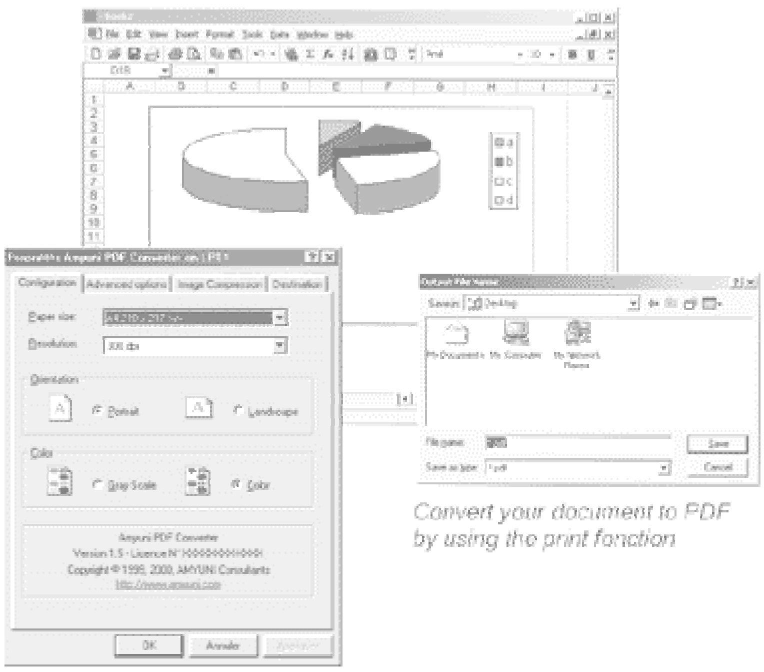
Amyuni PDF Converter

Version 1.5

User's Manual



Main property page

Convert your document to PDF by using the print fonction

AMYUNI Consultants
www.amyuni.com

Contents

Legal Information

Acknowledgments

Special thanks to:

Jean-loup Gailly (jloup@gzip.org)
Mark Adler (madler@alumni.caltech.edu)

for their work on the deflate compression.

This software is also based in part on the work of the Independent JPEG Group and on parts of the FreeType library.

What’s new in version 1.5

Features:

- Added JPEG and 256-colour compression algorithm
- Option to broadcast messages when PDF documents are generated
- Added the capability to easily add bookmarks to the PDF document
- Improved user interface for both Windows 95/98 and NT
- Added low 75 and 150 DPI resolutions
- Physical margins configurable through user interface or programming
- Ability to set output file name through user interface
- Optimised file sizes with some applications
- Added the ability to insert watermarks on each printed page

Bug fixes:

- Grey-scale option caused problems with 24 bits images under Windows NT
- Character spacing problems with some applications solved
- Concatenation failed with some applications under NT
- Blue screen problems with some applications and large documents
- Image alignment problems caused some images to be completely blurred
- Removed embedding of Arial and Times New Roman fonts
- Removed extraneous ‘>’ sign following image data that caused some PDF viewers/analysers to fail

For a complete version history, please refer to the Versions.txt file that comes with this product.

Installation

Before installing and/or using this product, please make sure you carefully read the copyright notice and agree on all its terms.

The install procedure described here applies to the standard version of the PDF Converter. For installing and using the developer version, please go to Appendix D at the end of this document.

Automatic Installation Procedure

An automatic installation procedure has been created to easily install the PDF printer on your machine. This procedure is only available for Windows 95, 98, NT, 2000 and later operating systems. For Windows 3.1, refer to the section on manual installation.

If you have received the product by e-mail, copy the file you received in an empty directory, unzip the file and launch “install.exe”.

If you have received the product by diskette or on CD-ROM, you can launch “install.exe” directly from the diskette or CD-ROM.

The installation procedure automatically detects your operating system; copies needed files to your system directory and installs the PDF printer in your system.

A log file named “install.log” is generated in the source directory. If the install procedure is launched from a read-only media such as CD-ROM, the log file will be generated in your system’s temporary directory. This file is required by the support staff to analyse problems that may occur during installation.

Manual Installation Procedure

To manually install the PDF printer into your system, we recommend you to follow the instructions in your operating system's manual.

The following options should be chosen to have the PDF driver correctly installed:
Output port LPT1: for Windows 95 version, PDF: or FILE: for NT version.
Spooler Off (or direct printing On)

Usage

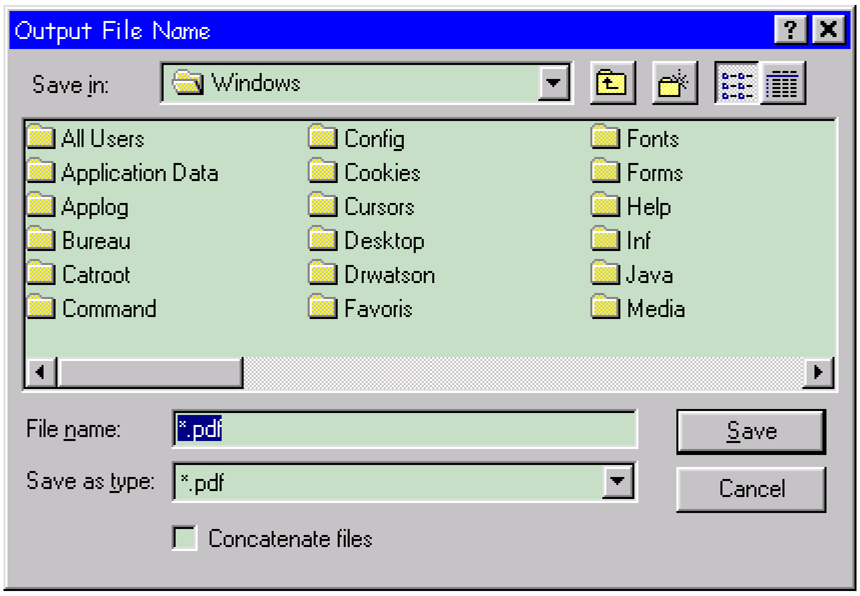
The PDF Converter works like any standard Windows® printer driver.

The main difference is that instead of directly printing to a printer, it generates a PDF 1.2 compatible file.

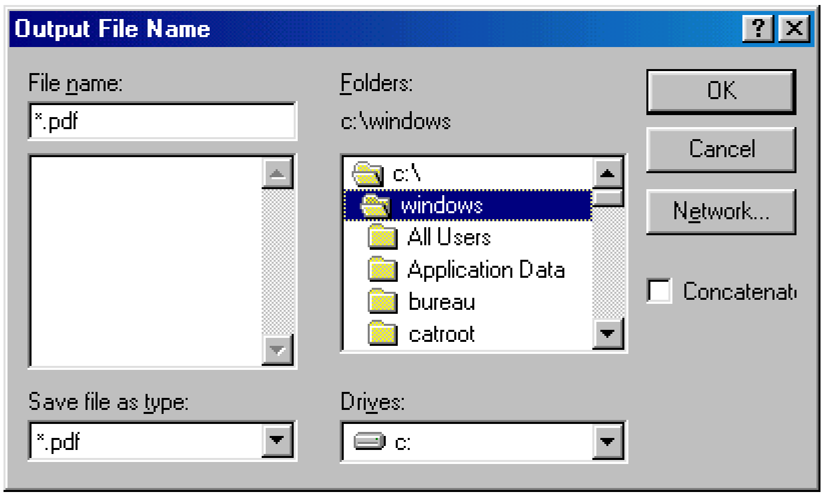
The PDF files generated by this driver can be viewed or printed to a physical printer by the free Adobe® Acrobat® viewer versions 3 or 4, the Amyuni PDF Creator, or by any other application capable of viewing standard PDF files.

When you use the "Print" option of your application, you will be prompted for the name of an output file:

Windows NT®



Windows® 3.1, 95 or 98



Just enter any valid file name with .pdf extension. Other extensions can be used but the generated file will not be automatically recognized by the system as a PDF file.

By default, the file is overwritten if it already exists. To append the document to an existing file instead of overwriting it, you can check the ‘Concatenate’ button.

