



# **KODAK Digital Lab Systems Software Developer's User Guide for COS File Creation for DLS**

Part No. 6B7233  
March 1, 2002

Eastman Kodak Company  
343 State Street  
Rochester, NY 14650

Kodak, FlashPix, Photo CD and PhotoNet are trademarks of Eastman Kodak Company.

©Eastman Kodak Company, 2002

## Table of Contents

<b>1. Construction of a COS File for DLS Fulfillment .....</b>	<b>5</b>
Rules for DLS fulfillment .....	5
COS Fields Used for Fulfillment on DLS.....	7
COS File Hierarchy.....	8
<b>2. Creating a COS file with Embedded Images.....</b>	<b>9</b>
<b>3. Creating a COS File with Referenced Images .....</b>	<b>17</b>
<b>4. Using Additional COS Fields .....</b>	<b>19</b>
Fields Used by DLS .....	19
<b>5. Reference - COS SDK Methods Used .....</b>	<b>21</b>
COSOrder::COSOrder .....	21
COSOrder::GetVendorInfo .....	21
COSOrder::GetConsumerInfo.....	22
COSOrder::GetFileInfo .....	22
COSOrder::AddNewProduct .....	22
COSOrder::AddImageDetail .....	23
COSOrder::AddNewImage .....	23
pTheName Values Note: .....	24
COSOrder::WriteOrder .....	24
COSOrder::~~COSOrder.....	24
COSVendorInfo::SetVendorOrderNumber.....	25
COSFileMod::SetApplication .....	25
COSVendorInfo::SetVendorOrderNumber.....	25
COSConsumerInfo::SetConsumerID .....	26
COSConsumerInfo::GetAddressInfo.....	26
COSAddressInfo::SetDayPhone.....	26
COSAddressInfo::SetNightPhone.....	27
COSAddressInfo::SetFaxNumber.....	27
COSAddressInfo::SetEmailAddress .....	27
COSAddressInfo::SetName .....	27
COSFileInfo::GetFileMod.....	28
COSFileMod::SetApplication .....	28
COSProduct::SetProductClass .....	28
COSProduct::SetProductDescription .....	28
COSProduct::GetFinishingInfo.....	29
COSProduct::GetPrintInfo .....	29

COSProduct::AddImageDetailRef .....	29
COSFinishingInfo::SetQuantity .....	29
COSPrintInfo::SetShortDimension .....	30
COSPrintInfo::SetLongDimension .....	30
COSPrintInfo::SetSizeUnit .....	30
COSPrintInfo::SetMediaSurface.....	31
COSPrintInfo::SetBackPrintMessage .....	31
COSImageDetail::SetImageType .....	31
COSImageDetail::SetRegionOfInterest.....	32
COSImageDetail::SetRotation.....	32
COSImageDetail::GetDigitalInfo.....	32
COSImageDetail::AddImageRef .....	32
COSDigitalInfo::SetFileFormat.....	33
COSDigitalInfo::SetFileColorSpace.....	33
COSDigitalInfo::SetDigitalSharpening.....	33
COSDigitalInfo::SetDigitizingSource .....	33
COSDigitalInfo::SetDigitizedResolution .....	33
COSImage::WriteImage .....	34
<b>6. Reference – KODAK Image Access Client SDK Methods Used .....</b>	<b>35</b>
KIASGetOrders .....	35
KIASGetImageRefs.....	35
KIASGetAvailableProducts .....	36
KIASGetAvailableOutputs .....	36
KIASSubmitCosOrder .....	36
KIASFree .....	37

# 1. Construction of a COS File for DLS Fulfillment

The Customer Order Specification (COS) SDK is used to construct COS files and provide a class library of tools for creating, writing, reading and modifying a photofinishing customer order. For information regarding the use of the COS SDK, see the *Customer Order Specification SDK User's Guide*, version 2.3, June 1999.

This document, *KODAK Digital Lab System Software Developer's User Guide for COS File Creation for DLS 2.0* discusses the creation of a COS order for fulfillment on a DLS.

Note: This document and all DLS limitations and expected values are specific to DLS Software. The Digital Lab System's client is responsible for constructing valid and conforming COS files.

## Rules for DLS fulfillment

There are several options when constructing a COS file. Figure 1 diagrams the most complex case that DLS supports. The order requests a 4x6 print of each image, an 8x10 of image 2 and an index print of all the images in the order. The index print product requires that a dummy image be linked. An index print is optional. The images may be either embedded or referenced.

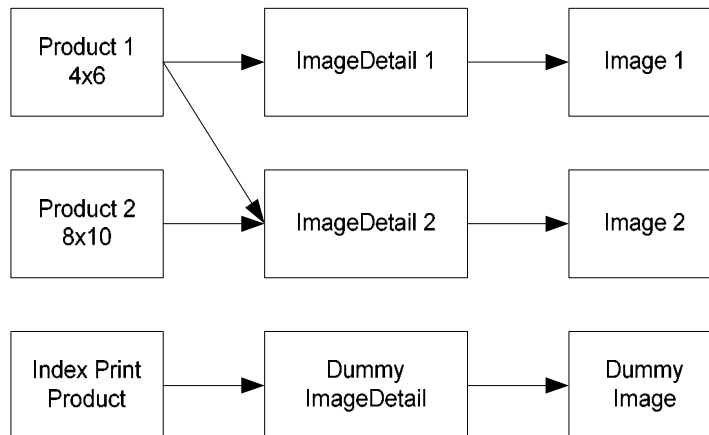


Figure 1 : m Products per-Image, n Images per-Product

The layout of Figure 1 is the same as the layout used by the DLS DIM utility. Another form of COS layout is to have one Image per Product as in Figure 2.

Figure 2 is constructed to request the exact same prints as Figure 1 but does so by specifying a duplicate 4x6 product. Having a duplicate product is required if you wish to specify a per-print backprint message because that message is part of the product object. The layout of Figure 2 may also yield simpler code because the link structure is less complex.

