The DaVince DLL Library

Creating PDF files programmatically

Version 2.2

www.davince.com

July 22, 2003

Table of Contents

Part 1: Using the Library

Introduction

The DaVince DLL Library is a DLL library that creates Portable Document Format (PDF) files. Some of the features of this library include:

- Convert TIFF and JPEG files to PDF
- Convert text files to PDF with word wrap, header, footer, paragraph, column and green bar options
- Create PDF pages from scratch by writing directly to the page stream
- Write PDF data to either a file or memory buffer
- Compression support
- Bookmark support
- OpenAction support
- Support for all built-in PDF fonts and encodings
- Multi-thread capable
- Support for C++ exceptions
- Support for the creation of large PDF files (+1,000 pages)

This library contains functions to easily convert TIFF, JPEG and text files to PDF, as well as allows one to create PDF pages from scratch. The DLL can write the PDF data either directly to a file or it can transfer it to the calling program using a memory buffer. The memory buffer approach is ideal in situations where the PDF data is not written to a file directly, like in a CGI or mailer program.

The library is based on the DaVince Tools suite of programs (web site: http://www.davince.com). Most of the profile parameters used in DaVince Tools can be used with the DLL library. All profile parameters are set using a single DLL function.

Using the Library With Your Application

Although the DaVince Tools DLL library allows one to easily create a PDF file programmatically, it does not mean it will always create a problem free PDF file. In particular, page streams created by the calling program with the writePage() function are not checked for errors. The developer will need to ensure these page streams conform to the Adobe PDF Specification. Viewing an uncompressed page stream using a hexadecimal editor/viewer may be beneficial in diagnosing problems. Page compression can be enabled or disabled using the setParameter() function.

For C/C++ applications, the include file "dvclib.h" will be used to define the DLL functions in your C/C++ program. You will also need to use a library for linking your program with the DLL. The library varies depending on the compiler used. For Microsoft Visual C++, use the library dvclibms.lib. The DaVince DLL library has been tested with MicroSoft Visual C++ version 6. For Borland C++ Builder, use the dvclibb.lib library. Borland C++ Builder version 5 has been tested with the library. These and later versions of the compilers should work with this library.

For other languages, the "dvclib.def" file is included to help integrate your application with the library.

Creating a PDF File With Your Application

When using the DaVince DLL Library, the first call you will make is to the createHandle() function. This handle defines a PDF instance in the DLL. Because the library is thread safe, multiple PDF handles may be opened (one for each PDF instance) simultaneously. This handle is used in all subsequent calls to the library until it is released by using the releaseHandle() function. The releaseHandle() function must be used in order to free up memory and temporary disk space used by the handle.

Creating Page Content

Probably the most important part of using the DaVince DLL Library is creating the page content. The library provides two mechanisms for creating page content: creating a page via file conversion, or creating page content manually.

The library supports three file formats for converting to PDF: TIFF, JPEG, and text files. These file types can be converted to page content using the convertFile() or the convertBuffer() functions. With TIFF and text files, these functions may actually create several pages depending on the files being converted. The convertBuffer() function works similarly to the convertFile() function, except that instead of converting a file stored on disk, it converts a file stored in a memory buffer.

Creating page content manually requires knowledge of the PDF specification for page descriptions. This information can be found in **the Portable Document Reference Manual**, published by Adobe. The addPage() and writePage() functions are used to create page content manually. Page handles are used to access individual pages in the PDF file. These page handles become invalid when the PDF data is released using the releaseHandle function.

The library supports text and graphics operators in the page content. When using text in the page content, the useFont() function is required. The getTextLength() function is needed if the length of a string of text is needed (for example, if right or center justifying text is required).

Both methods of creating page content can be combined within the same PDF file. For example, a cover page contaning text may be created manually using the addPage() and writePage() functions, then, an image file is added using the convertFile() method.

Writing the PDF Output

Two methods exist for writing PDF output: writing PDF output to a file, or writing PDF output to a memory buffer. The latter method is useful if the PDF does not immediately need to be written to disk. Composing an e-mail message with a PDF file attachment or writing out the PDF to standard output for a web page are examples where the PDF file may not need to be written to disk. Use the writePdf() function when writing the PDF data directly to a file. When writing to a memory buffer, use the preparePdfBuffer() and getPdfBuffer() functions.

