Amyuni Document Converter

PDF – HTML – RTF – Excel® – JPeg Converters

Version 2.10e

Updated July 31st, 2003

Developer's Manual

Amyuni Consultants – Amyuni Technologies

http://www.amyuni.com

Contents

Legal Information

Important Note to developers:

The activation code that is provided to you by Amyuni should be kept confidential and not be revealed to end-users, even in this case where the developer's products are sub-licensed to other developers.

Acknowledgments

Special thanks to:

Jean-loup Gailly (jloup@gzip.org) and 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.

Introduction

This manual is a supplement to the Amyuni Document Converter series of products. These include the PDF, HTML, RTF, Excel and JPeg Converter products or any combination of these formats. Each of these products ships with a user's manual that describes the overall operation of the product. The developer should be familiar with the operation of the specific product that he or she is using before reading the developer's manual.

All Amyuni Document Converter products share the same interface DLL named the "Common Driver Interface". This interface resides in a DLL named CDINTF210.DLL that should be in the system or system32 directory.

CDINTF210 provides the developer with three calling conventions. Depending on their development platform and programming habits, developers might chose one these two interfaces:

- A standard DLL interface
- An ActiveX interface
- A .NET Managed Code Interface

Using the developer version of the Document Converter Products

The developer version of the Amyuni Document Converter products is a special version of these products that can be distributed with the developers' applications without paying any additional royalties to Amyuni Technologies.

By special version, we mean a version that:

- does not need to be pre-installed on the client system (although recommended)
- does not have any properties dialog box
- does not have any "File Save As" dialog box

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

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 as follows: acfpdf.dll, acfpdfu.dll, acfpdfui.dll, acfpdf.drv, cdintf.dll, install.exe, acfpdf.txt.

Step 2 – Initialise the Document Converter printer on the end-user's system
This can be done in two different ways:

1. (Recommended) By launching Install.exe when the main application is being installed. Install.exe should be launched followed by a printer name specific to the developer's company or application. It can also be followed by the –s switch for silent mode, and by the license and activation code. E.g.: Install -s "My Company Printer" -n "Evaluation Developer License" -c "07ABCDA0ABCDABCDABCD012301230123.....".
2. By calling PDFDriverInit at the initialisation of the application. The documentation for PDFDriverInit provides details about how to use this function to initialise the printer and activate it.

Step 3 – Copy CDIntf to the system directory
CDINTF.DLL should be copied to the system directory and renamed as CDINTF210.DLL. This is done automatically by Install.exe and is needed only if the developer has chosen some other method for installing the printer.

Step 4 – Register the CDIntf ActiveX
When using the ActiveX interface, the ActiveX control should be registered in the system by calling: REGSVR32 CDINTF210.DLL, from the system directory. CDIntf can be used through the DLL convention without the need for registering or creating ActiveXs.

Step 5 – 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 DriverInit followed by the printer name. When using method 2 outlined above to install the printer, DriverInit is replaced with PDFDriverInit. Note that all the functions PDFDriverInit, HTMLDriverInit and RTFDriverInit are exactly the same in version 2.1 of the product.

Step 6 – 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, JPEG or Excel file, you need to set up the output file name using SetDefaultFileName, the file generation options SetFileNameOptions(NoPrompt + UseFileName + …) and print to the "My Company Printer" as you would do when printing to any other printer. The developer can be in one of three situations:

a. The developer licensed a product that generates only one format, e.g. PDF only or RTF only. In this case, the printer will generate the right format without any specific option to set.
b. The developer licensed a product that generates multiple formats including PDF, e.g. PDF and RTF. In this case, to export to either RTF, HTML, JPEG or Excel, the corresponding option should be set in the call to SetFileNameOptions. The PDF file will be generated in all cases and should be deleted by the developer if not needed. There is no method to have the Converter generate an RTF/HTML/JPEG/Excel file only without generating PDF.
c. The developer licensed a product that generates multiple formats excluding PDF, e.g. RTF and HTML. In this case, to export to either RTF, HTML, the corresponding option should be set in the call to SetFileNameOptions. A temporary file will be generated in all cases and will be deleted by the printer when the print job is finished.

Step 7 – Restore the printer to its previous setting
When printing is over, the developer needs to call SetFileNameOptions(0) to prevent other applications or users from overwriting the file that has just been generated from the application.

Step 8 – Uninitialise the printer before exiting
Before exiting the application, the DriverEnd function should be called. This function will remove the printer if installed using PDFDriverInit, otherwise it will simply disconnect from the printer.

Important Note to developers:
To avoid confusion with other applications and with the single-user versions of the Document Converter products, developers are required to use a printer name specific to their application or company. Using the default printer name of "Amyuni Document Converter" is not recommended.

The activation code that is provided to you by Amyuni should be kept confidential and not be revealed to end-users, even in this case where the developer's products are sub-licensed to other developers.

DLL Interface

Before using the DLL interface of CDINTF210, the C or C++ developer might need to download the header and library files from:
http://www.amyuni.com/downloads/cdintf210.zip

The zip file also contains the latest version of CDIntf210.DLL.

The library file is compatible with Visual Studio 6. When using other compilers such as Borland C++, the developer needs to import the library from the DLL instead of using the library from our downloads section.

Visual Basic users can import the included CDIntf210.txt file to get all the constant and function declarations of CDIntf210. VB users are encouraged however to use the ActiveX interface as it remains more versatile and easier to use from within VB.

Printer Installation and Activation

```
DriverInit
PDFDriverInit, HTMLDriverInit, RTFDriverInit
DriverEnd
SetDefaultPrinter
RestoreDefaultPrinter
EnablePrinter
SetPrinterAttributes, GetPrinterAttributes
SetPrinterLanguage, GetPrinterLanguage
```

DriverInit Function

The DriverInit function initializes the library for use with an already installed printer. DriverInit cannot install a new printer and will fail if the printer does not already exist.

Syntax

```
HANDLE DriverInit( LPCSTR szPrinter  );
```

Parameters

szPrinter

[in] Name of the printer as it shows in the printers control panel.

Return Value

If the function succeeds, the return value is an internal handle to the printer. The handle is used in most other functions of CDINTF. If the function fails, the return value is NULL. To get extended error information, call GetLastErrorMsg.

Remarks

This function will only fail if the printer does not exist. There are otherwise no known instances where the function will fail.

The hPrinter handle returned by this function is not to be confused with the hPrinter handle returned by the Windows API OpenPrinter and CreatePrinter functions.

Example

```
HANDLE    hPrinter = DriverInit("Amyuni PDF Converter")
if ( NULL == hPrinter )
{
    return FALSE;
}
```

PDFDriverInit, HTMLDriverInit, RTFDriverInit Functions

The PDFDriverInit, HTMLDriverInit and RTFDriverInit functions can be used to create a new printer when the application is launched and remove it when the application is closed. These three functions are aliases the same internal function. They have been kept in version 2.1 for compatibility with previous versions. Any one of these can be used with any Amyuni Document Converter product with exactly the same result.

Syntax

```
HANDLE PDFDriverInit( LPCSTR szPrinter );
HANDLE HTMLDriverInit( LPCSTR szPrinter );
HANDLE RTFDriverInit( LPCSTR szPrinter );
```

Parameters

szPrinter

[in] Name of the printer as it shows in the printers control panel.

Return Value

If the function succeeds, the return value is an internal handle to the printer. The handle is used in most other functions of CDINTF. If the function fails, the return value is NULL. To get extended error information, call GetLastErrorMsg.

Remarks

This function will first attempt to connect to an existing printer. If it does not find any printer with the name provided by szPrinter, it will attempt to install a new printer. The function fails if it cannot find a printer with the specified name and cannot install a new printer. The printer referenced to by szPrinter is removed when the application exits or the DriverEnd function is called.

Installing or removing a printer requires administrative rights under Windows NT/2000/XP.

The hPrinter handle returned by this function is not to be confused with the hPrinter handle returned by the Windows API OpenPrinter and CreatePrinter functions.

Example

```
HANDLE    hPrinter = HTMLDriverInit(“My Application HTML Converter”)
if ( NULL == hPrinter )
{
    return FALSE;
}
```

DriverEnd Function

The DriverEnd function closes the printer handle created by one of the DriverInit functions and frees any internally allocated memory. If the printer was installed using the PDF, HTML or RTFDriverInit functions, it will be removed by the call to DriverEnd.

Syntax

```
void DriverEnd( HANDLE hPrinter  );
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

Return Value

None.

Remarks

This function will simply detach from an existing printer if the handle was created using DriverInit, or attempt to remove the printer if the handle was created using PDF, HTML or RTFDriverInit.

Example

```
BOOL CDLLTestApp::InitInstance()
{
  // create and initialize a new printer
  m_converter = PDFDriverInit("DLLTestApp Document Converter");
  if ( NULL == m_converter )
  {
          // cannot initialize new printer, get the error message from CDINTF
          CHAR    buf[MAX_BUF];
          GetLastErrorMsg( buf, sizeof(buf) );
          // CString will convert error message to Unicode if necessary
          ::MessageBox( NULL, CString(buf), _T("Error initializing printer"),
                                      MB_OK | MB_ICONWARNING );
          return FALSE;
  }

  // set Amyuni Document Converter as system default printer
  SetDefaultPrinter( m_converter );

  // Standard initialization
  ...
}

int CDLLTestApp::ExitInstance()
{
  // close printer handle; this will also restore the default printer
  if ( NULL != m_converter )
  {
          DriverEnd( m_converter );
  }

  return CWinApp::ExitInstance();
}
```

SetDefaultPrinter Function

The SetDefaultPrinter function sets the system default printer to the one initialized by the DriverInit functions.

Syntax

```
long SetDefaultPrinter( HANDLE hPrinter  );
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

Return Value

If the system default printer was modified, this function returns 1. It returns 0 if the default printer was not modified. A return value of 0 usually indicates that the printer was already the system default printer before the call to SetDefaultPrinter.

Remarks

Some applications require that the printer be set as default before being able to print to that printer. The default printer that was set before calling this function is restored in one of three situations:

- The function RestoreDefaultPrinter is called
- DriverEnd is called
- The calling application is closed

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

Modifying the system default printer too frequently is a source of numerous printing errors. Developers are encouraged to use methods supplied by their application to specify the destination printer rather than calling this function.

Example

```
// create and initialize a new printer
m_converter = PDFDriverInit("DLLTestApp Document Converter");
if ( m_converter )
{
    // set Amyuni Document Converter as system default printer
    SetDefaultPrinter( m_converter );
}
```

RestoreDefaultPrinter Function

The RestoreDefaultPrinter function resets the system default printer to the printer that was the default before the call to SetDefaultPrinter.

Syntax

```
long RestoreDefaultPrinter( HANDLE hPrinter  );
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

Return Value

If the system default printer was modified, this function returns 1. It returns 0 if the default printer was not modified. A return value of 0 usually indicates that the default printer was not modified by a call to SetDefaultPrinter.

Remarks

This function is called automatically when DriverEnd is called.

Example

```
// restore the previous default printer and close printer handle
if ( NULL != m_converter )
{
        RestoreDefaultPrinter( m_converter );
        DriverEnd( m_converter );
}
```

EnablePrinter Function

The EnablePrinter function can be used to install the permanent license and activation code for the product. The license and activation code define which features of the product are available to the users depending on the product they pruchased. For example, users of the PDF Converter will have a different activation code than users of the mixed PDF/RTF/HTML/Excel Converter product. EnablePrinter is also used in the developer versions of the products to activate the printer before every printout.

Syntax

```
long EnablePrinter(HANDLE hPrinter, LPCTSTR szCompany, LPCTSTR szCode);
```

Parameters

hPrinter
: [in] Handle to printer returned by one of the DriverInit functions.

szCompany
: [in] Name of the company or private user having licensed the product.

szLicKey
: [in] License key provided by Amyuni Technologies when downloading or purchasing a product.

Return Value

The return value is False if the printer handle is invalid, True otherwise.

Remarks

This function is usually called after successful installation of the product. End-users would usually enter their license information through the printer configuration tab, whereas developers would call this function from their product installation routine. The default install.exe that is provided with the products can also be used to set the license information by adding the following switches to the command line:

-n CompanyName -c LicenseKey

Calling the EnablePrinter function after installing the printer requires administrative rights under Windows NT/2000/XP, calling the same function before every printout does not require administrative rights and can be used with users having limited priveleges on the system.

Note to Developers

In addition to calling EnablePrinter after installing the printer, this function should also be called right before printing any document. The printer will otherwise be disabled. If the developers have full control of the printing process, they can call this function right before calling their printing function. If the application is an archiving or document management application that runs in the background and waits for other documents to be printed, this function can be called from the EnabledPre event that is fired by the printer. A detailed sample is provided under the Printer Driver Events section of this manual.

Example

```
BOOL CDLLTestApp::InitInstance()
{
  // create and initialize a new printer
  m_converter = PDFDriverInit("DLLTestApp Document Converter");
  if ( NULL == m_converter )
  {
          // cannot initialize new printer, get the error message from CDINTF
          CHAR    buf[MAX_BUF];
          GetLastErrorMsg( buf, sizeof(buf) );
          // CString will convert error message to Unicode if necessary
          ::MessageBox( NULL, CString(buf), _T("Error initializing printer"),
                                      MB_OK | MB_ICONWARNING );
          return FALSE;
  }

  // set the license information
  EnablePrinter( m_converter, "Evaluation Version Developer",
  "07EFCDAB0100010024CE83E1E8DC2244455F2F2C87EFC6FFF82B2F90206F7EEE87AE0751CAECA86FED0207C
D295E03629A5726433B6E3BE1766876E2B5237F8F5" );
```

```
  // set Amyuni Document Converter as system default printer
  SetDefaultPrinter( m_converter );

  // Standard initialization
  ...
}

void CDLLTestDlg::OnPrint()
{
  HFONT    font, oldFont;
  HDC              dc;
  DOCINFO  di;

  // create a printer device context
  dc = CreateDC( "winspool", theApp.m_szPrinter, NULL, NULL );
  if ( NULL == dc )
  {
          ASSERT( FALSE );
          return;
  }

  // init the DOCINFO structure, set the output file name
  memset( di, 0, sizeof(di) );
  di.cbSize = sizeof( di );
  di.lpszDocName = "Test document";
  di.lpszOutput = "c:\\test.pdf";

  // activate printer before starting to print
  EnablePrinter( theApp.m_converter, "Evaluation Version Developer",
  "07EFCDAB01000100584F829683E1E8DC2244455F2F2C87EFC6FFF82B2F90206F7EEE87AE0751CAECA86FED02
07CD295E03629A5726433B6E3BE1766876E2B5237F8F5" );

  // start printing
  StartDoc( dc, &di );
  StartPage( dc );
  MoveToEx( dc, 100, 100, NULL );
  LineTo( dc, 400, 100 );
  MoveToEx( dc, 100, 100, NULL );
  LineTo( dc, 100, 400 );

  font = CreateFont( -24, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, _T("Verdana") );
  oldFont = (HFONT)SelectObject( dc, font );
  TextOut( dc, 100, 100, _T("Hi There"), 8 );
  if ( oldFont ) SelectObject( dc, oldFont );
  DeleteObject( font );

  font = CreateFont( -24, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, _T("Wingdings") );
  oldFont = (HFONT)SelectObject( dc, font );
  TextOut( dc, 100, 200, _T("Hi There"), 8 );
  if ( oldFont ) SelectObject( dc, oldFont );
  DeleteObject( font );

  EndPage( dc );
  EndDoc( dc );

  DeleteDC( dc );

  // printing ended, no need to disactivate printer
  // the printer will be disactivated automatically
}
```

SetPrinterAttributes, GetPrinterAttributes Functions

The SetPrinterAttributes and GetPrinterAttributes functions can be used to modify or read the default attributes of an installed printer.

Syntax

```
long SetPrinterAttributes( HANDLE hPrinter, DWORD dwAttributes );
DWORD GetPrinterAttributes( HANDLE hPrinter );
```

Parameters

hPrinter

[in] Handle to printer returned by one of the DriverInit functions.

dwAttributes

[in] Printer attributes as defined by the Windows® operating system.

Return Value

SetPrinterAttributes returns 1 if successful, 0 otherwise. GetPrinterAttributes returns the current attributes of the printer.

Remarks

Modifying the printer attributes requires administrative rights under Windows NT/2000/XP.

The list of attributes that can be set for a printer are defined in the MSDN documentation under the SetPrinter function.

Example

SetPrinterLanguage, GetPrinterLanguage Functions

The SetPrinterLanguage and GetPrinterLanguage functions can be used to modify or read the language used in the user interface of the printer.

Syntax

```
void SetPrinterLanguage( HANDLE hPrinter, long nLang );
long GetPrinterLanguage( HANDLE hPrinter );
```

Parameters

hPrinter

[in] Handle to printer returned by one of the DriverInit functions.

nLang

[in] User-interface language Id. nLang can be one of the following values:

0	default language set by the license key
1	English
2	French
3	German

Return Value

GetPrinterLanguage returns the current language Id.

Remarks

The license key provided with the product contains the default user-interface language that will be used after installing the product. SetPrinterLanguage can be used to modify the default value. The language can also be modified by the user from the printer configuration dialog-box.

Example

```
  // create and initialize a new printer
  m_szPrinter = _T("DLLTestApp Document Converter");
  m_converter = PDFDriverInit( (LPCSTR)m_szPrinter );

  // set the license information
  if ( !EnablePrinter( m_converter, "Evaluation Version Developer",
"07EFCDAB01000100584F8296873E1E8DC2244455F2F2C87EFC6FFF82B2F90206F7EEE87AE0751CAECA86FED02
07CD295E03629A5726433B6E3BE1766876E2B5237F8F5" ) )
  {
          ASSERT( FALSE );
          return FALSE;
  }

  // set language to French
  SetPrinterLanguage( m_converter, 2L );
```

Printer Configuration

```
SetDefaultDirectory
SetDefaultFileName
SetFileNameOptions

SetPaperSize, GetPaperSize
SetPaperWidth, GetPaperWidth
SetPaperLength, GetPaperLength
SetOrientation, GetOrientation
SetResolution, GetResolution
SetJPEGCompression, GetJPEGCompression
SetJPegLevel, GetJPegLevel
SetFontEmbedding, GetFontEmbedding
SetHorizontalMargin, GetHorizontalMargin, SetVerticalMargin, GetVerticalMargin
SetImageOptions, GetImageOptions
SetSimPostscript, GetSimPostscript

SetWatermark

SetPrinterParamStr, GetPrinterParamStr, SetPrinterParamInt, GetPrinterParamInt

SetDefaultConfig, SetDefaultConfigEx
```

SetDefaultDirectory Function

The SetDefaultDirectory function defines the default directory used to store the files generated by the Converter products.

Syntax

```
long SetDefaultDirectory( HANDLE hPrinter, LPCSTR szDir  );
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

szDir

[in] Default directory used to store output files.

Return Value

This function returns 1 if successful, 0 otherwise.

Remarks

The directory can be either a local or a network directory. In both cases, the directory should exist and the users have right to write to this directory. The printer driver will not attempt to create the directory if it does not exist.

This setting is only used in the case where the output file name is defined by the printer driver and not by the user or developer, i.e. when the FileNameOptions contain the NoPrompt but not the UseFileName options.

Example

```
HANDLE    hPrinter = DriverInit( "Amyuni Document Converter" );

If ( hPrinter )
{
  SetFileNameOptions( NoPrompt );           // output file name defined by the printer
  SetDefaultDirectory( "c:\\temp" );        // all files will be placed in c:\temp
}
```

SetDefaultFileName Function

The SetDefaultFileName function defines the destination file name for the PDF, RTF and Excel Converter products, or the root of the output file names for the DHTML and JPeg Converter products.

Syntax

```
long SetDefaultFileName( HANDLE hPrinter, LPCSTR szFile  );
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

szFile

[in] Output file name.

Return Value

This function returns 1 if successful, 0 otherwise.

Remarks

This setting is only used in the case where the output file name is defined by the developer and not by the user or the printer driver, i.e. when the FileNameOptions contain the NoPrompt and the UseFileName options.

The szFile parameter should contain both the destination directory and file name. The directory can be either a local or a network directory. In both cases, the directory should exist and the users have right to write to this directory. The printer driver will not attempt to create the directory if it does not exist.

In this case of the PDF, RTF and Excel converters, only one file is generated and the name of this file defined by the SetDefaultFileName call. In the case of the DHTML and JPeg Converters, multiple files can be generated from a single printout; in this case SetDefaultFileName defines the name of the main file, the other files being generated from the main file by appending numerical Ids at the end of each file.

Example

```
HANDLE    hPrinter = DriverInit( "Amyuni DHTML Converter" );

If ( hPrinter )
{
  SetFileNameOptions( NoPrompt + UseFileName );      // output file name defined by the
                                                     // calling application
  SetDefaultFileName( "c:\\temp\\test.htm" );        // the output files will be placed in
                                                     // c:\temp and named test1, test2, …
}
```

SetFileNameOptions Function

The SetFileNameOptions function defines a number of options used in the generation of the PDF, HTML, RTF, Excel and JPeg Converter products.

Syntax

```
long SetFileNameOptions( HANDLE hPrinter, int nOptions  );
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

nOptions

[in] Combination of options as defined below.

Return Value

This function returns 1 if successful, 0 otherwise.

Remarks

Options supported by each product:

Option	Value (Hex)	Description	PDF	HTM	RTF	JPG	XL
NoPrompt	1	Prevents the display of the file name dialog box	x	x	x	x	x
UseFileName	2	Use the file name set by SetDefaultFileName as the output file name	x	x	x	x	x
Concatenate	4	Concatenate with existing file instead of overwriting. This is useful only if the NoPrompt option is set	x	x	x	x	x
DisableCompression	8	Disable deflate (zip) compression of the page's content	x				
EmbedFonts	10	Enable embedding of fonts used in the source document	x				
BroadcastMessages	20	Send notification messages for document generation progress to all running application	x	x	x	x	x
PrintWatermark	40	Watermarks are printed on all printed pages	x	x	x	x	x
MultilingualSupport	80	Add supports for international character sets	x			x	
EncryptDocument	100	Encrypt resulting document	x				
FullEmbed	200	Embed full fonts as opposed to embedding the fonts partially	x				
UseTcpIpServer	400	Reserved					
SendByEmail	800	Send the document by email	x		x	x	x
ConfirmOverwrite	1000	If the file exists, confirm before overwriting	x	x	x	x	x
AppendExisting	2000	If the file exists, append to existing file	x	x	x	x	x
AddDateTime	3000	If the file exists, add date and time to file name	x	x	x	x	x
AddIdNumber	4000	If the file exists, add ID number to file name	x	x	x	x	x
LinearizeForWeb	8000	Activate web optimisation (Linearization) of PDF document	x				
PostProcessing	10000	Post process file using specified application. The application's full path should be in the registry	x				
JpegLevelLow	20000	Low quality JPeg compression of 24-bit images. This is equivalent to level 2 in the user interface.	x	x	x	x	
JpegLevelMedium	40000	Medium quality JPeg compression of 24-bit images; this is equivalent to level 7 in the user interface	x	x	x	x	
JpegLevelHigh	60000	High quality JPeg compression of 24-bit images; this is equivalent to level 9 in the user interface	x	x	x	x	
Colors2GrayScale	80000	Replaces all colors by an equivalent gray scale value	x	x	x	x	
ConvertHyperlinks	100000	Convert text beginning with http or www to a hyperlink	x	x			
EmbedStandardFonts	200000	Embed standard fonts such as Arial, Times, ...	x				
EmbedLicensedFonts	400000	Embed fonts requiring a license	x				
Color256Compression	800000	Activate 256 color compression	x	x	x	x	
EmbedSimulatedFonts	1000000	Embed Italic or Bold fonts that do not have an associated font file but are simulated by the system	x				
SendToCreator	2000000	Send the PDF data directly to the Amyuni PDF Creator product instead of saving to disk	x				

ExportToHTML	4000000	Export document to HTML format		x			
ExportToRTF	8000000	Export document to RTF format			x		
ExportToJPEG	10000000	Export document to JPEG format				x	
CCITTCompression	20000000	Activate CCITT Fax Level 4 compression for B&W images	x				
EncryptDocument128	40000000	Use 128 bits encryption compatible with Adobe Acrobat® 5	x				
AutoImageCompression	80000000	Use automatic image compression, i.e. the best compression option for each image in a document	x	x	x	x	

The export to Excel is configured differently from the RTF, HTML or JPeg exports. To export to Excel, the developer should use the SetPrinterParam functions in addition to SetFileNameOptions.

Example

```
HANDLE    hPrinter = DriverInit( "Amyuni RTF Converter" );

If ( hPrinter )
{
  SetFileNameOptions( NoPrompt + UseFileName );      // output file name defined by the
                                                     // calling application
  SetDefaultFileName( "c:\\temp\\test.rtf" );        // the output file will be named
                                                     // c:\temp\test.rtf
}
```

SetPaperSize, GetPaperSize Functions

The SetPaperSize and GetPaperSize functions are used to define the default output paper size. The default paper size is usually used by applications when creating a new document.