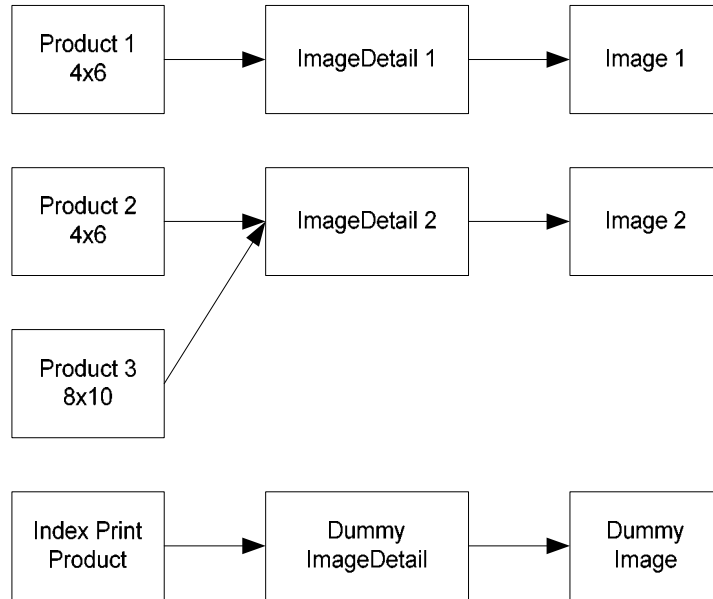


Figure 2 : m Products per-Image, one Image per-Product

The common thread between Figure 1 and Figure 2 is that there is always a one-to-one relationship between ImageDetails and Images. Either layout works fine with DLS. Contrary to Figure 1 and Figure 2, the COS specification lets one ImageDetail link to multiple images; this is not recommended with DLS.

The DLS expects COS files to adhere to the following COS file conventions:

- Each Source Image Detail Property Set should contain a reference to exactly one image.
- All images should either be embedded in the COS file or all images should refer to a specific archived order. A mixture of embedded and referenced images is not permitted. Referenced images must all be from the same archived order.
- The DLS has a limit of 40 images per order. The COS SDK does not enforce this; your application must do so.

## ***COS Fields Used for Fulfillment on DLS***

The following tree diagram shows the COS fields used by DLS. The COS format supports many other fields; however, DLS does not use them.

COSOrder – the COS file itself

VendorInfo – one of these per order file

VendorOrderNumber

ConsumerInfo – one of these per order file

ConsumerID

AddressInfo (optional)

DayPhone (optional)

NightPhone (optional)

FaxNumber (optional)

EmailAddress (optional)

Name (optional)

FileInfo (optional) – one of these per order file

FileMod (optional)

Application (optional)

Product – multiple products are allowed in an order file

ProductClass

ProductDescription

FinishingInfo

Quantity

PrintInfo

ShortDimension

LongDimension

SizeUnit

MediaSurface (optional)

BackprintMessage (optional)

ImageDetailRef – a link to an ImageDetail

ImageDetail – multiple ImageDetails are allowed in an order file

ImageType

RegionOfInterest (optional)

Rotation (optional)

DigitalInfo

FileFormat

FileColorSpace

DigitalSharpening (optional)

DigitizingSource (optional)

DigitizedResolution (optional)

ImageRef – a link to an Image

FileMod

Set Application

\*Image – you should have one image per ImageDetail

## **COS File Hierarchy**

A COS file is a hierarchical format, as the previous tree implies. The structure of the full tree is documented in the *Customer Order Specification Version 1.11* document. Files conforming to this specification are created using the KODAK COS SDK. The hierarchy of the tree is realized as either a MICROSOFT structured storage file or as an XML file.

*Important: Because DLS only recognizes the structured storage variant, we specifically recommend that you create your COS files with version 2.6 of the COS SDK.*

The names of the tree parts are taken from the COS SDK. For leaves of the tree, there is always a matching method to set that property. For instance, the diagram indicates that ConsumerID is a field of a ConsumerInfo object. This implies that ConsumerInfo has a method called SetConsumerID().

Each non-leaf node of the tree is an object. The first step of filling in an object is to create it. With the exception of the root COSOrder object, each object may be created by calling a creation method on its parent. These methods look like:

```
pParent->GetChildObj(ChildType*& pChild);
```

For example, in the case of the AddressInfo sub-object of ConsumerInfo, one would create a new AddressInfo with the call:

```
COSAddressInfo *pAddressInfo = 0;  
pConsumerInfo->GetAddressInfo(pAddressInfo);
```

The root COSOrder object has a constructor that you would call with the new operator. After getting each sub-object in the tree and filling each field, the final step of creating the COS file is to call the WriteOrder method and then the delete operator on the root COSOrder object.

Each COS SDK method, except the root COSOrder constructor, returns a COSStatus. This should be compared to COS\_OK to see if the method succeeded. The COSOrder constructor returns its COSStatus as a reference parameter.

The COS file should be constructed in the reverse order of the link arrows in the two earlier diagrams. To link to something, that file has to exist first.

## 2. Creating a COS file with Embedded Images

### 1. Create an Empty COS File

- a. Include the required headers; COS object types each have their own header. Link your code with cosekxapi.lib, cosapi.lib, and psets.lib from the COS SDK.

```
#include <COSTypes.h>
#include <COSDatatools.h>
#include <COSOrder.h>
#include <COSErrors.h>
#include <COSFileInfo.h>
#include <COSVendorInfo.h>
#include <COSProduct.h>
#include <COSAddressInfo.h>
#include <COSPrintInfo.h>
#include <COSFinishingInfo.h>
#include <COSImageDetail.h>
#include <COSDigitalInfo.h>
#include <COSImage.h>
#include <COSConsumerInfo.h>
#include <COSFileMod.h>
```

- b. Use the COSOrder constructor to create your new COS file. The path that you provide as the first parameter should be an ASCII string, char\*. This is one of the few places in the COS SDK that a string is not passed as wide characters, wchar\_t\*.

```
COSStatus Status;
COSOrder *pCOSOrder = new COSOrder("C:\\MyNewFile.cos",
COS_CREATE, Status);
if (!pCOSOrder || Status != COS_OK) goto error;
Add (use pCosOrd->AddGetFileMod() )-
>SetApplication("YourApplicationorCompanyName V3.5")
```

- c. Add your company name or application name to identify the source of the COS file.