The library supports PDF compression, which allows for compression of the PDF page stream. Since PDF compression is done during the writing of each PDF page, the setParameter function needs to be called prior to the writing of the first PDF page in the following manner:

```
setParameter(pdfhandle, "Compression","true")
```

Handling Errors

All DLL functions, with the exception of createHandle(), return a value of zero "0" upon successful completion, otherwise, an error code is returned. In contrast, createHandle() returns a nonzero number for a DLL handle upon successful completion, otherwise, it returns zero "0". Error codes describing the type of error are described in the file "dllerr.h" . Further information can be found about the error that occurred by calling getInfoText(). This string can be used as part of an error message displayed to the user.

C++ programmers may be disappointed that C++ exceptions cannot pass from a DLL to a calling program. However, a workaround to this limitation has been developed for use with the DaVince DLL library. The macro "DLLERRORMESSAGE", defined in "dvcexc.h", can be used to emulate a DLL thrown exception. This include file is automatically included when the include file "dvclib.h" is specified. This macro copies the error code and error string into its own exception class and throws an exception on behalf of the DLL. For example, the following code shows a sample of using this macro:

```
long pdfhandle = 0;
try
{
  pdfhandle = createHandle();
  DLLERRORMESSAGE(pdfhandle, setInfo(pdfhandle,"Title", "Hello
World"));
…
}
catch (DLLException &e)
{
  cerr << "*** pdfhello DLL error: " <<  e.getErrorText() << endl;
  releaseHandle(pdfhandle);
}
```

The DLLERRORMESSAGE macro throws exceptions of type DLLException. A string of the error message causing the exception can be retrieved using the getErrorText() function. The DLLERRORMESSAGE macro takes two arguments. The first argument is the PDF handle created by the createHandle() function, and the second argument is the actual function call to the DaVince DLL library. The PDF handle is needed by the macro in order to extract the text of the error message for use by getErrorText().

This macro does not work with the createHandle() function. An error condition exists with this function if it returns a value of zero for the PDF handle.

The Sample Programs

Three sample programs, available in both binary and source code form, are included in the DaVince Tools Plus distribution:

Program Name	**Description**
pdfclock.cpp	Show current time and date using an analog clock
pdfconv.cpp	Convert TIFF, JPEG and text files to PDF
pdfhello.cpp	Demonstration application displaying the words "Hello World" in a PDF file

Of the three programs, "pdfclock.cpp" and "pdfhello.cpp" create their page content manually using the addPage() and writePage() functions. "pdfconv.cpp" creates its page content automatically by calling the convertFile() function.

Deploying Your Application

In addition to the files associated with your program, you will also need to include the DaVince DLL library file, "dvclib.dll", and the license file, "davince.txt", in the deployment of your application. The location of the "dvclib.dll" file can vary depending on your application development environment. At a minimum, the DLL can reside in one of the following directories:

- The same directory as the calling executable

- The Windows directory (usually "c:\winnt" or "c:\windows")

Your application development environment may allow other directories.
The license file, "davince.txt", can reside in one of the following directories:

- The same directory as the calling executable
- The root directory of the calling program ("c:\", for example)
- The Windows directory (usually "c:\winnt" or c:\windows")

Writing CGI Programs That Create PDF Files

CGI (Common Gateway Interface) is a standard to allow applications to communicate with a web server. With the DaVince DLL Library, one can create a CGI program to create PDF files dynamically using a web browser interface. The sample program, pdfclock, can be run as either a command line application or a CGI application. Both binary and source code versions of pdfclock are included in the **DaVince Tools Plus** distribution.

More information can be found in the source code of pdfclock (written in C++), but the following pointers may help in creating problem free CGI applications:

- Use standard output (cout in C++) when writing the PDF file.
- Include the line "Content-Type: application/pdf" as the first line to standard output.
- Optionally include the line "Content-Length: " followed by the byte size of the PDF file in decimal units.
- A blank line must appear after the content header and before the PDF file.
- Use preparePdfBuffer and getPdfBuffer DLL functions to create the PDF file in memory.

- Standard output must be in binary mode instead of text mode in order to prevent corruption of the PDF data stream. See pdfclock source code for a way to do this in a C or C++ program.

