CosDictPutKeyString` method. This means that the stream will be compressed using Flate (ZIP) compression.

```
CosDictPutKeyString(dictJsStm, "Filter",
    CosNewNameFromString(cosDoc, false, "FlateDecode"));
```

Next, the Cos stream is created, using the stream data and attributes dictionary already created:

```
stm3Djscode = CosNewStream(cosDoc, true,
                    JsFileStm,   //the stream
                    0, true,
                    dictJsStm,   // the stream dictionary
                    CosNewNull(), -1);
```

and set it as the value of the **OnInstantiate** entry of the 3D stream dictionary.

```
CosDictPutKeyString(attrObj, "OnInstantiate", stm3Djscode);
```

Then some cleanup is done:

```
ASFileSysReleasePath (ASGetDefaultFileSys(), JsPathName);
ASStmClose(JsFileStm);
```

Setting the default view

A 3D *view* specifies parameters such as position, orientation, and projection style, which are applied to the *virtual camera* associated with the 3D annotation (see section 9.5.3 in the *PDF Reference*). The *default view* is the view that is chosen when the annotation is activated.

U3D data typically contains a default initial view. This view is used by default if not otherwise specified. In addition, views can be specified by the entries in a *view dictionary*.

The **VA** entry in the 3D stream dictionary is an array of view dictionaries. One of the views can be chosen as the default by means of the **3DV** entry in the 3D annotation dictionary or the **DV** entry in a 3D stream dictionary.

NOTE: The initial release of Acrobat 7.0 does not correctly display the default view in the U3D data. This behavior is expected to be corrected in future releases and hence, the rest of the code in this section, which provides an explicit view dictionary, will no longer be necessary.

The following code creates a view dictionary and specifies its entries. The code assumes the Cos objects `cosAnnot` for the annotation and `cosDoc` for the document have already been obtained. First, a view dictionary is created:

```
CosObj cosView = CosNewDict (cosDoc, true, 8);
```

Next, the code sets the following entries (see Table 9.37 in the *PDF Reference* for more detailed information):

- **Type** (optional): If present, must be the name **3DView**.
- **XN** (required): The name of the view, a string that can be displayed in the user interface.
- **IN** (optional): The internal name of the view, a string that can be used to refer to the view from other objects, such as in JavaScript code.

- **C2W** (optional): A transformation matrix specifying the camera position. To use this, it is also necessary to set the value of the **MS** entry to **M**.
- **CO** (optional): A number indicating the distance to the center of orbit for this view.

These are some constants used in the sample. You can replace them as appropriate for your own code:

```
char* externalViewName = "Default View";
char* internalViewName = "Sample3dView";

double gMatrixVals[12] =
     {1.0, 0.0, 0.0, 0.0, 0.0000000000000000612303, -1.0,
     0.0, 1.0, 0.0000000000000000612303, 82.9517, -883.324, 115.166};
float gCOvalue = (float) 725.305;
```

Now the values of the dictionary entries are set:

```
CosDictPutKeyString(cosView, "Type",
                    CosNewNameFromString(cosDoc, false,"3DView"));

CosDictPutKeyString(cosView, "XN", CosNewString(cosDoc, false,
     externalViewName, strlen(externalViewName)));

CosDictPutKeyString(cosView, "IN", CosNewString(cosDoc, false,
     internalViewName, strlen(internalViewName)));

CosDictPutKeyString(cosView, "MS",
                    CosNewNameFromString(cosDoc, false, "M"));

CosDictPutKeyString(cosView, "CO", CosNewFixed(cosDoc, false,
FloatToASFixed(gCOvalue)));
```

Here the **C2W** matrix is filled with the appropriate values:

```
CosObj matrixArray = CosNewArray(cosDoc, false, 12);
     for(int i=0; i<12; i++)
            CosArrayPut(matrixArray, i,
                    CosNewFloat(cosDoc, false, (float) gMatrixVals[i]));
CosDictPutKeyString(cosView, "C2W", matrixArray);
```

Last, the dictionary is set as the value of the the **3DV** key in the annotation dictionary:

```
CosDictPutKeyString(cosAnnot, "3DV", cosView);
```

Setting the annotation appearance

You may optionally provide a *poster* as the initial appearance of the annotation. The poster may be an image or other graphic content that is in a file or in memory. It must be converted to a PDF *form XObject* to be used as the annotation appearance (see section 4.9 of the *PDF Reference*).