2. Fill in the per-order fields.
  - a. Fill in the VendorInfo and ConsumerInfo branches of the tree. The ConsumerInfo.ConsumerID field should be filled with a recognizable customer identifier. An email address or a name works well. The VendorInfo.VendorOrderNumber field should be filled with a number from 1 to 999999. You may use any number in this range, but it is recommended that you increment it between orders so that it is possible to distinguish between orders with the same ConsumerID. These fields appear on the DLS user interface screens labeled as customer and orderID, respectively. These fields are technically both optional, but we recommend always filling them so that you can distinguish your orders. Both fields are wide character strings.

```
// fill ConsumerInfo.ConsumerID
COSConsumerInfo *pConsumerInfo = 0;
pCOSOrder->GetConsumerInfo(pConsumerInfo);
pConsumerInfo->SetConsumerID( L"me.myself@myisp.com"); // any string
```

```
// fill VendorInfo.VendorOrderNumber
COSVendorInfo *pVendorInfo = 0;
pCOSOrder->GetVendorInfo(pVendorInfo);
pVendorInfo->SetVendorOrderNumber(L"1234"); // string form of any
number 1-999999
```

- b. The AddressInfo sub-object of ConsumerInfo may now be filled. The fields of this object are used by the DLS in the event of a KODAK PHOTONET Online upload being requested for this order by the DLS operator. Future versions of DLS will support the requesting of an upload from within the COS file with no operator intervention. The AddressInfo object contains phone numbers, name, and e-mail address. As indicated in the example below, Names may be composed of multiple parts. The first parameter to SetName() is an index to indicate which part of the Name is being set. The entire AddressInfo object is optional. If you don't intend on using PhotoNet, there is no need to fill this object. AddressInfo is not supported in DLS.

```
COSAddressInfo *pAddressInfo = 0;
pConsumerInfo->GetAddressInfo(pAddressInfo);
pAddressInfo->SetDayPhone(L"(301) 123-4567");
pAddressInfo->SetNightPhone(L"(301) 123-4568");
pAddressInfo->SetFaxNumber(L"(301) 123-4569");
pAddressInfo->SetEmailAddress(L"bob@bobsisp.com");
pAddressInfo->SetName(0, L"Bob");
pAddressInfo->SetName(1, L"Smith");
```

3. Add an image to the order.
  - a. Embed the image file.

Embedding an image consists of calling `pCOSOrder->AddNewImage()` to create an image object and then calling `pImage->WriteImage()` to copy the image file into the order file. This is complicated by the fact that the `WriteImage()` method of the COS SDK expects to be passed a memory buffer, an 'unsigned char\*', rather than a filename. One technique would be to allocate a large chunk of memory, copy the file into it, and then pass the pointer to this memory to `WriteImage()`. While this works, it is highly inefficient. We recommend using the memory-mapping facilities of your operating system to make the image file appear to be in memory. In the following code, we demonstrate the WIN32 way of doing this. Similar techniques are available for other operating systems.

The first parameter to `AddNewImage()` is the name of the image. One's first thought is to fill this with the name of the image file. However, DLS expects the image index to be placed here. This should be a number, starting from 1, and incremented for each subsequent image added to the file. The DLS is completely unaware of the filename. DLS will accept a maximum of 40 images in a single order.

```

COSImage* pImage = 0;
pCOSOrder->AddNewImage("1", pImage); // name the image by number,
increment for each image
HANDLE hFile = CreateFile("c:\\myimage.jpg", GENERIC_READ
    ,FILE_SHARE_READ, 0, OPEN_EXISTING,
    FILE_ATTRIBUTE_NORMAL, 0);

// we will use NT file mapping to create a virtual memory buffer with the
contents of the file.
// we'll only handle files up to 2GB
long FileSize = GetFileSize(hFile, 0);
HANDLE hFileMap = CreateFileMapping(hFile, 0, PAGE_READONLY, 0,
0, 0);
unsigned char *pBuf =
reinterpret_cast<unsigned char*>(MapViewOfFile(hFileMap,
FILE_MAP_READ, 0, 0, 0));
pImage->WriteImage(pBuf, FileSize); // stream the image file into the order
file
UnmapViewOfFile(pBuf);
CloseHandle(hFileMap);
CloseHandle(hFile);

```

- b. Fill the descriptive ImageDetail for the Image.

An ImageDetail object must be created in the order file to describe each image that is present. The first thing to do after creating a new ImageDetail in an order is link the image it describes to it. If you don't do this, other ImageDetail methods will complain about the lack of an image.

```
COSImageDetail* pImageDetail = 0;  
pCOSOrder->AddNewImageDetail(pImageDetail);  
pImageDetail->AddImageRef(pImage);
```

Fill in the three object fields, ImageType, DigitalInfo.FileFormat, and DigitalInfo.FileColorSpace. ImageType must be set to either kEmbeddedDigitalData or kSeparateDigitalData, and since this is the embedded image example, we will set it to kEmbeddedDigitalData.

```
pImageDetail->SetImageType(COSImageDetail::kEmbeddedDigitalData);
```

The other 2 fields are in a sub-object called DigitalInfo. DLS expects the file format and colorspace to be set here. This won't be required with future versions of DLS. The possible file formats are: kFlashPix, kPhotoCD, kEXIFJPEG, kJPEG, kTIFF, and kBMPWin. Not all variants of these are supported, but you are safe with common truecolor flavors. Avoid 8 bit color. The colorspace should always be set to ksRGB unless you know that you have a kPhotoCD file that uses PhotoYCC. In this you should use kPhotoYCC as the color space. Specify ksRGB for your JPEGs even though they likely are not RGB.

```
COSDigitalInfo *pDigitalInfo = 0;  
pImageDetail->GetDigitalInfo(pDigitalInfo);  
pDigitalInfo->SetFileFormat(COSDigitalInfo::kJPEG);  
pDigitalInfo->SetFileColorSpace(COSDigitalInfo::ksRGB);
```