Dissecting pdfhello.cpp

This section describes how to use the library by way of example. Techniques will be discussed that were used in writing the C++ sample program pdfhello.cpp, which is included with "DaVince Tools".

Source code to pdfhello.cpp:

```
/*

  PDFHELLO Program
  A sample application using the DaVince Tools DLL
  DaVince Web site: www.davince.com

  Create a single page PDF file with the words "Hello World"
  Syntax: pdfhello

*/

#include <string>
#include <iostream>
#include <fstream>
#include <strstream>
#include <ctype.h>

#include "dvcdll.h"

using namespace std;

int main(int argc, char* argv[])
{
    long pdfhandle = 0;
    string filename = "pdfhello.pdf";

    try
    {
        // initialize PDF
        pdfhandle = createHandle();
        DLLERRORMESSAGE(pdfhandle, setInfo(pdfhandle,"Title", "Hello
World"));
        DLLERRORMESSAGE(pdfhandle, setParameter(pdfhandle,
"Compression","true"));

        // create page
                long pagehandle;
        DLLERRORMESSAGE(pdfhandle, addPage(pdfhandle, &pagehandle,
8.5*72, 11.0*72));
```

```
        // create page stream
        // define a buffer size that will be big enough to store our
page stream
        #define BUFFSIZE (1024*10)
        char buff[BUFFSIZE];
        strstream spage(buff, BUFFSIZE);

        //select font to use
        char shortname[20];
        DLLERRORMESSAGE(pdfhandle, useFont(pdfhandle, pagehandle,
"Helvetica", shortname, 20));

        // determine text length of "Hello World" in order to center it
horizontally on the page
        float textlength;
        DLLERRORMESSAGE(pdfhandle, getTextLength(pdfhandle,
"Helvetica", "Hello World", 72, &textlength));
        float horizontalpos = (8.5*72 - textlength)/2;

        // write page description to stream
        spage << "q ";
        spage << "BT ";
        spage << "/" << shortname << " 72 Tf ";

        spage << horizontalpos << " 576 Td (Hello World) Tj ";
        spage << "ET ";
        spage << "Q" << endl;
        long pagesize = spage.tellp();
        DLLERRORMESSAGE(pdfhandle, writePage(pdfhandle, pagehandle,
buff, pagesize));
        cout << "writing to " << filename << endl;
        DLLERRORMESSAGE(pdfhandle,
writePdf(pdfhandle,filename.c_str()));
        releaseHandle(pdfhandle);
    }
    catch (DLLException &e)
    {
        cerr << "*** pdfhello DLL error: " <<  e.getErrorText() <<
endl;
        releaseHandle(pdfhandle);
        return 1;
    }
    catch (...)
    {
        cerr << "*** pdfhello application error" << endl;
        releaseHandle(pdfhandle);
        return 1;
    }
    return 0;
}
```

The first part of the file defines the headers that will be used with the program:

```
#include <string>
#include <iostream>
#include <fstream>
#include <strstream>
#include <ctype.h>

#include "dvcdll.h"

using namespace std;
```

Note that the include file "dvcdll.h" is needed in order to use the DaVince DLL Library. The "using namespace" command is a C++ directive that some C++ compilers need to simplify coding of standard namespace functions, like string, cout, and endl. Without this, these functions would require a "std::" prefix each time they are used.

```
int main(int argc, char* argv[])
{
    long pdfhandle = 0;
    string filename = "pdfhello.pdf";

    try
    {
```

The try command is used to catch C++ exceptions.

```
        // initialize PDF
        pdfhandle = createHandle();
        DLLERRORMESSAGE(pdfhandle, setInfo(pdfhandle,"Title", "Hello
World"));
        DLLERRORMESSAGE(pdfhandle, setParameter(pdfhandle,
"Compression","true"));
```

This section of code creates the PDF handle that will be used in all subsequent calls to the DaVince DLL library for this PDF file. The setInfo function is used to set the title used in the PDF file (viewable by typing CTRL-D in Adobe Acrobat). The setParameter function is used to enable compression on the PDF file.