The output file name can be set by the application to avoid user interaction by using the Windows StartDoc escape. Here is a sample C code that can be used to set the output file name:

```
DOCINFO      di;
di.cbSize = sizeof( DOCINFO );
di.lpszDocName = (LPCTSTR)"My document";
di.lpszOutput = (LPCTSTR)"C:\\TEMP\\TEST.PDF";
StartDoc( hDC, &di );
```

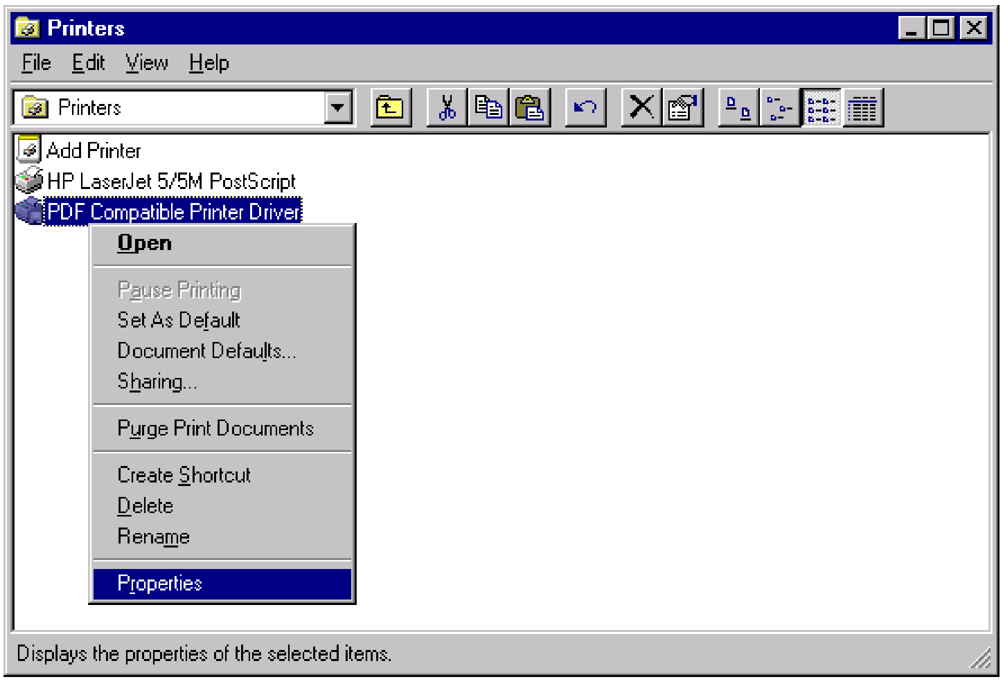
Setting the output file name using this method disables the concatenation feature under Windows NT.

Configuration

Information presented here only applies to the standard version of the PDF Converter. The developer version does not have any user interface; all printer configuration should be done programmatically as explained in the samples and technical notes. The information presented here will however help the programmer have a better understanding of the various printer options made available by the PDF Converter.

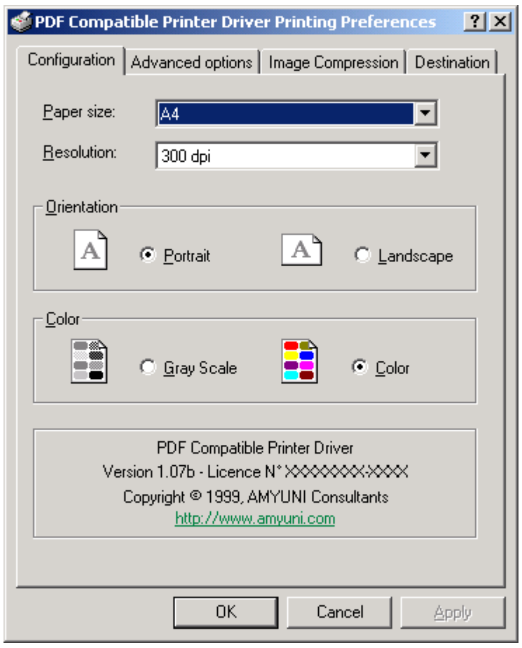
The files generated by the PDF printer driver are destined to be printed on any type of printer on any platform. The configuration settings supported by this driver are kept to a minimum to be supported by all kind of printers.

The PDF printer properties are configurable using your system control panel or your application "Print" dialog box. The printer properties are available under "Document defaults" under Windows NT/2000.



The printer properties are divided into four property pages as follows.

Property Page 1: Configuration


PDF Compatible Printer Driver Printing Preferences

Configuration | Advanced options | Image Compression | Destination

Paper size: A4

Resolution: 300 dpi

Orientation: [✓ Portrait] Landscape

Color: Gray Scale [✓ Color]

PDF Compatible Printer Driver
Version 1.07b - Licence N° XXXXXXXX-XXXX
Copyright © 1999, AMYUNI Consultants
http://www.amyuni.com

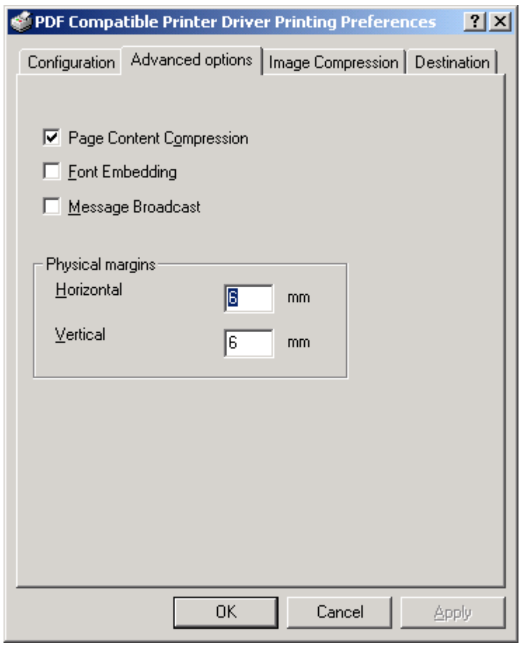
Paper size
This is the default paper size for new documents. All standard paper sizes supported by Windows are also supported by the PDF Converter. This parameter is usually set by the client application. The value entered here is only the default for new documents. The default value for the Paper size is either A4 or Letter depending on the countries where the product is used.

Resolution
75, 150, 300, 600 or 1200 Dots per inch (DPI). This concerns mainly image resolution. Text precision can also be improved by increasing this value. The default value is 300 DPI.

Orientation
Portrait or Landscape. This is also a default value used for new documents only. Applications usually set this parameter internally. The default value is Portrait.

Colour appearance
Colour or Grey-scale. Using grey scale can decrease document size for archival or transmission purposes. The default value is Colour.

Property Page 2: Advanced Options



☑ Page Content Compression
☐ Font Embedding
☐ Message Broadcast

Physical margins
Horizontal 6 mm
Vertical 6 mm

Page Content Compression

On or Off. This option enables page compression using the zlib/deflate compression algorithm. Removing this option can be useful for debugging or for applications that analyse the generated PDF file. The default value is On.

Font Embedding

On or Off. Setting this option to On will enable the driver to include all True Type® fonts used by the document in the output PDF file. This results in larger files but ensures consistent look of the document on any platform. There is no way in the current version to specify which fonts to include. However, the following fonts are never included: Arial, Times New Roman, fonts under license. The default value is Off.

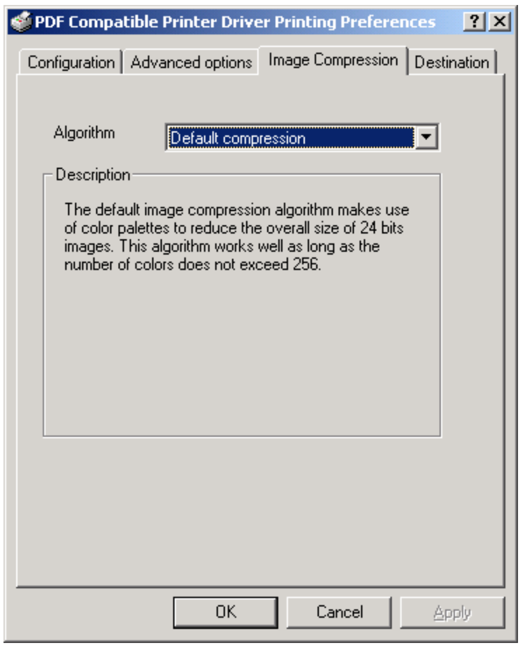
Message Broadcasting

On or Off. When activated, messages are broadcasted for all running applications each time a document is printed to PDF format. The following events result in messages being broadcasted: Start of document, End of document, Start of page, End of page. Refer to appendix A for the format of these messages and how to intercept them. The default value is Off.

Vertical and horizontal margins

Most printers have a minimum physical margin below which they cannot generate any output. Applications use this value to prevent users from drawing outside the printable area. The default value is 6 millimetres.

Property Page 3: Image Compression



Algorithm: Default compression

Description

The default image compression algorithm makes use of color palettes to reduce the overall size of 24 bits images. This algorithm works well as long as the number of colors does not exceed 256.

Image compression algorithm

Three algorithms are handled by the PDF Converter.

Default compression

The default image compression algorithm makes use of colour palettes to reduce the overall size of 24 bits images. This algorithm works well as long as the number of colours does not exceed 256.