4. Add a product to the order.

A product specifies how you would like your image printed. Just as we linked an Image to our ImageDetail in the last section, here we link an ImageDetail to the new product that we create. You may link as many ImageDetails as you wish; this product is produced for each of them. A single 4x6 product could link to all of the images in the order. When the product is created, the AddNewProduct() method has a first parameter of product name. You may set this to whatever you like, DLS completely ignores it. It is an ASCII string.

```
COSProduct *pProduct = 0;
pCOSOrder->AddNewProduct("Product 1", pProduct); // first parameter is
not used by DLS
pProduct->AddImageDetailRef(pImageDetail); // link to the ImageDetail,
repeat for each ImageDetail
```

ProductClass should be set to either kStandardPrint or kIndexPrint. Because this product is going to be a standard 4x6, set ProductClass to kStandardPrint. ProductDescription should be set to one of the ProductID's returned by the KODAK Image Access Client GetAvailableProducts() call. This field is not being used as a description but rather as a unique productID. You should not fill it with the product description returned by GetAvailableProducts().

```
pProduct->SetProductClass(COSProduct::kStandardPrint);

// get the list of available products from the KIAS server
// pSessionID is a valid KIAS session
wchar_t *pError[ERRSIZE];
KIASLocale Locale = {L"EN", L"US", L"USD"};
KIASINT productCount = 0;
KIASProduct *pProductsOut = 0;
KIASGetAvailableProducts(pError, pSessionID, Locale, &productCount,
&pProductsOut );

// suppose we wish to use the first available product
pProduct->SetProductDescription(pProductsOut[0].ProductID);
```

Within the sub-object FinishingInfo is the field Quantity. Set this to the number of copies of the product desired for each linked image.

```
COSFinishingInfo *pFinishingInfo = 0;
pProduct->GetFinishingInfo(pFinishingInfo);
pFinishingInfo->SetQuantity(2); // doubles
```

Within the sub-object PrintInfo, there are 5 fields to set. ShortDimension should be set to the length of the shortest dimension of the print. The unit of this length is specified in the SizeUnit field. The same is true for the field LongDimension. All three of these fields should be taken from the product information returned by the KODAK Image Access Client GetAvailableProducts() method. SizeUnit must be either kInch or kMillimeter. BackprintMessage is a wide character string that is printed on the back of each print. If you want distinct messages for each image, create a separate product for each image. MediaSurface specifies the desired paper type. This request may or may not be honored.

The legal values are: kMatte, kSemiMatte, kGlossy, kSmoothLuster, kUltraSmoothLuster, kFineGrainedLuster, kSilk, kLuster, kDeepMatte, kHighGloss, and kLaminate. Both the BackprintMessage and MediaSurface fields are optional.

```
COSPrintInfo *pPrintInfo = 0;
pProduct->GetPrintInfo(pPrintInfo);

// use result from prior call to KIASGetAvailableProducts()
float shortDim = pProductsOut[0].ImageWidth <
pProductsOut[0].ImageHeight
? pProductsOut[0].ImageHeight : pProductsOut[0].ImageWidth;
float longDim = pProductsOut[0].ImageWidth <
pProductsOut[0].ImageHeight
? pProductsOut[0].ImageWidth : pProductsOut[0].ImageHeight;
ushort unit = pProductsOut[0].LengthUnit == INCH ? COSPrintInfo::kInch :
COSPrintInfo::kMillimeter;

pPrintInfo->SetShortDimension( shortDim);
pPrintInfo->SetLongDimension( longDim);
pPrintInfo->SetSizeUnit( unit);
pPrintInfo->SetBackprintMessage( L"picture of Billy at pool");
pPrintInfo->SetMediaSurface( COSPrintInfo::kGlossy);

// remember to free the products array allocated by KIAS
KIASFree(pError, pProductsOut);
```

5. Add an index print to the order.

Adding an index print is an abbreviated version of steps 3 and 4. First create a dummy image, and then link it to an index print product. For a regular product, the images have names that are numbers. For an index print, the dummy image should be named "INDEX\_PRINT." There is no need to call WriteImage.

```
COSImage* pImage = 0;
pCOSOrder->AddNewImage("INDEX_PRINT", pImage);
```

Next create an ImageDetail for our dummy image. The ImageType field must be set to kSeparateDigitalData an actual image file is not embedded. There is no need to create and fill a DigitalInfo object for an index print.

```
COSImageDetail* pImageDetail = 0;
pCOSOrder->AddNewImageDetail(pImageDetail);
pImageDetail->AddImageRef(pImage);
pImageDetail->SetImageType(COSImageDetail::kSeparateDigitalData);
```

Create the index print product. As for a regular print, the product name does not matter. The ProductClass should be set to kIndexPrint. Instead of getting ProductDescription from the product information returned by the KODAK Image Access Client GetAvailableProducts() method, hard code it to either L"REGULAR" or L"JUMBO" for a regular or jumbo sized index print. You do not need to create or fill a PrintInfo object for an index print. Set the quantity to one.

```
COSProduct *pProduct = 0;
pCOSOrder->AddNewProduct("index product", pProduct); // product name
doesn't matter
pProduct->AddImageDetailRef(pImageDetail);
pProduct->SetProductClass(COSProduct::kIndexPrint);
pProduct->SetProductDescription(L"REGULAR");
COSFinishingInfo *pFinishingInfo = 0;
pProduct->GetFinishingInfo(pFinishingInfo);
pFinishingInfo->SetQuantity(1);
```

Finally, as the last step of COS file creation, be sure to flush the order to disk and delete the root COSOrder object.

```
pCOSOrder->WriteOrder();
delete pCOSOrder;
```

You may now submit your newly created COS file to the server for order fulfillment.

```
wchar_t *pOrderIDOut[IDSIZE];
KIASSubmitCosOrder(pError, pSessionID, L"C:\\MyNewFile.cos", L"",
pOrderIDOut );
```



### **3. Creating a COS File with Referenced Images**

An order contains either embedded images, as discussed in the prior section, or references to images already in the image archive. For DLS, an order may not have both embedded and referenced images. All images must be either embedded or referenced. Similarly, for an order with referenced images, the referenced images must all reside within the same archived order. You may not reference images from multiple archived orders in a single COS file.

