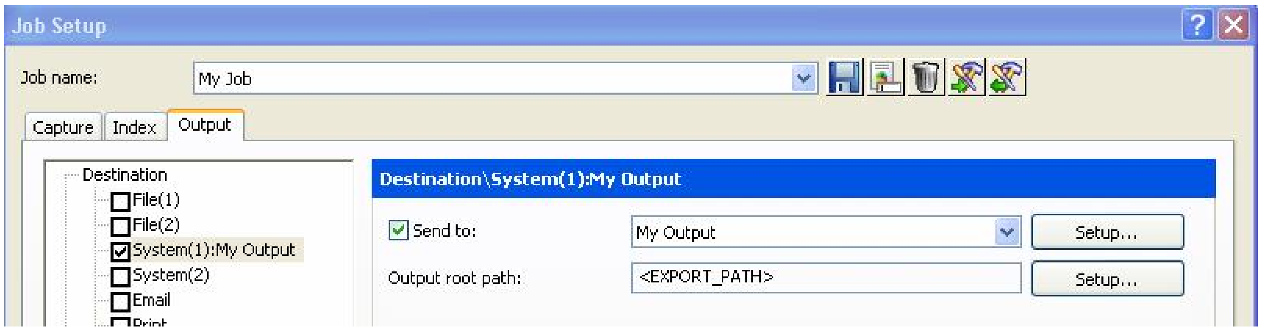
Kodak Capture Pro System Output Destination (SOD) API Documentation

Version 2.0 - July 2009



Job Setup

Job name: My Job

Capture | Index | Output

Destination
- ☐ File(1)
- ☐ File(2)
- ☑ System(1):My Output
- ☐ System(2)
- ☐ Email
- ☐ Print

Destination\System(1):My Output

☑ Send to: My Output Setup...

Output root path: <EXPORT_PATH> Setup...

This document is subject to change, for the latest version please visit:

www.kodak.com/go/capturepro

Table of Contents

1 Purpose of the System Output Destination (SOD)

The Batch Output Format Application Programming Interface makes it possible to integrate a custom batch output as part of Kodak Capture Pro Software.

The System Output Destination was formerly known as the Batch Output Format (BOF) in Capture Pro v1.x.

A custom SOD is a 32-bit DLL that is called from Kodak Capture Pro Software during batch Output and at other times as appropriate. E.g. setup dialog is called during Job setup but not during Output. During batch process, the DLL may produce another index file format, folder structure or image file format.

This document describes each core function of the SOD explaining its purpose complete with VC++ code snippets. A fully functioning source code example incorporating the example code snippets has been made available.

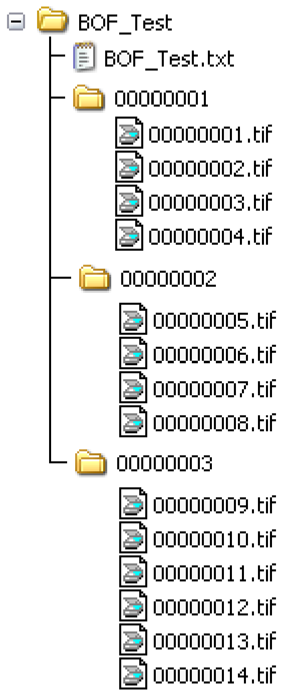
1.1 *Intended audience*

It is assumed programmers are accomplished developers who have created applications / DLL's for Microsoft Windows.

The System Output Destination is a standard Windows DLL which could be developed in any standard programming language. Although the programming language and compiler of the API are not limited to VC++, all declarations are made in C++. The demonstration source and accompanying documentation has been written for Microsoft Visual Studio (2005) programmers who are familiar with the use of DLL's and Class Modules within the Microsoft Visual Studio environment and have some knowledge of the Windows API.

1.2 *Sample source code*

This document was written around portions of source code built under Microsoft Visual Studio 2005 and does NOT use any managed code.
The entire sample source code has been made available with this document.



The Sample source code Output (shown on the right) will create a folder for the Batch of images/data in the path specified in the Job setup, Capture Tab, Output image location field. The Batch folder will contain a .TXT file with any index data and the path to each image that was output. Each Document will be in a sequentially numbered folder that contains the TIFF image files for that document.

1.3 *Using the sample source code*

Copy the sample source files to a suitable folder on your hard disk.

In Microsoft Visual Studio 2005 select File->Open->Project/ Solution and then select:

BOF_EXM\BOF_EXM\BOF_EXM\bof_exm.sln

The project is configured to build the output to the following folder.

BOF_EXM\BOF_EXM\bin\kcbofapi\bof_exm\bof_exm.dpl

The resulting DPL file should be placed in the folder as described in section *6 Installing your custom System BOF*.

In Job setup within Capture Pro, the System Output Destination will show as 'My Output'.

2 System Batch Output Format library description

2.1 Overview

The key objective for the System Output Destination API is to provide way to output a scanned batch into a 3rd party system specific “import” format. This may be a specific file or folder structure on the file system, a specific interface file or process to trigger automatic import of the scanned images and associated index data, or it could feed data directly to the 3rd party system through services over the network.

The System Output Destination needs to control the following operations:

- The setup dialog for SOD specific configuration.
- Copying and optionally formatting the TIFF files with any associated index data for output to the 3rd party system.
- System Output Destination identification.
- Help information for use within the Windows help system (if desired).
- Error recovery cleanup for any partially output batch.

To clarify the terminology and logic flow of the SOD API, we will first look at the logical hierarchy of the contents of a batch of scanned papers.

A batch is a logical structure to manage images and index data in a multiple level tree. The levels are defined as:
Batch Level
Document Level
Page Level
Image Level

The index data associated with these level groups can in theory be associated at any of the four levels, but for version 1.x of Kodak Capture Pro software only 2 levels are supported. These are the batch level index and the document level index.

A Batch is a collection of Documents. Associated Batch Level index data is related to all documents, pages and images contained within the batch.

A Document is a collection of pages that are logically related to each other (e.g. a report or multi-page letter). Associated Document Level index data is related to all pages and images contained within the document.

A Page is a collection of images that are the result from the digital imaging of a single piece of paper (front and rear). Currently Page Level index data is not supported. Note that scanning a Page can result in many individual image files taken from a single side of a piece of paper depending on system settings within the main Capture Pro software. E.g scanning an A4 page set to capture both front and back in both colour and black & white will result in a total of 4 image files (2 per side of the piece of paper).

An Image describes an individual image file as captured from the front or rear of a piece of paper. Currently Image Level index data is not supported.

Index data is transferred in a list of key pairs. For each key pair there is an index field name (key) and index field value (value). For example, if there is a batch level index named “BATCHNAME” and its value is “BATCH001”, then the key is “BATCHNAME” and the value is “BATCH001”.

The index key is defined in job setup. The index key name is the Index Field Label as defined in Job setup. The index value is assigned during scanning for automatically indexed index fields (e.g. barcode. OCR or system generated data) or during manual indexing. Index value is defined as a text string.

This means that during Job setup, the SOD API will be able to access the index key list but not the index value for the defined keys.

Data for the batch that is not specifically index data (output paths, job level information, user information or batch metrics) are available for use by the System Output Destination if required.

2.2 The SOD output process

As mentioned, for many of the SOD calls there are 4 ‘Levels’ that need to be dealt with:

Level Value	Level Name
1	Image
2	Page
3	Document
4	Batch

Many of the SOD functions will be called 4 times (at start of batch Output), in nested loops, once for each level. Not all functions are supported at all levels, but the function may still be called with NULL data pointers.

e.g. Indexing is only supported at Batch and Document level, but the ClearIndexKey function will be called 4 times at the start of the Output process:
At the Batch level with valid data pointers.
At the Document level with valid data pointers.
At the Page level with NULL data pointers.
At the Image level with NULL data pointers.
Developers of System Output Destination’s should allow for all 4 levels without the need to use data at the page and image level. This will allow for possible changes to the availability of Image and Page level index data pointers.