The DLLERRORMESSAGE macro is used to throw C++ exceptions originating in the DLL. Since DLLs cannot generate exceptions to their calling programs directly, this macro is used instead.

```
        // create page
        long pagehandle;
        DLLERRORMESSAGE(pdfhandle, addPage(pdfhandle, &pagehandle,
8.5*72, 11.0*72));
```

This call to the addPage function defines a new page 8 1/2 inches by 11 inches (one inch = 72 points).

```
        // create page stream
        // define a buffer size that will be big enough to store our
page stream
        #define BUFFSIZE (1024*10)
        char buff[BUFFSIZE];
        strstream spage(buff, BUFFSIZE);
```

A string stream is defined here to store the page description.

```
//select font to use
        char shortname[20];
        DLLERRORMESSAGE(pdfhandle, useFont(pdfhandle, pagehandle,
"Helvetica", shortname, 20));

        // determine text length of "Hello World" in order to center it
horizontally on the page
        float textlength;
        DLLERRORMESSAGE(pdfhandle, getTextLength(pdfhandle,
"Helvetica", "Hello World", 72, &textlength));
        float horizontalpos = (8.5*72 - textlength)/2;
```

The useFont function is used here to tell the DLL we will be using the "Helvetica" font for this page. The function also returns the short name of the font so we can use it in the page description.

The "Hello World" text is centered horizontally on the page, so the getTextLength function is used to determine horizontalpos, which defines the distance from the left edge of the page to the first word of the letter, in points.

```
        // write page description to stream
        spage << "q ";
        spage << "BT ";
        spage << "/" << shortname << " 72 Tf ";

        spage << horizontalpos << " 576 Td (Hello World) Tj ";
        spage << "ET ";
        spage << "Q" << endl;
```

The page description is written to the string stream so that it can later be sent to the DLL. The font short name and the horizontal distance between the left edge of the page and the words "Hello World", determined previously, are written to the stream.

```
        long pagesize = spage.tellp();
        DLLERRORMESSAGE(pdfhandle, writePage(pdfhandle, pagehandle,
buff, pagesize));
        cout << "writing to " << filename << endl;
        DLLERRORMESSAGE(pdfhandle,
writePdf(pdfhandle,filename.c_str()));
        releaseHandle(pdfhandle);
    }
```

The stream is written to the DLL using the writePage function. The actual size of the stream was determined by the tellp() function. The PDF file is actually written to disk using the writePdf function.

```
    catch (DLLException &e)
    {
        cerr << "*** pdfhello DLL error: " <<  e.getErrorText() <<
endl;
        releaseHandle(pdfhandle);
        return 1;
    }
    catch (...)
    {
        cerr << "*** pdfhello application error" << endl;
        releaseHandle(pdfhandle);
        return 1;
    }
    return 0;
}
```

The DLLERRORMESSAGE exception throws an exception of type DLLException if a non-zero return value is returned by a DaVince DLL library function. These exceptions are caught here.

Internet Resources

The following web sites contain information useful to the DaVince DLL Library programmer:

www.davince.com - product web site
www.davince.com/support - product support web site
www.adobe.com - Adobe Corporation's web site

Part 2: DaVince DLL Function Summary

addPage

Summary:
Append a page

Syntax:
```
long addPage(long handleid, long *pageid, float width,
float height);
```

Parameters:
handleid (input): DaVince DLL handle
pageid (output): page handle of newly created page
width (input): Width of page, in points
height (input): Height of page, in points

Return Value:
0 if successful, otherwise non-zero error code

Description:
This function appends a new page to the PDF handle at the given width and height, in points. Subsequent page content is written to the page using the **writePage** function.

See Also:
writePage

convertBuffer

Summary:
Convert file stored in memory to PDF

Syntax:
```
long convertBuffer(long handleid, const char *converter,
const char *filename, char *buff, long length);
```

Parameters:
handleid (input) - DaVince DLL handle
converter (input) - converter to use, must be either "tiff", "jpeg" or "text"
filename (input) - filename to use when generating error messages
buff (input) - buffer area containing file image
length (input) - length of buffer area, in bytes

Return Value:
0 if successful, otherwise non-zero error code

Description:
Take a file in memory and convert it to PDF. The file in memory must be in the format the converter is expecting. This is the memory buffer version of **convertFile**.