An order of referenced images is similar to an order of embedded images but with the following differences. You may not set the VendorOrderNumber to an arbitrary number between 1 and 999999 as you did for embedded images. Rather, you must set it to the orderID of an order that exists in archive. You should get this orderID by calling the KODAK Image Access Client GetOrders() method.

```
KIASINT orderCount = 0;
OrderInfo *pOrdersOut = 0;
KIASGetOrders(pError, pSessionID, L"Bob", L"", 0, 0, NOTFOUND, L"",
&orderCount, &pOrdersOut );

// we'll reference the first order in the order array
pVendorInfo->SetVendorOrderNumber(pOrdersOut[0].OrderID);
```

When you create an image, use an imageID for the image name. Get the imageID for each image in an order by calling the KODAK Image Access Client GetImageRefs() method on that order. Do not call WriteImage for a referenced image.

```
KIASINT imageCount = 0;
KIASImageInfo *pImages = 0;
KIASGetImageRefs(pError, pSessionID, pOrdersOut[0].OrderID, &imageCount,
&pImages );

// we'll add the first image in the referenced order
wchar_t imageID[16];
swprintf(imageID, L"%d", pImages[0].ImageID);
COSOrder->AddNewImage(imageID, pImage);

// remember to free the image and order arrays allocated by KIAS
KIASFree(pError, pImages);
KIASFree(pError, pOrdersOut);
```

When creating the ImageDetail for the image, set the ImageType to kSeparateDigitalData rather than kEmbeddedDigitalData. These are the only differences.

```
pImageDetail->SetImageType(COSImageDetail::kSeparateDigitalData);
```

## 4. Using Additional COS Fields

This section provides information on COS files that can be used by KODAK Image Access Client applications to crop, rotate, and more accurately render images. Information on attaching address information to uploads is also included.

### **Fields Used by DLS**

You may specify a crop box for each image. The DLS prints just the region specified. SetRegionOfInterest takes four parameters, Xpos, Ypos, Width, and Height. Xpos and Ypos are the distance from the upper left corner of the image to the upper left corner of the crop box. Width and Height are the width and height of the crop box. Units used are a percentage of total image width or height. The parameters must satisfy the following constraints:

```
0 <= Xpos <= 100
0 <= Ypos <= 100
0 <= Xpos + Width <= 100
0 <= Ypos + Height <= 100.
```

```
// request a region going from (25%, 25%) to (75%, 75%) of the full image
pImageDetail->SetRegionOfInterest(25.0, 25.0, 50.0, 50.0);
```

DLS maintains a table of device characteristics for various scanners. In most cases, the rendering that DLS performs by default is satisfactory. However, if DLS knows which device was used, an enhanced render may be performed. KODAK Image Access Client does not yet provide a mechanism for getting the list of devices for which DLS maintains device profiles. If your scanner or camera has a DLS profile (and the name of that profile), specify it in the DigitizingSource field. A related parameter is DigitizedResolution. Your scanner may have a slightly different hardware profile depending upon whether it scanned at 300 versus 600 dpi. If you know the scanned resolution, specify it. In most cases, the default processing works well.

```
// you used a ScanCo model X at 600 dpi to scan the image
pDigitalInfo->SetDigitizingSource(L"ScanCo model X");
pDigitalInfo->SetDigitizedResolution(600);
```

As part of the render process, DLS runs a sharpening algorithm upon each image. If you have already presharpened the image, request that DLS skip this step. `kSharpened` indicates that you have already sharpened the image. `kNoSharpening` indicates that you haven't.

```
pDigitalInfo->SetDigitalSharpening(COSDigitalInfo::kSharpened);
```

The `AddressInfo` sub-object of `ConsumerInfo` is optional. The `AddressInfo` object contains phone numbers, name, and email address. As indicated in the example below, Names may be composed of multiple parts. The first parameter to `SetName()` is an index to indicate which part of the Name is being set. Certain digital outputs require that this be filled in, i.e. 'Upload.'

```
COSAddressInfo *pAddressInfo = 0; pConsumerInfo-  
>GetAddressInfo(pAddressInfo); pAddressInfo->SetDayPhone(L"(301) 123-  
4567"); pAddressInfo->SetNightPhone(L"(301) 123-4568"); pAddressInfo-  
>SetFaxNumber(L"(301) 123-4569"); pAddressInfo-  
>SetEmailAddress(L"bob@bobsisp.com"); pAddressInfo->SetName(0, L"Bob");  
pAddressInfo->SetName(1, L"Smith");
```

Fill in the `Application` parameter so the DLS knows about the application that created the COS file.

```
COSFileInfo *pFileInfo = 0; COSFileMod *pFileMod = 0; pCOSOrder-  
>GetFileInfo(pFileInfo); pFileInfo->GetFileMod(0, pFileMod);
```

You may specify a rotation for the image to indicate which orientation of the image is 'right side up.' Specify the clockwise rotation, in increments of 90 degrees, which will result in the image being 'right side up.' Since DLS does not support this, it is recommended that you submit your images already correctly oriented.

```
pImageDetail->SetRotation(90.0f);
```

## 5. Reference - COS SDK Methods Used

### ***COSOrder::COSOrder***

#### **Description**

Construct the top level COS file object.

#### **Syntax**

`COSOrder(const char * path, COS_FILEOP fop, COSStatus*& fStatus)`

#### **Parameter**

path

fop

fStatus

#### **Description**

Path of COS file

Set to COS\_CREATE for creating a COS order

Returned status

### ***COSOrder::GetVendorInfo***

#### **Description**

Create a VendorOrder sub-object of COSOrder.