The variables that are available to the SOD are detailed later in the Job Settings and Batch Metrics Variables section of the documentation.

2.2.1 Function overview

A Kodak Capture Pro Software compatible batch output format library needs to support the following routines during the Output operation:

```
SetOutputFileType
ClearIndexKey
ClearIndexValue
AddIndexKey
SetIndexValue
BeginProcess
AddImageFile
EndProcess
AbortProcess
GetFilePathName
SetFilePathName
```

2.2.1.1 SetOutputFileType (dwFileTypeAll, dwFileTypeBW, dwFileTypeCG, dwGroupModeAll, dwGroupModeBW, dwGroupModeCG)

At the start of the output, the plugin is informed of the output file types (TIF, JPG, PDF, etc.) and group modes (single image, group by batch, group by document, group by page).

2.2.1.2 ClearIndexValue(Level)

ClearIndexValue() is called once per level to clear ALL index values from the SOD at the specified level when starting Output. It is then called prior to adding index values at the specified level. E.g. at the end of document 1, ClearIndexValue(3) will be called to clear the index values for document 1 before the values for document 2 are set using the SetIndexValue() function.

2.2.1.3 AddIndexKey(Level, Key Name)

AddIndexKey() is called once for each index field defined in Job setup for the specified level. E.g for a document level index key in a Job where 3 document level index fields are defined, AddIndexKey() will be called 3 times at the document level.

2.2.1.4 SetIndexValue(Level, Key Name, Value)

SetIndexValue() is called once for each index Key at the specified level.

2.2.1.5 BeginProcess()

BeginProcess() is called to signal to the SOD that the defined level can be processed by the SOD.
E.g. a BeginProcess(4) signals that the batch setup information is complete and we are about to begin processing the data within the batch (documents). At this point you may want to create the output folder to hold the processed documents. A BeginProcess(3) signals that the Document setup is complete and that we are about to process the Pages. At this point you may want to retrieve an index value to name the output document by.

2.2.1.6 AddImageFile ()

This is a level 1 operation. Capture Pro passes image information and the path to the TIFF file. This path is to a temporary copy of the source file. Any changes to this file will be discarded at the end of the Output process. See section 4.1 Image information for details of the information passed.

2.2.1.7 EndProcess()

EndProcess signals the end of the current level operation. E.g. EndProcess(3) is called at the end of a document, this can be taken as a trigger to close the current document.

2.2.1.8 AbortProcess()

AbortProcess() is called when Capture Pro would like to stop the currently processing batch output operation. This call will be used in the event that the output operation is cancelled by the user or a system level error happens. Capture Pro will invoke AbortProcess() anytime after the first occurrence of BeginProcess() had been invoked, so the SOD will probably need to do some cleaning up work of the partially output batch. E.g. if the SOD had created some files/folders during the preceding output steps, these would need to be removed in preparation for a retry of the output operation at a later time.

2.2.1.9 GetFilePathName()

GetFilePathName() is called when the application is ready to create a file with the image(s). The SOD will need to provide a file location and name, and can use a formula with elements such as <DEFAULT_EXT>. These formula elements are the same as in the File() File name Setup dialogs. The application will indicate the channel (black and white, color/grayscale or all).

2.2.1.10 SetFilePathName()

With SetFilePathName Capture Pro will set the file name in the SOD. If GetFilePathName had formula elements, they will have been evaluated when SetFilePathName is called.

3 SOD output process overview

The System Output Destination will be called from Kodak Capture Pro in response to a batch being queued for output. The outline calling process flow from the start of the Output operation is as detailed below:

```
SetOutputFileType

ClearIndexKey(Batch)
ClearIndexKey(Document)
ClearIndexKey(Page)
ClearIndexKey(Image)
ClearIndexValue(Batch)
ClearIndexValue(Document)
ClearIndexValue(Page)
ClearIndexValue(Image)

For Each Batch Index Field
      AddIndexKey(Batch)
      ClearIndexValue(Batch)
      SetIndexValue(Batch)

For Each Document Index Field
      AddIndexKey(Document)

BeginProcess(Batch)
      GetOutputFilePath (images grouped by batch)
      SetOutputFilePath (images grouped by batch)

      For Each Document Index Field
            ClearIndexValue(Document)
            SetIndexValue(Document)

      For Each Document in the batch
            BeginProcess(Document)
                  GetOutputFilePath (images grouped by document)
                  SetOutputFilePath (images grouped by document)

                  ClearIndexValue(Page)

                  For Each Page in the document
                        BeginProcess(Page)
                              GetOutputFilePath (images grouped by page)
                              SetOutputFilePath (images grouped by page)

                              For Each Image in the page
                                    BeginProcess(Image)
                                    AddImageFile()
                                    GetOutputFilePath (single-image file)
                                    SetOutputFilePath (single-image file)

                                    EndProcess(Image)
                        EndProcess(Page)
                  ClearIndexValue(Page)
            EndProcess(Document)
            ClearIndexValue(Document)
EndProcess(Batch)
```

4 Data structures

The Capture Pro SOD makes use of several data structures define in kcbofapi.h. Some of these structures are detailed below for clarity while reading the example functions.
For the full supporting data structures, refer to the kcadvbofapi.h, kcbofapi.h, myplugin.h and the other header files included in the sample project

```
struct KCBOFAPI_BEGINPROCESS_PARAM
{
   __int32                  nLevel;          // level
   DFAPI_TEXT128_TYPE       szLevelName;     // level name  (eg. "image", "page",
                                             "document" or “batch”
   DFAPI_TEXT128_TYPE       szLevelID;       // level ID, eg. image ID, page ID,
                                             document ID, etc.
};
```

KCBOFAPI_LEVEL : Level data definitions are as follows

```
KCBOFAPI_LEVEL_IMAGE            0x00000001  //!< image level
KCBOFAPI_LEVEL_PAGE             0x00000002  //!< page level
KCBOFAPI_LEVEL_DOCUMENT         0x00000003  //!< document level
KCBOFAPI_LEVEL_BATCH            0x00000004  //!< batch level
```

KCBOFAPI_IMAGESIDE: Image Side definitions are as follows

```
KCBOFAPI_IMAGESIDE_UNKNOWN      0x0000      // unknown (e.g Imported Image)
KCBOFAPI_IMAGESIDE_FRONT        0x0001      // front side
KCBOFAPI_IMAGESIDE_REAR         0x0002      // rear side
KCBOFAPI_IMAGESIDE_BOTH         0x0003      // both sides.eg. merged image with
                                            both sides)
```

KCBOFAPI_IMAGETYPE; Image Type definitions are as follow

```
KCBOFAPI_IMAGETYPE_UNKNOWN      0x0000      // unknown (e.g Imported Image)
KCBOFAPI_IMAGETYPE_BITONAL      0x0001      // bitonal image
KCBOFAPI_IMAGETYPE_GRAYSCALE    0x0002      // grayscale image
KCBOFAPI_IMAGETYPE_COLOR        0x0003      // color image
```

KCBOFAPI_FILETYPE

```
KCBOFAPI_FILETYPE_NONE                              0X00000000
KCBOFAPI_FILETYPE_TIFF                              0X00000001
KCBOFAPI_FILETYPE_PDF                               0X00000002
KCBOFAPI_FILETYPE_SPDF                              0X00000004
KCBOFAPI_FILETYPE_JPEG                              0X00000008
KCBOFAPI_FILETYPE_JPEG2000                          0X00000010
KCBOFAPI_FILETYPE_JBIG                              0X00000020
KCBOFAPI_FILETYPE_GIF                               0X00000040
KCBOFAPI_FILETYPE_PNG                               0X00000080
KCBOFAPI_FILETYPE_BMP                               0X00000100
KCBOFAPI_FILETYPE_TXT                               0X00000200
KCBOFAPI_FILETYPE_RTF                               0X00000400
KCBOFAPI_FILETYPE_DOC                               0X00000800
KCBOFAPI_FILETYPE_XML                               0X00001000
KCBOFAPI_FILETYPE_PDFD                              0X00002000
```