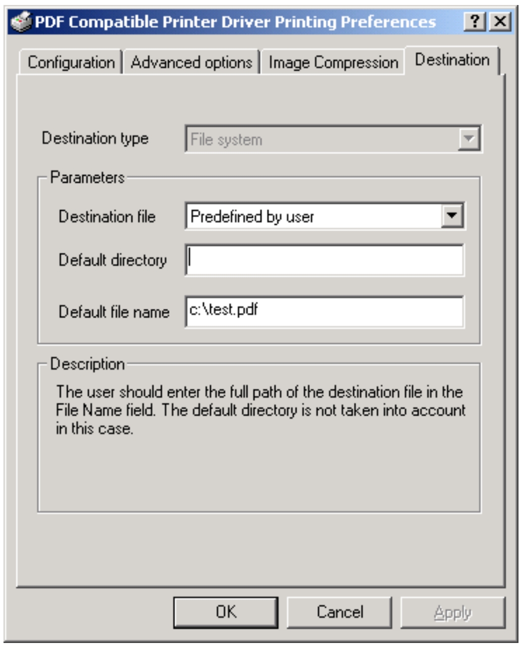
256-colour compression

The 256-colour compression algorithm calculates the most used 256 colours in an image and replaces all other colour pixels by their closest match. This lossless compression algorithm works usually very well with most applications but is relatively slow on large image with a large number of colours.

JPEG compression

The lossy JPEG compression algorithm produces excellent results with real life true colour images with a small distortion of the source image. It is not recommended for graphics than real pictures.

Property Page 4: Destination



Destination type: File system

Parameters

Destination file: Predefined by user

Default directory:

Default file name: c:\test.pdf

Description

The user should enter the full path of the destination file in the File Name field. The default directory is not taken into account in this case.

Destination type

The only value currently supported is “File system” to indicate that output is to be sent to a file on disk.

The destination file name can be set up in three different ways:

Prompt for file name:

The user is prompted for an output file name each time a document is printed to the PDF printer. If a default directory is entered, the File Save dialog box is initialised with this value. The Default file name is not used.

Document title.pdf

The output file name is set automatically by the PDF Converter by concatenating the Default directory, the document title and .pdf extension. The document title depends on every client application. No additional user prompt is needed. The Default file name is not used.

Predefined by user

The output file name is predefined by the user in the “Default file name” field. The same file name is used for all generated PDF documents. No additional user prompt is needed. The Default directory is not used.

Using the interface DLL (PDFIntf.dll standard edition)

Starting from version 1.5, this interface DLL has been superseded by the more versatile “Common Driver Interface”. This interface remains lighter to implement and the samples given here still apply.

Starting from version 1.01, an interface DLL (PdfIntf.dll) is installed on the system to allow more versatile interface to the driver.

This interface can be used from any programming language including VB/VBA.

The following functions are exported by this interface:

SetDefaultDirectory
Sets the default output directory for PDF files.

SetDefaultFileName
Sets the default file name. This should include complete path name and overrides the default directory set by SetDefaultDirectory.

Set FileNameOptions
Sets various options for generating files. The following options are currently available:

NoPrompt	prevents the display of the file name dialog box.
UseFileName	use the file name set by SetDefaultFileName as the output file name. If this option is not set and the NoPrompt option is set, the file name is constructed from the default directory, the document title and .pdf extension.
Concatenate	concatenate with existing file instead of overwriting. This is useful only if the NoPrompt option is set.
DisableCompression	disable deflate compression of the page’s content.
EmbedFonts	enable embedding of fonts used in the source document.
BroadcastMessages	send notification messages for PDF generation progress to all running application.

These options can be combined together using the addition (+) or logical OR operators. Do not forget to reset the options to none (0), as these will be valid for all applications using the driver.

VB/VBA declarations

```
Const NoPrompt As Integer = 1    ' do not prompt for file name
Const UseFileName As Integer = 2 ' use file name set by SetDefaultFileName
else use document name
Const Concatenate As Integer = 4 ' concatenate files, do not overwrite
Const DisableCompression As Integer = 8
                                 ' disable page content compression
Const EmbedFonts = 16            ' embed fonts used in the input document
Const BroadcastMessages = 32     ' enable broadcasting of PDF events

Declare Function SetDefaultDirectory Lib "PdfIntf" (ByVal Printer As String,
ByVal Directory As String) As Long
Declare Function SetDefaultFileName Lib "PdfIntf" (ByVal Printer As String,
ByVal FileName As String) As Long
Declare Function SetFileNameOptions Lib "PdfIntf" (ByVal Printer As String,
ByVal Options As Integer) As Long
```

Sample VB code

```
Sub PrintToPDF()

    ' set default directory for PDF files
    SetDefaultDirectory "PDF Compatible Printer Driver", "d:\temp"

    ' set default file name
    SetDefaultFileName "PDF Compatible Printer Driver", "d:\temp\d.pdf"

    ' set options
    SetFileNameOptions "PDF Compatible Printer Driver", NoPrompt + UseFileName
+ Concatenate

    ' print (this is to print an Excel Worksheet)
    ActiveWindow.SelectedSheets.PrintOut Copies:=1

    ' reset all options before returning
    SetFileNameOptions "PDF Compatible Printer Driver", 0

End Sub
```

C++ declarations

```
#define NoPrompt           1  // do not prompt for file name
#define UseFileName        2  // use file name set by SetDefaultFileName else
use document name
#define Concatenate        4  // concatenate files, do not overwrite
#define DisableCompression 8   // disable page content compression
#define EmbedFonts         16 // embed fonts used in the input document
#define BroadcastMessages 32 // enable broadcasting of PDF events

long SetDefaultDirectory( LPCTSTR Printer, LPCTSTR Directory );
long SetDefaultFileName( LPCTSTR Printer, LPCTSTR FileName );
long SetFileNameOptions( LPCTSTR Printer, short Options );
```

Sample C/C++ code

```
typedef long (CALLBACK* SetDefaultDirectoryProc)( LPCTSTR Printer, LPCTSTR
Directory );
typedef long (CALLBACK* SetDefaultFileNameProc)( LPCTSTR Printer, LPCTSTR
FileName );
typedef long (CALLBACK* SetFileNameOptionsProc)( LPCTSTR Printer, short
Options );

SetDefaultDirectoryProc   SetDefaultDirectory = NULL;
SetDefaultFileNameProc    SetDefaultFileName = NULL;
SetFileNameOptionsProc    SetFileNameOptions = NULL;

LPCTSTR      szPrinter = _T("PDF Generator");

HINSTANCE    hLib = NULL;

BOOL PDFPrint()
{
       // dynamically load the library. This should be done in InitInstance
       hLib = LoadLibrary( _T( "pdfintf.dll" ) );
       if ( NULL == hLib )
       {
              return FALSE;
       }

       // get exported functions addresses
       SetDefaultDirectory = (SetDefaultDirectoryProc)GetProcAddress( hLib, _T(
"SetDefaultDirectory" ) );
       SetDefaultFileName = (SetDefaultFileNameProc)GetProcAddress( hLib, _T(
"SetDefaultFileName" ) );
       SetFileNameOptions = (SetFileNameOptionsProc)GetProcAddress( hLib, _T(
"SetFileNameOptions" ) );

       if ( NULL == SetDefaultDirectory || NULL == SetDefaultFileName ||
             NULL == SetFileNameOptions
              )
       {
              return FALSE;
       }

       // set font embedding option
       (*SetFileNameOptions)( szPrinter, EmbedFonts );

       // get printer DC
       HDC    hDC = CreateDC( _T( "WINSPOOL" ), szPrinter, NULL, NULL );

       if ( NULL == hDC )
       {
              return FALSE;
```

```
	}

	// start notofication to detect end of file printing
	HANDLE		h = FindFirstChangeNotification( _T("c:\\temp"), FALSE,
FILE_NOTIFY_CHANGE_LAST_WRITE );

	// print something
	DOCINFO di;
	di.cbSize = sizeof( DOCINFO );
	di.lpszDocName = _T("My document");
	di.lpszOutput = _T("C:\\TEMP\\TEST.PDF");
	StartDoc( hDC, &di );
	StartPage( hDC );
	TextOut( hDC, 100, 100, _T( "My first PDF file" ), lstrlen( _T( "My
first PDF file" ) ) );
	EndPage( hDC );
	EndDoc( hDC );

	DeleteDC( hDC );

	// wait for my file to finish printing
	MsgWaitForMultipleObjects( 1, &h, FALSE, INFINITE, 0 );
	FindCloseChangeNotification( h );

	FreeLibrary( hLib );

	return TRUE;
}
```

Delphi declarations

```
const
  NoPrompt = 1;
  UseFileName = 2;
  Concatenate = 4;
  DisableCompression = 8;
  EmbedFonts = 16;
  BroadcastMessages = 32;

function SetDefaultDirectory(Printer, Directory : PChar) : LONG; stdcall;
external 'pdfintf.dll' name 'SetDefaultDirectory';

function SetDefaultFileName(Printer, FileName : PChar) : LONG; stdcall;
external 'pdfintf.dll' name 'SetDefaultFileName';

function SetFileNameOptions(Printer : PChar; Options : SmallInt) : LONG;
stdcall;
external 'pdfintf.dll' name 'SetFileNameOptions';
```

Using the interface DLL (PDFIntf.dll developer edition)

Starting from version 1.01, an interface DLL (PdfIntf.dll) is provided with the PDF Converter to allow more versatile interface to the driver.

This interface has been superseded by the “Common Driver Interface” as described in the accompanying manual.

This interface can be used from any programming language including VB/VBA.