#### **Syntax**

`COSStatus COSOrder:: GetVendorInfo (COSVendorInfo* pVendorInfo)`

#### **Parameter**

pVendorInfo

#### **Description**

Pointer to the created COSVendorInfo object

## ***COSOrder::GetConsumerInfo***

### **Description**

Create a ConsumerInfo sub-object of COSOrder.

### **Syntax**

```
COSStatus COSOrder:: GetConsumerInfo(COSConsumerInfo* pConsumerInfo)
```

### **Parameter**

pConsumerInfo

### **Description**

Pointer to the created  
COSConsumerInfo object

## ***COSOrder::GetFileInfo***

### **Description**

Create a FileInfo sub-object of COSOrder.

### **Syntax**

```
COSStatus COSOrder::GetFileInfo(COSFileInfo*& pFileInfo)
```

### **Parameter**

pFileInfo

### **Description**

Pointer to the created COSFileInfo object

## ***COSOrder::AddNewProduct***

### **Description**

Create an additional Product sub-object within the COSOrder.

### **Syntax**

```
COSStatus COSOrder:: AddNewProduct (const char* pTheNewProductName,  
COSProduct *& pTheProduct)
```

### **Parameter**

pTheNewProductName

### **Description**

The ProductName is not used by DLS. For diagnostic purposes, it is useful to use unique and obvious strings such as 'product 1', 'product 2', 'Index Print' etc.

pTheProduct

Pointer to the created COSProduct object

## ***COSOrder::AddImageDetail***

### **Description**

Create an additional ImageDetail sub-object within the COSOrder.

### **Syntax**

```
COSStatus COSOrder:: AddImageDetail (COSImageDetail*&
pImageDetail)
```

### **Parameter**

pImageDetail

### **Description**

Pointer to the created COSImageDetail object

## ***COSOrder::AddNewImage***

### **Description**

Create an additional Image sub-object within the COSOrder.

### **Syntax**

```
COSStatus COSOrder:: AddNewImage (char* pTheName, COSImage*&
pImage )
```

### **Parameter**

pTheName

### **Description**

Image name

### **COS reference**

9.9.27

### **Values**

Wide character string (1-31 characters) used to identify image of interest.

***pTheName Values Note:***

Embedded images: This is imageName or index ID.

*Important: This field must begin with a numeric value. Each name must begin with a unique numeric value (e.g 1, 2, 3 , etc).*

Referenced images: This should be the imageID as returned by the GetImageRefs method. All referenced images must be from the same archived order.

Embedded and Referenced images must not both be placed into the same order file. The COS format supports this, but DLS does not. All images must be either embedded or referenced.

For an index print, set the name to the string 'INDEX\_PRINT'. It is not required to call WriteImage on the new image.

The DLS has a limit of 40 images per an order

NOTE: ImageName is the name assigned to the property set and is NOT the same as the Image-> FileName

pImage                      Pointer to the new  
   COSImage object

***COSOrder::WriteOrder***

**Description**

Flush the COS file to the disk. Call just before the destructor.

**Syntax**

COSStatus COSOrder::WriteOrder()

***COSOrder::~~COSOrder***

**Description**

Destroy the COSOrder object.

**Syntax**

delete pCOSOrder;

***COSVendorInfo::SetVendorOrderNumber*****Description**

Set the OrderID for this order. The orderID is the string representation of a number from 1 to 999999. This number is printed on the back of each print.

**Syntax**

COSStatus COSConsumerInfo:: SetVendorOrderNumber (const COSWideChar \* VendorOrderNumber)

***COSFileMod::SetApplication*****Description**

The name of the creating application and the subsequent editing applications for the COS file. The range is any string(s).

**Syntax**

COSStatus SetApplication(const COS widechar\* pTheApplication)

**Parameter**

pTheApplication

**Description**

A pointer to a created TheApplication object.

***COSVendorInfo::SetVendorOrderNumber*****Description**

Set the OrderID for this order. The orderID must be the string representation of a number from 1 to 999999. This number will printed on the back of each print.

**Syntax**

COSStatus COSConsumerInfo:: SetVendorOrderNumber (const COSWideChar \* VendorOrderNumber)

## ***COSConsumerInfo::SetConsumerID***

### **Description**

Set the customerID for this order.

### **Syntax**

COSStatus COSConsumerInfo:: SetConsumerID(COSWideChar \* ConsumerID)

<b>Parameter</b>	<b>Description</b>	<b>COS reference</b>	<b>Values</b>
ConsumerID	Pointer to ConsumerID wide character string	9.2.1	String, unique identification string; e.g. customer's e-mail address or telephone number w/area code or country. Limited to 64 characters.

## ***COSConsumerInfo::GetAddressInfo***

### **Description**

Create an AddressInfo sub-object within the ConsumerInfo.

### **Syntax**

COSStatus COSConsumerInfo::GetAddressInfo (COSAddressInfo \*& pAddressInfo)

<b>Parameter</b>	<b>Description</b>
pAddressInfo	Pointer to the created COSAddressInfo object

## ***COSAddressInfo::SetDayPhone***

### **Description**

Set the daytime telephone number for this customer.

### **Syntax**

COSStatus COSAddressInfo::SetDayPhone(const COSWideChar \* pPhone)

***COSAddressInfo::SetNightPhone*****Description**

Set the nighttime telephone number for this customer.

**Syntax**

COSStatus COSAddressInfo::SetNightPhone(const COSWideChar \* pPhone)

***COSAddressInfo::SetFaxNumber*****Description**

Set the fax number for this customer.

**Syntax**

COSStatus COSAddressInfo::SetFaxNumber(const COSWideChar \* pPhone)

***COSAddressInfo::SetEmailAddress*****Description**

Set the email address for this customer.

**Syntax**

COSStatus COSAddressInfo::SetEmailAddress(const COSWideChar \* pEmailAddress)

***COSAddressInfo::SetName*****Description**

Set the name of this customer.