See Also:
convertFile

convertFile

Summary:
Convert file to PDF

Syntax:
```
long convertFile(long handleid, const char *converter,
const char *filename, const char *aliasname);
```

Parameters:
handleid (input) - DaVince DLL handle
converter (input) - converter to use, must be either "tiff", "jpeg" or "text
filename (input) - name of file to convert
aliasname (input) - text to use as bookmark, if bookmarks are enabled

Return Value:
0 if successful, otherwise non-zero error code

Description:
This function converts the specified file to PDF. The file must be in the format the converter is expecting. See **convertFile** for the memory buffer version of this function.

See Also:
convertBuffer

convertTiffBuffer

Summary:
Convert TIFF file stored as an image in memory.

Syntax:
```
long convertTiffBuffer(long handleid, char *filename, char
*buff, long length)
```

Parameters:
handleid (input) - DaVince DLL handle
filename (input) - text to use as bookmark, if bookmarks are enabled
buff (input) - buffer area containing image of TIFF file
length (input) - length of buffer area, in bytes

Return Value:
0 if successful, otherwise non-zero error code

Description:
This function is obsolete. Use the **convertBuffer** function instead and "tiff" as the converter parameter.

See Also:
convertBuffer

convertTiffFile

Summary:
Convert TIFF file.

Syntax:
```
long convertTiffFile(long handleid, char *filename, char
*aliasname)
```

Parameters:
handleid (input) - DaVince DLL handle
filename (input) - TIFF file to convert
aliasname (input) - Name to use for bookmark, if bookmarks are enabled

Return Value:
0 if successful, otherwise non-zero error code

Description:
This function is obsolete. Use the **convertFile** function instead and "tiff" as the converter parameter.

See Also:
convertFile

createHandle

Summary:
Create a handle to a PDF object for use with subsequent function calls

Syntax:
```
long createHandle()
```

Parameters:
none

Return Value:
non-zero handle id, otherwise zero if unsuccessful

Description:
All programs utilizing the "DaVince DLL Library" must first call **createHandle** before any other calls to the library are made. This function returns a handle to be used in subsequent calls to the library. A handle should be created for each PDF file that is generated. After a program is finished using the handle, it must be released using **releaseHandle** in order for the DLL to clear all memory that was utilized with the handle.

See Also:
releaseHandle

getInfoText

Summary:
Retrieve various information strings from the DLL.

Syntax:
```
long getInfoText(long handleid, char *key, char *buff, long
size);
```

Parameters:
handleid (input) : DaVince DLL handle
key (input) : key of value to retrieve, see below for supported list of keys
buff (output) : memory buffer used to retrieve the key value
size (input) : size of memory buffer

Return Value:
0 if successful, otherwise non-zero error code

Description:
getInfoText retrieves various strings based on the key parameter. The acceptable key values are:

ERROR	display the text corresponding to the last error generated for the specified handle

VERSION_DATE	The date when the DLL was published
VERSION_NUMBER	The version of the DLL

"ERROR" returns an empty string if no error has occurred. If an error has occurred and a string is returned, then the error string is reset to empty for future calls.

"VERSION_DATE" and "VERSION_NUMBER" change as new versions of the "DaVince DLL Library" is updated. Use these strings to determine or display the current version of the DLL.

getPageCount

Summary:
Retrieve the PDF page count of the specified DaVince DLL handle.

Syntax:
```
long getPageCount(long handleid, long *pagecount);
```

Parameters:
handleid (input) - DaVince DLL handle
pagecount (input) - page count of the DaVince DLL handle

Return Value:
0 if successful, otherwise non-zero error code

Description:
This function returns the current page count for the specified handle. If called just before or after a PDF file is complete, it can be used to provide page count statistics.

See Also:
getInfoText

getPdfBuffer

Summary:
Copy PDF file memory image just created with preparePdfBuffer() to the specified buffer

Syntax:
```
long getPdfBuffer(long handleid, char *buff, long length);
```

Parameters:
handleid (input) - DaVince DLL handle
buff (output) - buffer that will contain the PDF file image
length (input) - length of buffer to receive image

Return Value:
0 if successful, otherwise non-zero error code

Description:

In conjunction with **preparePdfBuffer()**, this function is used to write a PDF image in memory. This is useful when the PDF image is not needed to be written to disk. PDF writing web based CGI routines, which write a PDF to standard out, or compressing a PDF file prior to writing to disk are some examples where this function would be useful.