The following functions are exported by this interface:

PDFDriverInit
Initialize the PDF driver library. The printer name given as parameter should be used as the name of the printer on which printing should occur. This should be ideally called at the initialization of the application or at the start of the print function.

PDFDriverInit returns 0 for success. If an error occurs, a Windows system error code is returned. To get the error message associated with the error code, you can use the Error Lookup utility or look in the Windows API header files.

PDFDriverEnd
Uninitialize the PDF driver library. This should be ideally called at the end of the application or at the end of the print function.

SetDefaultDirectory
Sets the default output directory for PDF files.
This function returns 0 when an error occurs.

SetDefaultFileName
Sets the default file name. This should include complete path name and overrides the default directory set by SetDefaultDirectory.
This function returns 0 when an error occurs.

Set FileNameOptions
Sets various options for generating files. The following options are currently available:

NoPrompt	prevents the display of the file name dialog box.

UseFileName	use the file name set by SetDefaultFileName as the output file name. If this option is not set and the NoPrompt option is set, the file name is constructed from the default directory, the document title and .pdf extension.
Concatenate	concatenate with existing file instead of overwriting. This is useful only if the NoPrompt option is set.
DisableCompression	disable deflate compression of the page's content.
EmbedFonts	enable embedding of fonts used in the source document.
BroadcastMessages	send notification messages for PDF generation progress to all running application.

These options can be combined together using the addition (+) or logical OR operators. Do not forget to reset the options to none (0), as these will be valid for all applications using the PDF Converter.

This function returns 0 when an error occurs.

VB/VBA declarations

```
Const NoPrompt As Integer = 1    ' do not prompt for file name
Const UseFileName As Integer = 2 ' use file name set by SetDefaultFileName
else use document name
Const Concatenate As Integer = 4 ' concatenate files, do not overwrite
Const DisableCompression As Integer = 8
                                 ' disable page content compression
Const EmbedFonts = 16            ' embed fonts used in the input document

Declare Function PDFDriverInit Lib "PdfIntf" (ByVal PrinterName As String) As
Long
Declare Function PDFDriverEnd Lib "PdfIntf" () As Long
Declare Function SetDefaultDirectory Lib "PdfIntf" (ByVal Directory As String)
As Long
Declare Function SetDefaultFileName Lib "PdfIntf" (ByVal FileName As String)
As Long
Declare Function SetFileNameOptions Lib "PdfIntf" (ByVal Options As Integer)
As Long
```

Sample VB code

```
Sub PrintToPDF()

' initialize the PDF driver
PDFErr = PDFDriverInit("MyPDFPrinter")

If PDFErr Then
    Exit Sub
End If
```

```
' set default directory for PDF files
SetDefaultDirectory "d:\temp"

' set default file name
SetDefaultFileName "d:\temp\d.pdf"

' set options
SetFileNameOptions NoPrompt + UseFileName

' print (this is to print an Excel Worksheet)
ActiveWindow.SelectedSheets.PrintOut Copies:=1, ActivePrinter:="MyPDFPrinter"

' reset all options before returning
SetFileNameOptions 0

' Uninitialize the PDF driver
PDFDriverEnd

End Sub
```

C++ declarations

```
#define NoPrompt          1 // do not prompt for file name
#define UseFileName       2 // use file name set by SetDefaultFileName else
use document name
#define Concatenate       4 // concatenate files, do not overwrite
#define DisableCompression8 // disable page content compression
#define EmbedFonts        16// embed fonts used in the input document

long PDFDriverInit( LPCTSTR PrinterName );
long PDFDriverEnd();
long SetDefaultDirectory( LPCTSTR Directory );
long SetDefaultFileName( LPCTSTR FileName );
long SetFileNameOptions( short Options );
```

Sample C/C++ code

```
typedef long (CALLBACK* PDFDriverInitProc)( LPCTSTR Printer );
typedef long (CALLBACK* PDFDriverEndProc)();
typedef long (CALLBACK* SetDefaultDirectoryProc)( LPCTSTR Directory );
typedef long (CALLBACK* SetDefaultFileNameProc)( LPCTSTR FileName );
typedef long (CALLBACK* SetFileNameOptionsProc)( short Options );

PDFDriverInitProc   PDFDriverInit = NULL;
PDFDriverEndProc    PDFDriverEnd = NULL;
SetDefaultDirectoryProc   SetDefaultDirectory = NULL;
SetDefaultFileNameProc    SetDefaultFileName = NULL;
SetFileNameOptionsProc    SetFileNameOptions = NULL;

LPCTSTR      szPrinter = _T("PDF Generator");
HINSTANCE    hLib = NULL;
```

```
BOOL PDFPrint()
{
// dynamically load the library. This should be done in InitInstance
hLib = LoadLibrary( _T( "pdfintf.dll" ) );
if ( NULL == hLib )
{
       return FALSE;
}

// get exported functions addresses
PDFDriverInit = (PDFDriverInitProc)GetProcAddress( hLib, _T( "PDFDriverInit" )
);
PDFDriverEnd = (PDFDriverEndProc)GetProcAddress(hLib,_T( "PDFDriverEnd" ));
SetDefaultDirectory = (SetDefaultDirectoryProc)GetProcAddress( hLib, _T(
"SetDefaultDirectory" ) );
SetDefaultFileName = (SetDefaultFileNameProc)GetProcAddress( hLib, _T(
"SetDefaultFileName" ) );
SetFileNameOptions = (SetFileNameOptionsProc)GetProcAddress( hLib, _T(
"SetFileNameOptions" ) );

if ( NULL == PDFDriverInit || NULL == PDFDriverEnd ||
        NULL == SetDefaultDirectory || NULL == SetDefaultFileName ||
        NULL == SetFileNameOptions  )
{
       return FALSE;
}

// init the PDF Printer
DWORD  Err;
if ( Err = (*PDFDriverInit)( szPrinter ) )
{
       // system error, exit
       return FALSE;
}

// get printer DC
HDC    hDC = CreateDC( _T( "WINSPOOL" ), szPrinter, NULL, NULL );
if ( NULL == hDC )
{
       return FALSE;
}

// start notofication to detect end of file printing
HANDLE       h = FindFirstChangeNotification( _T("c:\\temp"), FALSE,
FILE_NOTIFY_CHANGE_LAST_WRITE );

// print something
DOCINFO di;
di.cbSize = sizeof( DOCINFO );
di.lpszDocName = _T("My document");
di.lpszOutput = _T("C:\\TEMP\\TEST.PDF");
StartDoc( hDC, &di );
StartPage( hDC );
TextOut( hDC, 100, 100, _T( "My first PDF file" ), lstrlen( _T( "My first PDF
file" ) ) );
EndPage( hDC );
EndDoc( hDC );
DeleteDC( hDC );

// wait for my file to finish printing
MsgWaitForMultipleObjects( 1, &h, FALSE, INFINITE, 0 );
FindCloseChangeNotification( h );

// exit PDF. This should be done in ExitInstance
(*PDFDriverEnd)();
FreeLibrary( hLib );

return TRUE;
}
```

Delphi declarations

```
const
  NoPrompt = 1;
  UseFileName = 2;
  Concatenate = 4;
  DisableCompression = 8 ;
  EmbedFonts = 16 ;

function PDFDriverInit( PrinterName : PChar ) : LONG; stdcall;
external 'pdfintf.dll' name 'PDFDriverInit';

function PDFDriverEnd() : LONG; stdcall;
external 'pdfintf.dll' name 'PDFDriverEnd';

function SetDefaultDirectory( Directory : PChar) : LONG; stdcall;
external 'pdfintf.dll' name 'SetDefaultDirectory';

function SetDefaultFileName( FileName : PChar) : LONG; stdcall;
external 'pdfintf.dll' name 'SetDefaultFileName';

function SetFileNameOptions( Options : SmallInt) : LONG; stdcall;
external 'pdfintf.dll' name 'SetFileNameOptions';
```

Appendix A: PDF Converter events format

When the message broadcast option is set, the PDF Converter broadcasts messages to all running applications each time one of the following events occur:

Start of document
Start of page
End of page
End of document

Reminder:
Each Windows messages is made of three parts:
A message Id. This is a 32 bits value.
A first parameter known as wParam. This is a 16 bits value under Windows 95/98 (Printer drivers under these systems are 16 bits), and a 32 bits value under Windows NT/2000.
A second parameter known as lParam. This is a 32 bits value.

The message Id generated by the PDF converter can be retrieved by calling the Windows API RegisterMessage function:

nPDFDriverMessage = RegisterWindowMessage(“PDF Driver Event”).

This is needed to intercept messages generated by the driver.