**Syntax**

COSStatus COSAddressInfo::SetName(ushort iNameIndex, COSWideChar \* pName)

## ***COSFileInfo::GetFileMod***

### **Description**

Create a FileMod sub-object of FileInfo.

### **Syntax**

```
COSStatus COSFileInfo::GetFileMod(COSFileMod*& pFileMod)
```

<b>Parameter</b>	<b>Description</b>
pFileMod	Pointer to the created COSFileMod object

## ***COSFileMod::SetApplication***

### **Description**

Set the name of the application creating the COS file. iModIndex should be 0 for the creator of the file and incremented by any application that performs further modification.

### **Syntax**

```
COSStatus COSFileMod::SetName(ushort iModIndex, COSWideChar * pName)
```

## ***COSProduct::SetProductClass***

### **Description**

Sets the class (normal vs index) of this product. Use COSProduct:: kStandardPrint or COSProduct:: kIndexPrint.

### **Syntax**

```
COSStatus COSProduct:: SetProductClass (const ushort ProductClass)
```

## ***COSProduct::SetProductDescription***

### **Description**

Set the productID of this product. Always use a ProductID returned by the KODAK Image Access Client GetAvailableProducts method. An exception is for index prints. In that case use L"REGULAR" or L"JUMBO."

### **Syntax**

```
COSStatus COSProduct:: SetProductDescription (COSWideChar * pProdDesc)
```

## ***COSProduct::GetFinishingInfo***

### **Description**

Get the FinishingInfo sub-object of the product object.

### **Syntax**

COSStatus COSProduct:: GetFinishingInfo (COSFinishingInfo\*& pFinishingInfo)

### **Parameter**

pFinishingInfo

### **Description**

Pointer to the created COSFinishingInfo object

## ***COSProduct::GetPrintInfo***

### **Description**

Create a PrintInfo sub-object for this Product.

### **Syntax**

COSStatus COSProduct::GetPrintInfo(COSPrintInfo\*& pPrintInfo)

### **Part**

pPrintInfo

### **Description**

Pointer to the newly created COSPrintInfo object

## ***COSProduct::AddImageDetailRef***

### **Description**

Add an ImageDetail link to this Product.

### **Syntax**

COSStatus COSProduct::AddImageDetailRef (COSImageDetail\* pImageDetail )

## ***COSFinishingInfo::SetQuantity***

### **Description**

Set the quantity of prints for each image linked to by this product.

### **Syntax**

COSStatus COSFinishingInfo:: SetQuantity (const long Quantity)

### **Parameter**

Quantity

### **Description**

Product print quantity

### **COS reference**

9.7.2

### **Values**

DLS limit is 500.

## ***COSPrintInfo::SetShortDimension***

### **Description**

Set the shortest dimension of this product.

### **Syntax**

COSStatus COSPrintInfo::SetShortDimension(float& ShortDimension)

<b>Parameter</b>	<b>Description</b>	<b>COS reference</b>	<b>Values</b>
ShortDimension	Print product height	9.5.12	Short Dimension of print (see 9.5.14 for units)

## ***COSPrintInfo::SetLongDimension***

### **Description**

Set the longest dimension of this product.

### **Syntax**

COSStatus COSPrintInfo::SetLongDimension(float& LongDimension)

<b>Parameter</b>	<b>Description</b>	<b>COS reference</b>	<b>Values</b>
LongDimension	Print product width	9.5.13	Long Dimension of print (see 9.5.14 for units)

## ***COSPrintInfo::SetSizeUnit***

### **Description**

Set the length unit of short and long dimension.

### **Syntax**

COSStatus COSPrintInfo::SetSizeUnit (ushort& SizeUnit)

<b>Parameter</b>	<b>Description</b>	<b>COS reference</b>	<b>Values</b>
SizeUnit	Product dimension unit, i.e. COSPrintInfo::kInch or COSPrintInfo::kMillimeter	9.5.14	inches, mm

## ***COSPrintInfo::SetMediaSurface***

### **Description**

Set the desired paper type.

### **Syntax**

COSStatus COSPrintInfo::SetMediaSurface(const ushort& Surface)

<b>Parameter</b>	<b>COS reference</b>	<b>Values</b>
Surface	9.5.17	COSPrintInfo::kLabDefault COSPrintInfo::kMatte COSPrintInfo::kSemiMatte COSPrintInfo::kGlossy

## ***COSPrintInfo::SetBackPrintMessage***

### **Description**

Set the backprint message for this product.

### **Syntax**

COSStatus COSPrintInfo::SetBackPrintMessage(const COSWideChar \* BackPrintMess)

<b>Parameter</b>	<b>COS reference</b>	<b>Values</b>
BackPrintMess	9.5.20	Any string. May contain a newline character. Up to two lines will be printed.

## ***COSImageDetail::SetImageType***

### **Description**

Set to COSImageDetail::kEmbeddedDigitalData for an embedded image. Set to COSImageDetail::kSeparateDigitalData for an index print or a referenced image.

### **Syntax**

COSStatus COSImageDetail:: SetImageType (ushort& ImageType)

## ***COSImageDetail::SetRegionOfInterest***

### **Description**

Set a crop box for an image. All 4 parameters are percentages from 0 to 100. Xpos and Ypos specify the position of the upper left corner of the crop box relative to the upper left corner of the full image.

### **Syntax**

COSStatus COSImageDetail::SetRegionOfInterest(double Xpos, double Ypos, double Width, double Height)

## ***COSImageDetail::SetRotation***

### **Description**

Apply a clockwise rotation to the image. This rotation indicates which side of the image is 'right side up.' Should be a multiple of 90 degrees. Not supported by DLS.

### **Syntax**

COSStatus COSImageDetail::SetRotation(float Rotation)

## ***COSImageDetail::GetDigitalInfo***

### **Description**

Get the DigitalInfo sub-object of an ImageDetail object.

### **Syntax**

COSStatus COSImageDetail:: GetDigitalInfo (COSDigitalInfo\* pDigitalInfo)

### **Parameter**

pDigitalInfo

### **Description**

Pointer to the internal COSDigitalInfo contained by COSImageDetail

## ***COSImageDetail::AddImageRef***

### **Description**

Link an image, pImage, to this ImageDetail.