C++ Example:

```
long handle, filesize;
handle = createHandle();
convertFile(handle, "tiff", "myfile.tif", "");
preparePdfBuffer(handle, &filesize);
// allocate memory for PDF buffer
char *buff = new char[filesize];
getPdfBuffer(handle, buff, filesize);

// write PDF buffer to cout as a CGI app
cout << "Content-type: application/pdf" << endl << endl;
cout.write(buff,filesize);
delete [] buff;
releaseHandle(handle);
```

See Also:
preparePdfBuffer
rewindPdfBuffer

getTextLength

Summary:
Calculate the text length of a string (in points) based on a given font and point size

Syntax:

```
long getTextLength(long handleid, const char *fontname,
const char *textstring, float pointsize, float
*textlength);
```

Parameters:
handleid (input) - DaVince DLL handle
fontname (input) - Font name (with optional encoding)
textstring (input) - The text string to be used to calculate the length
pointsize (input) - The point size of the text string
textlength (output) - the calculated text length

Description:
Use **getTextLength** to determine the length of a string (in points) given a specific font and point size. This function is used when writing to the page stream and the length of a string is needed. For example, right justifying or centering text on a page requires determining the length of the string. This function works only for a single font and point size. For strings using multiple fonts or point sizes, successive calls to this function will be needed and the individual text lengths summed.

preparePdfBuffer

Summary:
Create PDF file in memory for subsequent call by **getPdfBuffer**

Syntax:
```
long preparePdfBuffer(long handleid, long *pdfsize);
```

Parameters:
handleid (input) - DaVince DLL handle
pdfsize (output) - size of PDF memory image (return parameter)

Return Value:
0 if successful, otherwise non-zero error code

Description:
There are two ways to write a PDF structure pointed by a handle: either write the PDF structure to a file to create a PDF file, or write it to memory for additional manipulation (i.e. use within a CGI program, creating a mail attachment). In order to write the PDF structure to memory, two functions are required. This function, preparePDFBuffer, writes the PDF structure to memory and returns the size of the memory image for subsequent use by **getPdfBuffer**.

See Also:
getPdfBuffer
rewindPdfBuffer

releaseHandle

Summary:
Release DaVince DLL handle to free resources used by handle

Syntax:
```
long releaseHandle(long handleid)
```

Parameters:
handleid (input) - DaVince DLL handle

Return Value:
0 if successful, otherwise non-zero error code

Description:

releaseHandle is used to release memory allocated during the use of the handle. It must be called after the handle is no longer needed. It should be called even if a PDF file is never generated due to an error or other circumstance since failure to do so will result in an increase in unneeded system resources utilized by the DLL.

See Also:
createHandle

rewindPdfBuffer

Summary:
Rewind memory stream for additional call to getPdfBuffer()

Syntax:
```
long rewindPdfBuffer(long handleid);
```

Parameters:
handleid (input) - DaVince DLL handle

Return Value:
0 if successful, otherwise non-zero error code

Description:

See Also:
getPdfBuffer
preparePdfBuffer

setParameter

Summary:
Set a parameter to alter the execution of a DLL function

Syntax:
```
long setParameter(long handleid, char *key, char *value);
```

Parameters:
handleid (input) - DaVince DLL handle
key (input) - parameter key
value (input) - parameter value

Return Value:
0 if successful, otherwise non-zero error code

Description:
This function sets parameters used by the DLL functions. These parameters are case insensitive and are the same as those used in the .ini files in "DaVince Tools". As an example, the following line compresses the page stream and should be called prior to the **writePage**, **convertFile** and **convertBuffer** functions:

```
setParameter(myhandle, "compression", "true");
```

See Also:
setInfo

setInfo

Summary:
Set information values for a PDF file

Syntax:
```
long setInfo(long handleid, char *key, char *value)
```

Parameters:
handleid (input) - DaVince DLL handle
key (input) - Info object key ("Author", "Title", "Subject", etc.)
value (input) - value of key

Return Value:

0 if successful, otherwise non-zero error code

Description:
Use this function to set the Author, Title or Subject values for a PDF file.