Syntax

```
long SetPaperSize( HANDLE hPrinter, long nPaperSize );
long GetPaperSize( HANDLE hPrinter );
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

nPaperSize

[in] Paper size identifier as defined by the Windows® operating system. The default value is either Letter or A4 depending on the country where the product is used.

Return Value

SetPaperSize returns 1 if successful, 0 otherwise. GetPaperSize returns the current value for the specified printer.

Remarks

Sample PaperSize 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

These functions can be used in two situations:

1. The developer needs to modify the default values for the printer; in this case SetDefaultConfig should be called after modifying this setting to set the value as default for all applications. Application developers should not call this function for every printout but only after printer initialization.
2. The printer device context is created using the function CDICreateDC. This function uses the settings provided by these functions and there is no need to call SetDefaultConfig.

Example

```
HANDLE    hPrinter;

hPrinter = PDFDriverInit( "My Application Converter" );      // create a new printer
If ( hPrinter )
{
  SetPaperSize( hPrinter, 5 );              // set default paper size to Legal
  // set other printer defaults here
  ...
  SetDefaultConfig( hPrinter );             // make the value default
}
```

SetPaperWidth, GetPaperWidth, SetPaperLength, GetPaperLength Functions

The SetPaperWidth, GetPaperWidth, SetPaperLength and GetPaperLength functions are used to define custom output paper sizes.

Syntax

```
long SetPaperWidth( HANDLE hPrinter, long nPaperWidth );
long GetPaperWidth( HANDLE hPrinter );
long SetPaperLength( HANDLE hPrinter, long nPaperLength );
long GetPaperLength( HANDLE hPrinter );
```

Parameters

hPrinter
: [in] Handle to printer returned by any of the DriverInit function calls.

nPaperWidth
: [in] Paper width in 10^{th} of a millimeter. The default value is 1000 or 10 centimeters.

nPaperLength
: [in] Paper length in 10^{th} of a millimeter. The default value is 1000 or 10 centimeters.

Return Value

SetPaperWidth and SetPaperLength return 1 if successful, 0 otherwise. GetPaperWidth and GetPaperLength return the current value for the specified printer.

Remarks

These functions can be used in two situations:

1. The developer needs to modify the default values for the printer; in this case SetDefaultConfig should be called after modifying these settings to set the value as default for all applications. Application developers should not call this function for every printout but only after printer initialization.
2. The printer device context is created using the function CDICreateDC. This function uses the settings provided by these functions and there is no need to call SetDefaultConfig.

These functions automatically modify the PaperSize value to 256 for 'Custom'.

Example

```
HANDLE    hPrinter;

hPrinter = PDFDriverInit( "My Application Converter" );      // create a new printer
If ( hPrinter )
{
  SetPaperWidth( hPrinter, 2000 );          // set default paper width to 20 centimeters
  SetPaperLength( hPrinter, 3000 );         // set default paper length to 30 centimeters
  // set other printer defaults here
  ...
  SetDefaultConfig( hPrinter );             // make the values default
}
```

SetOrientation, GetOrientation Functions

The SetOrientation and GetOrientation functions are used to define the default paper orientation. The default orientation is usually used by application when creating a new document.

Syntax

```
long SetOrientation( HANDLE hPrinter, long nOrientation );
long GetOrientation( HANDLE hPrinter );
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

nOrientation

[in] Paper orientation.

Return Value

SetOrientation returns 1 if successful, 0 otherwise. GetOrientation returns the current value for the specified printer.

Remarks

Orientation value:

Orientation	Orientation value
Portrait	1
Landscape	2

These functions can be used in two situations:

1. The developer needs to modify the default values for the printer; in this case SetDefaultConfig should be called after modifying these settings to set the value as default for all applications. Application developers should not call this function for every printout but only after printer initialization.
2. The printer device context is created using the function CDICreateDC. This function uses the settings provided by these functions and there is no need to call SetDefaultConfig.

Example

```
HANDLE    hPrinter;

hPrinter = PDFDriverInit( "My Application Converter" );      // create a new printer
If ( hPrinter )
{
  SetOrientation( hPrinter, 2 );            // set default paper orientation to Landscape
  SetDefaultConfig( hPrinter );             // make the values default
}
```

SetResolution, GetResolution Functions

The SetResolution and GetResolution functions are used to define the default printer resolution. This value is mainly used when outputting images but has an effect also on the quality of text output.

Syntax

```
long SetResolution( HANDLE hPrinter, long nResolution );
long GetResolution( HANDLE hPrinter );
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

nResolution

[in] Printer resolution.

Return Value

SetResolution returns 1 if successful, 0 otherwise. GetResolution returns the current value for the specified printer.

Remarks

Resolution value:

Orientation	Orientation value
72 Dots Per Inch	72
150 Dots Per Inch	150
300 Dots Per Inch	300
600 Dots Per Inch	600
1200 Dots Per Inch	1200

These functions can be used in two situations:

1. The developer needs to modify the default values for the printer; in this case SetDefaultConfig should be called after modifying these settings to set the value as default for all applications. Application developers should not call this function for every printout but only after printer initialization.
2. The printer device context is created using the function CDICreateDC. This function uses the settings provided by these functions and there is no need to call SetDefaultConfig.

Example

```
HANDLE    hPrinter;

hPrinter = PDFDriverInit( "My Application Converter" );      // create a new printer
If ( hPrinter )
{
  SetResolution( hPrinter, 600 );           // set default resolution to 600 DPI
  SetDefaultConfig( hPrinter );             // make the values default
}
```

SetJPEGCompression, GetJPEGCompression Functions

The SetJPEGCompression and GetJPEGCompression functions are used to activate or disactivate the JPeg compression option for 24 bits images. JPeg compression heavily reduces the size of documents containing real life images but has a slight effect on image quality.

Syntax

```
long SetJPEGCompression( HANDLE hPrinter, long bCompression );
long GetJPEGCompression( HANDLE hPrinter );
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

bCompression

[in] This parameter should be True or 1 to set JPeg compression, 0 otherwise.

Return Value

SetJPEGCompression returns 1 if successful, 0 otherwise. GetJPEGCompression returns the current value for the specified printer.

Remarks

These functions can be used in two situations:

1. The developer needs to modify the default values for the printer; in this case SetDefaultConfig should be called after modifying these settings to set the value as default for all applications. Application developers should not call this function for every printout but only after printer initialization.
2. The printer device context is created using the function CDICreateDC. This function uses the settings provided by these functions and there is no need to call SetDefaultConfig.

When the JPeg compression option and JPeg level need to be changed for a specific printout and not set as default for all printouts, the SetFileNameOptions function provides a more efficient way to set JPeg compression.

Example

```
HANDLE    hPrinter;

hPrinter = PDFDriverInit( "My Application Converter" );      // create a new printer
If ( hPrinter )
{
  SetJPEGCompression( hPrinter, TRUE );     // activate JPEG compression
  SetJPegLevel( hPrinter, 3 );              // low quality but high compression level
  SetDefaultConfig( hPrinter );             // make the values default
}
```

SetJPegLevel, GetJPegLevel Functions

The SetJPegLevel and GetJPegLevel functions are used to set or read the level of JPeg compression for 24 bits images. JPeg compression heavily reduces the size of documents containing real life images but has a slight effect on image quality. The level of compression from 1 to 9 determines if the printer should output highly compressed low quality images or good quality images with reduced compression.

Syntax

```
long SetJPegLevel( HANDLE hPrinter, long nLevel );
long GetJPegLevel( HANDLE hPrinter );
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

nLevel

[in] This parameter can vary from 1 to 9, the default value is 7 for good quality with medium compression.

Return Value

SetJPegLevel returns 1 if successful, 0 otherwise. GetJPegLevel returns the current value for the specified printer.

Remarks

These functions can be used in two situations:

1. The developer needs to modify the default values for the printer; in this case SetDefaultConfig should be called after modifying these settings to set the value as default for all applications. Application developers should not call this function for every printout but only after printer initialization.
2. The printer device context is created using the function CDICreateDC. This function uses the settings provided by these functions and there is no need to call SetDefaultConfig.

When the JPeg compression option and JPeg level need to be changed for a specific printout and not set as default for all printouts, the SetFileNameOptions function provides a more efficient way to set JPeg compression.

Sample JPeg Level values:

JPeg Level	JPeg level value
Low quality, High compression	3
Good quality, Good compression	7
High quality, Low compression	9

Example

```
HANDLE    hPrinter;

hPrinter = PDFDriverInit( "My Application Converter" );      // create a new printer
If ( hPrinter )
{
  SetJPEGCompression( hPrinter, TRUE );     // activate JPEG compression
  SetJPegLevel( hPrinter, 3 );              // low quality but high compression level
  SetDefaultConfig( hPrinter );             // make the values default
}
```

SetFontEmbedding, GetFontEmbedding Functions

The SetFontEmbedding and GetFontEmbedding functions are used to activate or disactivate the TrueType® font embedding features of the PDF Converter product. These settings have no effect on the other Converter products. Font embedding can be used in the PDF files to ensure portability among various systems.

Syntax

```
long SetFontEmbedding( HANDLE hPrinter, long bEmbedding );
long GetFontEmbedding( HANDLE hPrinter );
```

Parameters

hPrinter
: [in] Handle to printer returned by any of the DriverInit function calls.

bEmbedding
: [in] This parameter should be True or 1 to set font embedding, 0 otherwise.

Return Value

SetFontEmbedding returns 1 if successful, 0 otherwise. GetFontEmbedding returns the current value for the specified printer.

Remarks

These functions can be used in two situations:

1. The developer needs to modify the default values for the printer; in this case SetDefaultConfig should be called after modifying these settings to set the value as default for all applications. Application developers should not call this function for every printout but only after printer initialization.
2. The printer device context is created using the function CDICreateDC. This function uses the settings provided by these functions and there is no need to call SetDefaultConfig.

When the font embedding option needs to be changed for a specific printout and not set as default for all printouts, the SetFileNameOptions function provides a more efficient way to set font embedding.

Example

```
HANDLE    hPrinter;

hPrinter = PDFDriverInit( "My Application Converter" );      // create a new printer
If ( hPrinter )
{
  SetFontEmbedding( hPrinter, TRUE );        // embed TrueType fonts with the document
  SetJPEGCompression( hPrinter, TRUE );      // activate JPEG compression
  SetJPegLevel( hPrinter, 3 );               // low quality but high compression level
  SetDefaultConfig( hPrinter );              // make the values default
}
```

SetHorizontalMargin, GetHorizontalMargin, SetVerticalMargin, GetVerticalMargin Functions

The SetHorizontalMargin, GetHorizontalMargin, SetVerticalMargin and GetVerticalMargin functions are used to set or read the minimum printer margins. Applications cannot print below the minimum margins defined by the printer. The Converter products being virtual printers, the minimum margins can be set to 0, this might cause some clipping if the generated document is later printed to a physical printer. The default value is set to 6 millimeters in order to accomodate most hardware printers available in the market.

Syntax

```
long SetHorizontalMargin( HANDLE hPrinter, long nMargin );
long GetHorizontalMargin( HANDLE hPrinter );
long SetVerticalMargin( HANDLE hPrinter, long nMargin );
long GetVerticalMargin( HANDLE hPrinter );
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

nMargin

[in] Minimum printer margin in 10^{th} of a millimeter unit.

Return Value

SetHorizontalMargin and SetVerticalMargin return 1 if successful, 0 otherwise. GetHorizontalMargin and GetVerticalMargin return the current value for the specified printer in 10^{th} of a millimeter. The default values for both vertical and horizontal margins is 6 mm, the return value should be 60 if these settings are not modified.

Remarks

These functions can be used in two situations:

1. The developer needs to modify the default values for the printer; in this case SetDefaultConfig should be called after modifying these settings to set the value as default for all applications. Application developers should not call this function for every printout but only after printer initialization.
2. The printer device context is created using the function CDICreateDC. This function uses the settings provided by these functions and there is no need to call SetDefaultConfig.

These settings define the minimum margin below which an application cannot print. Applications usually define their own margin settings and send a warning to the user whi attempts to set margins lower than the printer's minimum value.

Example

```
HANDLE    hPrinter;

hPrinter = PDFDriverInit( "My Application Converter" );      // create a new printer
If ( hPrinter )
{
  SetVerticalMargin( hPrinter, 0 );          // set minimum vertical margin to 0
  SetHorizontalMargin( hPrinter, 0 );        // set minimum horizontal margin to 0
  SetFontEmbedding( hPrinter, TRUE );        // embed TrueType fonts with the document
  SetJPEGCompression( hPrinter, TRUE );      // activate JPEG compression
  SetJPegLevel( hPrinter, 3 );               // low quality but high compression level
  SetDefaultConfig( hPrinter );              // make the values default
}
```

SetImageOptions, GetImageOptions

The SetImageOptions and GetImageOptions functions are used to set or read additional image conversion options.

Syntax

```
long GetImageOptions(HANDLE hPrinter);
void SetImageOptions(HANDLE hPrinter, long lImageOptions);
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

lImageOptions

[in] Additional options for image conversion.

Return Value

SetImageOptions returns 1 if successful, 0 otherwise. GetImageOptions returns the current value for the additional image options.

Remarks

Values for ImageOptions:

Description	Value
Remove Duplicate Images	1
Downsample high-resolution images	2

The options can be combined using the Addition or the OR operators.

The call should be followed by a call to SetDefaultConfig or SetDefaultConfigEx for these settings to take effect.

Example

```
HANDLE    hPrinter;

hPrinter = PDFDriverInit( "My Application Converter" );      // create a new printer
If ( hPrinter )
{
  SetVerticalMargin( hPrinter, 0 );          // set minimum vertical margin to 0
  SetHorizontalMargin( hPrinter, 0 );        // set minimum horizontal margin to 0
  SetImageOptions( hPrinter, 2 );            // downsample images to fit output resolution
  SetDefaultConfig( hPrinter );              // make the values default
}
```

SetSimPostscript, GetSimPostscript

The SetSimPostscript and GetSimPostscript functions are used to set or read the Postscript Simulation option in the printer driver. Some operations are not allowed on Postscript and PDF printers because on these printers the application cannot read from the destination device. Some applications query the printer and modify their drawing operations if the printer is a Postscript printer. This option is useful for printing Wordarts using Office XP.

Syntax

```
BOOL GetSimPostscript(HANDLE hIntf);
void SetSimPostscript(HANDLE hIntf, BOOL bSimPostscript);
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

bSimPostscript

[in] Flag that indicates if Postscript simulation is On or Off.

Return Value

SetSimPostscript returns 1 if successful, 0 otherwise. GetSimPostscript returns the current value for the Postscript Simulation option.

Remarks

The call should be followed by a call to SetDefaultConfig or SetDefaultConfigEx for the setting to take effect.

Example

```
HANDLE    hPrinter;

hPrinter = PDFDriverInit( "My Application Converter" );      // create a new printer
If ( hPrinter )
{
  SetVerticalMargin( hPrinter, 0 );          // set minimum vertical margin to 0
  SetHorizontalMargin( hPrinter, 0 );        // set minimum horizontal margin to 0
  SetSimPostscript( hPrinter, TRUE );        // simulate a Postscript printer
  SetDefaultConfig( hPrinter );              // make the values default
}
```

SetWatermark Function

The SetWatermark function is used to configure the printer to print a watermark message on all pages of a document. This method can be used to set only the simple text watermarks and cannot be used to set the watermark option to "External PDF File".

Syntax

```
long SetWatermark( HANDLE hPrinter, LPTSTR szWatermark, LPTSTR szFont, short fontSize,
                   short Orientation, COLORREF color, LONG xPos, LONG yPos, BOOL bForeground  );
```

Parameters

hPrinter
: [in] Handle to printer returned by any of the DriverInit function calls.

Watermark
: [in] Text to print on each page.

FontName
: [in] Font name used to print text.

FontSize
: [in] Font size in 0.1 inch units.

Orientation
: [in] Text orientation in 0.1 degree units.

Color
: [in] RGB value for watermark color.

HorzPos
: [in] Horizontal text position in 0.1 inch units. This a positive value measured from the left of the page and should take into account the minimum printer margin.

VertPos
: [in] Vertical text position in 0.1 inch units. This a negative value measured from the top of the page and should take into account the minimum printer margin.

Foreground
: [in] flag that indicates whether the watermark should be above or below the page content.

Return Value

SetWatermark returns 0 if successful, or a Windows specific error code if it fails.

Remarks

This method only sets the watermark parameters for the printer, to actually print watermarks on a document, the option PrintWatermark (Hex 40) should be added to the FileNameOptions property.

This method can be called anytime before or while printing a document to modify the watermarks settings.

To print watermark on specific pages only, e.g. the first page only, the printer events can be captured and the watermark parameters modified while the document is being printed.

Example

```
void CDLLTestAppDlg::OnPrintWord()
{
    // print an MS Word document using OLE Automation
    // also print a DRAFT watermark on the first page only of the document

    _Application   *wordApp = NULL;
    Documents      *documents = NULL;

    LPDISPATCH     pdisp;

    __try {
          // open MS Word
          wordApp = new _Application();
          wordApp->CreateDispatch( _T("Word.Application") );
```

```
            // open a Word document in Read-only mode
            pdisp = wordApp->GetDocuments();
            documents = new Documents( pdisp );
            documents->Open( COleVariant( _T("c:\\test.doc") ),         // file name
                                        COleVariant(),                  // no conversions
                                        COleVariant( (short)1 )   // read only
                                        );

            // set the ActivePrinter to ours
            wordApp->SetActivePrinter( theApp.m_szPrinter );

            // set watermark parameters
            SetWatermark( theApp.m_converter, " D  R  A  F  T ", "Verdana", 48, 450,
                                    0xFF00FF, 120, -120, TRUE );

            SetFileNameOptions( theApp.m_converter, NoPrompt + UseFileName +
                                    BroadcastMessages + PrintWatermark );

            SetDefaultFileName( theApp.m_converter, "c:\\test.pdf" ); // set output file

            // print the Word document without background printing
            wordApp->PrintOut( COleVariant( 0L ) );
    }
    __finally
    {
            // reset options after printing
            SetFileNameOptions( theApp.m_converter, 0 );

            // close MS Word
            if ( wordApp )
                    wordApp->Quit( COleVariant(), COleVariant(), COleVariant() );
            delete documents;
            delete wordApp;
    }
}

LONG CDLLTestAppDlg::OnPDFEvent( UINT wParam, LONG lParam )
{
// a PDF event has occured
//
  WORD    eventId;

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

  switch ( eventId )
  {
  case DOCUMENTEVENT_ENABLEDPRE:
            // activate printer before starting to print
            EnablePrinter( theApp.m_converter, "Evaluation Version Developer",
            "07EFCDAB01000100584F829697D0749DAF7E37EF1621AEBEBF2975B" );
            break;

  case DOCUMENTEVENT_ENDPAGE:
            // EndPage handled, remove watermarks
            SetFileNameOptions( theApp.m_converter, NoPrompt + UseFileName +
                                    BroadcastMessages );
            break;

  default:
            return DOCUMENTEVENT_UNSUPPORTED;
  }
  return DOCUMENTEVENT_SUCCESS;
}
```

SetPrinterParamStr, GetPrinterParamStr, SetPrinterParamInt, GetPrinterParamInt Functions

The SetPrinterParamStr, GetPrinterParamStr, SetPrinterParamInt and GetPrinterParamInt functions are used to set or read custom printer parameters. The printer parameters depend on each type of Document Converter Printer and can be either in string or long integer format.

Syntax

```
int SetPrinterParamStr( HANDLE hPrinter, LPCSTR szParam, LPCSTR szValue );
int GetPrinterParamStr( HANDLE hPrinter, LPCSTR szParam, LPSTR szValue, int nLen );
int SetPrinterParamInt( HANDLE hPrinter, LPCSTR szParam, long nValue );
long GetPrinterParamInt( HANDLE hPrinter, LPCSTR szParam );
```

Parameters

hPrinter
: [in] Handle to printer returned by any of the DriverInit function calls.

szParam
: [in] Name of parameter to change. The comments section contains a list of valid names.

szValue
: [in, out] String value of the parameter if the parameter is of type string.

nLen
: [in] Size of the szValue buffer where the string value should be returned.

nValue
: [in] Long integer value of the parameter if the parameter is of type integer.

Return Value

SetPrinterParamStr, GetPrinterParamStr and SetPrinterParamInt return 1 if successful, 0 otherwise. GetPrinterParamInt returns the value for the custom printer parameter, or 0 if the parameter is not found.

Remarks

For more details about the meaning of each of these parameters, refer to the user's manual for the product in question.

There is no need to call SetDefaultConfig after modiying these parameters as they are immediately taken into account by all applications.

Custom parameters for the RTF Converter:

Parameter	Type	Description	Values
RTF Format	Integer	Option for formatting the RTF output	0 – Advanced RTF 1 – Full RTF 2 – Formatted Text 3 – Non-formatted Text Only
Use Tabs	Integer	Option to replace tabs by spaces	0 – Do not replace 1 – Replace

Custom parameters for the DHTML Converter:

Parameter	Type	Description	Values
HTML MultiPage	Integer	Option for formatting the HTML output	1 – Use Layers 2 – Single Page HTML 3 – Multiple HTML files

Custom parameters for the JPEG Converter:

Parameter	Type	Description	Values
JPEG Resolution	Integer	Image resolution at which the document is converted to JPeg	0 – 75 DPI 1 – 150 DPI 2 – 300 DPI 3 – 600 DPI
JPEG Compression	Integer	JPeg compression level. The higher the level, the better the quality and larger the file size.	1 to 9

Custom parameters for the Excel Converter:

Parameter	Type	Description	Values
EXCEL MultiSheets	Integer	Excel output option	0 – No Excel output 16 – All document pages on a single sheet 17 – One excel sheet per document page

Example 1

```
HANDLE    hPrinter;

hPrinter = PDFDriverInit( "My HTML Converter" );     // create a new printer
if ( hPrinter )
{
  // create HTML document containing one layer for every page
  SetPrinterParamInt( hPrinter, "HTML MultiPage", 1 );
}
```

Example 2

```
HANDLE    hPrinter;

hPrinter = PDFDriverInit( "My RTF Converter" );      // create a new printer
if ( hPrinter )
{
  // create RTF document containing with objects not positioned in frames
  SetPrinterParamInt( hPrinter, "RTF Format", 1 );
}
```

Example 3

```
HANDLE    hPrinter;

hPrinter = PDFDriverInit( "My JPeg Converter" );     // create a new printer
if ( hPrinter )
{
  // create compact JPeg files at low resolution
  SetPrinterParamInt( hPrinter, "JPeg Resolution", 2 );      // 150 DPI
  SetPrinterParamInt( hPrinter, "JPeg Compression", 3 );     // high compression
}
```

Example 4

```
HANDLE    hPrinter;

hPrinter = PDFDriverInit( "My Excel Converter" );    // create a new printer
if ( hPrinter )
{
  // create an Excel file with one sheet per page
  SetPrinterParamInt( hPrinter, "Excel MultiSheets", 17 );
}
```

SetDefaultConfig, SetDefaultConfigEx Functions

The SetDefaultConfig and SetDefaultConfigEx functions are used set the current printer configuration as default for all applications. These calls are need when modifying some of the printer properties so that applications can take these properties into account. The document of all printer parameters or properties specifies when this call is needed.