The **AP** entry of the annotation dictionary specifies an *appearance dictionary*. This dictionary contains one or more *appearance streams* (see section 8.4.4 of the *PDF Reference*) that are PDF content streams (form XObjects) rendered inside the annotation rectangle.

For 3D annotations, the appearance stream is used in the following situations:

- To provide an annotation appearance for PDF viewers that do not support 3D.
- To provide an initial appearance for the annotation prior to activation. The settings in the activation dictionary determine whether this appearance is ever displayed.

There are several ways to get the poster. The function described below, `GetFormXObjectFromFile`, illustrates one method. The appearance is generated from a separate PDF file containing an image or other content. You call this function as follows:

```
CosObj formXObject = GetFormXObjectFromFile
                         (gsPosterFilePath, //the external file
                         pdDoc);
```

The function returns a Cos object, `formXObject`, which is the form XObject to be used as the appearance.

```
CosObj cosAnnot = PDAnnotGetCosObj(newAnnot);
CosDoc cosDoc = CosObjGetDoc(cosAnnot);
```

Then you create the appearance dictionary:

```
CosObj apprDict = CosNewDict(cosDoc, false, 1);
```

and set its **N** (normal) entry to the appearance stream obtained above.

```
CosDictPutKeyString(apprDict, "N", formXObject);
CosDictPutKeyString(cosAnnot, "AP", apprDict);
```

The following is the `GetFormXObjectFromFile` function:

```
CosObj GetFormXObjectFromFile
        (char* pdfImageFilePath, //path of image PDF file
        PDDoc TargetPdDoc)  // the current document
   {
   PDDoc posterPDFDoc = NULL;     //initialization code
   PDPage pdPageImage = NULL;
   ASPathName asPathName;
   CosObj contentFormXObject = CosNewNull();
   CosObj formXObject = CosNewNull();
```

First, the PDF file containing the image is opened:

```
   if(strlen(pdfImageFilePath) > 0 ) {
        char sPathFlag[16] = "Cstring";
   #ifdef MAC_PLATFORM
   if (!strchr(pdfImageFilePath,(int)':'))
        strcpy (sPathFlag, "POSIXPath");
   #endif

   asPathName = ASFileSysCreatePathName (ASGetDefaultFileSys(),
                    ASAtomFromString(sPathFlag), pdfImageFilePath, 0);
```

The content to be used is expected to be on the first page of the PDF file. The **`PDDocAcquirePage`** method returns a **`PDPage`** object for the first page.

```
pdPageImage = PDDocAcquirePage(posterPDFDoc, 0);
```

The code then uses PDE-layer (PDFEdit) methods that work with the content streams on the PDF page. (See the *API Overview* for more information on how these methods work.)

The **`PDPageAcquirePDEContent`** method returns a **`PDEContent`** object representing the page's contents. The first parameter is the **`PDPage`** and the second identifies the caller: for PDF Libray, it is zero; for plug-ins, it should be the **`gExtensionID`** extension:

```
PDEContent pdeContent = PDPageAcquirePDEContent (pdPageImage, 0);
```

The **`PDEContentGetAttrs`** method gets information about the content in a **`PDEContentAttrs`** structure:

```
PDEContentAttrs pdeContentAttrs;
PDEContentGetAttrs
     (pdeContent, &pdeContentAttrs, sizeof(pdeContentAttrs));

CosObj contentResources = CosNewNull();
CosDoc pdDocCos = PDDocGetCosDoc(posterPDFDoc);
```

The **`PDEContentToCosObj`** method converts the **`PDEContent`** to a form XObject Cos object.

```
PDEContentToCosObj (pdeContent,
                    kPDEContentToForm,  // to Form XObject
                    &pdeContentAttrs,// PDEContentAttrsP
                    sizeof(pdeContentAttrs),        // attrsSize,
                    pdDocCos,           // The CosDoc
                    NULL,               // PDEFilterArrayP
                    &contentFormXObject,  // resulting form Cos object
                    &contentResources);   // resulting resource Cos object
```

The parameters to this method are:

- The **`PDEContent`** object.
- A flag indicating what type of Cos object should be created; in this case, a form XObject.
- The **`PDEContentAttrs`** structure containing information about the **`PDEContent`**.
- The size of the **`PDEContentAttrs`** structure.
- The Cos document.
- A pointer indicating which filters to use to encode the contents (in this case, null).
- The resulting Cos object (in this case, the form XObject which is the variable **`contentFormXObject`**).
- The resulting Cos object representing the resources needed by the Cos object. These resources can include fonts and other items (see section 3.7.2 of the *PDF Reference*).

```
if (!CosObjEqual(contentFormXObject, CosNewNull()) &&
     !CosObjEqual(contentResources, CosNewNull())) {
```