For each printer event, belongs an event code which is given in the wParam parameter. The following event codes are defined by the Windows Device Driver Kit:

```
// printer driver events
#define DOCUMENTEVENT_STARTDOCPRE   5   // before StartDoc is handled
#define DOCUMENTEVENT_STARTDOCPOST 13   // after StartDoc is handled
#define DOCUMENTEVENT_STARTPAGE     6   // StartPage handled
#define DOCUMENTEVENT_ENDPAGE       7   // EndPage handled
#define DOCUMENTEVENT_ENDDOCPRE     8   // before EndDoc is handled
#define DOCUMENTEVENT_ENDDOCPOST   12   // after EndDoc is handled

// the following can be returned from the event handling function
#define DOCUMENTEVENT_SUCCESS     1
#define DOCUMENTEVENT_UNSUPPORTED 0
#define DOCUMENTEVENT_FAILURE     -1
```

Windows 95/98

Event	wParam	lParam	
		LOWORD	HIWORD
Before StartDoc is handled	DOCUMENTEVENT_STARTDOCPRE	JobId	hDC
After StartDoc is handled	DOCUMENTEVENT_STARTDOCPOST	JobId	hDC
StartPage	DOCUMENTEVENT_STARTPAGE	JobId	hDC
EndPage	DOCUMENTEVENT_ENDPAGE	JobId	hDC
Before EndDoc is handled	DOCUMENTEVENT_ENDDOCPRE	JobId	hDC
After EndDoc is handled	DOCUMENTEVENT_ENDDOCPOST	JobId	hDC

Windows NT

Event	wParam		lParam
	LOWORD	HIWORD	
Before StartDoc is handled	**DOCUMENTEVENT_STARTDOCPRE**	**JobId**	**hDC**
After StartDoc is handled	**DOCUMENTEVENT_STARTDOCPOST**	**JobId**	**hDC**
StartPage	**DOCUMENTEVENT_STARTPAGE**	**JobId**	**hDC**
EndPage	**DOCUMENTEVENT_ENDPAGE**	**JobId**	**hDC**
Before EndDoc is handled	**DOCUMENTEVENT_ENDDOCPRE**	**JobId**	**hDC**
After EndDoc is handled	**DOCUMENTEVENT_ENDDOCPOST**	**JobId**	**hDC**

Sample message handling function

```
///////////////////////////////////////////////////////////////////////////////
LONG CMessageTestDlg::OnPDFEvent( UINT wParam, LONG lParam )
///////////////////////////////////////////////////////////////////////////////
{
// a PDF event has occured
//
       DWORD  jobID;
       HDC    hDC;
       WORD   eventId;
```

```
	// get event ID
	eventId = LOWORD( wParam );

	if ( m_bNT )
	{	// Windows NT
		jobId = HIWORD( wParam );
		hDC = lParam;
	}
	else
	{	// Windows 95/98
		jobId = LOWORD( lParam );
		hDC = HIWORD( lParam );
	}

	switch ( eventId )
	{
	case DOCUMENTEVENT_STARTDOCPRE:		// before StartDoc is handled
		m_log.AddString( "StartDoc called but not handled yet by the
driver" );
		break;
	case DOCUMENTEVENT_STARTDOCPOST:	// after StartDoc is handled
		m_log.AddString( "StartDoc called and handled by the driver" );
		break;
	case DOCUMENTEVENT_STARTPAGE:		// StartPage handled
		m_log.AddString( "StartPage called and handled" );
		break;
	case DOCUMENTEVENT_ENDPAGE:		// EndPage handled
		m_log.AddString( "EndPage called and handled" );
		break;
	case DOCUMENTEVENT_ENDDOCPRE:		// before EndDoc is handled
		m_log.AddString( "EndDoc called but not handled yet" );
		break;
	case DOCUMENTEVENT_ENDDOCPOST:		// after EndDoc is handled
		m_log.AddString( "EndDoc called and handled" );
		break;
	default:
		return DOCUMENTEVENT_UNSUPPORTED;
	}
	return DOCUMENTEVENT_SUCCESS;
}
```

Appendix B: Adding bookmarks to the PDF document

Version 1.5 of the PDF Converter, added the ability to easily include bookmarks in the resulting PDF document while it is being generated.

Bookmarks can be added programmatically, either by using Windows Escape function, or the SetBookmark CDI ("Common Driver Interface") function.

The following function which is implemented in the "Common Driver Interface" allows embedding of bookmarks:

```
////////////////////////////////////////////////////////////////////////
long WINAPI SetBookmark( HDC hDC, long lParent, LPCTSTR szTitle )
////////////////////////////////////////////////////////////////////////
{
// MAKE SURE COMPILER STRUCT ALIGNEMENT IS SET TO 1 OR 2.
//
        struct
        {
                long    lParent;
                char    szTitle[255];
        } Bookmark;

        Bookmark.lParent = lParent;
        lstrcpyn( Bookmark.szTitle, szTitle, 254 );

        return ExtEscape( hDC, 250, sizeof( DWORD ) + lstrlen( Bookmark.szTitle
) + 1,
(LPCSTR)&Bookmark, 0, NULL );
}
```

The bookmark will be inserted at the location where the last text drawing operation occurred. Example: if you draw text in the middle of page 3 of your document and call SetBookmark immediately after, the bookmark will point to the middle of page 3 of the PDF document.

PDF bookmarks are structured in a tree. The root has an id of 0. This is where the lParent parameter is used. Each time you insert a bookmark, the ExtEscape function returns the bookmark ID which can be used to insert other bookmarks as children.

VB sample for inserting bookmarks

```
Public pdf As New CDIntf.CDIntf

Private Sub Form_Load()

    ' initialize PDF printer and set it as default
    pdf.DriverInit "PDF Compatible Printer Driver"
    pdf.SetDefaultPrinter

    ' draw some text
    Printer.CurrentX = 200
    Printer.CurrentY = 400
    Printer.Print "Bookmark 1"
    ' set a bookmark on page 1
    pdf.SetBookmark Printer.hDC, 0, "Bookmark 1"

    ' go to next page
    Printer.NewPage

    ' draw some text and set a new bookmark
    Printer.CurrentX = 100
    Printer.CurrentY = 100
    Printer.Print "Bookmark 2"
    Parent = pdf.SetBookmark(Printer.hDC, 0, "Bookmark 2")

    ' set a bookmark as child of another bookmark
    Printer.CurrentX = 100
    Printer.CurrentY = 800
    Printer.Print "Submark 2-1"
    pdf.SetBookmark Printer.hDC, Parent, "Submark 2-1"

    Printer.EndDoc

    pdf.DriverEnd

End Sub
```

Appendix C: Using the Amyuni Printer Drivers with Visual FoxPro 6

Objective:
To provide Visual FoxPro users with technical information to interface with the Amyuni series of printer drivers.

The full technical note is available from http://www.amyuni.com/support.htm. Along with this note, you will find: the FLL interface for Visual FoxPro (FLLINTF.FLL) and a sample FoxPro application.

This information applies to:
Microsoft Visual FoxPro versions 5 and 6
Amyuni PDF Converter / Printer Driver
Amyuni DHTML Converter / Printer Driver
Amyuni RTF Converter / Printer Driver
Amyuni EMF Converter / Printer Driver

To facilitate the use of the Amyuni Printer Drivers with VFP, two interface DLLs have been developed and made available through www.amyuni.com:
CDINTF.DLL (or “Common Driver Interface” DLL) contains two types of interfaces: DLL and ActiveX
FLLINTF.FLL contains a specific VFP extension library interface. This FLL does not work alone but with conjunction of CDINTF.DLL.

The Amyuni converters behave like standard printers that are installed in the printers’ folder in one of two ways:
For the standard version of the converters, the printers are installed using the supplied “install.exe” utility and remain in the printers’ folder.
For the developer version of the same products, the printers are installed programmatically at the start of the application, and removed before exiting.

The VFP application programmer can choose between three interfacing methods:
The ActiveX interface that is also provided by CDINTF.DLL
The FLL interface that is provided by FLLINTF.DLL
The DLL interface that is provided by CDINTF.DLL

All three interfaces provide similar services. The FLL interface is however not complete and only the services not available through standard FoxPro programming are implemented in the FLL. To know which functions are available through the VFP interface, please look at each function’s documentation in the “Common Driver Interface” manual

The use of the DLL interface is not recommended for VFP users, as it adds complexity and no real benefits to the programmer.

Prior to reading this note, the user should be familiar with the CDI functions by going through its documentation.

Using the ActiveX interface from within FoxPro

Prior to using the ActiveX interface, the DLL should be registered in the system by executing:

REGSVR32 CDINTF.DLL

from the location where the DLL has been installed.

NOTE: bookmarks cannot be added to PDF files when using the ActiveX or DLL interfaces. They are only available through the FLL interface. However, nothing prevents developers form using both interfaces at the same time.

The ActiveX object needs to be created first with the CreateObject FoxPro function:

```
Pdf = CreateObject( "CDINTF.CDINTF" )
```

The printer can now be initialised using one of the following functions:

```
Pdf.DriverInit( PrinterName )                && for the standard versions
PrinterName should be the name of the printer as it appears in the Printers
folder
```

Or

```
Pdf.PDFDriverInit( NewPrinterName )    && this will add a new printer to the
printers folder, to be used with the developer’s versions
```

Initialisation should be done as early as possible within the main application.