Syntax

```
long SetDefaultConfig( HANDLE hPrinter );
long SetDefaultConfigEx( HANDLE hPrinter );
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

Return Value

SetDefaultConfig and SetDefaultConfigEx return 1 if successful, 0 otherwise.

Remarks

The SetDefaultConfig function posts a Windows message to all running applications, including the application that calls this function, to notify the applications that the printer settings have changed. The notification can take a few seconds if there are a number of applications running. SetDefaultConfigEx does the same as SetDefaultConfig but without the notification. It can be useful when modifying the printer settings when installing the application or before the application is launched.

Example

```
HANDLE    hPrinter;

hPrinter = PDFDriverInit( "My Application Converter" );      // create a new printer
If ( hPrinter )
{
  SetFontEmbedding( hPrinter, TRUE );       // embed TrueType fonts with the document
  SetJPEGCompression( hPrinter, TRUE );     // activate JPEG compression
  SetJPegLevel( hPrinter, 3 );              // low quality but high compression level
  SetDefaultConfig( hPrinter );             // make the values default
}
```

Email Fucntions

```
SetEmailFieldFrom
SetEmailFieldTo
SetEmailFieldCC
SetEmailFieldBCC
SetEmailSubject
SetEmailMessage
SetEmailPrompt
SetEmailOptions, GetEmailOptions

SetSmtpServer, SetSmtpPort

SendMail, SendMailW
SendSmtpMail, SendSmtpMailW
```

SetEmailFieldFrom, SetEmailFieldTo, SetEmailFieldCC, SetEmailFieldBCC, SetEmailSubject, SetEmailMessage, SetEmailPrompt Functions

The SetEmailFieldFrom, SetEmailFieldTo, SetEmailFieldCC, SetEmailFieldBCC, SetEmailSubject, SetEmailMessage and SetEmailPrompt functions are used to set the various email parameters that can be found in the Destination property tab of the Amyuni Converter products.

Syntax

```
int SetEmailFieldFrom(HANDLE hPrinter, LPCSTR szEmailFieldFrom);
int SetEmailFieldTo( HANDLE hPrinter, LPCSTR szEmailFieldTo);
int SetEmailFieldCC( HANDLE hPrinter, LPCSTR szEmailFieldCC);
int SetEmailFieldBCC( HANDLE hPrinter, LPCSTR szEmailFieldBCC);
int SetEmailSubject( HANDLE hPrinter, LPCSTR szEmailSubject);
int SetEmailMessage( HANDLE hPrinter, LPCSTR szEmailMessage);
int SetEmailPrompt( HANDLE hPrinter, BOOL bEmailPrompt);
```

Parameters

hPrinter
: [in] Handle to printer returned by any of the DriverInit function calls.

szEmailFieldFrom
: [in] Name of the email sender as it shows to the person receiving the email. This parameter is used only in the case of sending emails through SMTP.

szEmailFieldTo
: [in] Name and email address of the email destinator. Multiple addresses can be specified by separating them with semi-colons (;).

szEmailFieldCC
: [in] Name and email address of the email Carbon Copy list. Multiple addresses can be specified by separating them with semi-colons (;).

szEmailFieldBCC
: [in] Handle Name and email address of the email Blind Carbon Copy list. Multiple addresses can be specified by separating them with semi-colons (;).

szEmailSubject
: [in] Subject of the email.

szEmailMessage
: [in] Email message.

bEmailPrompt
: [in] If set to True, the user is prompted with the message details before the email is sent. This parameter is used only in the case of sending emails through MAPI.

Return Value

These functions return 1 if successful, 0 otherwise.

Remarks

Emails can be sent using either MAPI compliant email systems, or directly using SMTP protocol without the use of any installed email client.

Example

Check the **SetEmailOptions** function for a complete sample

SetEmailOptions, GetEmailOptions Functions

The SetEmailOptions and SetEmailOptions are used to set or read options specific to sending emails through the Amyuni Converter products.

Syntax

```
long SetEmailOptions( HANDLE hPrinter, long nEmailOptions );
long GetEmailOptions( HANDLE hPrinter );
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

nEmailOptions

[in] Options specific to sending emails. The list of supported options is in the Remarks section below.

Return Value

The SetEmailOptions function returns 1 if successful, 0 otherwise. The GetEmailOptions function returns the options currently set in the printer.

Remarks

Emails can be sent using either MAPI compliant email systems, or directly using SMTP protocol without the use of any installed email client.

These functions supecede the SetEmailPrompt function as they provide some additional options.

nEmailOptions values:

Option	Option value
Prompt before sending email	2
Send emails using SMTP	4

Example

```
void CDLLTestDlg::OnPrint()
{
  HFONT    font, oldFont;
  HDC      dc;
  DOCINFO  di;

  // create a printer device context
  dc = CreateDC( "winspool", theApp.m_szPrinter, NULL, NULL );
  if ( NULL == dc )
  {
          ASSERT( FALSE );
          return;
  }

  // init the DOCINFO structure
  memset( &di, 0, sizeof(di) );
  di.cbSize = sizeof( di );
  di.lpszDocName = "Test document";
  di.lpszOutput = NULL;

  // activate printer before starting to print
  EnablePrinter( theApp.m_converter, "Evaluation Version Developer",
  "07EFCDAB01000100584F8296BFE4DB671FE259A5726433B6E3BE1766876E2B5237F8F5" );

  // create an RTF document and send it by email
  SetFileNameOptions( theApp.m_converter, NoPrompt + ExportToRTF + SendByEmail );
```

```
  // create RTF files with no frames
  SetPrinterParamInt( theApp.m_converter, "RTF Format", 2 );

  // set email parameters to automatically send document by email
  SetEmailFieldTo( theApp.m_converter, "info@amyuni.com" );
  SetEmailFieldCC( theApp.m_converter, "support@amyuni.com" );
  SetEmailSubject( theApp.m_converter, "Testing email capabilities" );
  SetEmailMessage( theApp.m_converter, "Please find attached the requested document\nin RTF
format." );
  SetEmailOptions( theApp.m_converter, SMO_PROMPT_RECIPIENT );

  // start printing
  StartDoc( dc, &di );
  StartPage( dc );
  MoveToEx( dc, 100, 100, NULL );
  LineTo( dc, 400, 100 );
  MoveToEx( dc, 100, 100, NULL );
  LineTo( dc, 100, 400 );

  font = CreateFont( -24, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, _T("Verdana") );
  oldFont = (HFONT)SelectObject( dc, font );
  TextOut( dc, 100, 100, _T("Hi There"), 8 );
  if ( oldFont ) SelectObject( dc, oldFont );
  DeleteObject( font );

  EndPage( dc );

  EndDoc( dc );

  DeleteDC( dc );

  // printing ended, no need to disactivate printer
  // the printer will be disactivated automatically
}
```

SetSmtpServer, SetSmtpPort Functions

The SetSmtpServer and SetSmtpPort functions are used to set the server address and port number of the server used to send emails. These parameters are needed only when sending emails through direct SMTP without the use of MAPI.

Syntax

```
int SetSmtpServer( HANDLE hPrinter, LPTSTR szSmtpServer );
int SetSmtpPort( HANDLE hPrinter, long lSmtpPort );
```

Parameters

hPrinter
: [in] Handle to printer returned by any of the DriverInit function calls.

szSmtpServer
: [in] Name or IP address of server used to send emails.

lSmtpPort
: [in] SMTP port number on server used to send emails. The default value is 25

Return Value

The SetSmtpServer and SetSmtpPort functions return 1 if successful, 0 otherwise.

Remarks

Emails can be sent using either MAPI compliant email systems, or directly using SMTP protocol without the use of any installed email client.

Example

```
// set email parameters to automatically send document by email using SMTP
SetEmailFieldFrom( theApp.m_converter, "test@amyuni.com" );
SetEmailFieldTo( theApp.m_converter, "info@amyuni.com" );
SetEmailFieldCC( theApp.m_converter, "support@amyuni.com" );
SetEmailSubject( theApp.m_converter, "Testing email capabilities" );
SetEmailMessage( theApp.m_converter, "Please find attached the requested document." );
SetSmtpServer(  theApp.m_converter, "smtp10.bellnet.ca" );
SetSmtpPort(  theApp.m_converter, 25 );
SetEmailOptions( theApp.m_converter, SMO_SMTP_MAIL );
```

SendMail, SendMailW Functions

The SendMail function is used to send a message with one or more attachments using MAPI email. The SendMailW function is the unicode equivalent of the same function.

Syntax

```
long SendMail( LPCSTR szTo, LPCSTR szCC, LPCSTR szBCC, LPCSTR szSubject, LPCSTR szMessage, LPCSTR
                szFilenames, long lOptions);
long SendMailW( LPCWSTR szTo, LPCWSTR szCC, LPCWSTR szBCC, LPCWSTR szSubject, LPCWSTR szMessage, LPCWSTR
                szFilenames, long lOptions );
```

Parameters

szTo
: [in] Name and email address of the email destinator. Multiple addresses can be specified by separating them with semi-colons (;).

szCC
: [in] Name and email address of the email Carbon Copy list. Multiple addresses can be specified by separating them with semi-colons (;).

szBCC
: [in] Handle Name and email address of the email Blind Carbon Copy list. Multiple addresses can be specified by separating them with semi-colons (;).

szSubject
: [in] Subject of the email.

szMessage
: [in] Email message.

szFileNames
: [in] Series of file names and their captions as they should appear in the email attachment. The file name is the full path of the file to send, the caption is the name of the file as seen by the recipient. The syntax for sending multiple files is as follows: file1;caption1;file2;caption2;....

lOptions
: [in] Options specific to sending emails. The list of supported options is in the Remarks section below.

Return Value

The SendMail function returns the value MAPI_E_NOT_SUPPORTED if no MAPI compliant email is installed in the system. It returns 0 if successful or a MAPI specific error code if an error occurs when sending the email.

Remarks

lOptions values:

Option	Option value
Prompt before sending email	2

Depending on the security settings of the system from which the email is sent, the users might be prompted with a confirmation message stating that an external application is trying to send an email on their behalf. This is an important MAPI security measure which cannot and should not be overridden.

Example

```
// send multiple files using MAPI email without prompting the user
SendMail( "info@amyuni.com", "support@amyuni.com", "", "Testing email capabilities",
        "Please find attached the requested document.",
        "c:\\temp1.pdf;document1.pdf;c:\\temp2.pdf;document2.pdf", 2
        );
```

SendSmtpMail, SendSmtpMailW Functions

The SendSmtpMail function is used to send a message with one or more attachments using direct SMTP email. The SendSmtpMailW function is the unicode equivalent of the same function.

Syntax

```
long SendSmtpMail( LPCSTR szHostname, long lPort, LPCSTR szFrom, LPCSTR szTo, LPCSTR szCC, LPCSTR szBCC,
                   LPCSTR szSubject, LPCSTR szMessage, LPCSTR szFilenames, long lOptions );
long SendSmtpMailW( LPCWSTR szHostname, long lPort, LPCWSTR szFrom, LPCWSTR szTo, LPCWSTR szCC, LPCWSTR
                   szBCC, LPCWSTR szSubject, LPCWSTR szMessage, LPCWSTR szFilenames, long lOptions );
```

Parameters

szHostName
: [in] Name or IP address of server used to send emails.

lPort
: [in] SMTP port number on server used to send emails. The default value is 25

szFrom
: [in] Name of the email sender as it shows to the person receiving the email.

szTo
: [in] Name and email address of the email destinator. Multiple addresses can be specified by separating them with semi-colons (;).

szCC
: [in] Name and email address of the email Carbon Copy list. Multiple addresses can be specified by separating them with semi-colons (;).

szBCC
: [in] Handle Name and email address of the email Blind Carbon Copy list. Multiple addresses can be specified by separating them with semi-colons (;).

szSubject
: [in] Subject of the email.

szMessage
: [in] Email message.

szFileNames
: [in] Series of file names and their captions as they should appear in the email attachment. The file name is the full path of the file to send, the caption is the name of the file as seen by the recipient. The syntax for sending multiple files is as follows: file1;caption1;file2;caption2;....

Return Value

The SendSmtpMail function returns 0 if successful or an SMTP specific error code if an error occurs when sending the email.

Remarks

Example

```
// send multiple files using SMTP email
SendSmtpMail( "smtp.amyuni.com", 25, "test.amyuni.com", "info@amyuni.com",
                "support@amyuni.com", "", "Testing email capabilities",
                "Please find attached the requested document.",
                "c:\\temp1.pdf;document1.pdf;c:\\temp2.pdf;document2.pdf"
                );
```

Print Job Locking Functions

```
Lock
Unlock
SetDocFileProps
```

Lock Function

The Lock function can be used in multi-threading situations to avoid conflicts between multiple applications or multiple threads requesting simultaneous access the Converter products. The CDIntf library uses the registry to interact with the printer drivers. This can cause conflicts when multiple applications use CDIntf to access the printer drivers.

Syntax

```
int Lock( HANDLE hPrinter, LPCSTR szLockName );
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

szLockName

[in] Lock identifier, this should be the document title as it appears in the print spooler when printing any document.

Return Value

The return value is 0 if the function is successful, or a Windows specific error code if the function fails.

Remarks

The Lock function is only needed for applications using CDIntf to set the destination file name and options. The technical notes on www.amyuni.com provide alternative ways for using the Document Converter products in multi-threading situations without the need for CDIntf or the Lock functions.

When the Lock function is used, the output file name and options are set using the SetDocFileProps function and not the more common SetDefaultFileName and SetFileNameOptions functions.

Job locking is needed in the following scenario:

1. Application or Thread A uses SetDefaultFileName or SetDefaultDirectory to set the output file name before starting to print
2. Application B also also uses SetDefaultFileName or SetDefaultDirectory to set the output file name
3. Application A starts to print, but prints to the file set by application B and not application A

To solve this issue, the printer driver should be locked for the few microseconds it takes for application A from the time it sets the output file name, to the time it starts to print. As soon as the Application A starts to print, the lock can be released to allow Application B to print in parallel.

Example

```
DWORD WINAPI ThreadFunc( HANDLE converter )
{
  // this method opens and prints a Word document using Automation
  // it uses the Locking mechanism of CDIntf to avoid multi-threading conflicts

  static   int      index = 1;       // the index is incremented for every printout
  TCHAR    fileName[MAX_PATH];

  // the lock name in the case of MS Word is the same as the file name
  TCHAR    LockName[] = _T("test.doc");

  // these classes are created when importing the MS Word objects
  _Application     *wordApp = NULL;
  Documents        *documents = NULL;

  ::OleInitialize( NULL );
```

```
  LPDISPATCH               pdisp;

  __try {
          // open MS Word
          wordApp = new _Application();
          wordApp->CreateDispatch( _T("Word.Application") );

          // open a Word document in Read-only mode
          pdisp = wordApp->GetDocuments();
          documents = new Documents( pdisp );
          documents->Open( COleVariant( _T("c:\\wutemp\\test.doc") ),// file name
                                          COleVariant(),
                                          COleVariant( (short)1 )
                                          );

          // set the ActivePrinter to ours
          wordApp->SetActivePrinter( theApp.m_szPrinter );

          // lock printer before starting to print
          Lock( converter, LockName );
          wsprintf( fileName, "c:\\test%d.pdf", index++ );
          SetDocFileProps( converter, LockName,
                  NoPrompt + UseFileName,   // output file name defined by our application
                   _T(""),                  // default directory is not used
                  fileName                  // set the output file name to "testN.pdf"
                  );

          // enable printer before printing (developer version only)
          EnablePrinter( converter, "Evaluation Version Developer",
  "07EFCDAB0100010024CE83E1E8DC2244455F2F2C87EFC6FFF82B2F90206F7EEE87AE0751CAECA86FED0207C
D295E03629A5726433B6E3BE1766876E2B5237F8F5" );

          // print the Word document without background printing
          wordApp->PrintOut( COleVariant( 0L ) );
  }
  __finally
  {
          // the printer is unlocked automatically as soon as it starts to print
          // Unlock is needed only in the case an error occurs while printing
          Unlock( converter, LockName, 1000 );

          if ( wordApp )
                  wordApp->Quit( COleVariant(), COleVariant(), COleVariant() );
          delete documents;
          delete wordApp;
  }

  ::OleUninitialize();

  return 0;
}

void CDLLTestAppDlg::OnPrintWord()
{
  // this function simulates two threads printing a Word document in parallel
  DWORD    dwThreadId;
  HANDLE   hThread1 = CreateThread( NULL, 0, ThreadFunc, (LPVOID)theApp.m_converter, 0,
          &dwThreadId );
  HANDLE   hThread2 = CreateThread( NULL, 0, ThreadFunc, (LPVOID)theApp.m_converter, 0,
          &dwThreadId );

  CloseHandle( hThread1 );
  CloseHandle( hThread2 );
}
```

Unlock Function

The Unlock function can be used in multi-threading situations to avoid conflicts between multiple applications or multiple threads requesting simultaneous access the Converter products. The CDIntf library uses the registry to interact with the printer drivers. This can cause conflicts when multiple applications use CDIntf to access the printer drivers.

Syntax

```
int Unlock( HANDLE hPrinter, LPCSTR szLockName, long dwTimeout );
```

Parameters

hPrinter
: [in] Handle to printer returned by any of the DriverInit function calls.

szLockName
: [in] Lock identifier, this should be the document title as it appears in the print spooler when printing any document.

dwTimeout
: [in] Timeout in milliseconds after which the function returns.

Return Value

The return value is 0 if the function is successful, or a Windows specific error code if the function fails.

Remarks

The Unlock function is used after printing has ended to make sure another printout can start. The call to Unlock is needed only in the case where an error occurs, the printer will otherwise call Unlock internally as soon as printing starts to allow another printout to occur in parallel.

Example

```
See the Lock function for a complete sample.
```

SetDocFileProps Function

The SetDocFileProps function can be used in multi-threading situations to avoid conflicts between multiple applications or multiple threads requesting simultaneous access the Converter products. The CDIntf library uses the registry to interact with the printer drivers. This can cause conflicts when multiple applications use CDIntf to access the printer drivers.

Syntax

```
int SetDocFileProps( HANDLE hPrinter, LPCSTR szDocTitle, long lOptions, LPCSTR szFileDir,
                     LPCSTR szFileName );
```

Parameters

hPrinter
: [in] Handle to printer returned by any of the DriverInit function calls.

szDocTitle
: [in] The document title as it appears in the print spooler when printing any document, this should be the same as the parameter szLockName used for Lock and Unlock.

lOptions
: [in] Output file name options equivalent to the function SetFileNameOptions.

szFileDir
: [in] Destination directory, equivalent to the SetDefaultDirectory function call.

szFileName
: [in] Output file name, equivalent to the function SetDefaultFileName.

Return Value

The return value is 0 if the function is successful, or a Windows specific error code if the function fails.

Remarks

This function replaces the calls to SetDefaultDirectory, SetDefaultFileName and SetFileNameOptions in the cases where print job locking is needed. The documentation for these three functions contains a complete description of the lOptions, szFileDir and szFileName parameters.

Example

```
See the Lock function for a complete sample.
```

PDF File Processing

```
ConcatenateFiles
MergeFiles

SetLicenseKeyA, SetLicenseKeyW

EncryptPDFDocument, EncryptPDFDocument128
LinearizePDFDocument

PrintPDFDocument, PrintPDFDocumentEx

PDF2RTF
PDF2HTML
PDF2EXCEL
PDF2JPEG
```

ConcatenateFiles Function

The ConcatenateFiles function appends or concatenates a second PDF file to a first one.

Syntax

```
BOOL ConcatenateFiles( LPCSTR file1, LPCSTR file2, LPCSTR file3 );
```

Parameters

file1

[in] Full path of the first file.

file2

[in] Full path of the second file to be appended to file1.

file3

[in] Full path of the destination or output file.

Return Value

The return value is True if the files were appended, False if an error occurred appending files.

Remarks

This function is not guaranteed to process files created from PDF generation tools other than the Amyuni PDF Converter, or Amyuni PDF Creator. It will however work with files coming from most applications.

Example

```
    ConcatenateFiles( "c:\\test1.pdf", "c:\\test2.pdf", "c:\\appended.pdf" );
```

MergeFiles Function

The MergeFiles function merges two PDF documents by combining the contents of every page of the first document with a page from the second document.

Syntax

```
BOOL MergeFiles( LPCSTR file1, LPCSTR file2, LPCSTR file3, long Options );
```

Parameters

file1

[in] Full path of the first file containing the watermark.

file2

[in] Full path of the second file to which file1 should be merged.

file3

[in] Full path of the destination or output file.

Options

[in] Options for merging files. The list of options is in the remarks section.

Return Value

The return value is True if the files were merged, False otherwise.

Remarks

This function is not guaranteed to process files created from PDF generation tools other than the Amyuni PDF Converter, or Amyuni PDF Creator. It will however work with files coming from most applications.

Options values:

Option	Option value	Description
Repeat first pages	1	The first pages of the second document are repeated in the first document
Second document above first	2	The contents of the second document are printed above the contents of the first document

If the documents do not have the same number of pages, say file1 has N1 pages and file2 has N2 pages where N1 < N2, then the developer can choose to:

- either merge file1 with the N1 pages of file2 and keep the remaining N2-N1 pages of file2 unchanged, in this case the Repeat option should be set to 0

- or merge the first block of N1 pages of file2 with the N1 pages of file1, merge the second block of N1 pages of file1 with the N1 pages of file1 and so on, in this case Repeat should be set to 1

Example

```
// If file1 contains the company’s letterhead in PDF format as one page, file2 is a two
// page invoice in PDF format generated with the accounting package, we can call:

// to repeat the company’s letterhead on all the invoice pages
MergeFiles( "file1.pdf", "file2.pdf", "output.pdf", 1 );

// to insert the company’s letterhead on the first page only
MergeFiles( "file1.pdf", "file2.pdf", "output.pdf", 0 );
```

SetLicenseKeyA, SetLicenseKeyW Functions

The SetLicenseKeyA (Ansi) and SetLicenseKeyW (Unicode) functions should be used after loading the CDINTF210.DLL file to activate the advanced functions that require the printer activation code to work properly. The advanced DLL functions that require a call to SetLicenseKey are: EncryptPDFDocument, LinearizePDFDocument, PrintPDFDocument, PDF2RTF, PDF2HTML, PDF2EXCEL, PDF2JPEG.

Syntax

```
BOOL SetLicenseKeyA(LPCSTR szCompany, LPCSTR szLicKey);      // ANSI Version
BOOL SetLicenseKeyW(LPCWSTR szCompany, LPCWSTR szLicKey);    // UNICODE Version
```

Parameters

szCompany
: [in] Name of the company or private user having licensed the product.

szLicKey
: [in] License key provided by Amyuni Technologies when downloading or purchasing a product

Return Value

The return value is True if the license key is valid, False otherwise.

Remarks

When CDINTF210 is loaded dynamically, this function is called right after the call to LoadLibrary. When CDINTF210 is statically linked to the application, this function can be called in the InitInstance of the main application.

Example

```
typedef BOOL (CALLBACK *lpfnSetLicenseKeyA)(LPCSTR Company, LPCSTR LicKey);
lpfnSetLicenseKeyA SetLicenseKey;
HINSTANCE hModule = LoadLibrary( _T("CDINTF210.DLL") );
if ( NULL == hModule )
{
    return FALSE;
}
SetLicenseKey = (lpfnSetLicenseKeyA)GetProcAddress( hModule, _T("SetLicenseKeyA") );
ASSERT( SetLicenseKey != NULL );
SetLicenseKey("Evaluation Version Developer",
"07EFCDAB010001005244455F2F2C87EFC6FFF82B2E0751CAECA86FED0207CD295E03629A5726433B6E3BE1766
876E2B5237F8F5" );
```

EncryptPDFDocument, EncryptPDFDocument128 Functions

The EncryptPDFDocument and EncryptPDFDocument128 functions can be used to password protect a PDF document and restrict users to viewing, modifying or even printing the document. These functions requires a call to SetLicenseKey before it can be used. The EncryptPDFDocument function uses 40 bits encryption, whereas the EncryptPDFDocument128 function uses 128 bits encryption compatible with Adobe® Acrobat® 5 and higher.

Syntax

```
BOOL EncryptPDFDocument( LPCSTR FileName, LPCSTR Owner, LPCSTR User,
                         DWORD Permissions );
BOOL EncryptPDFDocument128( LPCSTR FileName, LPCSTR Owner, LPCSTR User,
                         DWORD Permissions );
```

Parameters

FileName
: [in] Full path of PDF file to encrypt

Owner
: [in] Owner password

User
: [in] User password

Permissions
: [in] Options to restrict users opening the document using the User password

Return Value

The return value is True if the document was encrypted, False otherwise.

Remarks

This function is only available if the activation code is for a professional version of the Amyuni PDF Converter. In the case of the evaluation version, the passwords are always set to "aaaaaa" and "bbbbbb" and cannot be changed.

Permissions values:

Permission	Permission value
Enable Printing	-64 + 4
Enable document modification	-64 + 8
Enable copying text and graphics	-64 + 16
Enable adding and changing notes	-64 + 32

To combine multiple options, use -64 plus the values 4, 8, 16 or 32. E.g. to enable both printing and adding notes, use -64 + 4 + 32 = -28. To disable all 4 options, use -64.

Owner and user passwords

Two passwords are associated to an encrypted PDF document. The owner password is for the author of the document, and the user password for the destinator or user of the document.

The owner password is mandatory and allows the author having this password to do any operation he/she wishes on this document, including modifying its security settings.

The user password is optional and can be one of the following:

- A blank password. In this case, the user is not prompted for a password when opening a document, but is restricted to the operations allowed by the author.
- The same password as the owner. In this case the user is prompted for a password and the author of the document will not be able to open this document as an owner to change its security settings.
- A password different from the owner. In this case, the user will not be able to open the document unless he/she enters a valid password. When a valid password is entered, the document can be viewed but its usage restricted to the operations allowed by the author.

User settings

- Enable changing the document content. When this option is cheked, the user is allowed to change the contents of the PDF document.

- Enable printing of document. The user cannot print the PDF document to any printer unless this option is checked.
- Enable copying text and graphics from the document. When this option is checked, the user can copy parts of the text of graphics from the PDF document.
- Enable addings notes or modifying form fields. The main body of the document cannot be changed but the user can add annotation or enter data in the form fields if there are any.

NOTE: These options are managed by the tool used to view the document and not by the PDF Converter. Once a valid password is entered, it is up to the viewer or editor to make sure that these security settings are respected.

Example