C++ Example:

```
setInfo(handleid, "Title", "The Declaration of Independence");
```

useFont

Summary:
Specify a font to use for a page

Syntax:
`long useFont(long handleid, long pageid, const char *fontname, const char *buff, long length);`

Parameters:
handleid (input) - DaVince DLL handle
pageid (input) - Page where font is to be used
fontname (input) - Name of font to use (with optional encoding)
buff (output) - Short name of font (return value)
length (input) - Length of buff where short name will be stored

Description:
This function tells the library that the specified font will be used for the page pointed to by pageid. Use the short name returned by this function in the content for this page.

Allowable font names are any of the base 14 fonts available to all PDF files:

> Courier, Courier-Bold, Courier-BoldOblique, Courier-Oblique, Helvetica, Helvetica-Bold, Helvetica-BoldOblique, Helvetica-Oblique, Times-Roman, Times-Bold, Times-Italic, Times-BoldItalic, ZapfDingbats

Allowable encodings are any of the standard encodings available to all PDF files:

> Default, WinAnsiEncoding, MacRomanEncoding, PdfDocEncoding

More information on how encodings are used can be found in **the Portable Document Format Reference Manual**, available from Adobe's web site.

A font and encoding combination is specified by putting a space between them. For example, the following are valid font/encoding combinations for use with the fontname parameter:

Helvetica WinAnsiEncoding
Courier
TimesRoman PdfDocEncoding

When no encoding is specified as shown above with Courier, the default encoding is assumed.

The return value of the shortname paramater is used in the page description, which is written to the PDF file using the writePage() function. The shortname is preceded by a slash "/" character in the page description, as shown in the following C++ sample code:

```
//select font to use
char shortname[20];
useFont(pdfhandle, pagehandle, "Helvetica", shortname, 20);

// write page description to stream
spage << "q ";
spage << "BT ";
spage << "/" << shortname << " 72 Tf ";

spage << "72 576 Td (Hello World) Tj ";
spage << "ET ";
spage << "Q" << endl;
long pagesize = spage.tellp();
writePage(pdfhandle, pagehandle, buff, pagesize);
```

Return Value:
0 if successful, otherwise non-zero error code

See Also:
addPage
getTextLength
writePage

writePage

Summary:
write page content stored in a buffer to the PDF page.

Syntax:

`long writePage(long handleid, long pageid, const char *buff, long length);`

Parameters:
handleid (input) - DaVince DLL handle
pageid (input) - Page where font is to be used
buff (input) - Buffer containing page content
length (input) - Length of buffer

Return Value:
0 if successful, otherwise non-zero error code

Description:
This function writes the page content stored in buff to the specified page. The **useFont** function is required if text is going to be used on the page. More information about developing page content can be found in Adobe's "Portable Document Format Specification" in the "Page Description" chapter.

C++ Example:

```
        // create page stream
        // define a buffer size that will be big enough to store our
page stream
        #define BUFFSIZE (1024*10)
        char buff[BUFFSIZE];
        strstream spage(buff, BUFFSIZE);

        //select font to use
        char shortname[20];
        useFont(pdfhandle, pagehandle, "Helvetica", shortname, 20);

        // write page description to stream
        spage << "q ";
        spage << "BT ";
        spage << "/" << shortname << " 72 Tf ";

        spage << "72 576 Td (Hello World) Tj ";
        spage << "ET ";
        spage << "Q" << endl;
        long pagesize = spage.tellp();
        writePage(pdfhandle, pagehandle, buff, pagesize);
```

See Also:
addPage
useFont

writePdf

Summary:
Write PDF handle to disk

Syntax:
`long writePdf(long handleid, char *filename)`

Parameters:
handleid (input) - DaVince DLL handle
filename (input) - name of file to write

Return Value:
0 if successful, otherwise non-zero error code

Description:
There are two ways to write a PDF structure pointed by a handle: either write the PDF structure to a file to create a PDF file, or write it to memory for additional manipulation (i.e. use within a CGI program, creating a mail attachment). This function takes data referred to by the PDF handle and writes it to a PDF file on disk.

See Also:
getPdfBuffer
preparePdfBuffer
rewindPdfBuffer