KCBOFAPI_GROUPMODE

```
KCBOFAPI_GROUPMODE_SINGLE                            0X00000000
KCBOFAPI_GROUPMODE_PAGE                              0X00000001
KCBOFAPI_GROUPMODE_DOCUMENT                          0X00000002
KCBOFAPI_GROUPMODE_BATCH                             0X00000004

KCBOFAPI_CHANNEL
KCBOFAPI_CHANNEL_ALL                                 0X00000001
KCBOFAPI_CHANNEL_BW                                  0X00000002
KCBOFAPI_CHANNEL_CG                                  0X00000003
```

4.1 Image information

Image information is passed to the SOD with each pointer to the temporary copy of the images within the batch. The information passed is as below:

KCBOFAPI_IMAGEINFO: ImageInfo data structure definitions

```
dwID              // image ID. Count of the pieces of paper scanned in the
                  current scan session
wScannedSide      // Original side: KCBOFAPI_IMAGESIDE ID as captured by CP
wCurrentSide      // Current side: KCBOFAPI_IMAGESIDE ID as processed by CP
dwHWPartNo        // Hardware part number: Not currently used
dwSWPartNo        // Software part number: Identifies the individual section of
                  a source image when the image ‘Split’ Function has been used,
                  set to 0 if Split not used
wImageType        // Image Type: Indentifies the source Image Type as defined by
                  KCBOFAPI_IMAGETYPE
dwImageSeq        // Image Sequence: Image sequence within the document. Always
                  starts at 1 for every document
dwPageSeq         // Page Sequence: Identifies the Page Sequence within a
                  document. Always starts at 1 for every document
dwDocumentSeq     // Document Sequence: Document Sequence within the batch.
                  Always starts at 1 for every batch

bIAInfo           // TRUE | FALSE. If TRUE the Scanner assigned Image Address
                  (IA) information is valid and can be used as identified by ‘IA
                  Info’ in the following descriptions
wIALevel          // IA Info: Image Address level
szIAFixed         // IA Info: Fixed field
szIALevel1        // IA Info: Level 1
szIALevel2        // IA Info: Level 2
szIALevel3        // IA Info: Level 3
szIAEntire        // IA Info: Entire Image Address
szPrintString     // Print string: If defined, contains the Print String used by
                  the scanner’s printer for this image
dwFlags;          // reserved, for future use
```

Please note: the Scanner assigned Image Address (IA) references 3 IA levels. These levels are related to image address levels within the scanner and NOT Capture Pro levels. The IA levels have nothing to do with the 4 levels used during the SOD output.

4.1.1 Images Flagged within Capture Pro

The status of any images that have been Flagged within Capture Pro is currently not supported by the System Output Destination API. The Flag bit is recorded in the TIFF tag 269 (document name tag). The details of the information that Capture Pro places in this TIFF tag is described below.

TIFF Tag 269: The Document Name tag contains 54 bytes of data with the following structure:

Tag Field	# of Bytes	Value
Structure Version	4	Always "DN0P"
Scanner Serial Number	16	Left-justified, padded with spaces. Available if the scanner has an accessible serial number
Session number	10	Internal Session number
Page No.	10	Capture Pro Page ID. Right justified; padded with zero’s
Front/Rear Process	1	Front/Rear Status as output processed (0=Front; 1=Rear)
Front/Rear Scanned	1	Front/Rear Status as scanned (0=Front; 1=Rear)
Fragment Index	2	When splitting images (in page setup) this field contains fragment info. Value is "00" when not splitting a page.
Flagged Status	1	Flag status (0=Not Flagged, 1=Flagged)
Filler	9	Always ".----!--" followed by trailing Null character.

Note: The Front/Rear Process state could be different from the Front/Rear scanned state. The Front/Rear Scanned state refers to the image position on the page as scanned. The Front/Rear Processed state refers to the logical image position when the batch is sent for Output. The image side can be changed when manually changing the image order or when images are split/re-ordered as part of a folded page (folded brochure scanning) page setup.

5 Functions in detail

5.1 Unicode

IMPORTANT. Kodak Capture Pro software supports Unicode SODs only so ensure your project is compiled for Unicode.

5.2 SOD naming and identification

The file name and folder name of the SOD must conform to the following pattern.
BOF_XYZ for the folder and hence the filename will be **BOF_XYZ.DPL**

The **BOF_** must remain as it signifies this as a System Output Destination.
The **XYZ** is a unique 3-digit identification part to identify this individual System Output Destination. E.g the example supplied builds to a naming of BOF_EXM, the EXM naming was chosen because this is an EXaMple SOD.
This name should be unique within the KCBOFAPI folder.

The SOD needs to respond to calls from Capture Pro for the following:

- The Name to display as the SOD name in the System Output Destination list. This name is returned to Capture Pro in response to the GetPluginInfoEx() call.

- The description and version information for the DLL listing within the Help, About Capture Pro, System Info… screen. This description and version information should match the version information from the project resource file.

5.3 Default Functions and supporting code

Several functions listed within the example source code are not detailed in this document. These functions call supporting functions in the inline code and should not be changed. An example of this type of function is the CreateUI() function.

5.4 Debug settings

By default *debug* is not supported within the Capture Pro environment and therefore you cannot *debug* your SOD during testing on a standard installation. To use Debug within your SOD during development, you must first enable debug within the Capture Pro environment. To do this simply add the following entry into the **general** section of the **env.info** file. The *env.info* file is located in the Capture Pro\System folder in the program files folder on your system. On an English Operating System the path for this file will be ***C:\Program Files\Kodak\Capture Pro\System\env.info***

```
[general]
Foregroundprocess = 1
```

If this file does not exist on your system, it can be manually created whilst KC is not running. If there is no `[general]` section in the *env.info* file, add it to the bottom of the file.

Remember to remove this setting after testing is completed & before testing the final non-debug version of your SOD.

5.5 Index Functions

Index data is stored in pairs, Index Field (Key) and Index Data (Value),
The data is stored within a *CList* as defined below (*MyPlugin.h*)

```
struct Index_Pair{
            int         nLevel;
            CString     strKey;
            CString     strValue;
      };
      CList<Index_Pair, Index_Pair&>    m_lstIndex;
```

Please refer to the KCBOFAPI.h header file for details of the index function data PARAM* definitions.

5.5.1 Function ClearIndexKey

	Data Type	**Comment**
Input	KCBOFAPI_CLEARINDEXKEY_PARAM*	Index Level
return	bool	TRUE if no errors else FALSE

This function is used to reset the index list "Key" data, typically in preparation to receive index data for the appropriate index level. Indexing is only supported at level 4 (Batch) and level 3 (Document)

```
DFAPI_PLUGIN_IMPLEMENT_STD_INTERFACE(CMyPlugin, KCBOFAPI_ClearIndexKey)
{
      KCBOFAPI_CLEARINDEXKEY_PARAM* p = (KCBOFAPI_CLEARINDEXKEY_PARAM*) param;
      if (p == NULL)
            return FALSE;
      // Level
      // 1  Image
      // 2  Page
      // 3  Document
      // 4  Batch

      // Get head of list
      POSITION pos = m_lstIndex.GetHeadPosition();
      while (pos)
      {     // traverse down list setting Value to “” for this level
            Index_Pair& item = m_lstIndex.GetNext(pos);
            if (item.nLevel == p->nLevel)
            {
                  item.strKey = _T("");
            }
      }
      return TRUE;
}
```