```
// enable advanced functions such as Encryption or Linearization
SetLicenseKeyA( "Evaluation Version Developer",
"07EFCDAB01000100584F829697ECDB6776F3CC948D0749DAF7E37EF1621AEBEBF2975B" );

// password protect the PDF document
// users are only allowed to print
EncryptPDFDocument( "c:\\test.pdf", "ownerpass", "userpass", -64 + 4 );
```

LinearizePDFDocument Function

The LinearizePDFDocument function can be used to optimize a document for web viewing. PDF documents usually need to be completely downloaded before thay can be viewed. A linearized document can be viewed one page at a time without the need to completely download the document. This function requires a call to SetLicenseKey before it can be used.

Syntax

```
BOOL LinearizePDFDocument( LPCSTR FileName );
```

Parameters

FileName

[in] Full path of PDF file to optimize.

Return Value

The return value is True if the document was linearized, False otherwise.

Remarks

This function is only available if the activation code is for a professional version of the Amyuni PDF Converter. In the case of the evaluation version, a message-box appears to indicate that the document is being optimized.

Example

```
// enable advanced functions such as Encryption or Linearization
SetLicenseKeyA( "Evaluation Version Developer",
"07EFCDAB01000100584F829697ECDB6776F3CC948D0749DAF7E37EF1621AEBEBF2975B" );

// optimize the PDF document for web downloading
LinearizePDFDocument( "c:\\test.pdf" );

// password protect the PDF document
// this should be done after web optimization and will keep the document linearized
EncryptPDFDocument( "c:\\test.pdf", "ownerpass", "userpass", -64 + 4 );
```

PrintPDFDocument, PrintPDFDocumentEx Functions

The PrintPDFDocument and PrintPDFDocumentEx functions can be used to print a PDF document to any standard printer. The PrintPDFDocumentEx version should be used when the document is password encrypted.

Syntax

```
BOOL PrintPDFDocument( LPCSTR FileName, LPCSTR PrinterName, long StartPage, long EndPage,
                       long Copies );
BOOL PrintPDFDocumentEx( LPCSTR FileName, LPCSTR Password, LPCSTR PrinterName,
                       long StartPage, long EndPage, long Copies );
```

Parameters

FileName
: [in] Full path of PDF file to print.

PrinterName
: [in] Destination printer. If empty, print to the system default printer.

Password
: [in] Password used to open protected PDF file.

StartPage
: [in] Starting page number from which to print.

EndPage
: [in] End page number to print.

Copies
: [in] Number of copies to print document.

Return Value

The return value is True if the document was printed successfully, False otherwise.

Remarks

This function is only available if the activation code is for a professional version of the Amyuni PDF Converter. In the case of the evaluation version, only one page at a time can be printed.

If the number of pages is not known and we need to print the whole document, a very large integer can be used for the EndPage parameter. C/C++ developers would typically use MAX_INT (0x7fffffff).

Example

```
// enable advanced functions such as Encryption, Linearization or Printing
SetLicenseKeyA( "Evaluation Version Developer",
"07EFCDAB01000100584F829697ECDB6776F3CC948D0749DAF7E37EF1621AEBEBF2975B" );

// Print document to the system default printer
PrintPDFDocument( "c:\\test.pdf", "", 1, MAX_INT, 1 );
```

PDF2RTF Function

The PDF2RTF function converts a PDF document to an RTF document. This function is only available with the RTF Converter product and requires a call to SetLicenseKey before it can be used.

Syntax

```
long PDF2RTF( LPCSTR InputFile, LPCSTR Password, LPCSTR OutputFile, long Options,
                    long OptimizeLevel );
```

Parameters

InputFile
: [in] Full path of PDF file to convert to RTF.

Password
: [in] Password, if any, needed to open the PDF file.

OutputFile
: [in] Full path of resulting RTF file.

Options
: [in] RTF generation options.

OptimizeLevel
: [in] Optimization level to apply to PDF document before exporting to RTF.

Return Value

The return value is True if the document was converted, False otherwise.

Remarks

This function is only available if the activation code is for the RTF Converter product, or an RTF Converter product combined with other Document Converter products.

Options values:

Option	Option value (Hex)
Advanced RTF: using frames to position objects	0000
Full RTF: Text, Graphics and images with no frames	0001
Formatted Text only	0002
Simple text, non-formatted	0003
Simple text, non-formatted with spaces replacing tabs	10003

OptimizeLevel values:

Optimization Level	OptimizeLevel value
No optimization	0
Line optimization (Recommended)	1
Paragraph optimization	2

Example

```
// enable advanced functions such as RTF Export
SetLicenseKeyA( "Evaluation Version Developer",
"07EFCDAB01000100584F829697ECDB6776F3CC948D0749DAF7E37EF1621AEBEBF2975B" );

// convert a PDF document to RTF
PDF2RTF( "c:\\test.pdf", "", "c:\\test.rtf", RTFOPTION_FULL, 1 );
```

PDF2HTML Function

The PDF2HTML function converts a PDF document to an HTML document. This function is only available with the DHTML Converter product and requires a call to SetLicenseKey before it can be used.

Syntax

```
long PDF2HTML( LPCSTR InputFile, LPCSTR Password, LPCSTR OutputFile, long Options,
                    long OptimizeLevel );
```

Parameters

InputFile
: [in] Full path of PDF file to convert to HTML.

Password
: [in] Password, if any, needed to open the PDF file.

OutputFile
: [in] Full path of resulting HTML file.

Options
: [in] HTML generation options.

OptimizeLevel
: [in] Optimization level to apply to PDF document before exporting to HTML.

Return Value

The return value is True if the document was converted, False otherwise.

Remarks

This function is only available if the activation code is for the DHTML Converter product, or a DHTML Converter product combined with other Document Converter products.

Options values:

Option	Option value (Hex)
Use Layers: Multiple pages in a single HTML file using layers	0001
Single HTML: All pages in a single HTML file	0002
Multiple HTML files: Each page in a separate HTML file	0003

OptimizeLevel values:

Optimization Level	OptimizeLevel value
No optimization	0
Line optimization	1
Paragraph optimization (Recommended)	2

Example

```
// enable advanced functions such as HTML Export
SetLicenseKeyA( "Evaluation Version Developer",
"07EFCDAB01000100584F829697ECDB6776F3CC948D0749DAF7E37EF1621AEBEBF2975B" );

// convert a PDF document to HTML
PDF2HTML( "c:\\test.pdf", "", "c:\\test.html", HTMLOPTION_SINGLE_PAGE, 2 );
```

PDF2EXCEL Function

The PDF2EXCEL function converts a PDF document to a Microsoft® Excel® document. This function is only available with the Excel Converter product and requires a call to SetLicenseKey before it can be used.

Syntax

```
long PDF2EXCEL( LPCSTR InputFile, LPCSTR Password, LPCSTR OutputFile, long Options,
                    long OptimizeLevel );
```

Parameters

InputFile
: [in] Full path of PDF file to convert to Excel.

Password
: [in] Password, if any, needed to open the PDF file.

OutputFile
: [in] Full path of resulting Excel file.

Options
: [in] Excel generation options.

OptimizeLevel
: [in] Optimization level to apply to PDF document before exporting to Excel.

Return Value

The return value is True if the document was converted, False otherwise.

Remarks

This function is only available if the activation code is for the Excel Converter product, or an Excel Converter product combined with other Document Converter products.

Options values:

Option	Option value (Hex)
Single Sheet: All pages in one sheet	0
Multiple Sheets: one sheet per page	1

OptimizeLevel values:

Optimization Level	OptimizeLevel value
No optimization	0
Line optimization	1
Paragraph optimization (Recommended)	2
Table optimization	3

Example

```
// enable advanced functions such as Excel Export
SetLicenseKeyA( "Evaluation Version Developer",
"07EFCDAB01000100584F829697ECDB6776F3CC948D0749DAF7E37EF1621AEBEBF2975B" );

// convert a PDF document to Excel
PDF2EXCEL( "c:\\test.pdf", "", "c:\\test.xls", EXCELOPTION_MULTIPLE_SHEETS, 2 );
```

PDF2JPEG Function

The PDF2JPEG function converts a PDF document to a JPEG document. This function is only available with the JPeg Converter product and requires a call to SetLicenseKey before it can be used.

Syntax

```
long PDF2JPEG( LPCSTR InputFile, LPCSTR Password, LPCSTR OutputFile, long Options,
                    long OptimizeLevel );
```

Parameters

InputFile
: [in] Full path of PDF file to convert to JPeg.

Password
: [in] Password, if any, needed to open the PDF file.

OutputFile
: [in] Full path of resulting JPeg file(s).

Options
: [in] JPeg generation options.

OptimizeLevel
: [in] Optimization level to apply to PDF document before exporting to HTML.

Return Value

The return value is True if the document was converted, False otherwise.

Remarks

This function is only available if the activation code is for the JPeg Converter product, or a JPeg Converter product combined with other Document Converter products.

When the PDF document consists of multiple pages, one JPeg file is generated for every page with the page index appended to the supplied file name.

The Options value is a combination of the image resolution in the high order word, and the JPeg compression level from 1 (highest) to 9 (lowest) in the low order word.

Options values:

Option	Option value (Hex)
Default : 300 DPI, medium compression level	0x00000000
Low : 150 DPI, high compression level	0x00960003
Medium: 300 DPI, medium compression level	0x012C0007
High : 600 DPI, low compression level	0x02580009

OptimizeLevel values:

Optimization Level	OptimizeLevel value
No optimization (Recommended)	0
Line optimization	1
Paragraph optimization	2

Example

```
// enable advanced functions such as JPeg Export
SetLicenseKeyA( "Evaluation Version Developer",
"07EFCDAB01000100584F829697ECDB6776F3CC948D0749DAF7E37EF1621AEBEBF2975B" );

// convert a PDF document to JPeg
PDF2JPEG( "c:\\test.pdf", "", "c:\\test.jpeg", JPEGOPTION_DEFAULT, 0 );
```

General Functions

```
CDICreateDC
SetBookmark
SetHyperLink
CaptureEvents
GetLastErrorMsg
BatchConvertEx
SetTargetPrinterName
SetPageProcessor

SetServerAddress
SetServerPort
SetServerUsername
```

CDICreateDC Function

The CDICreateDC function creates a printer device context using the various parameters set using CDIntf function calls. The parameters include paper size, resolution, margins, font embedding, jpeg compression, ...

Syntax

```
HDC CreateDC( HANDLE hPrinter );
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

Return Value

The return value is a printer device context (hDC) if the function succeeds, NULL otherwise.

Remarks

Example

```
HANDLE    hPrinter;

hPrinter = PDFDriverInit( "My Application Converter" );      // create a new printer
If ( hPrinter )
{
  SetPaperSize( hPrinter, 5 );              // set default paper size to Legal
  // set other printer defaults here
  ...
  HDC     printerDC = CDICreateDC( hPrinter );
}
```

SetBookmark Function

The SetBookmark function creates a bookmark on the current printing page and location. Users will then be able to browse through the document by clicking on the bookmarks tree. Bookmarks are currently available in PDF files only.

Syntax

```
long SetBookmark( HANDLE hDC, long lParent, LPCSTR szTitle );
```

Parameters

hDC

[in] Handle to printer device context returned by a call to CreateDC.

lParent

[in] Id of parent bookmark.

szTitle

[in] Bookmark title.

Return Value

The return value is the identifier of the bookmark that was created, or 0 if the function fails.

Remarks

PDF bookmarks are structured in a tree. The root has an id of 0. SetBookmark returns the bookmark ID which can be used to insert other bookmarks as children of the current bookmark.

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

Example

```
Please check the method CDIntfEx.SetBookmark for a complete sample
```

SetHyperlink Function

The SetHyperlink function creates a hyperlink on the current printing text or image object. Users will then be able to click on the hyperlink to browse to another location inside the document or to an external URL. Hyperlinks are currently available in PDF and HTML files only.

Syntax

```
long SetHyperLink( HANDLE hDC, LPCSTR szDestination );
```

Parameters

hDC

[in] Handle to printer device context returned by a call to CreateDC.

szDestination

[in] Hyperlink destination.

Return Value

The return value is always 0.

Remarks

Hyperlinks can be set to an external URL such as http://www.amyuni.com or to a bookmark in the same document. To add a hyperlink to an internal bookmark, the szDestination should start with the # sign followed by the bookmark title.

The hyperlink will be inserted at the location where the last text drawing operation occurred. E.g.: if we draw text in the middle of page 3 of the document and call SetHyperlink immediately after, the hyperlink will be set on the text in the middle of page 3 of the PDF document.

Example

```
Please check the method CDIntfEx.SetHyperlink for a complete sample
```

GetLastErrorMsg Function

The GetLastErrorMsg function returns the error message generated by the last call to a CDIntf function call.

Syntax

```
void GetLastErrorMsg( LPSTR Msg, long MaxMsg );
```

Parameters

Msg

[out] Address of buffer that will contain the error message.

MaxMsg

[in] Size of the Msg buffer. The error message will be truncated if it is larger than the input buffer.

Return Value

On return, the Msg parameter will contain a description of the last error generated by a call to a CDIntf function.

Remarks

Example

```
HANDLE    hPrinter = DriverInit("Amyuni PDF Converter")
if ( NULL == hPrinter )
{
    // DriverInit failed, get error message generated by CDIntf
    char  error[256];
    GetLastErrorMsg( error, sizeof(error) );
    MessageBox( hWnd, error, "Printer Error", MB_OK | MB_ICONWARNING );
    return FALSE;
}
```

BatchConvertEx Function

The BatchConvertEx function converts a number of files to PDF, RTF, HTML, Excel or JPeg formats in batch mode.

Syntax

```
long BatchConvertEx( HANDLE hPrinter, LPCSTR szFileName );
```

Parameters

hPrinter
: [in] Handle to printer returned by any of the DriverInit function calls.

szFileName
: [in] File or group of files to be converted.

Return Value

This function returns the number of files having been converted, or 0 if no files were converted.

Remarks

The Document Converter printer should be configured with the destination file name and all other options before calling this function. The printer should also be set as default printer. This function launches the application that is associated with a specific file and issues a print command to convert the document.

The FileName parameter can contain wildcards to convert a number of documents in the same directory.

Example

```
HANDLE    hPrinter = DriverInit("Amyuni PDF Converter")
if ( NULL != hPrinter )
{
    SetDefaultPrinter( hPrinter );                  // printer needs to be set as default
    SetDefaultDirectory( hPrinter, "c:\temp\pdf_files" );    // set destination directory

    // the file name is to be generated automatically from the document name
    SetFileNameOptions( hPrinter, NoPrompt );

    // convert all MS Word documents to PDF format
    BatchConvertEx( hPrinter, "c:\temp\*.doc" );

    // close printer
    DriverEnd( hPrinter );
}
```

Printer Driver Message Handling

```
PDF Driver Event message
SendMessagesTo
GetDocumentTitle
GetGeneratedFilename
```

PDF Driver Event Message

When the message broadcast option is set in the Document Converter products, the printer driver broadcasts messages to all running applications each time one of the following events occur:

Printer Enabled
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-bit value.
- A first parameter known as wParam. This is a 16-bit value under Windows 9x and a 32-bit value under Windows NT/2000/XP.
- A second parameter known as lParam. This is a 32-bit value.

The message Id generated by the Document Converter products can be retrieved by calling the Windows API RegisterMessage function:

nPDFDriverMessage = RegisterWindowMessage(_T("PDF Driver Event")).

This is needed to intercept messages generated by the printer driver.

Associated with each printer event, is 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
#define DOCUMENTEVENT_LAST              14

// this event is privately defined by the Document Converter products
#define DOCUMENTEVENT_ENABLEDPRE (DOCUMENTEVENT_LAST + 1)   // before checking if printer is enabled

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

Event Parameters:

Windows 95/98/Me

Event	wParam	LOWORD(lParam)	HIWORD(lParam)
Before checking if printer is enabled	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/2000/XP

Event	LOWORD(wParam)	HIWORD(wParam)	lParam
Before checking if printer is enabled	DOCUMENTEVENT_ENABLEDPRE	0	hDC
Before StartDoc is handled	DOCUMENTEVENT_STARTDOCPRE	0	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

The drawback of this method of intercepting messages, is that the printer driver will broadcast messages to all running applications which might result in slowing down the whole system. To avoid this message broadcast, the printer driver can be programmed to send messages only to a specific application or window by calling te SendMessagesTo function.

Sample message handling function

```
UINT  nPDFEventMessage = 0;        // this is the ID of the message sent by the driver

BEGIN_MESSAGE_MAP(CDLLTestAppDlg, CDialog)

  // trap PDF driver events
  ON_REGISTERED_MESSAGE( nPDFEventMessage, OnPDFEvent )

END_MESSAGE_MAP()

BOOL CDLLTestAppDlg::OnInitDialog()
{
  CDialog::OnInitDialog();

  // get windows version number
  OSVERSIONINFO    osvi;
  memset( &osvi, 0, sizeof( osvi ) );
  osvi.dwOSVersionInfoSize = sizeof( osvi );
  GetVersionEx( &osvi );
  m_bNT = (osvi.dwPlatformId == VER_PLATFORM_WIN32_NT);

  // get ID of message sent by PDF driver
  nPDFEventMessage = RegisterWindowMessage( "PDF Driver Event" );

  // send messages to our application only
  TCHAR    className[64];
  GetClassName( m_hWnd, className, sizeof(className) );
  SendMessagesTo( theApp.m_converter, className );

  // make sure event broadcast option is set
  SetFileNameOptions( theApp.m_converter, BroadcastMessages );

  return TRUE;
}

LONG CDLLTestAppDlg::OnPDFEvent( UINT wParam, LONG lParam )
{
// a Document Converter event has occured
//
  DWORD    jobId;
  HDC      hDC;
  WORD     eventId;

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

  if ( m_bNT )
  {        // Windows NT/2000/XP
           jobId = HIWORD( wParam );
           hDC = (HDC)lParam;
  }
  else
  {        // Windows 95/98/Me
           jobId = LOWORD( lParam );
           hDC = (HDC)HIWORD( lParam );
  }

  switch ( eventId )
  {
  case DOCUMENTEVENT_ENABLEDPRE:            // before checking if printer is enabled
           m_log.AddString( "Before checking if printer is enabled" );
           // activate printer before starting to print
           // this is needed with the developer version only
           EnablePrinter( theApp.m_converter, "Evaluation Version Developer",
  "07EFCDAB01000100584F829697ECDB6776E2564CF3CC948D0749DAF7E37EF1621AEBEBF2975B" );
```

```
            break;
  case DOCUMENTEVENT_STARTDOCPRE:           // before StartDoc is handled by the printer
            char    buf[255] = "";
            // display file name of document being printed
            GetGeneratedFilename( theApp.m_converter, buf, sizeof( buf ) );
            m_log.AddString( CString( "Destination file: " ) + buf );
            m_log.AddString( "StartDoc called but not handled yet" );
            break;
  case DOCUMENTEVENT_STARTDOCPOST:          // after StartDoc is handled by the printer
            m_log.AddString( "StartDoc called and handled by the driver" );
            break;
  case DOCUMENTEVENT_STARTPAGE:             // StartPage handled by the printer
            m_log.AddString( "StartPage called and handled" );
            break;
  case DOCUMENTEVENT_ENDPAGE:               // EndPage handled by the printer
            m_log.AddString( "EndPage called and handled" );
            break;
  case DOCUMENTEVENT_ENDDOCPRE:             // before EndDoc is handled by the printer
  {
            // get title of document being printed
            char    buf[255] = "";
            GetDocumentTitle( theApp.m_converter, jobId, buf, sizeof( buf ) );
            // display title of document being printed
            m_log.AddString( CString( "Document title: " ) + buf );

            m_log.AddString( "EndDoc called but not handled yet" );
            break;
  }
  case DOCUMENTEVENT_ENDDOCPOST:            // after EndDoc is handled by the printer
            m_log.AddString( "EndDoc called and handled" );
            break;
  default:
            return DOCUMENTEVENT_UNSUPPORTED;
  }
  return DOCUMENTEVENT_SUCCESS;
}
```

SendMessagesTo Function

The SendMessagesTo function is used to direct the Document Converter product to send all event messages to a specific window or application. This is more efficient than the default behaviour of the printer driver, where it would send messages to all running applications and thus slow down the system significantly.

Syntax

```
long SendMessagesTo( HANDLE hPrinter, LPCSTR szWndClass );
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

szWndClass

[in] Class of the Window to which the events are sent.

Return Value

The return value is 1 if the function is successful, 0 otherwise.

Remarks

This function should be used in conjunction with the BroadcastMessages option of the Document Converter products.

Example

```
// send messages to our application only
TCHAR   className[64];
GetClassName( m_hWnd, className, sizeof(className) );
SendMessagesTo( theApp.m_converter, className );
```

GetDocumentTitle Function

The GetDocumentTitle function returns the title of the document that is currently being printed. This function should be called in the message handling loop and will return invalid results as soon as the document finishes printing.

Syntax

```
void GetDocumentTitle( HANDLE hPrinter, long JobID, LPSTR Buf, long MaxBuf );
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

JobID

[in] Job ID as retrieved by the PDF message handling function.

Buf

[out] Address of buffer that will contain the document title.

MaxBuf

[in] Size of the Buf buffer. The document title will be truncated if it is larger than the input buffer.

Return Value

On return, the Buf parameter will contain the title of the document that is currently being printed.

Remarks

This function should be called from a PDF message handling function before the EndDocPost message is sent.

Example

```
Please refer to the PDF Driver Event section for a full sample
```

GetGeneratedFilename Function

The GetGeneratedFilename function returns the full path of the file that is currently being generated. This function should be called in the message handling loop and will return invalid results as soon as the processing of StartDoc is finished.

Syntax

```
void GetGeneratedFilename( HANDLE hPrinter, LPSTR Buf, int MaxBuf );
```

Parameters

hPrinter

[in] Handle to printer returned by any of the DriverInit function calls.

Buf

[out] Address of buffer that will contain the file name.

MaxBuf

[in] Size of the Buf buffer. The file name will be truncated if it is larger than the input buffer.

Return Value

On return, the Buf parameter will contain the full path of the file that is currently being generated.

Remarks

This function should be called from a PDF message handling function before the StartDocPost message is sent, i.e. in the StartDocPre event handler.