### **Syntax**

COSStatus COSImageDetail:: AddImageRef (COSImage\* pImage)

## ***COSDigitalInfo::SetFileFormat***

### **Description**

Set the file format of the image. The choices are: `COSDigitalInfo::kPhotoCD`, `kEXIFJPEG`, `kJPEG`, `kTIFF`, `kFlashPix`, or `kBMPWin` }

### **Syntax**

`COSStatus COSDigitalInfo::SetFileFormat(const ushort FileFormat)`

## ***COSDigitalInfo::SetFileColorSpace***

### **Description**

Set the colorspace of the file. Always set to `COSDigitalInfo::ksRGB` unless you have a photoYCC PhotoCD image, in which case set to `COSDigitalInfo::KPhotoYCC`.

### **Syntax**

`COSStatus COSDigitalInfo::SetFileColorSpace(const ushort FileColorSpace)`

## ***COSDigitalInfo::SetDigitalSharpening***

### **Description**

Indicate whether the image has been presharpened. Set to either `COSDigitalInfo::kNoSharpening` if you haven't presharpened or to `COSDigitalInfo::kSharpened` if you have.

### **Syntax**

`COSStatus COSDigitalInfo::SetDigitalSharpening(const ushort sharpening)`

## ***COSDigitalInfo::SetDigitizingSource***

### **Description**

Set to the name of a scan hardware profile known by DLS for optimal rendering.

### **Syntax**

`COSStatus COSDigitalInfo::SetDigitizingsource(const COSWideChar* Source)`

## ***COSDigitalInfo::SetDigitizedResolution***

### **Description**

Set to the resolution, in dpi, of your DigitizingSource.

### **Syntax**

`COSStatus COSDigitalInfo::SetDigitizedResolution(const ULONG Resolution)`

## ***COSImage::WriteImage***

### **Description**

If the source buffer is a large file, use the WINDOWS memory mapping methods. See the MSDN documentation for CreateFileMapping and MapViewOfFile. If the image is a referenced image, or if the dummy image is being referenced by an index product, there is no need to call this function (WriteImage).

### **Syntax**

COSStatus COSImage:: WriteImage (unsigned char\* buffer, unsigned int ImageStreamSize )

<b>Parameter</b>	<b>Description</b>
buffer	Image buffer
ImageStreamSize	Image size

## 6. Reference – KODAK Image Access Client SDK Methods Used

This section provides an overview of the KODAK Image Access Client SDK methods. For more information, refer to the *KODAK Image Access Client SDK Version 2.0 Developer's Guide*.

### **KIASGetOrders**

#### **Description**

Get a list of archived orders from the server. Call this method when creating an order file with referenced images. Use the OrderID field of a returned OrderInfo structure to populate the VendorOrderNumber COS field. Other, similar methods such as GetOrdersAdmin may also be used for the same purpose. With a DLS server, you should always set the OrderStatus and pOrderID parameters to NOTFOUND and L"", respectively.

#### **Syntax**

```
KIASINT KIASGetOrders( wchar_t *pError, wchar_t *pSessionID,
wchar_t *pCustomerID, wchar_t *pCustomerPassword, KIASINT
StartDate, KIASINT EndDate, KIASINT OrderStatus, wchar_t
*pOrderID, KIASINT* pNumReturned, OrderInfo **pOrdersOut );
```

### **KIASGetImageRefs**

#### **Description**

Get a list of images within an archived order. Call this method when creating an order file with referenced images. Use the ImageID field of the returned KIASImageInfo structure to name the referenced image when calling AddNewImage.

#### **Syntax**

```
KIASINT KIASGetImageRefs( wchar_t *pError, wchar_t *pSessionID,
wchar_t *pOrderID, KIASINT *pNumReturned, KIASImageInfo **pImages
);
```

## ***KIASGetAvailableProducts***

### **Description**

Get a list of supported products from the server. Always call this method prior to populating the ProductDescription, ShortDimension, LongDimension, or SizeUnit COS fields. The list of supported products varies from server to server. Only request products that the server you are using supports. Use a returned KIASProduct to populate the mentioned COS fields as discussed in the 'add a product to the order' section. Place the Description fields of the returned KIASProduct array into a user selectable list on your UI so that the end user may choose the product.

### **Syntax**

```
KIASINT KIASGetAvailableProducts( wchar_t *pError, wchar_t
*pSessionID, KIASLocale Locale, KIASINT* NumReturned, KIASProduct
**pProductsOut );
```

## ***KIASGetAvailableOutputs***

### **Description**

Get a list of supported outputs from the server. The list of supported outputs varies from server to server. Only request outputs that the server you are using supports. Use a returned KIASOutput to populate the mentioned COS fields as discussed in the 'add a digital output to the order' section. Place the Description fields of the returned KIASOutput array into a user selectable list on your UI so that the end user may choose the output.

### **Syntax**

```
KIASINT KIASGetAvailableOutputs( wchar_t *pError, wchar_t
*pSessionID, KIASLocale Locale, KIASINT* NumReturned, KIASOutput
**pProductsOut );
```

## ***KIASSubmitCosOrder***

### **Description**

Send your COS file to the server.

### **Syntax**

```
KIASINT KIASSubmitCosOrder( wchar_t *pError, wchar_t *pSessionID,
wchar_t *pCosFilePath, wchar_t *pOrderIDIn, wchar_t *pOrderIDOut
);
```

## ***KIASFree***

### **Description**

Free the arrays allocated by GetOrders, GetAvailableProducts, or GetImages. It is important to call this method exactly once on each array when you are done using that array.

### **Syntax**

```
KIASINT KIASFree( wchar_t *pError, void* array );
```



EASTMAN KODAK COMPANY  
Kodak Consumer Imaging Division  
Rochester NY 14650  
[www.kodak.com/go/KIASdevelopers](http://www.kodak.com/go/KIASdevelopers)  
© Eastman Kodak Company, 2002