January 2006

DEVELOPER NEWSLETTER

Volume 11, Issue 1

Inside This Issue:

Phone: (603) 673-1019
Fax: (603) 672-4112
sales@blackice.com
www.blackice.com

New Language Support Added to the RTK

New language support has been added to existing sample applications in the Resource Tool Kit. Restrict Printing, Use Document Name and Plug-in sample applications now have language support for VB, VB.NET, C#, and Delphi.

New Tiff Format Added for Printer Drivers to Support Cisco Routers!

Black Ice Software released the monochrome printer driver with a new file format for Cisco routers. The Cisco routers require a tiff file in a special format. The Cisco tiff file begins with an 8-byte tiff image file header identifier which contains the offset for the first Image File Directory (IFD), followed by the first Image File Directory (IFD), and the Image data. Followed by the next IFD, the next Image data is a chain of images in the TIFF file. See diagram on next page.

A conventional tiff file begins with an 8-byte image file header which contains the offset for the first Image File Directory (IFD), followed by the Image data, which is followed by the first Image File Directory (IFD). The actual compressed data is before the Image File Directory (Tags). In the Image File Directory there is offset

(Continued on page 2)

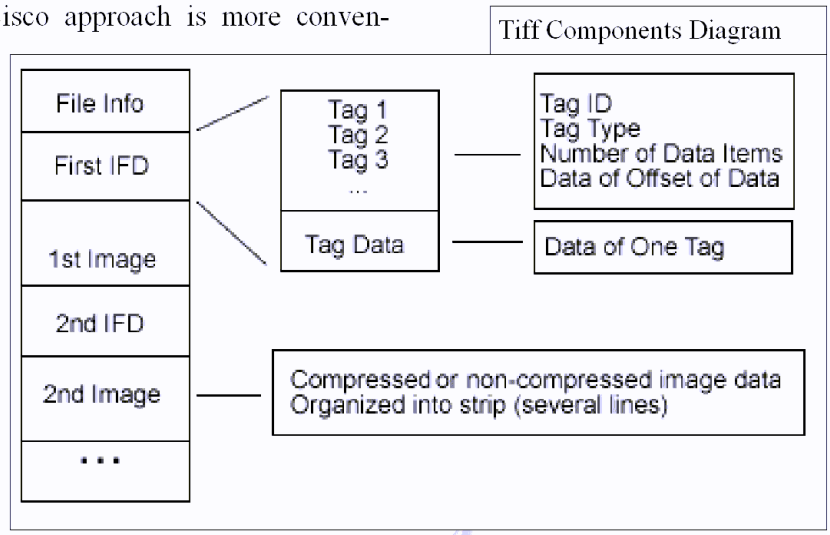
Black Ice Printer Driver - Restrict Print Job

The Black Ice printer driver can se set up to refuse any print jobs other than those coming from the developer's printing application. This is not a hardcore security feature. It is more likely to be used to make sure that users will not accidentally print from an unsupported application. The restriction is made using a password stored in the INI file and the printer driver will accept print jobs only from the application that uses the correct password. The password can be any string. The string should be passed to the printer driver by the printing application in the document name. If the password string is present in the document name, the files are generated, if the document name does not contain the password, the print job will be aborted.

(Tiff Cisco Support Continued from page 1)

for the image which is pointing to the beginning of the file. It looks a little backward, but there is a reason to it. TIFF File format originated in the mid 1980s when memory was very limited. The image was decompressed and compressed from the video buffer. An image was segmented and cut up to strips. Each strip contained 32k or 64k of data and each strip's offset and length was written into the IFD. As the image was compressed one did not know in advance how big the compressed image data would be. Therefore one could not fill out the IFD tags until the entire image strips were written to the file. Once the image is written to the file then one can write the IFD tags. To make it more complicated each strip in the image header is an entry, so one could not compute how big the IFD would be in advance. Therefore it makes sense to write the image data first, and then write the IFD tags.

The TIFF specification allows both approaches. The conventional tiff file structure was dictated by necessity at the time. The Cisco approach is more convenient in an IP application to process the data stream. One can start to decompress the image before the entire image is received since the header information is in front of the image.



Add a Custom Plug-in to the Black Ice Printer Drivers!

Version 9.75 of the Black Ice printer drivers introduces new Plug-In support which is accessed through a dynamic linked library or DLL, and is loaded by the printer driver during printing. The printer driver calls pre-defined functions from the Plug-in at each phase of printing, and will not continue printing until the function returns. Developers will have full access to the DLL they can also insert their own custom code to each DLL function to perform a variety of tasks as required by their application.

How It Works

The Black Ice printer driver will load the Plug-In DLL with each print job and will pass to each DLL function a pointer to the devmode of the current print job. When the print job starts, at the Start Document phase, the Plug-In DLL can pass custom data to the printer driver. The custom data will be stored by the driver for the duration of the print job. The size of the custom data is not limited by the printer driver, however using a large data block is not recommended since the printing can be negatively impacted. Specifying a data block of no larger than 10-20 KB should be sufficient in most cases and will not have an impact on performance. The printer driver will return a pointer to the custom data in each additional function call to the Plug-in DLL. The Plug-in DLL can then modify or use the custom data in any way desired.

The Plug-in DLL functions must return TRUE on success or FALSE on any failure. If any of the custom functions return FALSE, the Black Ice Printer driver will abort the current print job.

Imaging Tips and Tricks - Programmatically Zooming

The Black Ice Imaging Toolkits include BiDisp controls which allow developers to display images for their users. You can zoom into an image with BiDisp control either through user intervention or programmatically. This article focuses on how to programmatically zoom using the **BZoomStep** or **ZoomArea** functions or methods.

[VB]

```
Private Sub ZoomImage()
        Dim iLeft As Long
        Dim iTop As Long
        Dim iRight As Long
        Dim iBottom As Long

        iLeft = 10
        iTop = 10
        iRight = 30
        iBottom = 30

        ` parameters: Zooming rectnagle coordinates
        BiDisp.ZoomArea (iLeft, iTop, iRight, iBottom)
End Sub
```

[C++]

```
#include "BiDisp.h"

void CMyView::ZoomStep()
{
        // 1. parameter (HWND): Window handle to display image
// 2. parameter (LPPOINT): Coordinates of the mouse pointer
// 3. parameter (ZOOMFLAG): Determines the operation of zoom.
// 4. parameter (LPZOOMSTRUCT): Zoom structure
// 5. parameter (DISPLAYSTRUCT): Display information structure
        BZoomStep(m_hWnd, &rScale, zoomflg, &m_zm, &sDisplay);
        DispZoom();
}

void CMyView::DispZoom()
{
        CopyRect(&rScale, &m_zm.rScaleOut);
        SetRect(&sDisplay.rOrigo, 0, 0, 0, 0);
        sDisplay.rOrigo.left  = m_zm.pOrigoOut.x;
        sDisplay.rOrigo.top   = m_zm.pOrigoOut.y;

        if(hDib)
        {
          DisplayDIBStart(m_hWnd, hDib, &sDisplay.rOrigo,
                &rScale, wDisplayMode, &sDisplay);
        }
}
```

[C#]

```
private void ZoomImage()
{
        int iLeft, iTop, iBottom, iRight;

        iLeft = iTop = 10;
        iRight = iBottom = 30;

        // parameters: Zooming rectnagle coordinates
        BiDisp.ZoomArea(iLeft, iTop, iRight, iBottom);
}
```