To modify printer settings, you can change one or more of the ActiveX properties. For ex, to set the resolution to 600 DPI:

```
Pdf.Resolution = 600
```

The modified parameters will not be in effect unless SetDefaultConfig is called. This notifies FoxPro of the change in printer properties:

```
Pdf.SetDefaultConfig
```

Setting the output file name and options can be done using:
```
Pdf.FileNameOptions = 1 + 2      && no prompt + use file name
Pdf.DefaultFileName = "test.pdf" && set output file name

* printing can be done here
SET PRINTER TO NAME "PDF Compatible Printer Driver"
REPORT FORM authors.frx NOEJECT NOCONSOLE TO PRINTER
```

Before exiting the application, allocated memory has to freed and the printer removed (in the case of the developer versions) by calling:
```
Pdf.DriverEnd
```

Using the FLL interface

Prior to using the FLL interface, the FLL should be registered in FoxPro by executing:
```
SET LIBRARY TO FLLINTF.FLL ADDITIVE
```

The printer can be initialised using one of the following functions:
```
DriverInit( PrinterName )              && for the standard versions,
PrinterName should be the name of the printer as it appears in the Printers
folder
```
Or
```
Pdf.PDFDriverInit( NewPrinterName )    && this will add a new printer to the
printers folder, to be used with the developer’s versions
```

Initialisation should be done as early as possible within the main application.

Setting the output file name and options can be done using:
```
FileNameOptions( 1 + 2 + 32 )    && no prompt + use file name + broadcast
DefaultFileName( "test.pdf" )    && set output file name

* Printing can be done here
SET PRINTER TO NAME "PDF Compatible Printer Driver"
REPORT FORM authors.frx NOEJECT NOCONSOLE TO PRINTER
```

To add bookmarks to PDF files requires some specific handling as compared to other applications. Bookmarks can only be added while a report is printed and this requires a handle to a Printer Device Context that is readily available from most applications but is hidden in VFP. To overcome this problem, the FLL installs a hook to read the Device Context of the job being printed. This is why the “Broadcast Messages” option should be on to enable bookmarks.

To insert a bookmark at the current print location, call:

```
SetBookmark( Parent, Title )
```

This call can be placed in a user defined function that can be called from within reports:

```
* Adds a bookmark to the report at the current print location
function AddBookmark
PARAMETERS PageNo
=SetBookmark( 0, "Test" + Str(PageNo) )
return ""
```

Before exiting the application, allocated memory has to freed and the printer removed (in the case of the developer versions) by calling:

```
DriverEnd
```

Sample Application

The sample application provided with this note demonstrates the use of the ActiveX and FLL interfaces. It is made up of a simple project, one report named Authors, one database named Books with one Authors database. It also contains FLLINTF.FLL and CDINTF.DLL and this technical note.

The “vfptest.prg” program initialises the FLL and the PDF or HTML printer, prints the Authors report while inserting bookmarks for each page of the report.

You will note that to generate bookmarks, a hidden field is inserted in the page header. The purpose of this field is to call the SetBookmark function for each page.

Here is a listing of vfptest.prg:

```
* the FLL should be in the application directory along with CDINTF.DLL
* load the interface FLL
set default to "c:\vfptest"      && modifiy this line to point to the real
application location
set library to "FllIntf.fll" ADDITIVE

On ERROR DO errhand;

PDF = DriverInit( "PDF Compatible Printer Driver" )        && for the
standard version
&&PDF = PDFDriverInit( "My PDF Printer" )     && for the developer version
&&HTML = HTMLDriverInit( "My HTML Printer" )&& for the HTML driver

DefaultFileName( PDF, "c:\test.pdf" )  && set output file name
FileNameOptions( PDF, 1 + 2 + 32 )     && no prompt + use file name +
broadcast messages
*
```

```
* Note: in order to insert bookmarks from VFP, message broadcasting should be
enbaled
On ERROR && restore system error handler

set printer to name "PDF Compatible Printer Driver"

REPORT FORM authors.frx NOEJECT NOCONSOLE TO PRINTER

DriverEnd( PDF )    && for the developer versions: remove the PDF or HTML
printer, otherwise only clears some memory

set library to

return

* adds a bookmark to the report at the current print location
function AddBookmark
PARAMETERS PageNo
=SetBookmark( 0, "Test" + Str(PageNo))
return ""

* error handling: displays last error message
Procedure errhand
Clear
? 'Error message: ' + LASTERRORMSG()
cancel
```

Appendix D: Using the developer version of the PDF Converter

Objective:
To help developers integrate the developer versions of the Amyuni Converters inside their applications.

You can find the full technical note by visiting http://www.amyuni.com/support.htm. Along with this note, you will find: the install utility for the PDF Converter that may be needed under some circumstances.

The developer version of the Amyuni Converters is a special version of these products that can be distributed with the licensees' applications without paying any additional royalties to Amyuni Consultants.

By special version, we mean a version that:
does not need to be pre-installed on the client system (no install utility)
does not have any properties dialog box
does not have any "File Save As" dialog box

The reference manual for the developer versions is the "Developer's manual" provided with each product. The user's manual also provided helps understand the overall operation of the products. The dialog boxes that are shown in this manual are not available in the developer version.

All printer configuration, file destination and options settings should be done programmatically by the main application.

The "Common Driver Interface" (CDINTF.DLL) provided with each of the mentioned products, is a DLL that contains both a standard DLL and an ActiveX interface to easily configure the printers.

Step by step procedure for using the Amyuni Converters, developer version

Step 1 – Copy all distributable files to the application's main directory
The application's main directory is usually where the executable file is located. The list of distributable files is given in the developer manual of each product.

Step 2 – Register the CDIntf ActiveX
If you are using the ActiveX interface, you need to register the ActiveX by calling: REGSVR32 CDINTF.DLL, from the directory where you copied the printer DLLs. You can use CDIntf through the DLL convention without the need for registering or creating ActiveXs.

Step 3 – Initialise the printer at start-up of your application
An application needs to initialise the printer when it is launched. To initialise the printer, you need to call PDFDriverInit, HTMLDriverInit, RTFDriverInit or EMFDriverInit depending on which product you are using. Each of these functions take as parameter the name of the printer that will be added to the printers folder as long as the application is running. Ex: PDFDriverInit("MyCompany PDF Export") will add a printer named "MyCompany PDF Export" to the list of printers available in the system. To avoid problems, use a name that is likely to be unique on the system where your application is running. Adding your company's name or your application's name or both to the printer name will help ensure this.

Step 4 – Export to the format of your choice by printing from your application
When the user chooses the export function of your application to generate a PDF, HTML, RTF or EMF file, you need to set up the output file name using SetDefaultFileName, the file generation options SetFileNameOptions(NoPrompt + UseFileName + …) and print to the "MyCompany PDF Export" as you would do when printing to any other printer.

Step 5 – Restore the printer to its previous setting
When printing is over, you need to call SetFileNameOptions(0) to prevent other applications or users from overwriting the file that has just been generated from your application.

Step 6 – Remove the printer before exiting
Before exiting the application, call DriverEnd to remove the printer from the list of available printers.

Frequently encountered problems / asked questions

Q – *My application is deployed on NT/2000 systems where users do not have permission to add / remove a printer. How can I use these products under this configuration ?*

A – To use the Amyuni Converters in situations where users do not have enough rights to add or remove a printer, the printer needs to be installed in these systems by an administrator at the same time as the application is installed or at some later stage. To install the printer, you can use one of two methods:
In your application's install procedure, call PDFDriverInit("MyCompany PDF Export") or any other xxxDriverInit function. If you are using the ActiveX interface, the printer will be removed as soon as you exit the install application, to avoid this call DriverInit("MyCompany PDF Export") immediately after PDFDriverInit("MyCompany PDF Export").
Use the attached install utility while logged in as an administrator. This applies only to the PDF Converter, for other formats please contact support@amyuni.com. You can specify a printer name by using (install "MyCompany PDF Export"). You can also avoid the display of any dialog box by activating silent mode with the /s switch.
Your application's code does not need to be changed under this situation.

Q – *When frequently using one the Converter products, the DriverInit function fails or succeeds but printing fails. This happens randomly with no apparent reason.*

A – This problem usually occurs when xxxDriverInit / DriverEnd are called each time a document is converted. It is very important that these functions be called once at application initialization (for xxxDriverInit) and once before exiting (for DriverEnd)

Q – *What is the difference between the DriverInit function and one of the xxxDriverInit functions ?*

A – The DriverInit function only attaches to an existing printer. If the specified printer does not exist, it is not created and the function fails. The xxxDriverInit functions attach to the printer if it already exists, otherwise they attempt to create it. In both cases, xxxDriverInit functions cause the printer to be removed when DriverEnd is called., whereas DriverInit leaves the printer in the system.

Technical Support

You can obtain technical support on our web site by visiting the following page:

http://www.amyuni.com/support.htm

Before contacting technical support, please take a look at the technical notes for answers to common questions.

You need to be a registered user of our products to obtain technical support.

To be able to service you in an effective manner, include when possible the original document where the problem occurred and the PDF file generated by our driver.