5.5.2 Function ClearIndexValue

```
            Data Type                           Comment
Input       KCBOFAPI_CLEARINDEXVALUE_PARAM*     Index Level
return      bool                                TRUE if no errors else FALSE
```

This function is used to reset the index list “Value” data, typically in preparation to receive new index values for the appropriate index level. Indexing is only supported at level 4 (Batch) and level 3 (Document)

```
DFAPI_PLUGIN_IMPLEMENT_STD_INTERFACE(CMyPlugin, KCBOFAPI_ClearIndexValue)
{
      KCBOFAPI_CLEARINDEXVALUE_PARAM* p = (KCBOFAPI_CLEARINDEXVALUE_PARAM*) param;
      if (p == NULL)
            return FALSE;

      // Level
      // 1  Image
      // 2  Page
      // 3  Document
      // 4  Batch

      // Get head of list
      POSITION pos = m_lstIndex.GetHeadPosition();
      while (pos)
      {     // traverse down list setting Value to “” for this level
            POSITION posOld = pos;
            Index_Pair& item = m_lstIndex.GetNext(pos);
            if (item.nLevel == p->nLevel)
            {
                  item.strValue = _T("");
            }
      }
      return TRUE;
}
```

5.5.3 Function AddIndexKey

	Data Type	Comment
Input	KCBOFAPI_ADDINDEXKEY_PARAM*	Index Level and Key name
return	bool	TRUE if no errors else FALSE

This function adds the defined index field (key) to the previously cleared list. This function will be called repeatedly, once for each defined index field.

```
DFAPI_PLUGIN_IMPLEMENT_STD_INTERFACE(CMyPlugin, KCBOFAPI_AddIndexKey)
{
      KCBOFAPI_ADDINDEXKEY_PARAM* p = (KCBOFAPI_ADDINDEXKEY_PARAM*) param;
      if (p == NULL)
            return FALSE;

      // Level
      // 1  Image
      // 2  Page
      // 3  Document
      // 4  Batch

      // Define local CArray object
      Index_Pair item;
      // add level
      item.nLevel = p->nLevel;
      // and key value
      item.strKey = p->szIndexKey;
      // and add to list
      m_lstIndex.AddTail(item);

      //
      return TRUE;
}
```

5.5.4 Function SetIndexValue

	Data Type	Comment
Input	KCBOFAPI_ADDINDEXKEY_PARAM*	Index Level, Key name and Value data
return	bool	TRUE if no errors else FALSE

This function adds the defined index data to the index Key. This function will be called repeatedly, once for each defined index field.

```
DFAPI_PLUGIN_IMPLEMENT_STD_INTERFACE(CMyPlugin, KCBOFAPI_SetIndexValue)
{
      KCBOFAPI_SETINDEXVALUE_PARAM* p = (KCBOFAPI_SETINDEXVALUE_PARAM*) param;
      if (p == NULL)
            return FALSE;

      // Level
      // 1  Image
      // 2  Page
      // 3  Document
      // 4  Batch

      // get head of list
      POSITION pos = m_lstIndex.GetHeadPosition();
      while (pos)
      {     // Traverse through list setting the value data where the level
            // and key match
            Index_Pair& item = m_lstIndex.GetNext(pos);
            if( item.nLevel == p->nLevel && item.strKey == p->szIndexKey )
            {
                  item.strValue = p->szIndexValue;
            }
      }
      //
      return TRUE;
}
```

5.6 *Process Functions*

5.6.1 Function BeginProcess

	Data Type	**Comment**
Input	KCBOFAPI_BEGINPROCESS_PARAM*)	Level and Level data
return	bool	TRUE if no errors else FALSE

This function is called at the start of each Level (Batch, Document, Page and Image). This is typically where you create the output folder (for Batch), create the output document and its index data (for Document), or output the actual image (for Image)

```
DFAPI_PLUGIN_IMPLEMENT_STD_INTERFACE(CMyPlugin, KCBOFAPI_BeginProcess)
{
      KCBOFAPI_BEGINPROCESS_PARAM* p = (KCBOFAPI_BEGINPROCESS_PARAM*) param;
      if (p == NULL)
            return FALSE;


      //begin process pass details of Level, Name and ID
      return BeginProcess((int) p->nLevel, p->szLevelName, p->szLevelID);
}
```

5.6.1.1 Function CMyPlugin::BeginProcess

Locally defined function to make the sample code more readable. This will handle the BeginProcess function call based on level.
For clarity this function is used to ensure we are where we think we are and then call the appropriate Process function based on the current level.

```
BOOL CMyPlugin::BeginProcess(int nLevel, LPCTSTR lpszLevelName, LPCTSTR lpszLevelID)
{
      //check level
      switch (nLevel){
            case KCBOFAPI_LEVEL_BATCH:
                  if (m_nCurrentLevel != -1)
                        return FALSE;
                  m_nCurrentLevel = KCBOFAPI_LEVEL_BATCH;
                  m_strBatchLevelName = lpszLevelName;
                  m_strBatchLevelID = lpszLevelID;
                  return OnBeginProcessBatch();
            case KCBOFAPI_LEVEL_DOCUMENT:
                  if (m_nCurrentLevel != KCBOFAPI_LEVEL_BATCH)
                        return FALSE;
                  m_nCurrentLevel = KCBOFAPI_LEVEL_DOCUMENT;
                  m_strDocumentLevelName = lpszLevelName;
                  m_strDocumentLevelID = lpszLevelID;
                  return OnBeginProcessDocument();

            case KCBOFAPI_LEVEL_PAGE:
                  if (m_nCurrentLevel != KCBOFAPI_LEVEL_DOCUMENT)
                        return FALSE;
                  m_nCurrentLevel = KCBOFAPI_LEVEL_PAGE;
                  m_strPageLevelName = lpszLevelName;
                  m_strPageLevelID = lpszLevelID;
                  return OnBeginProcessPage();

            case KCBOFAPI_LEVEL_IMAGE:
                  if (m_nCurrentLevel != KCBOFAPI_LEVEL_PAGE)
                        return FALSE;
                  m_nCurrentLevel = KCBOFAPI_LEVEL_IMAGE;
                  m_strImageLevelName = lpszLevelName;
                  m_strImageLevelID = lpszLevelID;
                  return OnBeginProcessImage();
      }

      return FALSE;
}
```

5.6.2 Function AddImageFile

	Data Type	**Comment**
Input	KCBOFAPI_ADDIMAGEFILE_PARAM*)	Image Path and image data structure
return	bool	TRUE if no errors else FALSE

This function is called to do the actual image output. Typically the file will be saved in the required format to a location set in the Job setup.

```
DFAPI_PLUGIN_IMPLEMENT_STD_INTERFACE(CMyPlugin, KCBOFAPI_AddImageFile)
{
      KCBOFAPI_ADDIMAGEFILE_PARAM* p = (KCBOFAPI_ADDIMAGEFILE_PARAM*) param;
      if (p == NULL)
            return FALSE;

      //add image file
      return OnAddImageFile(p->szImageFilePath, p->info);
}
```

5.6.2.1 Function CMyPlugin::OnAddImageFile

Locally defined function to make the sample code more readable. This will handle the AddImageFile function. For clarity this function is used to save the image file to the Job output folder and name it based on the image sequence number.

```
BOOL CMyPlugin::OnAddImageFile(LPCTSTR lpszImageFilePath, KCBOFAPI_IMAGEINFO&
info)
{


      // While not used in this example, the following line gets the type of
image that is being processed
      BOOL bColorGrayScale = (info.wImageType == KCBOFAPI_IMAGETYPE_GRAYSCALE ||
info.wImageType == KCBOFAPI_IMAGETYPE_COLOR);

      CString strDst;
      strDst.Format(_T("%s%08d.txt"),m_strBatchDir, m_nImageCounter);

      m_nImageCounter++;

      //write image path name to .txt file if appropriate
      if (!m_data.m_lBatchImport)
      {
            if (!m_indexWriter.WriteImagePath (strDst))
            {
                  SetLastError(EC_WRITE_FILE);
                  return FALSE;
            }
      }

      // Add target Image name to array.
      m_arrImages.Add(strDst);

      return TRUE;

}
```

Note: In earlier versions of this API images would be copied in this method or appended to multi-page formatted files. These files are now created by the application.