The returned resources must be put into the form XObject's **Resources** dictionary:

```
        CosDictPutKeyString(contentFormXObject, "Resources",
                            contentResources);
```

The **BBox** entry of the form XObject is required and is set to the value of the page's media box:

```
        ASFixedRect boundingBox;
        PDPageGetMediaBox(pdPageImage, &boundingBox);
        CosObj BBoxArray = CosNewArray(pdDocCos, 4, false);
        CosArrayPut(BBoxArray,0, CosNewInteger(pdDocCos, false,
   ASFixedRoundToInt16(boundingBox.left)));
        CosArrayPut(BBoxArray,1, CosNewInteger(pdDocCos, false,
   ASFixedRoundToInt16(boundingBox.bottom)));
        CosArrayPut(BBoxArray,2, CosNewInteger(pdDocCos, false,
   ASFixedRoundToInt16(boundingBox.right)));
        CosArrayPut(BBoxArray,3, CosNewInteger(pdDocCos, false,
   ASFixedRoundToInt16(boundingBox.top)));
        CosDictPutKeyString(contentFormXObject, "BBox", BBoxArray);
        // set matrix key in form object
```

The **Matrix** entry of the form XObject is set to the values obtained from the page by means of the `PDPageGetDefaultMatrix` method:

```
        ASFixedMatrix defaultMatrix;
        PDPageGetDefaultMatrix(pdPageImage, &defaultMatrix);
        CosObj MatrixArray = CosNewArray(pdDocCos, 6, false);
        CosArrayPut(MatrixArray,0, CosNewFixed(
              pdDocCos, false, defaultMatrix.a));
        CosArrayPut(MatrixArray,1, CosNewFixed
               (pdDocCos, false, defaultMatrix.b));
        CosArrayPut(MatrixArray,2, CosNewFixed
               (pdDocCos, false, defaultMatrix.c));
        CosArrayPut(MatrixArray,3, CosNewFixed
               (pdDocCos, false, defaultMatrix.d));
        CosArrayPut(MatrixArray,4, CosNewFixed
               (pdDocCos, false, defaultMatrix.h));
        CosArrayPut(MatrixArray,5, CosNewFixed
               (pdDocCos, false, defaultMatrix.v));
        CosDictPutKeyString(contentFormXObject, "Matrix", MatrixArray);
        }
```

Finally, the `CosObjCopy` method is used to copy the Cos object `contentFormXObject` into the current PDF document. The parameters to this method are:

- The `CosObj` to copy.
- The `CosDoc` for the document in which to copy it.
- A boolean: `true` means that all indirectly referenced objects from the source should be copied to the destination.

```
   formXObject = CosObjCopy (contentFormXObject,
                    PDDocGetCosDoc(TargetPdDoc), true);
```

And finally, there is some cleanup code:

```
ASFileSysReleasePath (ASGetDefaultFileSys(), asPathName);
PDPageRelease(pdPageImage);
return formXObject;
}
```

Setting the activation dictionary

The optional **3DA** entry of the 3D annotation specifies an *activation dictionary* whose entries indicate when the annotation should be activated and deactivated and the state of the 3D content at these times.

When an annotation is inactive, it displays its *normal appearance*. (See "Setting the annotation appearance" on page 10.) When it is activated, one of its views (specified by the **3DV** entry) is displayed.

First the dictionary is created and set as the **3DA** entry of the 3D annotation:

```
CosObj activationDict = CosNewDict(CosObjGetDoc(cosAnnot), false, 1);
CosDictPutKeyString (cosAnnot, "3DA", activationDict);
```

It is not necessary to set any entries whose default values are acceptable. Here the non-default entries are set.

The **DIS** entry of the activation dictionary specifies the state of the 3D content when it is deactivated. In this case, it is set to **I**, meaning that it should be instantiated. (The default is **U** for uninstantiated.)

```
CosDictPutKeyString (activationDict, "DIS",
     CosNewNameFromString (cosDoc, false, "I"));
```

The code provides a variable to determine the value of the **A** entry. The default value is **XA**, meaning that the annotation needs to be explicitly activated. **PO** means that the annotation should be activated as soon as the page containing the annotation is opened:

```
// optional activation choice
if(gbShowDefaultViewWhenOpenPage == true)
     CosDictPutKeyString(activationDict, "A",
             CosNewNameFromString (cosDoc, false, "PO"));
```