Common Driver Interface

Version 1.02

Developer's Manual

AMYUNI Consultants

www.amyuni.com

Contents

Introduction

The « Common Driver Interface » (**CDI**) has been developed to make easier the common configuration tasks for printers and printer drivers.

It is currently available with the following products :

- Amyuni PDF Converter / PDF Compatible Printer Driver
- Amyuni DHTML Converter / Dynamic HTML Printer Driver
- Amyuni EMF Converter / EMF Printer Driver
- Amyuni RTF Converter / RTF Printer Driver

When used with one the previously mentioned products, the user should first familiarize with each product's operation by going through the user's or developer's manual.

This interface resides in a DLL named CDINTF.DLL, which has two calling conventions :

- A standard DLL calling convention
- An ActiveX calling convention

Users wishing to use this interface as an ActiveX from any ActiveX aware application should register the DLL in the system by calling:

Regsvr32 CDINTF.DLL

from the location where this DLL is installed.

CDI Properties

PaperSize

Type	: short integer
Description	: set or retrieve the paper size for all documents
DLL calls	: long SetPaperSize(HANDLE hPrinter, **long** PaperSize)
	: long GetPaperSize(HANDLE hPrinter)
FLL calls	: SetPaperSize(Printer, PaperSize)
	: paperSize = GetPaperSize(Printer)

Possible values for this property are predefined by Windows. Some common paper size values :

Paper Size	*PaperSize value*
Letter 8 1/2 x 11 in	1
Legal 8 1/2 x 14 in	5
A4 210 x 297 mm	9
A3 297 x 420 mm	8
Custom size	256

Note :
The hPrinter handle passed as a first parameter to the DLL functions, is the handle returned by a call to one of the DriverInit, PDFDriverInit, HTMLDriverInit, EMFDriverInit or RTFDriverInit calls.

PaperWidth / PaperLength

Type	: long integer
Description	: set or retreive custom paper size
DLL calls	: long SetPaperLength(HANDLE hPrinter, long PaperLength)
	: long GetPaperLength(HANDLE hPrinter)
	: long SetPaperWidth(HANDLE hPrinter, long PaperWidth)
	: long GetPaperWidth(HANDLE hPrinter)
FLL calls	: Not available

The unit value of these property is one thenth of a millimeter (mm/10). These properties are set to zero for all standard paper sizes. When one of these properties is changed, the PaperSize property automatically switches to 256 (Custom Size).

Orientation

Type	: short integer
Description	: set or retreive default paper orientation
DLL calls	: long SetOrientation(HANDLE hPrinter, short Orientation)
	: long GetOrientation(HANDLE hPrinter)
FLL calls	: Not available

Possible values of this property are:

Paper Orientation	*Orientation value*
Portrait	1
Landscape	2

Resolution

Type	: long integer
Description	: set or retreive default image resolution
DLL calls	: long SetResolution(HANDLE hPrinter, long Resolution)
	: long GetResolution(HANDLE hPrinter)
FLL calls	: Not available

Possible values of this property are:

Image Resolution	*Resolution value*
75 dots per inch	75
150 dots per inch	150
300 dots per inch	300
600 dots per inch	600
1200 dots per inch	1200

HorizontalMargin / VerticalMargin

Type	: long integer
Description	: set or retrieve default physical margins
DLL calls	: long SetHorizontalMargin(HANDLE hPrinter, long Margin)
	: long GetHorizontalMargin(HANDLE hPrinter)

	: long SetVerticalMargin(HANDLE hPrinter, long Margin)
	: long GetVerticalMargin(HANDLE hPrinter)
FLL calls	: Not available

The unit value of these property is one thenth of a millimeter (mm/10). The default values for these properties is 60 (6mm), which is on average the lowest physical margin most printers can handle.

FontEmbedding

Type	: boolean
Description	: set or retreive font embedding option
DLL calls	: long SetFontEmbedding(HANDLE hPrinter, long FontEmbedding)
	: long GetFontEmedding(HANDLE hPrinter)
FLL calls	: Not available

Note :
This property is only available for the Amyuni PDF Converter / PDF Compatible Printer Driver

Compression

Type	: boolean
Description	: set or retreive page content compression option
DLL calls	: long SetCompression(HANDLE hPrinter, long Compression)
	: long GetCompression(HANDLE hPrinter)
FLL calls	: Not available

Note :
This property is only available for the PDF Compatible Printer Driver

JPEGCompression

Type	: boolean
Description	: set or retreive JPEG compression of 24 bit images option
DLL calls	: long SetJPEGCompression(HANDLE hPrinter, long Compression)
	: long GetJPEGCompression(HANDLE hPrinter)
FLL calls	: Not available

Note :

This property is only available for the Dynamic HTML Printer Driver and the PDF Compatible Printer Driver.

HTMLUseLayers

Type	: boolean
Description	: when set, create one HTML document with layers for multi-page documents
DLL calls	: long SetHTMLUseLayers(HANDLE hPrinter, long UseLayers)
	: long GetHTMLUseLayers (HANDLE hPrinter)
FLL calls	: Not available

Note :
This property is only available for the Dynamic HTML Printer Driver

HTMLMultiplePages

Type	: boolean
Description	: when set, create multiple HTML files for multi-page documents
DLL calls	: long SetHTMLMultiplePages(HANDLE hPrinter, long MultiHTML)
	: long GetHTMLMultiplePages (HANDLE hPrinter)
FLL calls	: Not available

Note :
This property is only available for the Dynamic HTML Printer Driver

The HTMLUseLayers overrides the setting for this property. When both properties are cleared (= FALSE), only one HTML file is generated with all pages appended one after the other.

DefaultDirectory

Type	: string
Description	: set or retreive default directory for output files
DLL calls	: long SetDefaultDirectory(HANDLE hPrinter, LPTSTR szDir)
	: Get function not available
FLL calls	: DefaultDirectory(Printer, Directory)
	: Get function not available

DefaultFileName

Type	: string
Description	: set or retreive default file name for output files
DLL calls	: long SetDefaultFileName(HANDLE hPrinter, LPTSTR szFileName)
	: Get function not available
FLL calls	: DefaultFileName(Printer, FileName)
	: Get function not available

Note :
For the Dynamic HTML Printer Driver, this is the name of the first file to be generated. Other HTML files will be suffixed by the page number, and images will be suffixed by image number and .bmp or .jpg extension.

FileNameOptions

Type	: short integer
Description	: set or retreive file name options
DLL calls	: long SetFileNameOptions(HANDLE hPrinter, int nOptions)
	: Get function not available
FLL calls	: FileNameOptions(Printer, Options)
	: Get function not available

The options depend on the printer driver being used. Please refer to each driver user’s or developer’s manual for possible values.

CDI Methods

DriverInit(PrinterName)

Return type	: long
Description	: Initializes the library for use with any printer
DLL call	: HANDLE DriverInit(LPTSTR szPrinter)
FLL call	: Printer = DriverInit(PrinterName)

The return value of this function is an internal data handle that is to be passed as a first parameter to all other functions. In the case of the ActiveX interface, this handle is not used.

A return value of 0, indicates that an error occured. You can use the GetLastError Windows API to get the error code, or the GetLastErrorMsg CDI call to convert the error code to an error message.

The PrinterName parameter should be the name of the printer as it appears in printers control panel.

Note :
This function should be used with the standard versions of the above mentioned products or with any other printer driver. Developers using the special versions of the drivers should use PDFDriverInit, HTMLDriverInit, EMFDriverInit or RtfDriverInit.

PDFDriverInit(PrinterName)

Return type	: long
Description	: Initializes the PDF printer
DLL call	: HANDLE PDFDriverInit(LPTSTR szPrinter)
FLL call	: Printer = PDFDriverInit(PrinterName)

The return value of this function is an internal data handle that is to be passed as a first parameter to all other functions. In the case of the ActiveX interface, this handle is not used.

A return value of 0, indicates that an error occured. You can use the GetLastError Windows API to get the error code, or the GetLastErrorMsg CDI call to convert the error code to an error message.

Note :
This function is only available for developers who have licensed the “PDF Compatible Printer Driver”. Developers using the standard version of this driver should use DriverInit instead.

HTMLDriverInit(PrinterName)

Return type	: long
Description	: Initializes the HTML printer
DLL call	: HANDLE HTMLDriverInit(LPTSTR szPrinter)
FLL call	: Printer = HTMLDriverInit(PrinterName)

The return value of this function is an internal data handle that is to be passed as a first parameter to all other functions. In the case of the ActiveX interface, this handle is not used.

A return value of 0, indicates that an error occured. You can use the GetLastError Windows API to get the error code, or the GetLastErrorMsg CDI call to convert the error code to an error message.

This function is only available for developers who have licensed the “Dynamic HTML Printer Driver”. Developers using the standard version of this driver should use DriverInit instead.

EMFDriverInit(PrinterName)

Return type	: long
Description	: Initializes the EMF printer
DLL call	: HANDLE EMFDriverInit(LPTSTR szPrinter)
FLL call	: Printer = HTMLDriverInit(PrinterName)

The return value of this function is an internal data handle that is to be passed as a first parameter to all other functions. In the case of the ActiveX interface, this handle is not used.