5.6.2.2 Function GetFilePathName

	Data Type	Comment
Input	KCBOFAPI_FILEPATHNAME_PARAM*	channel, file path
return	bool	TRUE if no errors else FALSE

This function is called whenever the application needs to generate a file, so this depends on whether single-image or a multi-image file type was selected and in the case of a mult-image file type, what the batch mode is. The plugin may return a formula with <keywords> as used by the file outputs among others.

```
DFAPI_PLUGIN_IMPLEMENT_STD_INTERFACE(CMyPlugin, KCBOFAPI_GetFilePathName)
{
      KCBOFAPI_FILEPATHNAME_PARAM* p = (KCBOFAPI_FILEPATHNAME_PARAM*) param;
      if (p == NULL) return FALSE;

      CString strFilePathFormula;
      if(GetFilePathName(p->dwChannel, strFilePathFormula))
      {
            p->szPathName.SetEx(strFilePathFormula);
            return TRUE;
      }
      return FALSE;
}
```

5.6.2.3 Function GetFilePathName

	Data Type	Comment
Input	DWORD*	channel
Ouput	LPCTSTR	lpszFilePathName
return	bool	TRUE if no errors else FALSE

Locally defined function to make the sample code more readable. This will handle the AbortProcess function call.

```
BOOL CMyPlugin::GetFilePathName(DWORD dwChannel, CString &strFilePathFormula)
{
	CString strImageName;
	strImageName.Format(_T("%08d"),m_nImageCounter);

	// The application will interpret the returned formula before generating 
the
	// image file.
	strFilePathFormula = m_strBatchDir + strImageName + _T("<DEFAULT_EXT>");

	return TRUE;
}
```

5.6.2.4 SetFilePathName

	Data Type	Comment
Input	KCBOFAPI_FILEPATHNAME_PARAM*	channel, file path
return	bool	TRUE if no errors else FALSE

This function is called whenever the application needs to generate a file, so this depends on whether single-image or a multi-image file type was selected and in the case of a mult-image file type, what the batch mode is. The application has first call GetFilePathName() to request a filename formula from the plugin, SetFilePathName sends the evaluated name back to the plugin.

```
DFAPI_PLUGIN_IMPLEMENT_STD_INTERFACE(CMyPlugin, KCBOFAPI_SetFilePathName)
{
	KCBOFAPI_FILEPATHNAME_PARAM* p = (KCBOFAPI_FILEPATHNAME_PARAM*) param;
	if (p == NULL) return FALSE;

	return SetFilePathName(p->dwChannel, p->szPathName);

}
```

5.6.2.5 Function SetFilePathName

	Data Type	Comment
Input	DWORD*	channel
Ouput	LPCTSTR	lpszFilePathName
return	bool	TRUE if no errors else FALSE

Locally defined function to make the sample code more readable. This will handle the SetFilePathname function call.

```
BOOL CMyPlugin::SetFilePathName(DWORD dwChannel, LPCTSTR lpszFilePathName)
{
      // If a formula element was used in GetFilePathName(), it will be
substituted // when SetFilePathName is called

      //switch(dwChannel)
      //{
      //case KCBOFAPI_CHANNEL_ALL:
      //    m_strCurFilePathAll = lpszFilePathName;
      //    break;
      //case KCBOFAPI_CHANNEL_BW:
      //    m_strCurFilePathBW = lpszFilePathName;
      //    break;
      //case KCBOFAPI_CHANNEL_CG:
      //    m_strCurFilePathCG = lpszFilePathName;
      //    break;
      //default:
      //    return FALSE;
      //}

      m_strCurProcessFile = lpszFilePathName;
      return TRUE;
}
```

5.6.2.6 Function   EndProcess

	Data Type	**Comment**
Input	KCBOFAPI_ENDPROCESS_PARAM*	Level
return	bool	TRUE if no errors else FALSE

This function is called at the end of each Level (Batch, Document, Page and Image). This is typically where you close the document and update any statistics based file, or close the batch and create a trigger file for the 3rd party system.

```
DFAPI_PLUGIN_IMPLEMENT_STD_INTERFACE(CMyPlugin, KCBOFAPI_EndProcess)
{
      KCBOFAPI_ENDPROCESS_PARAM* p = (KCBOFAPI_ENDPROCESS_PARAM*) param;
      if (p == NULL)
            return FALSE;

      return EndProcess(p->nLevel);
}
```

5.6.2.7 Function   CMyPlugin::EndProcess

Locally defined function to make the sample code more readable. This will handle the EndProcess function call based on level.
For clarity this function is used to ensure we are where we think we are and then call the appropriate Process function based on the current level.

```
BOOL CMyPlugin::EndProcess(int nLevel)
{
      // based on Level, set global variables so the BOF
      // knows where it is in the process.
      // This will ensure we will complete one process before
      // another is started then call the appropriate EndProcess
      // function for the current level

      switch (nLevel)
```

```
	{
		case KCBOFAPI_LEVEL_BATCH:
			if (m_nCurrentLevel != KCBOFAPI_LEVEL_BATCH)
				return FALSE;
			if (!OnEndProcessBatch())
				return FALSE;
			m_nCurrentLevel = -1;
			m_strBatchLevelName.Empty();
			m_strBatchLevelID.Empty();
			return TRUE;

		case KCBOFAPI_LEVEL_DOCUMENT:
			if (m_nCurrentLevel != KCBOFAPI_LEVEL_DOCUMENT)
				return FALSE;
			if (!OnEndProcessDocument())
				return FALSE;
			m_nCurrentLevel = KCBOFAPI_LEVEL_BATCH;
			m_strDocumentLevelName.Empty();
			m_strDocumentLevelID.Empty();
			return TRUE;

		case KCBOFAPI_LEVEL_PAGE:
			if (m_nCurrentLevel != KCBOFAPI_LEVEL_PAGE)
				return FALSE;
			if (!OnEndProcessPage())
				return FALSE;
			m_nCurrentLevel = KCBOFAPI_LEVEL_DOCUMENT;
			m_strPageLevelName.Empty();
			m_strPageLevelID.Empty();
			return TRUE;

		case KCBOFAPI_LEVEL_IMAGE:
			if (m_nCurrentLevel != KCBOFAPI_LEVEL_IMAGE)
				return FALSE;
			if (!OnEndProcessImage())
				return FALSE;
			m_nCurrentLevel = KCBOFAPI_LEVEL_PAGE;
			m_strImageLevelName.Empty();
			m_strImageLevelID.Empty();
			return TRUE;
	}

	// Default is to return FALSE
	return FALSE;
}
```

5.6.3 Function AbortProcess

	Data Type
Input	None
return	void

This function is called to trigger any SOD specific cleanup. Typically this will be called in response to the SOD returning FALSE in an earlier call or if Capture Pro has detected an error. Now is the time to remove any partially output batch from the target location so that after the error condition is cleared, the user can re-submit the batch for processing.

```
DFAPI_PLUGIN_IMPLEMENT_STD_INTERFACE(CMyPlugin, KCBOFAPI_AbortProcess)
{
      // This function is called in response to an error
      KCBOFAPI_ABORTPROCESS_PARAM* p = (KCBOFAPI_ABORTPROCESS_PARAM*) param;
      if (p == NULL)
            return FALSE;

      // Cleanup anything we have started
      OnAbortProcess();

      return TRUE;
}
```