Example

```
Please refer to the PDF Driver Event section for a full sample
```

ActiveX Interface

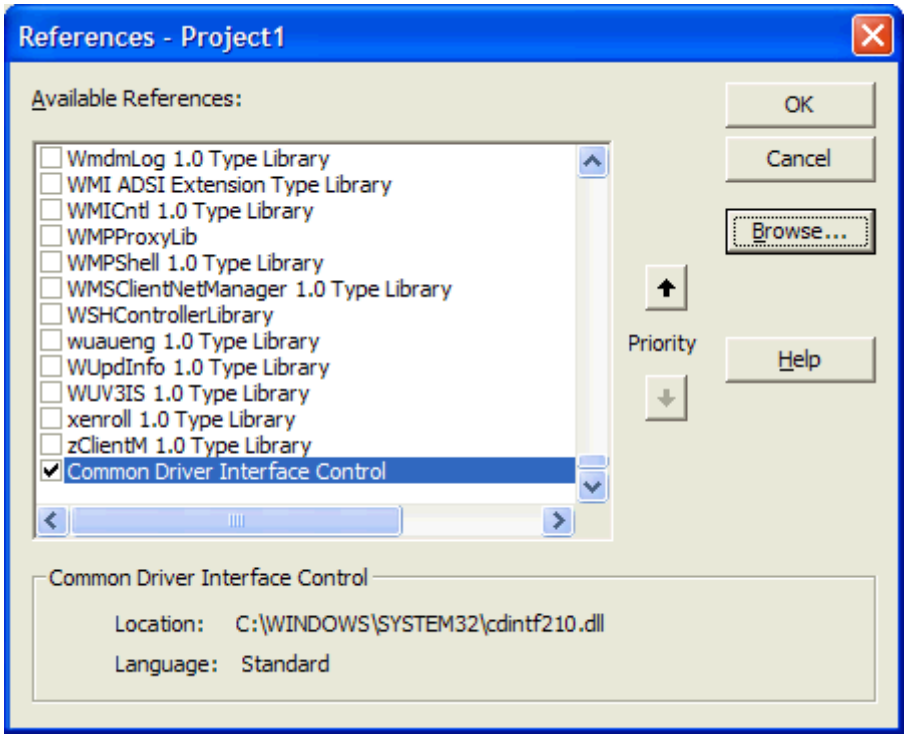
Before using the Common Driver Interface as an ActiveX component from any ActiveX aware application, the developer should register the DLL in the system by calling:

```
Regsvr32 CDINTF210.DLL
```

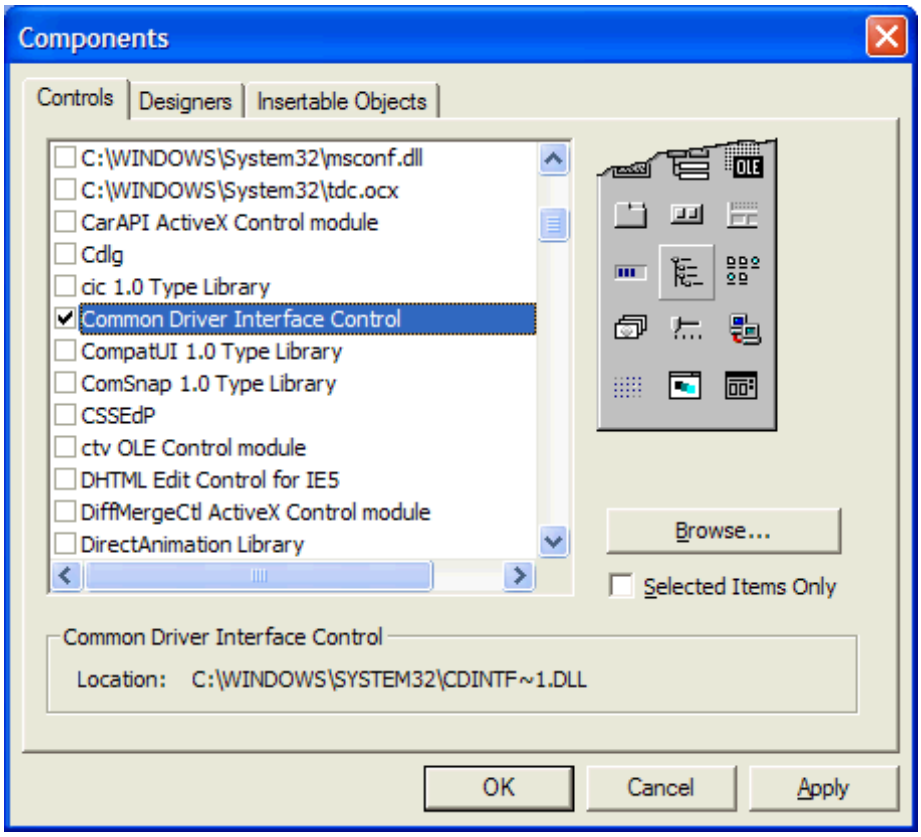
from the location where this DLL is installed. This is done automatically by the default installation procedure of any of the document converter product.

The CDIntf control in version 2.1 has been renamed to CDIntfEx to avoid confusion with the previous version. The control can either be created dynamically through code. Or placed on a form like any other VB control. When placed on a form, the control remains invisible at run time.

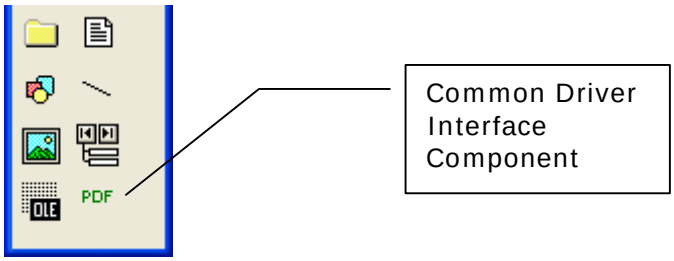
To dynamically create the CDIntfEx object, the CDINTF210 DLL should first be imported into the project. The procedure is quite similar in all programming environments. Here is what it would look like in VB:



To place the CDIntfEx control on a form, it should be imported into the project using the Project à Components menu:

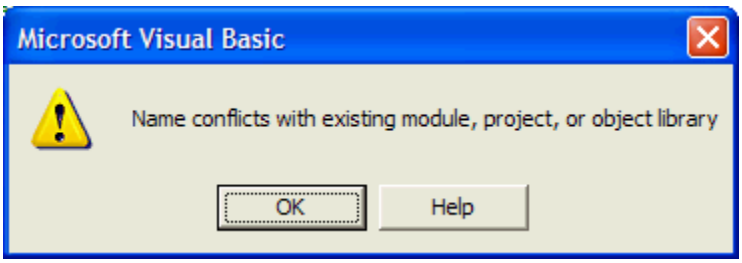


It should then appear in the components toolbar as follows:



Note to VB users:

VB users who had used previous versions of CDIntf or are trying to switch from dynamically created object to the visual component placed on a form, might receive the following error message from VB:



If this happens, the control should be manually removed from the project by following these steps:

1. Close the VB project
2. Open the .VBP file using any text editor
3. Locate and remove the line
   Reference=*\G{4856F146-7516-11D3-BBE5-D53DCBD65107}#1.0#0# c:\WINDOWS\System32\cdintf.dll#CDIntf
4. Reload the project in VB and import the control

Printer Installation and Activation

```
CDIntfEx.DriverInit
CDIntfEx.PDFDriverInit, CDIntfEx.HTMLDriverInit, CDIntfEx.RTFDriverInit
CDIntfEx.DriverEnd
CDIntfEx.SetDefaultPrinter
CDIntfEx.RestoreDefaultPrinter
CDIntfEx.EnablePrinter
CDIntfEx.PrinterAttributes
CDIntfEx.PrinterLanguage
```

CDIntfEx.DriverInit Method

The DriverInit method initializes the library for use with an already installed printer. DriverInit cannot install a new printer and will fail if the printer does not already exist.

Syntax

```
Function DriverInit (PrinterName As String) As Long
```

Parameters

PrinterName

[in] Name of the printer as it shows in the printers control panel.

Return Value

If the function succeeds, the return value is 0. If the function fails, it launches an exception that should be trapped, there is no return value in this case.

Remarks

This function will only fail if the printer does not exist. There are otherwise no known instances where the function will fail.

Example

```
Private Sub Form_Load()
    ' Case 1: the CDIntfEx object is created dynamically in memory
    Dim cdi As New CDIntfEx.CDIntfEx

    ' Case 2: the CDIntfEx object was placed on a form
    ' Set cdi = CDIntfEx1

    On Error GoTo printer_error
    ' attach to existing printer
    cdi.DriverInit ("Amyuni Document Converter")

    Exit Sub

printer_error:
    MsgBox "Sorry, printer not found"
End Sub
```

CDIntfEx.PDFDriverInit, CDIntfEx.HTMLDriverInit, CDIntfEx.RTFDriverInit Methods

The PDFDriverInit, HTMLDriverInit and RTFDriverInit methods can be used to create a new printer when the application is launched and remove it when the application is closed. These three functions are aliases the same internal function. They have been kept in version 2.1 for compatibility with previous versions. Any one of these can be used with any Amyuni Document Converter product with exactly the same result.

Syntax

```
Function PDFDriverInit(PrinterName As String) As Long
Function RTFDriverInit(PrinterName As String) As Long
Function HTMLDriverInit(PrinterName As String) As Long
```

Parameters

PrinterName

[in] Name of the printer as it shows in the printers control panel.

Return Value

If the function succeeds, the return value is 0. If the function fails, it launches an exception that should be trapped, there is no return value in this case. To get extended error information, call GetLastErrorMsg.

Remarks

This function will first attempt to connect to an existing printer. If it does not find any printer with the name provided by PrinterName, it will attempt to install a new printer. The function fails if it cannot find a printer with the specified name and cannot install a new printer. The printer referenced to by PrinterName is removed when the application exits or the DriverEnd function is called.

Installing or removing a printer requires administrative rights under Windows NT/2000/XP.

Example

```
Private Sub Form_Load()
    ' Case 1: the CDIntfEx object is created dynamically in memory
    Dim cdi As New CDIntfEx.CDIntfEx

    ' Case 2: the CDIntfEx object was placed on a form
    ' Set cdi = CDIntfEx1

    On Error GoTo printer_error
    ' install a new printer attach to existing printer
    cdi.PDFDriverInit ("Amyuni Document Converter")

    ' the printer will be destroyed as soon as the function exits
    Exit Sub

printer_error:
    MsgBox "Sorry, printer not found"
End Sub
```

CDIntfEx.DriverEnd Method

The DriverEnd method closes the printer handle created by one of the DriverInit functions and frees any internally allocated memory. If the printer was installed using the PDF, HTML or RTFDriverInit functions, it will be removed by the call to DriverEnd.

Syntax

```
Sub DriverEnd()
```

Parameters

Return Value

None.

Remarks

This function will simply detach from an existing printer if the handle was created using DriverInit, or attempt to remove the printer if the handle was created using PDF, HTML or RTFDriverInit.

Example

```
Public cdi As CDIntfEx.CDIntfEx
Private Sub Form_Load()
    ' Case 1: the CDIntfEx object is created dynamically in memory
    Dim cdi As New CDIntfEx.CDIntfEx

    ' Case 2: the CDIntfEx object was placed on a form
    ' Set cdi = CDIntfEx1

    On Error GoTo printer_error
    ' install a new printer attach to existing printer
    cdi.PDFDriverInit ("Amyuni Document Converter")

    Exit Sub

printer_error:
    MsgBox "Sorry, could not install printer"
End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' close the printer handle and remove printer
    cdi.DriverEnd
    Set cdi = Nothing
End Sub
```

CDIntfEx.SetDefaultPrinter Method

The SetDefaultPrinter method sets the system default printer to the one initialized by the DriverInit functions.

Syntax

```
Function SetDefaultPrinter() As Boolean
```

Parameters

Return Value

If the system default printer was modified, this function returns True. It returns False if the default printer was not modified. A return value of False usually indicates that the printer was already the system default printer before the call to SetDefaultPrinter.

Remarks

Some applications require that the printer be set as default before being able to print to that printer. The default printer that was set before calling this function is restored in one of three situations:

- The function RestoreDefaultPrinter is called
- DriverEnd is called
- The calling application is closed

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

Modifying the system default printer too frequently is a source of numerous printing errors. Developers are encouraged to use methods supplied by their application to specify the destination printer rather than calling this function.

Example

```
    On Error GoTo printer_error
    ' install a new printer attach to existing printer
    cdi.PDFDriverInit ("Amyuni Document Converter")

    ' set printer as system default
    cdi.SetDefaultPrinter
```

CDIntfEx.RestoreDefaultPrinter Method

The RestoreDefaultPrinter method resets the system default printer to the printer that was the default before the call to SetDefaultPrinter.

Syntax

```
Function RestoreDefaultPrinter() As Boolean
```

Parameters

Return Value

If the system default printer was modified, this function returns True. It returns False if the default printer was not modified. A return value of False usually indicates that the default printer was not modified by a call to SetDefaultPrinter.

Remarks

This function is called automatically when DriverEnd is called.

Example

```
    ' restore default printer before removing our printer
    cdi.RestoreDefaultPrinter
    ' close the printer handle and remove printer
    cdi.DriverEnd
```

CDIntfEx.EnablePrinter Method

The EnablePrinter method can be used to install the permanent license and activation code for the product. The license and activation code define which features of the product are available to the users depending on the product they pruchased. For example, users of the PDF Converter will have a different activation code than users of the mixed PDF/RTF/HTML Converter product. EnablePrinter is also used in the developer versions of the products to activate the printer before every printout.

Syntax

```
Function EnablePrinter(Company As String, Code As String) As Long
```

Parameters

Company
: [in] Name of the company or private user having licensed the product.

Code
: [in] License key provided by Amyuni Technologies when downloading or purchasing a product.

Return Value

The return value is always False.

Remarks

This function is usually called after successful installation of the product. End-users would usually enter their license information through the printer configuration tab, whereas developers would call this function from their product installation routine. The default install.exe that is provided with the products can also be used to set the license information by adding the following switches to the command line:

-n CompanyName -c LicenseKey

Calling the EnablePrinter function after installing the printer requires administrative rights under Windows NT/2000/XP, calling the same function before every printout does not require administrative rights and can be used with users having limited priveleges on the system.

Note to Developers

In addition to calling EnablePrinter after installing the printer, this function should also be called right before printing any document. The printer will otherwise be disabled. If the developers have full control of the printing process, they can call this function right before calling their printing function. If the application is an archiving or document management application that runs in the background and waits for other documents to be printed, this function can be called from the EnabledPre event that is fired by the printer. A detailed sample is provided under the Printer Driver Events section of this manual.

Example

```
Public cdi As CDIntfEx.CDIntfEx

Private Sub Form_Load()
    ' Case 1: the CDIntfEx object is created dynamically in memory
    Dim cdi As New CDIntfEx.CDIntfEx

    ' Case 2: the CDIntfEx object was placed on a form
    ' Set cdi = CDIntfEx1

    On Error GoTo printer_error
    ' install a new printer attach to existing printer
    cdi.PDFDriverInit "Amyuni Document Converter"

    ' First time activation of printer
    cdi.EnablePrinter "Evaluation Version Developer",
"07EFCDAB01000100584F829687E4DB671FE2564622F2C87EFC6FFF82B2F90206F7EEE87AE0751CAECA86FED02
07CD295E03629A5726433B6E3BE1766876E2B5237F8F5"

    ' set printer as system default
    cdi.SetDefaultPrinter
```

```
    Exit Sub

printer_error:
    MsgBox "Sorry, could not install printer"
End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' restore default printer before removing our printer
    cdi.RestoreDefaultPrinter
    ' close the printer handle and remove printer
    cdi.DriverEnd
    Set cdi = Nothing
End Sub

Private Sub Print_Click()
    ' activate printer before printing
    cdi.EnablePrinter "Evaluation Version Developer",
"07EFCDAB01000100584F829687E4DB671FE2564622F2C87EFC6FFF82B2F90206F7EEE87AE0751CAECA86FED02
07CD295E03629A5726433B6E3BE1766876E2B5237F8F5"

    cdi.FileNameOptionsEx = 1 + 2       ' NoPrompt + UseFileName
    cdi.DefaultFileName = "c:\test.pdf" ' set output file name

    ' draw some text
    Printer.CurrentX = 200
    Printer.CurrentY = 400
    Printer.FontName = "Arial"
    Printer.Print "Hi There"

    Printer.EndDoc

    ' printing ended, no need to disactivate printer
    ' the printer will be disactivated automatically

End Sub
```

Attributes Property

The Attributes property can be used to modify or read the default attributes of an installed printer.

Syntax

```
Attributes As Long
```

Parameters

Attributes

[in, out] Printer attributes as defined by the Windows® operating system.

Remarks

Modifying the printer attributes requires administrative rights under Windows NT/2000/XP.

The list of attributes that can be set for a printer are defined in the MSDN documentation under the SetPrinter function.

Example

```

```

PrinterLanguage Property

The PrinterLanguage property can be used to modify or read the language used in the user interface of the printer.

Syntax

```
PrinterLanguage As Long
```

Parameters

PrinterLanguage

[in, out] User-interface language Id. nLang can be one of the following values:

0 default language set by the license key
1 English
2 French
3 German

Remarks

The license key provided with the product contains the default user-interface language that will be used after installing the product. PrinterLanguage can be used to modify the default value. The language can also be modified by the user from the printer configuration dialog-box.

Example

```
Private Sub Form_Load()
    ' Case 1: the CDIntfEx object is created dynamically in memory
    Dim cdi As New CDIntfEx.CDIntfEx

    ' Case 2: the CDIntfEx object was placed on a form
    ' Set cdi = CDIntfEx1

    On Error GoTo printer_error
    ' install a new printer attach to existing printer
    cdi.PDFDriverInit ("VB Test Application Converter")

    ' First time activation of printer
    cdi.EnablePrinter "Evaluation Version Developer",
"07EFCDABDC2244455F2F2C87EFC6FFF82B2F90206F7EEE87AE0751CAECA86FED0207CD295E03629A5726433B6
E3BE1766876E2B5237F8F5"

    ' set language to French
    cdi.PrinterLanguage = 2

    ' set printer as system default
    cdi.SetDefaultPrinter

    Exit Sub

printer_error:
    MsgBox "Sorry, could not install printer"
End Sub
```

Printer Configuration

```
CDIntfEx.DefaultDirectory
CDIntfEx.DefaultFileName
CDIntfEx.FileNameOptions, CDIntfEx.FileNameOptionsEx

CDIntfEx.PaperSize
CDIntfEx.PaperWidth
CDIntfEx.PaperLength
CDIntfEx.Orientation
CDIntfEx.Resolution
CDIntfEx.JPEGCompression
CDIntfEx.JPegLevel
CDIntfEx.FontEmbedding
CDIntfEx.HorizontalMargin, CDIntfEx.VerticalMargin
CDIntfEx.SetWatermark
CDIntfEx.ImageOptions
CDIntfEx.SimPostscript

CDIntfEx.PrinterParamStr, CDIntfEx.PrinterParamInt

CDIntfEx.SetDefaultConfig, CDIntfEx.SetDefaultConfigEx
```

CDIntfEx.DefaultDirectory Property

The DefaultDirectory property defines the default directory used to store the files generated by the Converter products.

Syntax

```
DefaultDirectory As String
```

Parameters

DefaultDirectory

[in, out] Default directory used to store output files.

Remarks

The directory can be either a local or a network directory. In both cases, the directory should exist and the users have right to write to this directory. The printer driver will not attempt to create the directory if it does not exist.

This setting is only used in the case where the output file name is defined by the printer driver and not by the user or developer, i.e. when the FileNameOptions contain the NoPrompt but not the UseFileName options.

Example

```
Private Sub Print_Click()

    cdi.FileNameOptionsEx = 1           ' NoPrompt
    cdi.DefaultDirectory = "c:\temp"    ' set output directory

End Sub
```

CDintfEx.DefaultFileName Property

The DefaultFileName property defines the destination file name for the PDF, RTF and Excel Converter products, or the root of the output file names for the DHTML and JPeg Converter products.

Syntax

```
DefaultFileName As String
```

Parameters

DefaultFileName

[in, out] Output file name.

Remarks

This setting is only used in the case where the output file name is defined by the developer and not by the user or the printer driver, i.e. when the FileNameOptions contain the NoPrompt and the UseFileName options.

The szFile parameter should contain both the destination directory and file name. The directory can be either a local or a network directory. In both cases, the directory should exist and the users have right to write to this directory. The printer driver will not attempt to create the directory if it does not exist.

In this case of the PDF, RTF and Excel converters, only one file is generated and the name of this file defined by the SetDefaultFileName call. In the case of the DHTML and JPeg Converters, multiple files can be generated from a single printout; in this case SetDefaultFileName defines the name of the main file, the other files being generated from the main file by appending numerical Ids at the end of each file.

Example

```
Private Sub Print_Click()

    cdi.FileNameOptionsEx = 1 + 2       ' NoPrompt + UseFileName
    cdi.DefaultFileName = "c:\test.pdf" ' set output file name

    ' draw some text
    Printer.CurrentX = 200
    Printer.CurrentY = 400
    Printer.FontName = "Arial"
    Printer.Print "Hi There"

    Printer.EndDoc

End Sub
```

CDIntfEx.FileNameOptions, CDIntfEx.FileNameOptionsEx Properties

The FileNameOptions and FileNameOptionsEx properties define a number of options used in the generation of the PDF, HTML, RTF, Excel and JPeg Converter products.

Syntax

```
FileNameOptions As Integer
FileNameOptionsEx As Long
```

Parameters

FileNameOptions

[in, out] Combination of options as defined below. This property has been superceded by FileNameOptionsEx but was kept for backward compatibility only.

FileNameOptionsEx

[in, out] Combination of options as defined below.

Remarks

Options supported by each product:

Option	Value (Hex)	Description	PDF	HTM	RTF	JPG	XL
NoPrompt	1	Prevents the display of the file name dialog box	x	x	x	x	x
UseFileName	2	Use the file name set by SetDefaultFileName as the output file name	x	x	x	x	x
Concatenate	4	Concatenate with existing file instead of overwriting. This is useful only if the NoPrompt option is set	x	x	x	x	x
DisableCompression	8	Disable deflate (zip) compression of the page's content	x				
EmbedFonts	10	Enable embedding of fonts used in the source document	x				
BroadcastMessages	20	Send notification messages for document generation progress to all running application	x	x	x	x	x
PrintWatermark	40	Watermarks are printed on all printed pages	x	x	x	x	x
MultilingualSupport	80	Add supports for international character sets	x			x	
EncryptDocument	100	Encrypt resulting document	x				
FullEmbed	200	Embed full fonts as opposed to embedding the fonts partially	x				
UseTcpIpServer	400	Reserved					
SendByEmail	800	Send the document by email	x		x	x	x
ConfirmOverwrite	1000	If the file exists, confirm before overwriting	x	x	x	x	x
AppendExisting	2000	If the file exists, append to existing file	x	x	x	x	x
AddDateTime	3000	If the file exists, add date and time to file name	x	x	x	x	x
AddIdNumber	4000	If the file exists, add ID number to file name	x	x	x	x	x
LinearizeForWeb	8000	Activate web optimisation (Linearization) of PDF document	x				
PostProcessing	10000	Post process file using specified application. The application's full path should be in the registry	x				
JpegLevelLow	20000	Low quality JPeg compression of 24-bit images. This is equivalent to level 2 in the user interface.	x	x	x	x	
JpegLevelMedium	40000	Medium quality JPeg compression of 24-bit images; this is equivalent to level 7 in the user interface	x	x	x	x	
JpegLevelHigh	60000	High quality JPeg compression of 24-bit images; this is equivalent to level 9 in the user interface	x	x	x	x	
Colors2GrayScale	80000	Replaces all colors by an equivalent gray scale value	x	x	x	x	
ConvertHyperlinks	100000	Convert text beginning with http or www to a hyperlink	x	x			
EmbedStandardFonts	200000	Embed standard fonts such as Arial, Times, ...	x				
EmbedLicensedFonts	400000	Embed fonts requiring a license	x				
Color256Compression	800000	Activate 256 color compression	x	x	x	x	
EmbedSimulatedFonts	1000000	Embed Italic or Bold fonts that do not have an associated font file but are simulated by the system	x				
SendToCreator	2000000	Send the PDF data directly to the Amyuni PDF Creator product instead of saving to disk	x				
ExportToHTML	4000000	Export document to HTML format		x			
ExportToRTF	8000000	Export document to RTF format			x		

ExportToJPEG	10000000	Export document to JPEG format				x	
CCITTCompression	20000000	Activate CCITT Fax Level 4 compression for B&W images	x				
EncryptDocument128	40000000	Use 128 bits encryption compatible with Adobe Acrobat® 5	x				
AutoImageCompression	80000000	Use automatic image compression, i.e. the best compression option for each image in a document	x	x	x	x	

The export to Excel is configured differently from the RTF, HTML or JPeg exports. To export to Excel, the developer should use the PrinterParam properties in addition to FileNameOptionsEx.

Example

```
Private Sub Print_Click()

    Dim cdi As New CDIntfEx.CDIntfEx

    ' attach to existing printer
    cdi.DriverInit "Amyuni RTF Converter"

    cdi.FileNameOptionsEx = 1 + 2       ' output file name defined by the application
    cdi.DefaultFileName = "c:\test.pdf" ' set output file name

    ' draw some text
    Printer.CurrentX = 200
    Printer.CurrentY = 400
    Printer.FontName = "Arial"
    Printer.Print "Hi There"

    Printer.EndDoc
    cdi.DriverEnd

End Sub
```

CDIntfEx.PaperSize Property

The PaperSize property is used to define the default output paper size. The default paper size is usually used by applications when creating a new document.

Syntax

```
PaperSize As Integer
```

Parameters

PaperSize

[in, out] Paper size identifier as defined by the Windows® operating system. The default value is either Letter or A4 depending on the country where the product is used.

Remarks

Sample PaperSize 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

These functions can be used in two situations:

1. The developer needs to modify the default values for the printer; in this case SetDefaultConfig should be called after modifying this setting to set the value as default for all applications. Application developers should not call this function for every printout but only after printer initialization.
2. The printer device context is created using the function CDICreateDC. This function uses the settings provided by these functions and there is no need to call SetDefaultConfig.

Example

```
Dim cdi As New CDIntfEx.CDIntfEx

' attach to existing printer
cdi.DriverInit "Amyuni PDF Converter"

cdi.PaperSize = 5      ' set default paper size to Legal
cdi.SetDefaultConfig   ' make value default for all apllications

cdi.DriverEnd
```

CDIntfEx.PaperWidth, CDIntfEx.PaperLength Properties

The PaperWidth and PaperLength properties are used to define custom output paper sizes.

Syntax

```
PaperWidth As Long
PaperLength As Long
```

Parameters

PaperWidth

[in, out] Paper width in 10^{th} of a millimeter. The default value is 1000 or 10 centimeters.

