March Developer Newsletter

Black Ice Software

Volume 14, Issue 3

March 2009

Inside this issue:

Phone: (603) 882-7711

Fax: (603) 882-1344

sales@blackice.com

www.blackice.com

Intermix 32 and 64-bit Printer Applications

This article shows how you can use an application using the BlackIceDEVMODE.dll to change printing preferences of a printer driver. The BlackIceDEVMODE.dll has two separate versions: one for 32-bit and another for 64-bit. If the program is a 64-bit application it can only be used on 64-bit operating systems. This program can not be started on 32-bit operating systems.

If the program is a 32-bit application it can be used on both 32 and 64 bit operating systems, because the 64-bit operating systems can handle 32-bit applications. Because the 32-bit version of the BlackIceDEVMODE.dll is able to also work with 64-bit Black Ice printer drivers, the 32-bit application can set the printer settings programmatically without any code changes. Of course the 64-bit program can use only the 64-bit version of BlackIceDEVMODE.dll and the 32-bit program can use only the 32-bit version of BlackIceDEVMODE.dll.

How to Restrict Printer Driver File Formats

You are able to restrict the available file formats on the File Formats tab of the Printing Preferences of the printer driver. You can select which file formats (only one or more) that can be used for printing. The available file formats can be specified through the printer driver INI file. The following example enables the JPEG file format and disables the bitmap file format.

```
[UI FileFormats]
JPEG File=1
Microsoft DIB Format=0
```

Note: The same file format name has to be used in the INI file as it is seen in the file format combo box.

Update ActiveX Controls by a CAB File

There have been some questions recently about updating ActiveX controls in the client machines with Black Ice CAB files. This article will show a technique to keep up to date the ActiveX controls in the client machines.
One may read about the HTML object tag's CODEBASE attribute on the following web site (you may need to copy/paste the link):
http://msdn.microsoft.com/en-us/library/ms533576.aspx
If one wants to force the client to reload the CAB file, one can set the version number of the current ActiveX control in the HTML code. If the client's version number is lower, the browser will download the CAB. It will check the version number in the .inf file. If the specified version is larger, it will replace the older file.

Here is a sample code snippet to the version number:

```
<object id="BIDISP" width=450 height=420 classid="CLSID:19B50C95-6BB5-
4DFD-B20C-5B9A61FA1C0D" CODEBASE="http://192.168.0.127/4/
Image.cab#Version=10,9,8,0"></object>
```

Compressions in the Low Level Interface for PDF SDK

As in previous articles it has been mentioned that there are properties of saving a PDF document to a file that one may set. Some of these properties are about PDF document compression. To reduce file size, PDF supports a number of industry-standard compression filters. Most of these filters are supported by the Black Ice PDF SDK. The following compression filters are supported for the various data types:

- Non-image streams: FlateDecode (STREAM_FILTER_FLATE)
- Color images: FlateDecode (CIMAGE_FILTER_FLATE), RunLength (CIMAGE_FILTER_RUNLENGTH), Jpeg (CIMAGE_FILTER_JPEG)
- Monochrome images: FlateDecode (MIMAGE_FILTER_FLATE), RunLength (MIMAGE_FILTER_RUNLENGTH), CCITT Fax (MIMAGE_FILTER_CCITT_FAX)

If one sets the compression for a PDF stream type but the SDK cannot compress it with the specified algorithm, the PDF SDK tries to compress the PDF stream with a different algorithm using the following rule:

- CCITT Fax > RunLength > FlateDecode > No compression
- Jpeg > FlateDecode > RunLength > No compression
- FlateDecode > No compression

The following code snippet shows how to set the compressions before saving the PDF document:

```
CPDF pdf;

// set metadata

CPDFPage page = pdf.AddPage(700.0, 900.0);

// load the content into page

// set encryption and passwords

// set compression
STREAM_FILTER_TYPE streamFilterType      = STREAM_FILTER_FLATE;
MONO_IMAGE_FILTER_TYPE monoFileterType   = MIMAGE_FILTER_CCITT_FAX;
COLOR_IMAGE_FILTER_TYPE colorFileterType = CIMAGE_FILTER_JPEG;

pdf.SetStreamFilter(streamFilterType);
pdf.SetMonoImageFilter(monoFileterType);
pdf.SetColorImageFilter(colorFileterType);
pdf.SetColorImageQuality(95);

// set font embedding

pdf.SavePDFToFile("c:\\hello.pdf");
```

This code snippet sets the default compression for non-image streams to the Flate algorithm, CCITT Fax algorithm for monochrome image streams and Jpeg for the color image streams. If Jpeg compression is used one can set the quality of the compression by using the SetColorImageQuality method. The parameter passed can be between 1 and 100, 100 being the best quality. During the PDF file generation the PDF SDK will try to compress the streams based on the settings and the rules mentioned previously.

Using the High Level Interface for PDF SDK to Modify and Save the Contents of the CPDFDoc Objects

This article continues from where the "Using the PDF Interface to load a PDF document into the object oriented structure" article has ended. It will explain how a modified page and document can be saved. It will work on a loaded/created document with one modified page in it.

When one wishes to modify the contents of a page, it is very important to keep track of if there have been any modifications. It is therefore the user's responsibility to set a CPDFPageObj object's m_bModified flag to true whenever one changes an attribute of one of the members of the page. This flag is however managed by the page object's member functions.
When the m_bModified flag of a page object is set to true, the page needs to be saved:

```
// Save the unloaded page if it was modified
    if (page->IsModified())
    {
        if (page->WriteToPDF()
        {
           // Error handling
        }
    }
```

This will write the page's contents into the PDF SDK's internal structure, but it will NOT save anything to disk.
Assuming one wouldn't need to access the page's objects again, it is also advisable to unload the page data from memory. (The changed data can still be loaded again if needed.)

```
    // Unload the old page's data
    page->Clear();
```

It is also possible to change a document's metadata in the CPDFDoc object and should be reviewed before saving the document to disk. The document object contains some important member objects that modify the way a PDF file is encoded. Compression settings can be accessed in the CPDFDoc object's m_compression member variable. In that member object, one may set the used compression algorithm for saved PDF streams, color images and monochrome images.

Passwording/Encryption settings can be accessed in the CPDFDoc object's m_encryption member variable. In that member object, one may set the user (required to open the document) and owner (required to edit the document) passwords, the user access permissions and the encryption type and key length.

Document information metadata can be accessed in the CPDFDoc object's m_info member variable. In that member object, one may set the document's title, author, subject, keywords, creator application and trapping information.

After having made the above changes, one is ready to save the document to file. This can be done the following way:

```
    // Save to file
    if (!pPDFDocument->SavePDFFile(lpszPathName))
    {
        // Error handling
    }
```

When closing the PDF document, one can just call:

```
    pPDFInterface->Destroy();
```