5.6.3.1 *Function CMyPlugin::AbortProcess*

Locally defined function to make the sample code more readable. This will handle the AbortProcess function call.

```
void CMyPlugin::OnAbortProcess()
{
        // This function will be called if any of the above functions have
        // returned FALSE
        // or Capture Pro has dectected an error.
        // It is the responsibility of this BOF to clearup in the event
        // of an error

        // If the index file is open, close it
        if (m_indexWriter.IsOpen())
                m_indexWriter.Close();
        // and then delete it
        DeleteFile(m_strIndexFile);

        //During process, we added the path to each image written to
        // an array.
        // Delete each image we have output
        for(int i=0; i<m_arrImages.GetCount(); i++)
        {
                DeleteFile(m_arrImages[i]);
        }
        // And finally remove the batch folder we created.
        RemoveDirectory(m_strBatchDir);
}
```

5.6.4 Function GetErrorMsg

	Data Type	Comment
Input	LPCTSTR	Language
Input	DWORD	Error Code
Input	CString&	Error Message
return	bool	TRUE if no errors else FALSE

Typically GetErrorMsg() will be called in response to the SOD returning FALSE in an earlier call. This function returns an error description relating to the error number that was previously sent to Capture Pro in an earlier call where you called SetLastError().

```
BOOL CMyPlugin::GetErrorMsg(LPCTSTR lpszLanguage, DWORD dwErrCode, CString&
strMsg)
{
      // If you encountered an error, call SetLastError with an
      // Error No and then returned FALSE.
      // After the BOF has returned FALSE, Capture Pro will call this
      // function to get the error description.

      // IMPORTANT
      // You should not display a MessageBox in this fuction

      switch (dwErrCode)
    {
    case EC_CREATE_FOLDER:
        strMsg = _T("Cannot create Batch folder!");
        break;
      case EC_COPY_FILE:
        strMsg = _T("Failed to copy file!");
        break;
      case EC_OPEN_INDEX:
            strMsg = _T("Cannot open index file!");
            break;
      case EC_WRITE_FILE:
              strMsg = _T("Failed to write index file!");
            break;
      case EC_NOINDEX:
            strMsg = _T("This format requires that at least one valid document
level index field");
            break;
    default:
        break;
    }
    return TRUE;
}
```

5.7 *Configuration and support Functions*

5.7.1 Function LoadConfig()

	Data Type	Comment
Input	LPCTSTR	Config File
Input	DFAPI_DATA_TYPE&	Data Block

```
return      void
```

This function is called to load the SOD specific configuration data from file into the provided data block. This function is called at various points in the Capture Pro process.

```
BOOL CMyPlugin::CFGACI_LoadConfig(LPCTSTR lpszFilePath,
                                  DFAPI_DATA_TYPE& config)
{
      BOOL bRet = TRUE;

      // Load the BOF configuration data
      if (g_bRegularPersistData)    // g_bRegularPersistData is a global
                                    variable defaulted to TRUE
      {  //From binary data file
            bRet = __super::CFGACI_LoadConfig(lpszFilePath, config);
      }
      else
      {  //Load data from INI format file
            CMyCFGACIData data;
            data.LoadConfig(lpszFilePath);
            if(bRet )
                  bRet = data.ToDataBlock(config);
      }

      return bRet;
}
```

5.7.2 Function SaveConfig()

	Data Type	Comment
Input	LPCTSTR	Config File
Input	DFAPI_DATA_TYPE&	Data Block
return	void	

This function is called to save the SOD specific configuration data to file from the provided data block. This function is typically called in response to an OK from the SOD setup screen

```
BOOL CMyPlugin::CFGACI_SaveConfig(LPCTSTR lpszFilePath,
                                  DFAPI_DATA_TYPE& config)
{
      BOOL bRet = TRUE;

      // Save the BOF configuration data
      if (g_bRegularPersistData)
      {  //To binary format data file
            bRet = __super::CFGACI_SaveConfig(lpszFilePath, config);
      }
      else
      {  //To INI format file
            CMyCFGACIData data;
            bRet = data.FromDataBlock(config);
            if (bRet)
                  data.SaveConfig(lpszFilePath);
      }
```

```
	return bRet;
}
```

5.7.3 Function GetPluginInfoEx

	Data Type	**Comment**
Input	LPCTSTR	Language
Input	CString&	Plugin Name
Input	CString&	Plugin Description
Input	CString&	Plugin Copyright information
return	void	

This function is called by Capture Pro to get the name of the System Output Destination to display in the SOD list under Job Setup. It is also used to populate the Help menu ->About Kodak Capture Pro.->System Info… details.

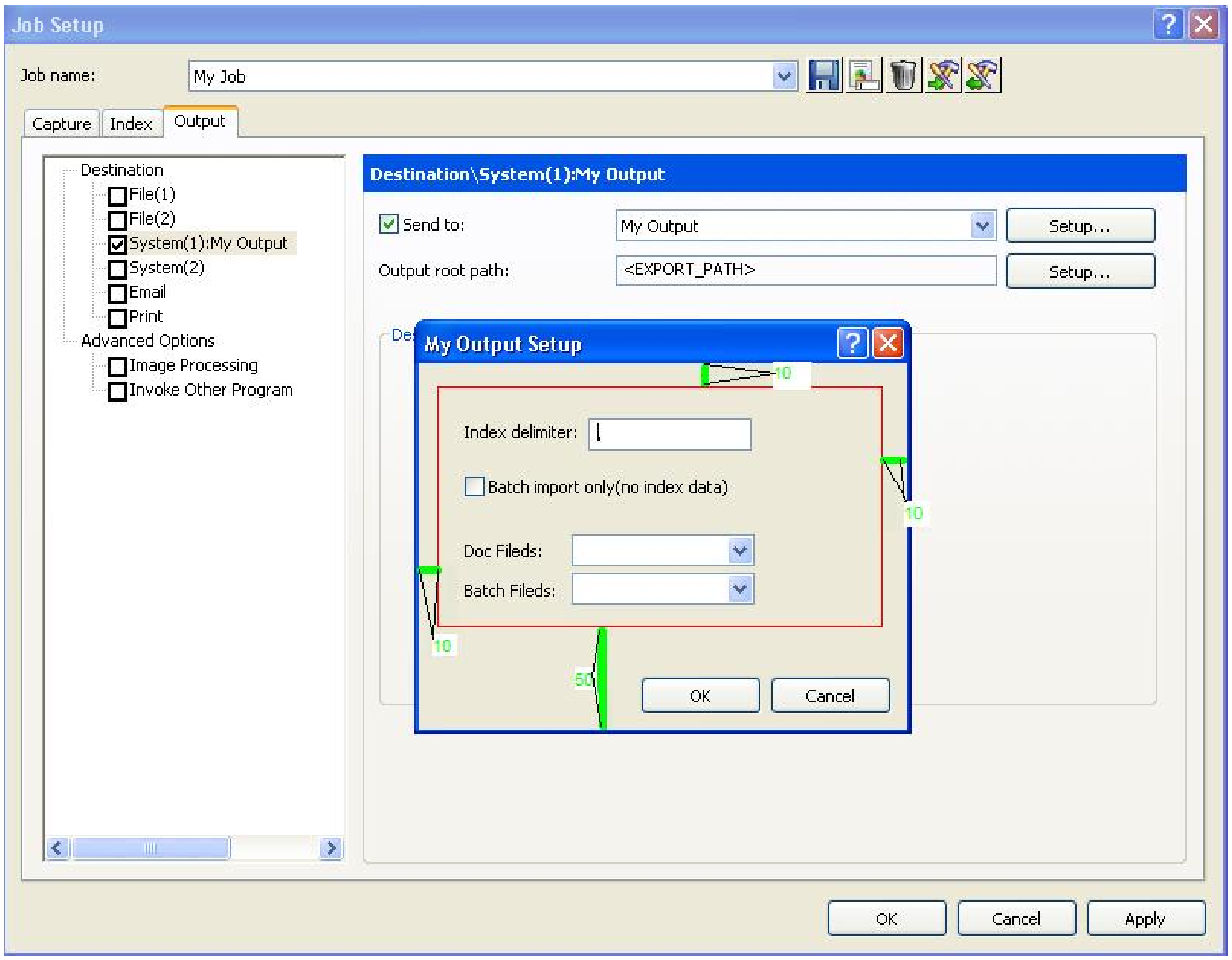
```
void CMyPlugin::GetPluginInfoEx(LPCTSTR lpszLanguage, CString& strName,
CString& strDescription, CString& strCopyright)
{
	// You should populate strName, strDescription and strCopyright with the
	appropriate data
	strName = _T("My Output");	// This is the name displayed in the Job
					Setup Screen
	strDescription = _T("This is an Example Plugin for Kodak Capture Pro.");
	strCopyright = _T("Copyright (C) 1998-2008 Eastman Kodak Company.  All
rights reserved.");
#ifdef	_DEBUG
	strName += _T(" (Debug)");
#endif	//_DEBUG
}
```