PaperLength

[in, out] Paper length in 10^{th} of a millimeter. The default value is 1000 or 10 centimeters.

Remarks

These functions can be used in two situations:

1. The developer needs to modify the default values for the printer; in this case SetDefaultConfig should be called after modifying these settings to set the value as default for all applications. Application developers should not call this function for every printout but only after printer initialization.
2. The printer device context is created using the function CDICreateDC. This function uses the settings provided by these functions and there is no need to call SetDefaultConfig.

These functions automatically modify the PaperSize value to 256 for 'Custom'.

Example

```
Dim cdi As New CDIntfEx.CDIntfEx

' attach to existing printer
cdi.DriverInit "Amyuni PDF Converter"

cdi.PaperWidth = 2000  ' set default paper width to 20 centimeters
cdi.PaperLength = 2000 ' set default paper length to 20 centimeters
cdi.SetDefaultConfig   ' make value default for all apllications

cdi.DriverEnd
```

CDIntfEx.Orientation Property

The Orientation property is used to define the default paper orientation. The default orientation is usually used by an application when creating a new document.

Syntax

```
Orientation As Integer
```

Parameters

Orientation

[in, out] Paper orientation.

Remarks

Orientation value:

Orientation	Orientation value
Portrait	1
Landscape	2

This property can be used in two situations:

1. The developer needs to modify the default values for the printer; in this case SetDefaultConfig should be called after modifying these settings to set the value as default for all applications. Application developers should not call this function for every printout but only after printer initialization.
2. The printer device context is created using the function CDICreateDC. This function uses the settings provided by these functions and there is no need to call SetDefaultConfig.

Example

```
Dim cdi As New CDIntfEx.CDIntfEx

' attach to existing printer
cdi.DriverInit "Amyuni PDF Converter"

cdi.PaperSize = 5      ' set default paper size to Legal
cdi.Orientation = 2    ' set paper orientation to Landscape
cdi.SetDefaultConfig   ' make value default for all apllications

cdi.DriverEnd
```

CDIntfEx.Resolution Property

The Resolution property is used to define the default printer resolution. This value is mainly used when outputting images but has an effect also on the quality of text output.

Syntax

```
Resolution As Long
```

Parameters

Resolution

[in, out] Printer resolution.

Remarks

Resolution values:

Orientation	Orientation value
72 Dots Per Inch	72
150 Dots Per Inch	150
300 Dots Per Inch	300
600 Dots Per Inch	600
1200 Dots Per Inch	1200

This property can be used in two situations:

1. The developer needs to modify the default values for the printer; in this case SetDefaultConfig should be called after modifying these settings to set the value as default for all applications. Application developers should not call this function for every printout but only after printer initialization.
2. The printer device context is created using the function CDICreateDC. This function uses the settings provided by these functions and there is no need to call SetDefaultConfig.

Example

```
Dim cdi As New CDIntfEx.CDIntfEx

' attach to existing printer
cdi.DriverInit "Amyuni PDF Converter"

cdi.PaperSize = 5      ' set default paper size to Legal
cdi.Orientation = 2    ' set paper orientation to Landscape
cdi.Resolution = 600   ' set output resolution to 600 DPI
cdi.SetDefaultConfig   ' make value default for all apllications

cdi.DriverEnd
```

CDIntfEx.JPEGCompression Property

The JPEGCompression property is used to activate or disactivate the JPeg compression option for 24 bits images. JPeg compression heavily reduces the size of documents containing real life images but has a slight effect on image quality.

Syntax

```
JPEGCompression As Boolean
```

Parameters

JPEGCompression

[in, out] This parameter should be True to set JPeg compression, False otherwise.

Remarks

This property can be used in two situations:

1. The developer needs to modify the default values for the printer; in this case SetDefaultConfig should be called after modifying these settings to set the value as default for all applications. Application developers should not call this function for every printout but only after printer initialization.
2. The printer device context is created using the function CDICreateDC. This function uses the settings provided by these functions and there is no need to call SetDefaultConfig.

When the JPeg compression option and JPeg level need to be changed for a specific printout and not set as default for all printouts, the FileNameOptions property provides a more efficient way to set JPeg compression.

Example

```
Dim cdi As New CDIntfEx.CDIntfEx

' attach to existing printer
cdi.DriverInit "Amyuni PDF Converter"

cdi.Resolution = 600            ' set output resolution to 600 DPI
cdi.JPEGCompression = True      ' activate JPeg image compression
cdi.SetDefaultConfig            ' make value default for all apllications

cdi.DriverEnd
```

CDIntfEx.JPegLevel Property

The JPegLevel property is used to set or read the level of JPeg compression for 24 bits images. JPeg compression heavily reduces the size of documents containing real life images but has a slight effect on image quality. The level of compression from 1 to 9 determines if the printer should output highly compressed low quality images or good quality images with reduced compression.

Syntax

```
JPegLevel As Integer
```

Parameters

JPegLevel

[in, out] This parameter can vary from 1 to 9, the default value is 7 for good quality with medium compression.

Remarks

This property can be used in two situations:

1. The developer needs to modify the default values for the printer; in this case SetDefaultConfig should be called after modifying these settings to set the value as default for all applications. Application developers should not call this function for every printout but only after printer initialization.
2. The printer device context is created using the function CDICreateDC. This function uses the settings provided by these functions and there is no need to call SetDefaultConfig.

When the JPeg compression option and JPeg level need to be changed for a specific printout and not set as default for all printouts, the SetFileNameOptions function provides a more efficient way to set JPeg compression.

Sample JPeg Level values:

JPeg Level	JPeg level value
Low quality, High compression	3
Good quality, Good compression	7
High quality, Low compression	9

Example

```
Dim cdi As New CDIntfEx.CDIntfEx

' attach to existing printer
cdi.DriverInit "Amyuni PDF Converter"

cdi.Resolution = 600            ' set output resolution to 600 DPI
cdi.JPEGCompression = True      ' activate JPeg image compression
cdi.JPegLevel = 3               ' activate high compression to reduce file size
cdi.SetDefaultConfig            ' make value default for all apllications

cdi.DriverEnd
```

CDIntfEx.FontEmbedding Property

The FontEmbedding property is used to activate or disactivate the TrueType® font embedding features of the PDF Converter product. These settings have no effect on the other Converter products. Font embedding can be used in the PDF files to ensure portability among various systems.

Syntax

```
FontEmbedding As Boolean
```

Parameters

FontEmbedding

[in, out] This parameter should be True to set font embedding, False otherwise.

Remarks

This property can be used in two situations:

1. The developer needs to modify the default values for the printer; in this case SetDefaultConfig should be called after modifying these settings to set the value as default for all applications. Application developers should not call this function for every printout but only after printer initialization.
2. The printer device context is created using the function CDICreateDC. This function uses the settings provided by these functions and there is no need to call SetDefaultConfig.

When the font embedding option needs to be changed for a specific printout and not set as default for all printouts, the FileNameOptions property provides a more efficient way to set font embedding.

Example

CDIntfEx.HorizontalMargin, CDIntfEx.VerticalMargin Properties

The HorizontalMargin and VerticalMargin properties are used to set or read the minimum printer margins. Applications cannot print below the minimum margins defined by the printer. The Converter products being virtual printers, the minimum margins can be set to 0, this might cause some clipping if the generated document is later printed to a physical printer. The default value is set to 6 millimeters in order to accomodate most hardware printers available in the market.

Syntax

```
HorizontalMargin As Integer
VerticalMargin As Integer
```

Parameters

HorizontalMargin

[in, out] Minimum horizontal printer margin in 10^{th} of a millimeter unit.

VerticalMargin

[in, out] Minimum vertical printer margin in 10^{th} of a millimeter unit.

Remarks

These properties can be used in two situations:

1. The developer needs to modify the default values for the printer; in this case SetDefaultConfig should be called after modifying these settings to set the value as default for all applications. Application developers should not call this function for every printout but only after printer initialization.
2. The printer device context is created using the function CDICreateDC. This function uses the settings provided by these functions and there is no need to call SetDefaultConfig.

These settings define the minimum margin below which an application cannot print. Applications usually define their own margin settings and send a warning to the user whi attempts to set margins lower than the printer's minimum value.

The default value for these properties is 60, or 6.0 millimeters.

Example

CDIntfEx.SetWatermark Method

The SetWatermark method is used to configure the printer to print a watermark message on all pages of a document. This method can be used to set only the simple text watermarks and cannot be used to set the watermark option to "External PDF File".

Syntax

```
Function SetWatermark(Watermark As String, FontName As String, FontSize As Integer, Orientation As Integer,
                      Color As Long, HorzPos As Long, VertPos As Long, Foreground As Long) As Long
```

Parameters

Watermark
[in] Text to print on each page.

FontName
[in] Font name used to print text.

FontSize
[in] Font size in 0.1 inch units.

Orientation
[in] Text orientation in 0.1 degree units.

Color
[in] RGB value for watermark color.

HorzPos
[in] Horizontal text position in 0.1 inch units. This a positive value measured from the left of the page and should take into account the minimum printer margin.

VertPos
[in] Vertical text position in 0.1 inch units. This a negative value measured from the top of the page and should take into account the minimum printer margin.

Foreground
[in] flag that indicates whether the watermark should be above or below the page content.

Return Value

SetWatermark returns 0 if successful, or a Windows specific error code if it fails.

Remarks

This method only sets the watermark parameters for the printer, to actually print watermarks on a document, the option PrintWatermark (Hex 40) should be added to the FileNameOptions property.

This method can be called anytime before or while printing a document to modify the watermarks settings.

To print watermark on specific pages only, e.g. the first page only, the printer events can be captured and the watermark parameters modified while the document is being printed.

Example

```
Private Sub Print_Click()

    ' print an MS Word document using OLE Automation
    ' also print a DRAFT watermark on the first page only of the document

    Dim wordApp As New Word.Application
    Dim documents As Word.documents

    ' open a Word document in Read-only mode
    Set documents = wordApp.documents
    documents.Open "c:\test.doc", False, True

    ' set the ActivePrinter to ours
    savePrinter = wordApp.ActivePrinter
    wordApp.ActivePrinter = PrinterName

    ' set watermark parameters
    cdi.SetWatermark " D  R  A  F  T ", "Verdana", 48, 450, &HFF00FF, 120, -120, True
```

```
    ' NoPrompt + UseFileName + BroadcastMessage + EnableWatermarks
    cdi.FileNameOptionsEx = &H1 + &H2 + &H20 + &H40
    cdi.DefaultFileName = "c:\test.pdf"      ' set output file name

    ' activate printer before printing
    cdi.EnablePrinter "Evaluation Version",
"07EFCDAB01000100C71720CBDBA4ABF6AA49DDF7E8AA44A6575449D"

    ' print the Word document without background printing
    wordApp.PrintOut False

    ' reset options after printing
    cdi.FileNameOptionsEx = 0

    ' restore printer and close MS Word
    wordApp.ActivePrinter = savePrinter
    wordApp.Quit False
    Set documents = Nothing
    Set wordApp = Nothing

End Sub

Private Sub CDIntfEx1_EndPage(ByVal JobID As Long, ByVal hDC As Long)
    ' remove watermarks after the first page is printed
    cdi.FileNameOptionsEx = &H1 + &H2 + &H20 ' NoPrompt + UseFileName + BroadcastMessage
End Sub

Private Sub CDIntfEx1_EndDocPost(ByVal JobID As Long, ByVal hDC As Long)
    ' finished printing a new document
    List1.AddItem "New document printed..."
End Sub
```

CDIntfEx.ImageOptions Property

The ImageOptions property is used to set or read additional image conversion options.

Syntax

```
ImageOptions As Long
```

Parameters

ImageOptions

[in, out] Additional options for image conversion.

Remarks

Values for ImageOptions:

Description	Value
Remove Duplicate Images	1
Downsample high-resolution images	2

The options can be combined using the Addition or the OR operators.

The call should be followed by a call to SetDefaultConfig or SetDefaultConfigEx for these settings to take effect.

Example

```
Dim cdi As New CDIntfEx.CDIntfEx

' attach to existing printer
cdi.DriverInit "Amyuni PDF Converter"

cdi.JPEGCompression = True      ' activate JPeg image compression
cdi.JPegLevel = 3               ' activate high compression to reduce file size
cdi.ImageOptions = 2            ' downsampling adjusts the image to output resolution
cdi.SetDefaultConfig            ' make value default for all apllications

cdi.DriverEnd
```

CDIntfEx.SimPostscript Property

The SimPostscript property is used to set or read the Postscript Simulation option in the printer driver. Some operations are not allowed on Postscript and PDF printers because on these printers the application cannot read from the destination device. Some applications query the printer and modify their drawing operations if the printer is a Postscript printer. This option is useful for printing Wordarts using Office XP.

Syntax

```
SimPostscript As Boolean
```

Parameters

SimPostscript

[in] Flag that indicates if Postscript simulation is On or Off.

Remarks

The call should be followed by a call to SetDefaultConfig or SetDefaultConfigEx for the setting to take effect.

Example

```
Dim cdi As New CDIntfEx.CDIntfEx

' attach to existing printer
cdi.DriverInit "Amyuni PDF Converter"

cdi.SimPostscript = True        ' enable PostScript simulation
cdi.ImageOptions = 2            ' downsampling adjusts the image to output resolution
cdi.SetDefaultConfig            ' make value default for all apllications

cdi.DriverEnd
```

CDIntfEx.PrinterParamStr, CDIntfEx.PrinterParamInt Properties

The PrinterParamStr and PrinterParamInt properties are used to set or read custom printer parameters. The printer parameters depend on each type of Document Converter Printer and can be either in string or long integer format.

Syntax

```
PrinterParamStr(Param As String) As String
PrinterParamInt(Param As String) As Long
```

Parameters

Param

[in] Name of parameter to set or read.

PrinterParamStr

[in, out] String value of the parameter if the parameter is of type string.

PrinterParamInt

[in, out] Long integer value of the parameter if the parameter is of type integer.

Remarks

For more details about the meaning of each of these parameters, refer to the user's manual for the product in question.

There is no need to call SetDefaultConfig after modiying these parameters as they are immediately taken into account by all applications.

Custom parameters for the RTF Converter:

Parameter	Type	Description	Values
RTF Format	Integer	Option for formatting the RTF output	0 – Advanced RTF 1 – Full RTF 2 – Formatted Text 3 – Non-formatted Text Only
Use Tabs	Integer	Option to replace tabs by spaces	0 – Do not replace 1 – Replace

Custom parameters for the DHTML Converter:

Parameter	Type	Description	Values
HTML MultiPage	Integer	Option for formatting the HTML output	1 – Use Layers 2 – Single Page HTML 3 – Multiple HTML files

Custom parameters for the JPEG Converter:

Parameter	Type	Description	Values
JPEG Resolution	Integer	Image resolution at which the document is converted to JPeg	0 – 75 DPI 1 – 150 DPI 2 – 300 DPI 3 – 600 DPI
JPEG Compression	Integer	JPeg compression level. The higher the level, the better the quality and larger the file size.	1 to 9

Custom parameters for the Excel Converter:

Parameter	Type	Description	Values
EXCEL MultiSheets	Integer	Excel output option	0 – No Excel output 16 – All document pages on a single sheet 17 – One excel sheet per document page

Example 1

```
Dim cdi As New CDIntfEx.CDIntfEx

cdi.DriverInit "My HTML Converter"  ' attach to existing printer

' create HTML document containing one layer for every page
cdi.PrinterParamInt("HTML MultiPage") = 1

cdi.DriverEnd
```

Example 2

```
Dim cdi As New CDIntfEx.CDIntfEx

cdi.DriverInit "My RTF Converter"   ' attach to existing printer

' create RTF document containing with objects not positioned in frames
cdi.PrinterParamInt("RTF Format ") = 1

cdi.DriverEnd
```

Example 3

```
Dim cdi As New CDIntfEx.CDIntfEx

cdi.DriverInit "My JPeg Converter"  ' attach to existing printer

' create compact JPeg files at low resolution
cdi.PrinterParamInt("JPeg Resolution") = 2          ' 150 DPI
cdi.PrinterParamInt("JPeg Compression") = 3         ' high compression

cdi.DriverEnd
```

Example 4

```
Dim cdi As New CDIntfEx.CDIntfEx

cdi.DriverInit "My Excel Converter"                 ' attach to existing printer

' create an Excel file with one sheet per page
cdi.PrinterParamInt("Excel MultiSheets") = 17

cdi.DriverEnd
```

CDIntfEx.SetDefaultConfig, CDIntfEx.SetDefaultConfigEx Methods

The SetDefaultConfig and SetDefaultConfigEx methods are used set the current printer configuration as default for all applications. These calls are need when modifying some of the printer properties so that applications can take these properties into account. The document of all printer parameters or properties specifies when this call is needed.

Syntax

```
Function SetDefaultConfig() As Boolean
Function SetDefaultConfigEx() As Boolean
```

Parameters

Return Value

SetDefaultConfig and SetDefaultConfigEx return True if successful, False otherwise.

Remarks

The SetDefaultConfig method posts a Windows message to all running applications, including the application that calls this function, to notify the applications that the printer settings have changed. The notification can take a few seconds if there are a number of applications running. SetDefaultConfigEx does the same as SetDefaultConfig but without the notification. It can be useful when modifying the printer settings when installing the application or before the application is launched.

Example

```
Dim cdi As New CDIntfEx.CDIntfEx

cdi.PDFDFDriverInit( "My Application Converter" );   ' create a new printer
With cdi
  .FontEmbedding = True             ' embed TrueType fonts with the document
  .JPEGCompression = True           ' activate JPEG compression
  .JpegLevel = 3                    ' low quality but high compression level
  .SetDefaultConfig                 ' make the values default
End With
```

Email Fucntions

```
SetEmailFieldFrom
SetEmailFieldTo
SetEmailFieldCC
SetEmailFieldBCC
SetEmailSubject
SetEmailMessage
SetEmailPrompt
SetEmailOptions, GetEmailOptions

SetSmtpServer, SetSmtpPort

SendMail, SendMailW
SendSmtpMail, SendSmtpMailW
```

SetEmailFieldFrom, SetEmailFieldTo, SetEmailFieldCC, SetEmailFieldBCC, SetEmailSubject, SetEmailMessage, SetEmailPrompt Functions

The SetEmailFieldFrom, SetEmailFieldTo, SetEmailFieldCC, SetEmailFieldBCC, SetEmailSubject, SetEmailMessage and SetEmailPrompt functions are used to set the various email parameters that can be found in the Destination property tab of the Amyuni Converter products.

Syntax

```
int SetEmailFieldFrom(HANDLE hPrinter, LPCSTR szEmailFieldFrom);
int SetEmailFieldTo( HANDLE hPrinter, LPCSTR szEmailFieldTo);
int SetEmailFieldCC( HANDLE hPrinter, LPCSTR szEmailFieldCC);
int SetEmailFieldBCC( HANDLE hPrinter, LPCSTR szEmailFieldBCC);
int SetEmailSubject( HANDLE hPrinter, LPCSTR szEmailSubject);
int SetEmailMessage( HANDLE hPrinter, LPCSTR szEmailMessage);
int SetEmailPrompt( HANDLE hPrinter, BOOL bEmailPrompt);
```

Parameters

szEmailFieldFrom
: [in] Name of the email sender as it shows to the person receiving the email. This parameter is used only in the case of sending emails through SMTP.

szEmailFieldTo
: [in] Name and email address of the email destinator. Multiple addresses can be specified by separating them with semi-colons (;).

szEmailFieldCC
: [in] Name and email address of the email Carbon Copy list. Multiple addresses can be specified by separating them with semi-colons (;).

szEmailFieldBCC
: [in] Handle Name and email address of the email Blind Carbon Copy list. Multiple addresses can be specified by separating them with semi-colons (;).

szEmailSubject
: [in] Subject of the email.

szEmailMessage
: [in] Email message.

bEmailPrompt
: [in] If set to True, the user is prompted with the message details before the email is sent. This parameter is used only in the case of sending emails through MAPI.

Return Value

These functions return 1 if successful, 0 otherwise.

Remarks

Emails can be sent using either MAPI compliant email systems, or directly using SMTP protocol without the use of any installed email client.

Example

```
Check the SetEmailOptions function for a complete sample
```

SetEmailOptions, GetEmailOptions Functions

The SetEmailOptions and SetEmailOptions are used to set or read options specific to sending emails through the Amyuni Converter products.

Syntax

```
long SetEmailOptions( HANDLE hPrinter, long nEmailOptions );
long GetEmailOptions( HANDLE hPrinter );
```

Parameters

nEmailOptions

[in] Options specific to sending emails. The list of supported options is in the Remarks section below.

Return Value

The SetEmailOptions function returns 1 if successful, 0 otherwise. The GetEmailOptions function returns the options currently set in the printer.

Remarks

Emails can be sent using either MAPI compliant email systems, or directly using SMTP protocol without the use of any installed email client.

These functions supecede the SetEmailPrompt function as they provide some additional options.

nEmailOptions values:

Option	Option value
Prompt before sending email	2
Send emails using SMTP	4

Example

```
void CDLLTestDlg::OnPrint()
{
  HFONT    font, oldFont;
  HDC      dc;
  DOCINFO  di;

  // create a printer device context
  dc = CreateDC( "winspool", theApp.m_szPrinter, NULL, NULL );
  if ( NULL == dc )
  {
          ASSERT( FALSE );
          return;
  }

  // init the DOCINFO structure
  memset( &di, 0, sizeof(di) );
  di.cbSize = sizeof( di );
  di.lpszDocName = "Test document";
  di.lpszOutput = NULL;

  // activate printer before starting to print
  EnablePrinter( theApp.m_converter, "Evaluation Version Developer",
  "07EFCDAB01000100584F8296BFE4DB671FE256462D679B50F9AEB6BDBA542926639B79ECC556F310DB8592F1
C68BF146A4D4E86EB8C0E7824CE83E1D295E03629A5726433B6E3BE1766876E2B5237F8F5" );

  // create an RTF document and send it by email
  SetFileNameOptions( theApp.m_converter, NoPrompt + ExportToRTF + SendByEmail );

  // create RTF files with no frames
```

```
  SetPrinterParamInt( theApp.m_converter, "RTF Format", 2 );

  // set email parameters to automatically send document by email
  SetEmailFieldTo( theApp.m_converter, "info@amyuni.com" );
  SetEmailFieldCC( theApp.m_converter, "support@amyuni.com" );
  SetEmailSubject( theApp.m_converter, "Testing email capabilities" );
  SetEmailMessage( theApp.m_converter, "Please find attached the requested document\nin RTF
format." );
  SetEmailOptions( theApp.m_converter, SMO_PROMPT_RECIPIENT );

  // start printing
  StartDoc( dc, &di );
  StartPage( dc );
  MoveToEx( dc, 100, 100, NULL );
  LineTo( dc, 400, 100 );
  MoveToEx( dc, 100, 100, NULL );
  LineTo( dc, 100, 400 );

  font = CreateFont( -24, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, _T("Verdana") );
  oldFont = (HFONT)SelectObject( dc, font );
  TextOut( dc, 100, 100, _T("Hi There"), 8 );
  if ( oldFont ) SelectObject( dc, oldFont );
  DeleteObject( font );

  EndPage( dc );

  EndDoc( dc );

  DeleteDC( dc );

  // printing ended, no need to disactivate printer
  // the printer will be disactivated automatically
}
```

SetSmtpServer, SetSmtpPort Functions

The SetSmtpServer and SetSmtpPort functions are used to set the server address and port number of the server used to send emails. These parameters are needed only when sending emails through direct SMTP without the use of MAPI.

Syntax

```
int SetSmtpServer( HANDLE hPrinter, LPTSTR szSmtpServer );
int SetSmtpPort( HANDLE hPrinter, long lSmtpPort );
```

Parameters

szSmtpServer

[in] Name or IP address of server used to send emails.

lSmtpPort

[in] SMTP port number on server used to send emails. The default value is 25.

Return Value

The SetSmtpServer and SetSmtpPort functions return 1 if successful, 0 otherwise.

Remarks

Emails can be sent using either MAPI compliant email systems, or directly using SMTP protocol without the use of any installed email client.

Example

```
// set email parameters to automatically send document by email using SMTP
SetEmailFieldFrom( theApp.m_converter, "test@amyuni.com" );
SetEmailFieldTo( theApp.m_converter, "info@amyuni.com" );
SetEmailFieldCC( theApp.m_converter, "support@amyuni.com" );
SetEmailSubject( theApp.m_converter, "Testing email capabilities" );
SetEmailMessage( theApp.m_converter, "Please find attached the requested document." );
SetSmtpServer(  theApp.m_converter, "smtp10.bellnet.ca" );
SetSmtpPort(  theApp.m_converter, 25 );
SetEmailOptions( theApp.m_converter, SMO_SMTP_MAIL );
```

SendMail, SendMailW Functions

The SendMail function is used to send a message with one or more attachments using MAPI email. The SendMailW function is the unicode equivalent of the same function.