A return value of 0, indicates that an error occured. You can use the GetLastError Windows API to get the error code, or the GetLastErrorMsg CDI call to convert the error code to an error message.

This function is only available for developers who have licensed the “EMF Printer Driver”. Developers using the standard version of this driver should use DriverInit instead.

DriverEnd

Return type	: none
Description	: uninstalls the printer installed by one of the DriveInit methods
DLL call	: void DriverEnd(HANDLE hPrinter)
FLL call	: DriverEnd(Printer)

This method should be called before exiting the main application.

If SetDefaultPrinter has been called to change the Windows default printer, the previous default printer is restored when this function is called.

CDICreateDC

Return type	: Handle to a Printer Device Context
Description	: returns a DC associated to the printer and takes into account the parameters as set
	: by calls to the CDI properties or methods
DLL call	: HDC CreateDC(HANDLE hPrinter)
FLL call	: Not available

This is only a wrapper to the CreateDC Windows API.

SetDefaultConfig

Return type	: boolean
Description	: sets the printer parameters as defined by previous calls the the CDI as default
	: for all applications
DLL call	: long SetDefaultConfig(HANDLE hPrinter)
FLL call	: SetDefaultConfig(Printer)

You should use this function if you need your settings to be taken as default values by all applications. Otherwise, these settings will be true only for Device Contexts returned by the CreateDC function.

SetDefaultPrinter

Return type	: boolean
Description	: sets the current printer as the default printer for all applications
DLL call	: long SetDefaultPrinter(HANDLE hPrinter)
FLL call	: SetDefaultPrinter(Printer)

To make sure the previous default printer is not restored when calling DriverEnd or when the ActiveX object is destroyed, you can call this function twice.

RestoreDefaultPrinter

Return type	: boolean
Description	: restores the default printer to its value before the call to SetDefaultPrinter
DLL call	: long RestoreDefaultPrinter(HANDLE hPrinter)
FLL call	: RestoreDefaultPrinter (Printer)

This function is called automatically when PDFDriverInit is called.

GetLastErrorMsg

Return type	: string
Description	: gets the error message associated with the last error code
DLL call	: void GetLastErrorMsg(LPTSTR Msg, long MaxMsg)
FLL call	: message = LastErrorMsg()

This function can be called after a call to one of the DriverInit functions to get the system error message.

SetBookmark(hDC, Parent, Title)

Return type	: long
Description	: puts a bookmark at the current printing location
DLL call	: long SetBookmark(long hDC, long lParent, LPCTSTR szTitle)
FLL call	: Parent = SetBookmark(Parent, Title)

This function can be used with the PDF Printer Driver only. As bookmarks can be organized in a tree, this function returns the ID of the inserted bookmark, this ID can be used to insert bookmarks beneath another one.

SetWatermark(Watermark, FontName, FontSize, Orientation, Colour, HorzPos, VertPos, Foreground)

Return type : long
Description : adds a watermark to every page of a PDF document
DLL call : long SetWatermark(HANDLE hIntf, LPTSTR szWatermark, LPTSTR szFont, short fontSize, short Orientation, COLORREF colour, LONG xPos, LONG yPos, BOOL bForeground)
FLL call : Not available

Parameters	: Watermark	text to print on each page
	: FontName	font to be used to print text
	: FontSize	font size in 0.1 inch units
	: Orientation	watermark text orientation in 0.1 degree units
	: Colour	watermark colour
	: HorzPos	horizontal position of text in 0.1 inch units
	: VertPos	vertical position of text in 0.1 inch units
	: Foreground	flag that indicates whether the watermark should be above or below the page content

This function can be used with the PDF Printer Driver only.

Sample VB code using the ActiveX interface

This is a sample VBA macro that allows you to convert an Excel worksheet to a PDF file without user interaction.

```
Const NoPrompt As Integer = 1    ' do not prompt for file name
Const UseFileName As Integer = 2 ' use file name set by SetDefaultFileName
else use document name
Const Concatenate As Integer = 4 ' concatenate files, do not overwrite
Const DisableCompression As Integer = 8
                                 ' disable page content compression
Const EmbedFonts = 16            ' embed fonts used in the input document
Const BroadcastMessages = 32     ' enable broadcasting of PDF events

Sub PrintToPDF()

Dim PDFPrinter As New CDIntf.CDIntf

' this adds a new printer named MyPDFPrinter
PDFPrinter.PDFDriverInit "MyPDFPrinter"


' this attaches the CDIntf object to an existing printer
' it will fail if the printer does not already exist
'PDFPrinter.PDFDriverInit "PDF Compatible Printer Driver"

PDFPrinter.PaperSize = 1  ' set paper size to letter
PDFPrinter.Orientation = 2' landscape

PDFPrinter.SetDefaultConfig
                          ' set default configuration for all applications
PDFPrinter.SetDefaultPrinter
                          ' set printer as default

' set default directory for PDF files
PDFPrinter.DefaultDirectory = "d:\temp"

' set default file name
PDFPrinter.DefaultFileName = "d:\temp\test.pdf"

' set options
PDFPrinter.FileNameOptions = NoPrompt + UseFileName

' print (this line should be changed according to the target application)
ActiveWindow.SelectedSheets.PrintOut Copies:=1, ActivePrinter:="MyPDFPrinter"

' reset all options before returning
PDFPrinter.FileNameOptions = 0

Set PDFPrinter = Nothing  ' also removes the PDF printer
```

```
End Sub
```

Sample VB code using the DLL interface

Here is some sample VB code that uses the DLL interface of CDINTF to access the PDF Converter.

```
' under VB, these should be in a separate module
Declare Function DriverInit Lib "CDIntf" (ByVal Printer As String) As Long
Declare Function PDFDriverInit Lib "CDIntf" (ByVal Printer As String) As Long
Declare Sub DriverEnd Lib "CDIntf" (ByVal hPrinter As Long)

Declare Function SetDefaultDirectory Lib "CDIntf" (ByVal hPrinter As Long,
ByVal Directory As String) As Long
Declare Function SetDefaultFileName Lib "CDIntf" (ByVal hPrinter As Long,
ByVal FileName As String) As Long
Declare Function SetFileNameOptions Lib "CDIntf" (ByVal hPrinter As Long,
ByVal Options As Integer) As Long

Declare Function SetResolution Lib "CDIntf" (ByVal hPrinter As Long, ByVal
Resolution As Long) As Long
Declare Function GetResolution Lib "CDIntf" (ByVal hPrinter As Long) As Long

Declare Function SetBookmark Lib "CDIntf" (ByVal hDC As Long, ByVal Parent As
Long, ByVal Title As String) As Long

Declare Function SetDefaultConfig Lib "CDIntf" (ByVal hPrinter As Long) As
Long
Declare Function SetDefaultPrinter Lib "CDIntf" (ByVal hPrinter As Long) As
Long
```

```
Const NoPrompt As Integer = 1    ' do not prompt for file name
Const UseFileName As Integer = 2 ' use file name set by SetDefaultFileName
else use document name
Const Concatenate As Integer = 4 ' concatenate files, do not overwrite
Const DisableCompression As Integer = 8
                                 ' disable page content compression
Const EmbedFonts = 16            ' embed fonts used in the input document
Const BroadcastMessages = 32     ' enable broadcasting of PDF events


Private Sub PrintToPDF_Click()

    ' initialize PDF printer and set it as default
    ' this is for the standard version
    pdf = DriverInit("PDF Compatible Printer Driver")
    ' this adds a printer named My PDF Printer
    ' pdf = PDFDriverInit("My PDF Printer")

    If pdf = 0 Then
        MsgBox "Cannot initialize PDF printer"
        Exit Sub
    End If

    SetResolution pdf, 600
    SetDefaultConfig pdf        ' set 600 DPI as default for all printouts

    SetDefaultPrinter pdf       ' set this printer as default

    SetDefaultFileName pdf, "c:\test.pdf"
    SetFileNameOptions pdf, NoPrompt + UseFileName

    ' Print something here

    ' draw some text
    Printer.CurrentX = 200
    Printer.CurrentY = 400
    Printer.Print "Bookmark 1"
    ' set a bookmark on page 1
    SetBookmark Printer.hDC, 0, "Bookmark 1"

    ' go to next page
    Printer.NewPage

    ' draw some text and set a new bookmark
    Printer.CurrentX = 100
    Printer.CurrentY = 100
    Printer.Print "Bookmark 2"
    Parent = SetBookmark(Printer.hDC, 0, "Bookmark 2")

    ' set a bookmark as child of another bookmark
    Printer.CurrentX = 100
    Printer.CurrentY = 800
    Printer.Print "Submark 2-1"
    SetBookmark Printer.hDC, Parent, "Submark 2-1"

    Printer.EndDoc

    ' this will remove the printer if it had been installed with PDFDriverEnd
    ' will also restore default printer
    DriverEnd pdf
End Sub
```