5.7.4 Function GetUISize()

As a developer you will need to provide a setup dialog to allow the user to configure System Output Destination specific settings.

Capture Pro handles the display of the SOD setup dialog. CP displays the setup dialog, you as a SOD programmer need to provide the template. To enable CP to size this correctly you need to create the template and also supply the width & height of the template to CP.

During Job setup, Capture pro needs to know how much space is needed on the System Output Destination setup dialog to display the SOD's setup information.
To find this out Capture Pro will call the **GetUISize()** function. The SOD must return the size of the setup screen area required in device units, typically pixels.
Capture Pro will create a dialog and embed the SOD's screen area within it as shown below.


Job Setup

Job name: My Job

Capture | Index | Output

- Destination
  - ☐ File(1)
  - ☐ File(2)
  - ☑ System(1):My Output
  - ☐ System(2)
  - ☐ Email
  - ☐ Print
- Advanced Options
  - ☐ Image Processing
  - ☐ Invoke Other Program

Destination\System(1):My Output

☑ Send to: My Output [Setup...]

Output root path: <EXPORT_PATH> [Setup...]

My Output Setup

Index delimiter: |

☐ Batch import only(no index data)

Doc Fileds:

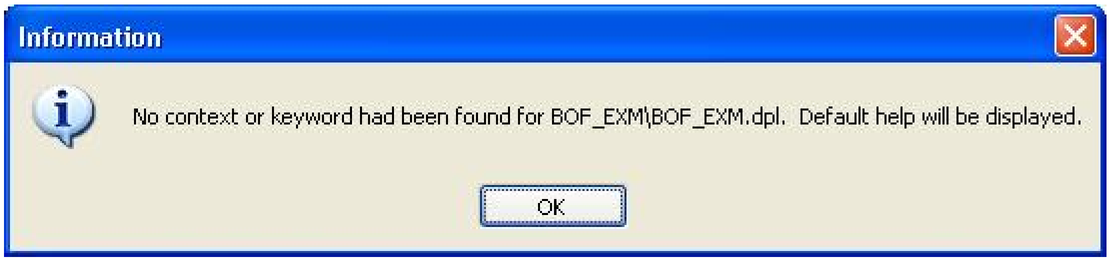
Batch Fileds:

The red box outlines the template screen area provided by the SOD (the red box will not appear on the dialog, it is for illustration only). This screen area has been placed in a section of the Setup dialog which was set to the size returned by the GetUISize() SOD call. Capture Pro will create the dialog complete with OK and Cancel buttons, the Dialog title is set to the SOD Name as defined in the GetPluginInfoEx function call. Capture Pro adds the margin space as shown in green on the screenshot above, 10 pixels top, left, right and 50 pixels on the bottom. If your SOD returns an area too small to display the template screen area you define, scroll bars will be added automatically within the red box.

5.8 Help file information

It is the responsibility of the SOD developer to provide a compiled Help file for their custom SOD if desired. To do this you must supply a compiled help file for your SOD.

If you do not include the appropriate help files with your SOD you will get the following message when you click on the ‘Help’ icon on the Setup dialog.


Information

No context or keyword had been found for BOF_EXM\BOF_EXM.dpl. Default help will be displayed.

When you OK this message, the Capture Pro help will be displayed but no SOD specific information will be available to the user.

To include help information with your SOD you must include the Windows compatible ‘help’ file, complied in Microsoft Help format (chm) and a *help.info* configuration file.

The *Help.info* configuration file is a plain ASCII text format file that tells the SOD Setup which *.chm* file to use for each language.

This is an example *help.info* file showing multiple languages:

```
[General]
ReportError = 1

[Language]
default = English_help_file.chm
en-us = English_help_file.chm
de-de = help_file_de.chm
fr-fr = help_file_fr.chm
it-it = help_file_it.chm
ja-ja = help_file_ja.chm
ko-ko = help_file_ko.chm
nl-nl = help_file_nl.chm
sv-se = help_file_sv_se.chm
zh-cn = help_file_zh_cn.chm
zh-tw = help_file_zh_tw.chm
```

If you are only going to include 1 help file for a single language then you can omit all the language specific entries and use only the default entry as follows

```
[General]
ReportError = 1

[Language]
default = General_help_file.chm
```

All help files and the *help.info* file should reside in the same folder as the SOD plugin file.

6 Installing your custom System Output Destination

Capture Pro Batch Output Formats are installed in their own folder in the following location on an English language system. On other language based systems, substitute the local folders in the path below.

```
C:\Program Files\Kodak\Capture Pro\Plugins\KCBOFAPI
```

To install your System Output Destination you will need to create a folder for it within the KCBOFAPI folder. Each SOD, together with any supporting files are installed in their own folder, the folder name must adhere to the format described earlier.
E.g for the example code supplied, create a folder under the KCBOFAPI called BOF_EXM and put the BOF_EXM.DPL file in the newly created BOF_EXM folder.

If you are supplying Windows Help with your System Output Destination, you will need to put the *.CHM and the *Help.info* file in this folder. See section 5.8 Help file information on page 24 for more information.

7 Job settings and batch metrics variables

The SOD API provides several functions to provide Kodak Capture Pro Job settings and batch metrics to the SOD.

7.1 Text Variables

Variable name **(Data Type)**

`TV_OUTPUTFOLDERPATH` `(String)`

Set to the output root path as defined in Job setup. Can be read to determine where the output should go.

`TV_JOBNAME` `(String)`

Set to the Capture Pro Job name for the batch to be output.

`TV_USERNAME` `(String)`

Set to the currently logged in user's name on the workstation sending the batch for output.

`TV_WORKSTATIONNAME` `(String)`

Set to the workstation name of the workstation sending the batch for output.

7.2 Data Variables

Variable name **(Data Type)**

`DV_WORKSTATIONID` `(DWORD)`

Set to the current workstation ID of the workstation sending the batch for output.

`DV_CREATETIMESTAMP` `(DATE)`

Set to the batch creation time.

`DV_PROCESSTIMESTAMP` `(DATE)`

Set to the current time at the point the batch output begins

`DV_FIRSTDOCUMENTID` `(DWORD)`

Set to the document ID of the first document within the batch. Can be used where the document ID increments across batches.

`DV_LASTDOCUMENTID` `(DWORD)`

Set to the document ID of the last document within the batch.

`DV_FRONTIMAGECOUNT` `(Long)`

Set to the total front side image count from the batch statistics.

`DV_BLANKFRONTIMAGECOUNT` `(Long)`

Set to the total (automatically deleted) blank front side image count from the batch statistics.

`DV_RESCANNEDFRONTIMAGECOUNT` `(Long)`

Set to the total rescanned front side image count from the batch statistics..

`DV_DELETEDFRONTIMAGECOUNT` `(Long)`

Set to the total (manually) deleted front side image count from the batch statistics.