Syntax

```
long SendMail( LPCSTR szTo, LPCSTR szCC, LPCSTR szBCC, LPCSTR szSubject, LPCSTR szMessage, LPCSTR
                szFilenames, long lOptions);
long SendMailW( LPCWSTR szTo, LPCWSTR szCC, LPCWSTR szBCC, LPCWSTR szSubject, LPCWSTR szMessage, LPCWSTR
                szFilenames, long lOptions );
```

Parameters

szTo
: [in] Name and email address of the email destinator. Multiple addresses can be specified by separating them with semi-colons (;).

szCC
: [in] Name and email address of the email Carbon Copy list. Multiple addresses can be specified by separating them with semi-colons (;).

szBCC
: [in] Handle Name and email address of the email Blind Carbon Copy list. Multiple addresses can be specified by separating them with semi-colons (;).

szSubject
: [in] Subject of the email.

szMessage
: [in] Email message.

szFileNames
: [in] Series of file names and their captions as they should appear in the email attachment. The file name is the full path of the file to send, the caption is the name of the file as seen by the recipient. The syntax for sending multiple files is as follows: file1;caption1;file2;caption2;....

lOptions
: [in] Options specific to sending emails. The list of supported options is in the Remarks section below.

Return Value

The SendMail function returns the value MAPI_E_NOT_SUPPORTED if no MAPI compliant email is installed in the system. It returns 0 if successful or a MAPI specific error code if an error occurs when sending the email.

Remarks

lOptions values:

Option	Option value
Prompt before sending email	2

Depending on the security settings of the system from which the email is sent, the users might be prompted with a confirmation message stating that an external application is trying to send an email on their behalf. This is an important MAPI security measure which cannot and should not be overridden.

Example

```
// send multiple files using MAPI email without prompting the user
SendMail( "info@amyuni.com", "support@amyuni.com", "", "Testing email capabilities",
        "Please find attached the requested document.",
        "c:\\temp1.pdf;document1.pdf;c:\\temp2.pdf;document2.pdf", 2
        );
```

SendSmtpMail, SendSmtpMailW Functions

The SendSmtpMail function is used to send a message with one or more attachments using direct SMTP email. The SendSmtpMailW function is the unicode equivalent of the same function.

Syntax

```
long SendSmtpMail( LPCSTR szHostname, long lPort, LPCSTR szFrom, LPCSTR szTo, LPCSTR szCC, LPCSTR szBCC,
                   LPCSTR szSubject, LPCSTR szMessage, LPCSTR szFilenames, long lOptions );
long SendSmtpMailW( LPCWSTR szHostname, long lPort, LPCWSTR szFrom, LPCWSTR szTo, LPCWSTR szCC, LPCWSTR
                   szBCC, LPCWSTR szSubject, LPCWSTR szMessage, LPCWSTR szFilenames, long lOptions );
```

Parameters

szHostName
: [in] Name or IP address of server used to send emails.

lPort
: [in] SMTP port number on server used to send emails. The default value is 25.

szFrom
: [in] Name of the email sender as it shows to the person receiving the email.

szTo
: [in] Name and email address of the email destinator. Multiple addresses can be specified by separating them with semi-colons (;).

szCC
: [in] Name and email address of the email Carbon Copy list. Multiple addresses can be specified by separating them with semi-colons (;).

szBCC
: [in] Handle Name and email address of the email Blind Carbon Copy list. Multiple addresses can be specified by separating them with semi-colons (;).

szSubject
: [in] Subject of the email.

szMessage
: [in] Email message.

szFileNames
: [in] Series of file names and their captions as they should appear in the email attachment. The file name is the full path of the file to send, the caption is the name of the file as seen by the recipient. The syntax for sending multiple files is as follows: file1;caption1;file2;caption2;....

Return Value

The SendSmtpMail function returns 0 if successful or an SMTP specific error code if an error occurs when sending the email.

Remarks

Example

```
// send multiple files using SMTP email
SendSmtpMail( "smtp.amyuni.com", 25, "test.amyuni.com", "info@amyuni.com",
                "support@amyuni.com", "", "Testing email capabilities",
                "Please find attached the requested document.",
                "c:\\temp1.pdf;document1.pdf;c:\\temp2.pdf;document2.pdf"
                );
```

Print Job Locking Functions

```
Lock
Unlock
SetDocFileProps
```

CDIntfEx.Lock Method

The Lock method can be used in multi-threading situations to avoid conflicts between multiple applications or multiple threads requesting simultaneous access the Converter products. The CDIntf library uses the registry to interact with the printer drivers. This can cause conflicts when multiple applications use CDIntf to access the printer drivers.

Syntax

```
Function Lock(szLockName As String) As Long
```

Parameters

szLockName

[in] Lock identifier, this should be the document title as it appears in the print spooler when printing any document.

Return Value

The return value is 0 if the function is successful, or a Windows specific error code if the function fails.

Remarks

The Lock method is only needed for applications using CDIntf to set the destination file name and options. The technical notes on www.amyuni.com provide alternative ways for using the Document Converter products in multi-threading situations without the need for CDIntf or the Lock functions.

When the Lock function is used, the output file name and options are set using the SetDocFileProps function and not the more common DefaultFileName and FileNameOptions properties.

Job locking is needed in the following scenario:

4. Application or Thread A uses DefaultFileName or DefaultDirectory to set the output file name before starting to print
5. Application B also also uses DefaultFileName or DefaultDirectory to set the output file name
6. Application A starts to print, but prints to the file set by application B and not application A

To solve this issue, the printer driver should be locked for the few microseconds it takes for application A from the time it sets the output file name, to the time it starts to print. As soon as the Application A starts to print, the lock can be released to allow Application B to print in parallel.

Example

```
Private Sub PrintWord_Click()
    ' this method opens and prints a Word document using Automation
    ' it uses the Locking mechanism of CDIntf to avoid multi-threading conflicts

    Dim fileName As String

    ' the lock name in the case of MS Word is the same as the file name
    Dim LockName As String
    LockName = "test.doc"

    Dim wordApp As New Word.Application
    Dim documents As Word.documents

    On Error GoTo unlock_printer

    ' open a Word document in Read-only mode
    Set documents = wordApp.documents
    documents.Open "c:\wutemp\test.doc", False, True

    ' set the ActivePrinter to ours
```

```
    wordApp.ActivePrinter = PrinterName

    ' lock printer before starting to print
    cdi.Lock LockName
    fileName = "c:\test" & index & ".pdf"
    index = index + 1
    cdi.SetDocFileProps LockName, 1 + 2, "", fileName
                         ' output file name defined by our application
                         ' default directory is not used
                         ' set the output file name to "testN.pdf"
    ' activate printer before printing
    cdi.EnablePrinter "Evaluation Version Developer",
"07EFCDAB0100018DC2244455F2F2C87EFC6FFF82B2F90206F7EEE87AE0751CAECA86FED0207CD295E03629A57
26433B6E3BE1766876E2B5237F8F5"
    ' print the Word document without background printing
    wordApp.PrintOut False

unlock_printer:
    ' the printer is unlocked automatically as soon as it starts to print
    ' Unlock is needed only in the case an error occurs while printing
    cdi.Unlock LockName, 1000

    wordApp.Quit False
    Set documents = Nothing
    Set wordApp = Nothing

End Sub
```

CDIntfEx.Unlock Method

The Unlock method can be used in multi-threading situations to avoid conflicts between multiple applications or multiple threads requesting simultaneous access the Converter products. The CDIntf library uses the registry to interact with the printer drivers. This can cause conflicts when multiple applications use CDIntf to access the printer drivers.

Syntax

```
Function Unlock(szLockName As String, dwTimeout As Long) As Long
```

Parameters

szLockName
: [in] Lock identifier, this should be the document title as it appears in the print spooler when printing any document.

dwTimeout
: [in] Timeout in milliseconds after which the function returns.

Return Value

The return value is 0 if the function is successful, or a Windows specific error code if the function fails.

Remarks

The Unlock method is used after printing has ended to make sure another printout can start. The call to Unlock is needed only in the case where an error occurs, the printer will otherwise call Unlock internally as soon as printing starts to allow another printout to occur in parallel.

Example

```
See the CDIntfEx.Lock method for a complete sample.
```

CDIntfEx.SetDocFileProps Method

The SetDocFileProps method can be used in multi-threading situations to avoid conflicts between multiple applications or multiple threads requesting simultaneous access the Converter products. The CDIntf library uses the registry to interact with the printer drivers. This can cause conflicts when multiple applications use CDIntf to access the printer drivers.

Syntax

```
Function SetDocFileProps(szDocTitle As String, lOptions As Long, szFileDir As String,
                         szFileName As String) As Long
```

Parameters

szDocTitle
: [in] The document title as it appears in the print spooler when printing any document, this should be the same as the parameter szLockName used for Lock and Unlock.

lOptions
: [in] Output file name options equivalent to the function SetFileNameOptions.

szFileDir
: [in] Destination directory, equivalent to the SetDefaultDirectory function call.

szFileName
: [in] Output file name, equivalent to the function SetDefaultFileName.

Return Value

The return value is 0 if the function is successful, or a Windows specific error code if the function fails.

Remarks

This method replaces the properties DefaultDirectory, DefaultFileName and FileNameOptions in the cases where print job locking is needed. The documentation for these three functions contains a complete description of the lOptions, szFileDir and szFileName parameters.

Example

```
See the CDIntfEx.Lock method for a complete sample.
```

```
Document.Title
Document.Subject
Document.Creator
Document.Author
Document.KeyWords
Document.PageMode
Document.Rotate
Document.PageCount

Document.Open, Document.OpenEx
Document.Save
Document.Append, Document.AppendEx
Document.Merge, Document.MergeEx

Document.ClearBookmarks
Document.SetBookmark
Document.SearchText

Document.Encrypt, Document.Encrypt128
Document.Linearized

Document.Optimize
Document.ExportToRTF
Document.ExportToHTML
Document.ExportToEXCEL
Document.ExportToJPEG

Document.Print

Document.SetLicenseKey
```

Document.Title, Document.Subject, Document.Creator, Document.Author, Document.KeyWords Properties

The Title, Subject, Creator, Author and KeyWords properties can be defined by the user or developer to identify a PDF document.

Syntax

```
Title As String
Subject As String
Creator As String
Author As String
KeyWords As String
```

Parameters

Title
: [in, out] Document title.

Subject
: [in, out] Document subject.

Creator
: [in, out] Application used to create the PDF document.

Author
: [in, out] Person who wrote the document.

KeyWords
: [in, out] List of keywords separated by semi-colons (;).

Remarks

All these properties are optional are not found in all PDF documents.

Example

Document.PageMode Property

The PageMode property determines how the document should be displayed when opened using Acrobat Reader®.

Syntax

```
PageMode As String
```

Parameters

PageMode

[in, out] String value representing the way the document should be opened. The list of possible values is in the remarks section

Remarks

Possible values for the PageMode property as of the PDF specifications 1.4:

UseNone	Neither document outline nor thumbnail images visible
UseOutlines	Document outline visible
UseThumbs	Thumbnail images visible
FullScreen	Full-screen mode, with no menu bar, window controls, or any other window visible

Example

Document.Rotate Property

The Rotate property determines the default rotation angle for all pages in a document.

Syntax

```
Rotate As Long
```

Parameters

PageMode

[in, out] Rotation angle of 0, 90 or -90 degrees.

Remarks

The page rotation can be overridden for each page separately, this property is taken into account only for pages that do not have this value overridden.

Example

```
```

Document.PageCount Method

The PageCount returns the number of pages in a PDF document.

Syntax

```
Function PageCount() As Long
```

Parameters

Remarks

Example

```
Dim document As New CDIntfEx.Document
Document.Open "c:\test.pdf"
MsgBox "test.pdf has " & document.PageCount & " pages
Set document = Nothing
```

Document.Open, Document.OpenEx Methods

The Open and OpenEx methods open a PDF file for processing. The OpenEx version can be used with password protected documents and will fail if the password is invalid.

Syntax

```
Function Open(FileName As String) As Boolean
Function OpenEx(FileName As String, Password As String) As Boolean
```

Parameters

FileName
: [in] Full path of the PDF file to open.

Password
: [in] Owner or user password associated with the document.

Return Value

The return value is True if the file was opened. If the Open or OpenEx method fails, an exception is raised and there is no return value.

Remarks

Opening password encrypted files is possible only with the professional version of the Amyuni PDF Converter. Some operations are restricted when the document is opened using the user password as opposed to the owner password.

Example

```
    ' create an object of type Document
    Dim doc As New CDIntfEx.Document

    On Error Goto invalid_file

    ' activate advanced functions
    doc.SetLicenseKey "Evaluation Version Developer",
"07EFCDAB01000100584F55F2F2C87EFC6FFF82B2F90206F7EEE87AE0751CAECA86FED0207CD295E03629A5726
433B6E3BE1766876E2B5237F8F5"

    ' open existing document
    doc.OpenEx "c:\test.pdf", "owner"

    ' modify the password
    doc.Encrypt "owner", "user", -6

    ' save document to some other file
    doc.Save "c:\encrypted.pdf"

invalid_file:
    Set doc = Nothing
```

Document.Save Method

The Save method saves a modified PDF document.

Syntax

```
Function Save(FileName As String) As Boolean
```

Parameters

FileName

[in] Full path of the PDF file to save.

Return Value

The return value is True if the file was save. If the Save method fails, an exception is raised and there is no return value.

Remarks

Example

```
See the Open method for sample code
```

Document.Append, Document.AppendEx Methods

The Appen and AppendEx methods append or concatenate a second PDF file to a first one.

Syntax

```
Function Append(FileName As String) As Boolean
Function AppendEx(Document As Object) As Boolean
```

Parameters

FileName
: [in] Full path of the second PDF file to append to the current one.

Document
: [in] Object reference of the second PDF file to append to the current one.

Return Value

The return value is True if the file was appended. If the Append or AppendEx method fails, an exception is raised and there is no return value.

Remarks

Append is used when the file to be appended is referenced by its full path. AppendEx is used when the second file was opened using the CDIntf.Document class.

Example 1

```
    ' create an object of type Document
    Dim doc As New CDIntfEx.Document

    On Error Goto invalid_file

    ' open existing document
    doc.Open "c:\test.pdf"
    ' append a second document to the first one
    doc.Append "c:\test1.pdf"
    ' save document to some other file
    doc.Save "c:\appended.pdf"

invalid_file:
    Set doc = Nothing
```

Example 2

```
    ' create two objects of type Document
    Dim doc1 As New CDIntfEx.Document
    Dim doc2 As New CDIntfEx.Document

    On Error Goto invalid_file

    ' open two existing documents
    doc1.Open "c:\test1.pdf"
    doc2.Open "c:\test2.pdf"
    ' append the second document to the first one
    doc1.AppendEx doc2
    ' save document to some other file
    doc1.Save "c:\appended.pdf"

invalid_file:
    Set doc1 = Nothing
    Set doc2 = Nothing
```

Document.Merge, Document.MergeEx Methods

The Merge and MergeEx methods merge two PDF documents by combining the contents of every page of the first document with a page from the second document.

Syntax

```
Function Merge(FileName As String, Options As Long) As Boolean
Function MergeEx(Document As Object, Options As Long) As Boolean
```

Parameters

FileName

[in] Full path of the second PDF file to merge with the current one.

Document

[in] Object reference of the second PDF file to merge with the current one.

Options

[in] Options for merging files. The list of options is in the remarks section.

Return Value

The return value is True if the file was merged. If the Merge or MergeEx method fails, an exception is raised and there is no return value.

Remarks

Merge is used when the file to be merged is referenced by its full path. MergeEx is used when the second file was opened using the CDIntf.Document class.

Options values:

Option	Option value	Description
Repeat first pages	1	The first pages of the second document are repeated in the first document
Second document above first	2	The contents of the second document are printed above the contents of the first document

If the documents do not have the same number of pages, say file1 has N1 pages and file2 has N2 pages where N1 < N2, then the developer can choose to:

- either merge file1 with the N1 pages of file2 and keep the remaining N2-N1 pages of file2 unchanged, in this case the Repeat option should be set to 0

- or merge the first block of N1 pages of file2 with the N1 pages of file1, merge the second block of N1 pages of file1 with the N1 pages of file1 and so on, in this case Repeat should be set to 1

E.g.: if file1 contains the company's letterhead in PDF format as one page, file2 is a two page invoice in PDF format generated with the accounting package, one can call:

Merge("file2.pdf", 1) to repeat the company's letterhead on all the invoice pages or

Merge("file2.pdf", 0) to insert the company's letterhead on the first page only.

Example

```
Private Sub Merge_Click()
    ' create two objects of type Document
    Dim doc1 As New CDIntfEx.Document
    Dim doc2 As New CDIntfEx.Document

    On Error GoTo invalid_file

    ' open the document containing the watermark
    doc1.Open "c:\test1.pdf"

    ' open the main document
    doc2.Open "c:\test2.pdf"
```

```
    ' merge the watermark document to the main one
    doc1.MergeEx doc2, 1

    ' save document to some other file
    doc1.Save "c:\merged.pdf"

invalid_file:
    Set doc1 = Nothing
    Set doc2 = Nothing

End Sub
```

Document.Encrypt, Document.Encrypt128 Methods

The Encrypt and Encrypt128 methods can be used to password protect a PDF document and restrict users to viewing, modifying or even printing the document. This function requires a call to Document.SetLicenseKey before it can be used. The Encrypt method uses 40 bits encryption, whereas the Encrypt128 method uses 128 bits encryption compatible with Adobe® Acrobat® 5 and higher.

Syntax

```
Function Encrypt(OwnerPassword As String, UserPassword As String, Permissions As Long)
                    As Boolean
Function Encrypt128(OwnerPassword As String, UserPassword As String, Permissions As Long)
                    As Boolean
```

Parameters

OwnerPassword

[in] Owner password.

UserPassword

[in] User password.

Permissions

[in] Options to restrict users opening the document using the User password.

Return Value

The return value is True if the document was encrypted, False otherwise. This method launches an exception if the encryption feature is not available.

Remarks

This function is only available if the activation code is for a professional version of the Amyuni PDF Converter. In the case of the evaluation version, the passwords are always set to "aaaaaa" and "bbbbbb" and cannot be changed.

Permissions values:

Permission	Permission value
Enable Printing	-64 + 4
Enable document modification	-64 + 8
Enable copying text and graphics	-64 + 16
Enable adding and changing notes	-64 + 32

To combine multiple options, use -64 plus the values 4, 8, 16 or 32. E.g. to enable both printing and adding notes, use -64 + 4 + 32 = -28. To disable all 4 options, use -64.

Owner and user passwords

Two passwords are associated to an encrypted PDF document. The owner password is for the author of the document, and the user password for the destinator or user of the document.

The owner password is mandatory and allows the author having this password to do any operation he/she wishes on this document, including modifying its security settings.

The user password is optional and can be one of the following:

- A blank password. In this case, the user is not prompted for a password when opening a document, but is restricted to the operations allowed by the author.
- The same password as the owner. In this case the user is prompted for a password and the author of the document will not be able to open this document as an owner to change its security settings.
- A password different from the owner. In this case, the user will not be able to open the document unless he/she enters a valid password. When a valid password is entered, the document can be viewed but its usage restricted to the operations allowed by the author.

User settings

- Enable changing the document content. When this option is cheked, the user is allowed to change the contents of the PDF document.

- Enable printing of document. The user cannot print the PDF document to any printer unless this option is checked.
- Enable copying text and graphics from the document. When this option is checked, the user can copy parts of the text of graphics from the PDF document.
- Enable addings notes or modifying form fields. The main body of the document cannot be changed but the user can add annotation or enter data in the form fields if there are any.

NOTE: These options are managed by the tool used to view the document and not by the PDF Converter. Once a valid password is entered, it is up to the viewer or editor to make sure that these security settings are respected.

Example

```
  Dim doc As New CDIntfEx.Document

  On Error Goto encrypt_failed

  // open a PDF document
  doc.Open "c:\test.pdf"

  // enable advanced functions such as Encryption or Linearization
  doc.SetLicenseKey "Evaluation Version Developer",
  "07EFCDAB01000100584F829697ECDB6776F3CC948D0749DAF7E37EF1621AEBEBF2975B"

  // password protect the PDF document
  // users are only allowed to print
  doc.Encrypt "ownerpass", "userpass", -64 + 4

  // save the encrypted document
  doc.Save "c:\encrypted.pdf"

encrypt_failed:
  Set doc = Nothing
```

Document.Linearized Property

The Linearized property can be used to optimize a document for web viewing. PDF documents usually need to be completely downloaded before thay can be viewed. A linearized document can be viewed one page at a time without the need to completely download the document. This function requires a call to SetLicenseKey before it can be used.

Syntax

```
Linearized As Long
```

Parameters

Linearized

[in, out] Flag that indicates if the document is linearized or not

Remarks

This function is only available if the activation code is for a professional version of the Amyuni PDF Converter. In the case of the evaluation version, a message-box appears to indicate that the document is being optimized.

Example

```
  Dim doc As New CDIntfEx.Document

  On Error Goto encrypt_failed

  // open a PDF document
  doc.Open "c:\test.pdf"

  // enable advanced functions such as Encryption or Linearization
  doc.SetLicenseKey "Evaluation Version Developer",
  "07EFCDAB01000100584F829697ECDB6776F3CC948D0749DAF7E37EF1621AEBEBF2975B"

  // set the Linearized property for this document
  doc.Linearized = True

  // password protect the PDF document
  // users are only allowed to print
  doc.Encrypt "ownerpass", "userpass", -64 + 4

  // save the encrypted document
  doc.Save "c:\encrypted_linearized.pdf"

encrypt_failed:
  Set doc = Nothing
```

Document.Optimize Method

The text inside a PDF file is usually split into multiple parts, a single sentence or paragraph can consist of multiple pieces of text positioned independently inside the PDF document. The Optimize method attempts to regroup lines or paragraphs prior to exporting the PDF file into another format, or to make the file easier to edit.

Syntax

```
Sub Optimize(Level As Integer)
```

Parameters

Level

[in] Optimization level to apply to PDF document.

Return Value

None.

Remarks

Level values:

Optimization Level	OptimizeLevel value
No optimization	0
Line optimization (Recommended)	1
Paragraph optimization	2
Table optimization	3

Example

```
See the ExportToRTF, ExportToHTML, ExportToEXCEL and ExportToJPEG  methods
```

Document.ExportToRTF Method

The ExportToRTF method converts a PDF document to an RTF document. This function is only available with the RTF Converter product and requires a call to SetLicenseKey before it can be used.

Syntax

```
Function ExportToRTF(FileName As String, RtfOption As acRtfExportOptions, UseTabs As Long)
                              As Boolean
```

Parameters

FileName

[in] Full path of resulting RTF file.

RtfOption

[in] RTF generation options.

UseTabs

[in] Replace tabs with spaces in the case of simple text output.

Return Value

The return value is True if the document was converted, False otherwise.

Remarks

This function is only available if the activation code is for the RTF Converter product, or an RTF Converter product combined with other Document Converter products.

RtfOption values:

Option	Option value (Hex)
Advanced RTF: using frames to position objects	0000
Full RTF: Text, Graphics and images with no frames	0001
Formatted Text only	0002
Simple text, non-formatted	0003

Example

```
Dim doc As New CDIntfEx.Document

' enable advanced functions such as RTF Export
doc.SetLicenseKey "Evaluation Version Developer",
"07EFCDAB01000100584F829697ECDB6776F3CC948D0749DAF7E37EF1621AEBEBF2975B"

' optimize document to line level before exporting
doc.Optimize 1

' save the PDF document as RTF
doc.ExportToRTF "c:\test.rtf", acRtfExportOptionAdvancedRTF, 0

Set doc = Nothing
```

Document.ExportToHTML Method

The ExportToHTML method converts a PDF document to an HTML document. This function is only available with the DHTML Converter product and requires a call to SetLicenseKey before it can be used.

Syntax

```
Function ExportToHTML(FileName As String, HtmlOption As acHtmlExportOptions) As Boolean
```

Parameters

FileName

[in] Full path of resulting HTML file.

HtmlOption

[in] HTML generation options.

Return Value

The return value is True if the document was converted, False otherwise.

Remarks

This function is only available if the activation code is for the DHTML Converter product, or a DHTML Converter product combined with other Document Converter products.

HtmlOption values:

Option	Option value (Hex)
Use Layers: Multiple pages in a single HTML file using layers	0001
Single HTML: All pages in a single HTML file	0002
Multiple HTML files: Each page in a separate HTML file	0003

Example

```
Dim doc As New CDIntfEx.Document

' enable advanced functions such as HTML Export
doc.SetLicenseKey "Evaluation Version Developer",
"07EFCDAB01000100584F829697ECDB6776F3CC948D0749DAF7E37EF1621AEBEBF2975B"

' optimize document to paragraph level before exporting
doc.Optimize 2

' save the PDF document as HTML
doc.ExportToHTML "c:\test.html", acHtmlExportOptionLayers

Set doc = Nothing
```

Document.ExportToEXCEL Method

The ExportToEXCEL method converts a PDF document to a Microsoft® Excel® document. This function is only available with the Excel Converter product and requires a call to SetLicenseKey before it can be used.

Syntax

```
Function ExportToEXCEL(FileName As String, ExcelOption As acExcelExportOptions) As Boolean
```

Parameters

FileName

[in] Full path of resulting Excel file.

ExcelOption

[in] Excel generation options.

Return Value

The return value is True if the document was converted, False otherwise.

Remarks

This function is only available if the activation code is for the Excel Converter product, or an Excel Converter product combined with other Document Converter products.

ExcelOption values:

Option	Option value (Hex)
Single Sheet: All pages in one sheet	0
Multiple Sheets: one sheet per page	1

Example

```
Dim doc As New CDIntfEx.Document

' enable advanced functions such as Excel Export
doc.SetLicenseKey "Evaluation Version Developer",
"07EFCDAB01000100584F829697ECDB6776F3CC948D0749DAF7E37EF1621AEBEBF2975B"

' optimize document to paragraph level before exporting
doc.Optimize 2

' save the PDF document as EXCEL
doc.ExportToEXCEL "c:\test.xls", acExcelExportOptionSingleSheet

Set doc = Nothing
```

Document.ExportToJPEG Method

The ExportToJPEG method converts a PDF document to a JPeg document. This function is only available with the JPEG Converter product and requires a call to SetLicenseKey before it can be used.

Syntax

```
Function ExportToJPEG(FileName As String, JPegOption As acJPegExportOptions) As Boolean
```

Parameters

FileName

[in] Full path of resulting JPeg file(s).

HtmlOption

[in] JPEG generation options.

Return Value

The return value is True if the document was converted, False otherwise.

Remarks

This function is only available if the activation code is for the JPEG Converter product, or a JPEG Converter product combined with other Document Converter products.

When the PDF document consists of multiple pages, one JPeg file is generated for every page with the page index appended to the supplied file name.

The Options value is a combination of the image resolution in the high order word, and the JPeg compression level from 1 (highest) to 9 (lowest) in the low order word.

JPegOption values:

Option	Option value (Hex)
Default : 300 DPI, medium compression level	0x00000000
Low : 150 DPI, high compression level	0x00960003
Medium: 300 DPI, medium compression level	0x012C0007
High : 600 DPI, low compression level	0x02580009

Example

```
Dim doc As New CDIntfEx.Document

' enable advanced functions such as JPEG Export
doc.SetLicenseKey "Evaluation Version Developer",
"07EFCDAB01000100584F829697ECDB6776F3CC948D0749DAF7E37EF1621AEBEBF2975B"

' save the PDF document as JPEG
doc.ExportToHTML "c:\test.jpeg", acJPegExportOptionMedium

Set doc = Nothing
```

Document.Print Method

The Print method can be used to print a PDF document to a hardware printer. This method is available only in the professional version of the Amyuni PDF Converter product.

Syntax

```
Function Print(PrinterName As String, StartPage As Long, EndPage As Long, Copies As Long)
                    As Boolean
```

Parameters

PrinterName

[in] Name of printer as it shows in the printers control panel. If this parameter is left empty, the document will print to the default printer.

StartPage

[in] Page number from which to start printing. The index of the first page is 1.

EndPage

[in] Page number at which to stop printing.

Copies

[in] Number of copies to print the document.

Return Value

The return value is True if printing was succesful, False otherwise.

Remarks

The security setting on a PDF file might disable printing, in this case the file should be opened with the owner password in order to print it.

The evaluation version of the Amyuni PDF Converter, professional version, will print only one page of the document.

Example

```
    ' create an object of type Document
    Dim doc As New CDIntfEx.Document

    ' activate advanced functions
    doc.SetLicenseKey "Evaluation Version Developer",
"07EFCDAB01000100584F55F2F2C87EFC6FFF82B2F90206F7EEE87AE0751CAECA86FED0207CD295E03629A5726
433B6E3BE1766876E2B5237F8F5"

    ' open existing document using the owner password
    doc.OpenEx "c:\test.pdf", "OwnerPass"

    ' print document to hardware printer
    doc.Print "HP LaserJet 4", 1, doc.PageCount, 1

    Set doc = Nothing
```

Document.SetLicenseKey Method

The SetLicenseKey method should be called after creating an object of type CDIntfEx.Document to activate the advanced methods that require the object activation code to work properly. The advanced methods that require a call to SetLicenseKey are: Encrypt, Linearize, ExportToRTF, ExportToHTML, ExportToJPeg.

Syntax

```
Function SetLicenseKey(Company As String, LicKey As String) As Boolean
```

Parameters

Company

[in] Name of the company or private user having licensed the product.

LicKey

[in] License key provided by Amyuni Technologies when downloading or purchasing a product.

Return Value

The return value is True if the license key is valid, False otherwise.

Remarks

Example

```
    ' create an object of type Document
    Dim doc As New CDIntfEx.Document

    ' activate advanced functions
    doc.SetLicenseKey "Evaluation Version Developer",
"07EFCDAB01000100584F55F2F2C87EFC6FFF82B2F90206F7EEE87AE0751CAECA86FED0207CD295E03629A5726
433B6E3BE1766876E2B5237F8F5"

    ' open existing document
    doc.Open "c:\test.pdf"

    ' encrypt document
    doc.Encrypt "owner", "user", -6

    ' save document to some other file
    doc.Save "c:\encrypted.pdf"
    Set doc = Nothing
```

General Functions

```
CDIntfEx.CreateDC
CDIntfEx.SetBookmark
CDIntfEx.SetHyperLink
CDIntfEx.GetLastErrorMsg
CDIntfEx.BatchConvert
SetTargetPrinterName
SetPageProcessor

SetServerAddress
SetServerPort
SetServerUsername
```

CDIntfEx.CreateDC Function

The CreateDC method creates a printer device context using the various parameters set using CDIntf function calls. The parameters include paper size, resolution, margins, font embedding, jpeg compression, ...

Syntax

```
Function CreateDC() As OLE_HANDLE
```

Parameters

Return Value

The return value is a printer device context (hDC) if the function succeeds, NULL otherwise.

Remarks

Example

```
Private Sub Form_Load()
    ' CDIntfEx object is created dynamically in memory
    Dim cdi As New CDIntfEx.CDIntfEx

    On Error GoTo printer_error
    ' attach to existing printer
    cdi.DriverInit ("Amyuni Document Converter")

    cdi.PaperSize = 5       ' set paper size to legal
    dc = cdi.CreateDC       ' create printer device context

    Exit Sub

printer_error:
    MsgBox "Sorry, printer not found"
End Sub
```

CDIntfEx.SetBookmark Method

The SetBookmark method creates a bookmark on the current printing page and location. Users will then be able to browse through the document by clicking on the bookmarks tree. Bookmarks are currently available in PDF files only.

Syntax

```
Function SetBookmark(hDC As Long, lParent As Long, Title As String) As Long
```

Parameters

hDC
: [in] Handle to printer device context returned by a call to CreateDC, or returned by the application.

lParent
: [in] Id of parent bookmark.

Title
: [in] Bookmark title.

Return Value

The return value is the identifier of the bookmark that was created, or 0 if the function fails.

Remarks

PDF bookmarks are structured in a tree. The root has an id of 0. SetBookmark returns the bookmark ID which can be used to insert other bookmarks as children of the current bookmark.

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

Example

```
Public pdf As New CDIntfEx.CDIntfEx

Private Sub Form_Load()
    ' initialize PDF printer and set it as default
    pdf.DriverInit "Amyuni PDF Converter"
    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
```

CDIntfEx.SetHyperlink Method

The SetHyperlink hyperlink creates a hyperlink on the current printing text or image object. Users will then be able to click on the hyperlink to browse to another location inside the document or to an external URL. Hyperlinks are currently available in PDF and HTML files only.

Syntax

```
Function SetHyperlink(hDC As Long, Destination As String) As Long
```

Parameters

hDC

[in] Handle to printer device context returned by a call to CreateDC or by the application.

szDestination

[in] Hyperlink destination.

Return Value

The return value is always 0.

Remarks

Hyperlinks can be set to an external URL such as http://www.amyuni.com or to a bookmark in the same document. To add a hyperlink to an internal bookmark, the Destination should start with the # sign followed by the bookmark title.

The hyperlink will be inserted at the location where the last text drawing operation occurred. E.g.: if we draw text in the middle of page 3 of the document and call SetHyperlink immediately after, the hyperlink will be set on the text in the middle of page 3 of the PDF document.

Example

```
    Dim pdf As New CDIntfEx.CDIntfEx

    ' initialize PDF printer
    pdf.DriverInit "Amyuni PDF Converter"
    ' set the output file
    pdf.DefaultFileName = "c:\test.pdf"
    pdf.FileNameOptions = NoPrompt + UseFileName

    ' draw some text
    Printer.CurrentX = 200
    Printer.CurrentY = 400
    Printer.FontName = "Arial"
    Printer.Print "Bookmark 1"
    ' set a bookmark on page 1
    pdf.SetBookmark Printer.hDC, 0, "Bookmark 1"
    ' add a hyperlink to our web site
    pdf.SetHyperLink Printer.hDC, "http://www.amyuni.com"
    ' 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 hyperlink to page 1
    pdf.SetHyperLink Printer.hDC, "#Bookmark 1"

    Printer.EndDoc
    pdf.DriverEnd

End Sub
```

CDIntfEx.GetLastErrorMsg Method

The GetLastErrorMsg method returns the error message generated by the last call to a CDIntf function call.

Syntax

```
Function GetLastErrorMsg() As String
```

Parameters

Return Value

This method returns a description of the last error generated by a call to a CDIntf function.

Remarks

Example

```
Dim cdi As New CDIntfEx.CDIntfEx

On Error Goto show_error
cdi.DriverInit("Amyuni PDF Converter")

Exit Sub

show_error:
    ' DriverInit failed, get error message generated by CDIntf
    MsgBox cdi.GetLastErrorMsg
    Exit Sub
```

CDIntfEx.BatchConvert Method

The BatchConvert method converts a number of files to PDF, RTF, HTML, Excel or JPeg formats in batch mode.

Syntax

```
Function BatchConvert(FileName As String) As Long
```

Parameters

FileName

[in] File or group of files to be converted.

Return Value

This method returns the number of files having been converted, or 0 if no files were converted.

Remarks

The Document Converter printer should be configured with the destination file name and all other options before calling this function. The printer should also be set as default printer. This function launches the application that is associated with a specific file and issues a print command to convert the document.

The FileName parameter can contain wildcards to convert a number of documents in the same directory.

Example

```
Dim cdi As New CDIntfEx.CDIntfEx

On Error Goto printer_error
cdi.DriverInit "Amyuni PDF Converter"

  cdi.SetDefaultPrinter                    ' printer needs to be set as default
  cdi.DefaultDirectory = "c:\temp\pdf_files"       ' set destination directory

  ' the file name is to be generated automatically from the document name
  cdi.FileNameOptionsEx = NoPrompt

  ' convert all MS Word documents to PDF format
  cdi.BatchConvert "c:\temp\*.doc"

  // close printer
  cdi.DriverEnd
```

Printer Driver Message Handling

```
CDIntfEx.StartDocPre
CDIntfEx.StartDocPost
CDIntfEx.EndDocPre
CDIntfEx.EndDocPost
CDIntfEx.StartPage
CDIntfEx.EndPage
CDIntfEx.EnabledPre

CDIntfEx.GetDocumentTitle
CDIntfEx.GetGeneratedFileName

CDIntfEx.CaptureEvents
```

CDIntfEx.StartDocPre, CDIntfEx.StartDocPost, CDIntfEx.EndDocPre, CDIntfEx.StartDocPost, CDIntfEx.StartPage, CDIntfEx.EndPage, CDIntfEx.EnabledPre Events

When the message broadcast option is set in the Document Converter products, the printer driver broadcasts messages to all running applications each time one of the following events occur:

Printer Enabled
Start of document
Start of page
End of page
End of document

These messages can be intercepted by the calling application using the CDIntfEx object. When placed on a form, the CDIntfEx object fires one or two VB events for each of the printer events listed above.

Syntax

```
Event EnabledPre()
Event StartDocPre()
Event StartDocPost(JobID As Long, hDC As Long)
Event StartPage(JobID As Long, hDC As Long)
Event EndPage(JobID As Long, hDC As Long)
Event EndDocPre(JobID As Long, hDC As Long)
Event EndDocPost(JobID As Long, hDC As Long)
```

Parameters

JobID

[out] Unique job ID generated by the spooler in the case of Windows NT/2000/XP, or internally by the printer driver in the case of Windows 9x.

hDC

[out] Printer Device Context.

Return Value

Remarks

The method CaptureEvents should be called before the CDIntfEx control starts sending events to the calling application. Events should be disabled before exiting the application by calling CaptureEvents False.

CaptureEvents adds the option BroadcastMessages (&H20) to FileNameOptions. If the user or developper changes the value for FileNameOptions, they should make sure they include this option with their other options, events will otherwise be disabled.

Example

```
    Private Sub Form_Load()

    On Error GoTo printer_error
    ' install a new printer attach to existing printer
    CDIntfEx1.PDFDriverInit (PrinterName)

    ' First time activation of printer
    CDIntfEx1.EnablePrinter "Evaluation Version",
"07EFCDAB01000100C717EA202478B0E1D72748B5D78870EC242CC8B9A4ABF6AA49DDF7E8AA44A6575449D"

    ' set printer as system default
    CDIntfEx1.SetDefaultPrinter

    ' capture printer events
    CDIntfEx1.CaptureEvents True

    Exit Sub

printer_error:
```

```
    MsgBox "Sorry, could not install printer"
End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' end event capture
    CDIntfEx1.CaptureEvents False

    ' restore default printer before removing our printer
    cdi.RestoreDefaultPrinter

    ' close the printer handle and remove printer
    cdi.DriverEnd
    Set cdi = Nothing
End Sub

Private Sub Print_Click()

    cdi.FileNameOptionsEx = &H1 + &H2 + &H20 ' NoPrompt + UseFileName + BroadcastMessages
    cdi.DefaultFileName = "c:\test.pdf"      ' set output file name

    ' draw some text
    Printer.CurrentX = 200
    Printer.CurrentY = 400
    Printer.FontName = "Arial"
    Printer.Print "Hi There"

    Printer.EndDoc
End Sub

Private Sub CDIntfEx1_EnabledPre()
    ' when using the developer version, we need to enable the printer here
    CDIntfEx1.EnablePrinter "Evaluation Version",
"07EFCDAB01000100C71728B0E1D72748B5D78870EC242CC8B9A4ABF6AA49DDF7E8AA44A6575449D"
End Sub

Private Sub CDIntfEx1_EndDocPost(ByVal JobID As Long, ByVal hDC As Long)
    ' finished printing a new document
    List1.AddItem "New document printed..."
End Sub
```

CDIntfEx.CaptureEvents Method

The CaptureEvents method should be called when the CDIntfEx control is placed on a form and the user or developer needs to capture events generated by the Document Converter products.

Syntax

```
Function CaptureEvents(bCapture As Long) As Long
```

Parameters

bCapture

[in] Set to True to start capturing events, False otherwise.

Return Value

The return value is 0 if the function succeeds. If the function fails, an exception is generated and there is no return value.

Remarks

The method CaptureEvents should be called before the CDIntfEx control starts sending events to the calling application. Events should be disabled before exiting the application by calling CaptureEvents False.

CaptureEvents adds the option BroadcastMessages (&H20) to FileNameOptions. If the user or developper changes the value for FileNameOptions, they should make sure they include this option with their other options, events will otherwise be disabled.

Example

```
    Private Sub Form_Load()

    On Error GoTo printer_error
    ' install a new printer attach to existing printer
    CDIntfEx1.PDFDriverInit (PrinterName)

    ' capture printer events
    CDIntfEx1.CaptureEvents True

    Exit Sub

printer_error:
    MsgBox "Sorry, could not install printer"
End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' end event capture
    CDIntfEx1.CaptureEvents False

    ' close the printer handle and remove printer
    cdi.DriverEnd
    Set cdi = Nothing
End Sub
```

CDIntfEx.GetDocumentTitle Method

The GetDocumentTitle method returns the title of the document that is currently being printed. This function should be called in the message handling loop and will return invalid results as soon as the document finishes printing.

Syntax

```
Function GetDocumentTitle(JobID As Long) As String
```

Parameters

JobID

[in] Job ID as retrieved by the PDF message handling function.

Return Value

This method returns the title of the document that is currently being printed.

Remarks

This function should be called from a PDF message handling function before the EndDocPost message is sent.

Example

```
Please refer to the PDF Driver Event section for a full sample
```

CDIntfEx.GetGeneratedFilename Method

The GetGeneratedFilename method returns the full path of the file that is currently being generated. This function should be called in the message handling loop and will return invalid results as soon as the processing of StartDoc is finished.

Syntax

```
Function GetGeneratedFilename() As String
```

Parameters

Return Value

This method returns the full path of the file that is currently being generated.

Remarks

This function should be called from a PDF message handling function before the StartDocPost message is sent, i.e. in the StartDocPre event handler.

Example

```
Please refer to the PDF Driver Event section for a full sample
```

Intercepting the data stream coming out from the document converter

Note: The information below apply to Windows 9x, 2000 and XP and does not apply to NT4.

The data stream generated by the Document Converter products can be intercepted for processing other than the default saving to a file or sending by email. The data stream can be sent to a remote location without being saved to the local hard drive, or sent to an application that needs to do some specific processing with the stream before saving it to file.

To intercept the data stream, the following steps should be followed:

1. Create a custom DLL following the framework described below and copy the DLL to the system or system32 directory.
2. Set the SendToCreator option in the FileNameOptions method or function call.
3. Set the "PageProcessor" parameter using SetPageProcessor method or function call. This parameter should be set to the name of the custom DLL and only when logged in as Administrator.

Example

```
    Dim hPrinter As Long

    Const PrinterName = "Amyuni PDF Converter"

    On Error GoTo printer_error

    ' attach to existing printer
    hPrinter = DriverInit(PrinterName)
    If hPrinter = 0 Then GoTo printer_error

    ' set output options
    SetFileNameOptions hPrinter, SendToCreator
    SetPageProcessor hPrinter, "custompp.dll"

    ' Print a sample Word document
    Dim wordApp As New Word.Application
    Dim documents As Word.documents

    ' open a Word document in Read-only mode
    Set documents = wordApp.documents
    documents.Open "c:\wutemp\test.doc", False, True

    ' set the ActivePrinter to ours
    wordApp.ActivePrinter = PrinterName

    ' print the Word document without background printing
    wordApp.PrintOut False

    ' reset printer options
    SetFileNameOptions hPrinter, 0

    wordApp.Quit False
    Set documents = Nothing
    Set wordApp = Nothing

    Exit Sub

printer_error:
    MsgBox "Sorry, could not find printer"
End Sub
```

```
*************** File name: custompp.def ******************
LIBRARY custompp

DESCRIPTION 'Custom Page Processor DLL'

EXPORTS

  PPInit
  PPStartDoc
  PPEndDoc
  PPStartPage
  PPEndPage
  PPStartObject
  PPEndObject
  PPWriteBuffer

*************** End of: custompp.def ******************

*************** File name: custompp.h ******************
typedef LPVOID DOCHANDLE;
typedef DOCHANDLE* LPDOCHANDLE;

int WINAPI PPInit();
int WINAPI PPStartDoc(LPDOCHANDLE phDoc, DWORD dwFlags);
int WINAPI PPEndDoc(DOCHANDLE hDoc);
int WINAPI PPStartPage(DOCHANDLE hDoc);
int WINAPI PPEndPage(DOCHANDLE hDoc);
int WINAPI PPStartObject(DOCHANDLE hDoc);
int WINAPI PPEndObject(DOCHANDLE hDoc);
int WINAPI PPWriteBuffer(DOCHANDLE hDoc, LPBYTE pData, DWORD dwSize);

*************** End of: custompp.h ******************

*************** File name: custompp.c ******************
#include "windows.h"
#include "custompp.h"

BOOL APIENTRY DllMain( HANDLE hModule, DWORD ul_reason_for_call, LPVOID lpReserved )
{
// standard DLL entry point
    return TRUE;
}

int WINAPI PPInit()
{
// called when the DLL is initialised, and before the call to StartDoc
  return 0;
}

int WINAPI PPStartDoc(LPDOCHANDLE phDoc, DWORD dwFlags)
{
// called when a new document is started
// dwFlags is 1 if the user requested the document to be printed to a physical printer
  if (!phDoc)
          return -1;

  // create a new PDF file
  *phDoc = (LPDOCHANDLE)CreateFile( "c:\\test.pdf", GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
0, NULL );
  if ( *phDoc )
          MessageBox( 0, "StartDoc", "Success", 0 );
  else
          MessageBox( 0, "StartDoc", "Failure", 0 );
  return 0;
}
```

```
int WINAPI PPEndDoc(DOCHANDLE hDoc)
{
// document printing has ended
  if (!hDoc)
          return -1;

  // close the destination PDF file
  CloseHandle( (HANDLE)hDoc );
  return 0;
}

int WINAPI PPStartPage(DOCHANDLE hDoc)
{
// a new page is started
  if (!hDoc)
          return -1;
  return 0;
}

int WINAPI PPEndPage(DOCHANDLE hDoc)
{
// page ended printing
  if (!hDoc)
          return -1;
  return 0;
}

int WINAPI PPStartObject(DOCHANDLE hDoc)
{
// a new PDF object is being started
// each document and page can contain a number of PDF objects
  if (!hDoc)
          return -1;
  return 0;
}

int WINAPI PPEndObject(DOCHANDLE hDoc)
{
// finished outputting a PDF object
  if (!hDoc)
          return -1;
  return 0;
}

int WINAPI PPWriteBuffer(DOCHANDLE hDoc, LPBYTE pData, DWORD dwSize)
{
// block of data data sent from document converter
// dwSize is the number of output bytes
  DWORD    dwWritten = 0;
  if (!hDoc)
          return -1;

  // simply write data to file
  if ( WriteFile( (HANDLE)hDoc, pData, dwSize, &dwWritten, NULL ) == FALSE )
          return -1;

  return 0;
}
```

.NET Managed Code Interface

The .NET Interface can be downloaded from the Amyuni Technologies web site at:
http://www.amyuni.com/downloads/cdintf210.zip

The interface is hosted in a file named CDIntfNet.DLL.

The .NET Interface was written using Borland® C# Builder and is compatible with Microsoft's Visual Studio .NET framework. This interface currently contains only a subset of the full CDIntf features and is being completed to support the full features. The supported classes and features can be seen when importing the DLL into the .NET environment. For documentation about each supported class, method, property or event, the developer should check the corresponding ActiveX control section.

Technical Support

If you have any questions or problems with our products, the following resources are available to you through our web site:

Frequently Asked Questions:
http://www.amyuni.com/en/support/faq.html

Technical Notes:
http://www.amyuni.com/en/support/technotes.html

User forum:
http://www.amyuni.com/forum/index.php

Posting questions to our technical support staff:
http://www.amyuni.com/en/support/index.html

Available Technical Notes:

Title	Description	Link
Multitasking the AMYUNI printer drivers	This note describes how to correctly use the AMYUNI printer drivers in multi-threaded environments such as web servers	www.amyuni.com/downloads/tn01.zip
Setting up printer properties using the DEVMODE structure	This note describes how to use the standard Windows DEVMODE structure to set general and specific printer properties	www.amyuni.com/downloads/tn02.htm
Using the Amyuni Printer Drivers with Visual FoxPro(r)	This note provides Visual FoxPro users with technical information to interface with the Amyuni series of printer drivers	www.amyuni.com/downloads/tn03.zip
Using the Amyuni Converters with Microsoft Access(r)	This note provides Microsoft Access users with technical information to interface with the Amyuni series of printer drivers	www.amyuni.com/downloads/tn05.zip
Using the Amyuni Converters with Sybase(r) PowerBuilder(r)	This note provides Sybase(r) PowerBuilder(r) users with technical information to interface with the Amyuni series of printer drivers	www.amyuni.com/downloads/tn06.zip
Using the Amyuni Converters with Borland(r) Delphi(r)	This note provides Borland(r) Delphi(r) users with technical information to interface with the Amyuni series of printer drivers	www.amyuni.com/downloads/tn07.zip
Using the Amyuni PDF Converter and Creator in ASP.NET	This note provides web developers with technical information for producing PDF files on a web server using ASP.NET and allow the users to view the PDF files in their web browser.	www.amyuni.com/downloads/tn08.zip