`DV_REARIMAGECOUNT` `(Long)`

Set to the total rear (back) side image count from the batch statistics.

`DV_BLANKREARIMAGECOUNT` `(Long)`

Set to the total (automatically deleted) blank rear (back) side image count from the batch statistics.

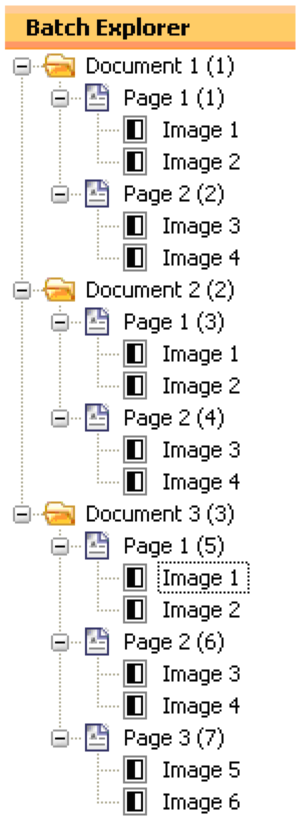
`DV_RESCANNEDREARIMAGECOUNT` `(Long)`

Set to the total rescanned rear (back) side image count from the batch statistics..

`DV_DELETEDREARIMAGECOUNT                    (Long)`
Set to the total (manually) deleted rear (back) side image count from the batch statistics.

8 Output process calling sequence example

For this example we will show the function calling process when Output is initiated. We will use the batch shown on the right. The Capture Pro Job for this Batch has 1 Batch level index field called “Patient” and 2 Document level index fields called “Doctor” and “Nurse”.
The Batch contains 3 Documents.
Document 1 has 2 pages, each Page has 2 Images.
Document 2 has 2 pages, each Page has 2 Images.
Document 3 has 3 pages, each Page has 2 Images.



```
Batch Explorer
Document 1 (1)
    Page 1 (1)
        Image 1
        Image 2
    Page 2 (2)
        Image 3
        Image 4
Document 2 (2)
    Page 1 (3)
        Image 1
        Image 2
    Page 2 (4)
        Image 3
        Image 4
Document 3 (3)
    Page 1 (5)
        Image 1
        Image 2
    Page 2 (6)
        Image 3
        Image 4
    Page 3 (7)
        Image 5
        Image 6
```

The Call Sequence is as below, comments are marked in //blue text:

//Start

```
LoadConfig        (Path to temporary copy of the configuration file)
```

//From this point forward, the Job settings and Batch metrics variables are available for query
//Now clear all index keys and values before starting.

```
ClearIndexKey   Level = 4
ClearIndexKey   Level = 3
ClearIndexKey   Level = 2
ClearIndexKey   Level = 1
ClearIndexValue Level = 4
ClearIndexValue Level = 3
ClearIndexValue Level = 2
ClearIndexValue Level = 1
```

//Next add the Batch level index key(s) and assign index data

```
AddIndexKey     Level = 4, Key = Patient
ClearIndexValue Level = 4
SetIndexValue   Level = 4, Key = Patient, Value = Chris
```

//Now add the Document level index key(s)

```
AddIndexKey     Level = 3, Key = Doctor
AddIndexKey     Level = 3, Key = Nurse
```

//Call BeginProcess() to signal that level 4 processing can be started

```
BeginProcess    Level = 4, Name = <Batch Name>, ID = batch
```

//Populate the Document level index data and call BeginProcess() to signal that level 3 processing can be started

```
ClearIndexValue Level = 3
SetIndexValue   Level = 3, Key = Doctor, Value = Richard
SetIndexValue   Level = 3, Key = Nurse, Value = Robert
BeginProcess    Level = 3, Name = 00000001, ID = document
```

//Clear the Page level index key and call BeginProcess() to signal that level 2 processing can be started

```
ClearIndexValue Level = 2
BeginProcess    Level = 2, Name = 00000001, ID = page
```

//Call BeginProcess() to signal that level 1 processing can be started

```
BeginProcess    Level = 1, Name = 00000001, ID = image
```

//Call AddImageFile() containing the path to the temporary copy of the source image file and the image related information

```
AddImageFile    ImagePath = (Path to the 1st TIFF file in the batch)
ID             = 7
ScannedSide    = 0
CurrentSide    = 1
HWPartNo       = 80
SWPartNo       = 0
ImageType      = 1
ImageSeq       = 1
PageSeq        = 1
DocumentSeq    = 1
IAInfo         = 0
IALevel        = 0
IALevel1       = (null)
IALevel2       = (null)
IALevel3       = (null)
IAEntire       = (null)
PrintString    = (null)
Flags          = 0
```

//Call EndProcess() to signal the end of the current level 1 operation

```
EndProcess      Level = 1
```

//Call BeginProcess() on the next level 1, AddImageFile() contains the details for the next image file

```
BeginProcess    Level = 1, Name = 00000002, ID = image
AddImageFile    ImagePath = (Path to the 2nd TIFF file in the batch)
     Image_Info_details…..
```

//Call EndProcess() twice to signal the end of the current level 1 and current level 2 operation

```
EndProcess      Level = 1
EndProcess      Level = 2
```

//Prepare the next Page level and call BeginProcess(), AddImageFile(), and EndProcess() on Page2 which contains image 3 and image 4

```
ClearIndexValue Level = 2
BeginProcess    Level = 2, Name = 00000002, ID = page
BeginProcess    Level = 1, Name = 00000003, ID = image
AddImageFile    ImagePath = (Path to the 3rd TIFF file in the batch)
     Image_Info_details…..
EndProcess      Level = 1
BeginProcess    Level = 1, Name = 00000004, ID = image
AddImageFile    ImagePath = (Path to the 4th TIFF file in the batch)
     Image_Info_details…..

EndProcess      Level = 1
EndProcess      Level = 2  //End of page
```

```
//As this is the end of Document 1, EndProcess(level 3) is called, then we prepare the next Document level
index data and start adding the Page/Image level details
EndProcess      Level = 3
ClearIndexValue Level = 3
SetIndexValue   Level = 3, Key = Doctor, Value = Ian
SetIndexValue   Level = 3, Key = Nurse, Value = Kathy
BeginProcess    Level = 3, Name = 00000002, ID = document
ClearIndexValue Level = 2
BeginProcess    Level = 2, Name = 00000003, ID = page
BeginProcess    Level = 1, Name = 00000005, ID = image
AddImageFile    ImagePath = (Path to the 5th TIFF file in the batch)
     Image_Info_details…..
EndProcess      Level = 1

//LOOP through the Images / Pages as for Document 1

//After processing all images / Pages for document 2, EndProcess(level3) to close Document 2 and prepare
the index etc. for Document 3
EndProcess      Level = 1
EndProcess      Level = 2
EndProcess      Level = 3
ClearIndexValue Level = 3
SetIndexValue   Level = 3, Key = Doctor, Value = Max
SetIndexValue   Level = 3, Key = Nurse, Value = Kelvin
BeginProcess    Level = 3, Name = 00000003, ID = document
ClearIndexValue Level = 2
BeginProcess    Level = 2, Name = 00000005, ID = page
BeginProcess    Level = 1, Name = 00000009, ID = image
AddImageFile    ImagePath = (Path to the 9th TIFF file in the batch)
     Image_Info_details…..
EndProcess      Level = 1

//LOOP through the Images / Pages as for the previous Documents

//After the last image has been processed, EndProcess() is called at each level to close the Image, Page,
Document and Batch
EndProcess      Level = 1
EndProcess      Level = 2
EndProcess      Level = 3
EndProcess      Level = 4
//End
```

9 Document revision history

Version	Date	Author	Changes:
1.0	Mar 2008	IJP/RJM	Initial version
2.0	Jul 2009	EPE/JDM	Updates for API Additions