

PDFDevice version 5

by Brainy Data Limited

About PDFDevice

Introduction

PDFDevice is an Omnis external component that adds PDF output to the Omnis printing device group. The component is fully integrated with Omnis Studio and allows the developer to take real control over PDF generation with very little effort. It installs directly in the Omnis tree and can be distributed as part of this tree without the need for further installations on the client machine.

For latest changes see History of Enhancements below

Downloading the Software and Examples

If you have not done so already you can download the demo or full release software components from the following locations:

Demo Software and examples: <http://www.brainydata.co.uk/demos/download.htm>

Release Software: http://www.brainydata.co.uk/support/pdfdevice_su.htm

Installing the Software

PDFDevice Desktop/Server Control

As a general rule, the downloaded folder containing the software will be organised so that you may follow these generic steps.

There are a number of components to install and the component names vary between platforms. It typically contains folders for different versions of studio, i.e. *studio_810*, *studio_1000*, *studio_1010* and *studio_1020*, referring to Studio versions 8.1.0, 10.0, 10.1 and 10.2 respectively.

1. Open the appropriate Omnis Studio folder. Always use the latest version that is not later in version than your version of Omnis Studio. For example, for Studio 8.1.7 you would use components from the *studio_810* folder and **not** the *studio_1000* folder.

The Omnis Studio folder will contain an additional folder called *XCOMP*. This may or may not be suffixed with a three letter platform identifier, i.e. *_mac*, *_win* or *_lin*.

2. Copy the components from inside XCOMP to your Omnis installation. You will find identical named folders inside the Omnis application support folder (macOS) or executable folder (winOS). On Mac OSX you may need to create this folder inside the *~/Library/Application Support/Omnis/Omnis Studio {version}/* folder. On windows, you will find the XCOMP folder alongside the Omnis executable.

Deploying your software

Please refer to the [license agreement](#) for rules on deployment.

Please also read the open source license information that is included in the software download tree.

History of Enhancements

Below is a summary of the most recent enhancements.

Version	Enhancements
5.0	<ul style="list-style-type: none"> New static \$file... functions: \$filemerge(), \$fileencrypt(), \$filesign(), \$filereaddata(), \$filegetlasterror(). Support for Acroform fields.
4.0	<ul style="list-style-type: none"> Support for transparency with RAW png and CS32 images. Better support for “assignpdf” and “showpdf” client commands. See latest Javascript Client examples. New Brainy Data PDF Server features. See section Multi-Threaded Server in the chapter “Designing PDFDevice”. New methods \$startserver(), \$stopserver(), \$initparams(), \$settempfilename() and \$setmemoryoutput(). New examples PDFDeviceAndJSClient.
3.1	<ul style="list-style-type: none"> New device parameter kDevPdfPrintScale, to manually apply scaling which was previously set automatically via the \$scale property.
3.0	<ul style="list-style-type: none"> Default device name changed to BrainyPDF (see technical note TN0021) PDF/A support. See new device parameters kDevPdfaEnabled, kDevPdfaOutputInfo and kDevPdfOutputProfile. Non-true-type Font substitution. See new device parameter kDevPdfSubstituteFont. Device parameter kDevPdfIgnoreFontStyle now obsolete. Bold and italic synthesising (see technical note TN0017) Improved font mapping and support for non-Western languages such as Thai, Korean, Chinese or Arabic when the chosen font does not support these languages. Printing to memory. New static method \$getmemoryoutput. Substantially updated to use latest OS X core text handling. Studio 6 support New version number support (see technical note TN0022)

www.brainydata.com

Introduction

PDFDevice is a simple to use external component that provides powerful PDF generation capabilities. Installation and preparation merely take a few minutes before you will be able to print your first PDF file.

Overview

The PDFDevice software consists of the following components

1. The external plug-in (pdfdevice.dll in winOS and pdfdevice.xcomp in macOS)
2. The [OWrite Document Manager](#) examples which demonstrate various uses of PDFDevice.
3. The [PDFDevice and JSClient example](#) which demonstrates various ways of producing and displaying PDF files when using the JS Client and Omnis JS Server.

For an introduction on how to integrate the PDFDevice software into your library, please read the chapter Designing PDFDevice.

Examples

In the previously mentioned example libraries, you will find a small number of classes that demonstrate the use of the PDFDevice software. These classes are mainly concerned with the PDF advanced options interface and consist of windows and the startup task. The JS Client examples are a little more comprehensive and include many more classes that demonstrate the various ways of using PDF device with JS Clients. For a description of the example classes please read the chapter Examples Reference.

External component Library

The external component library provides the printing device. A number of constants are defined for setting and getting device properties. These constants can be accessed from the Omnis Catalog -> Constants -> PDFDevice. For a description of the device parameters please read the chapter External Component Reference.

The external device does not implement any interface with the exception of the parameters pane for the Omnis destination dialog. All that is provided is an entry field and a browse button for selecting a destination file name, and the advanced options button.

PDFDevice Main Features

Acroform Fields Support (v5.0)

PDFDevice supports the embedding of Acroform fields in Omnis reports and in conjunction with the static \$filereaddata() method, it makes it now possible to produce PDF forms that can be completed by clients and then have Omnis extract the data from these forms. Please read the section “Acroform Fields” in the chapter “Designing PDFDevice” for more details.

Merging PDF Files (v5.0)

The static function `$filemerge()` can merge multiple PDF files. This makes it possible to append existing PDF files to reports, such as Standard appendages to Invoices perhaps.

Encryption and Signing of existing PDF Files (v5.0)

The static functions `$fileencrypt()` and `$filesign()` can encrypt and sign existing PDF files, thus widening the commercial scope of PDFDevice beyond producing PDF files from Omnis reports.

Printing to memory (v4.0)

It was first possible in PDFDevice version 3 to print and retrieve the binary output from an Omnis method. This way of producing PDF without the use of a file on disk has further been improved in version 4. You can now print directly to an Omnis binary variable. This means that all PDF generation can be handled in memory in a much more direct way, substantially improving logistics as well as performance.

Printing multiple Reports to the same file

Often considered as one of the most important features of PDFDevice is the ability to print multiple reports to a single PDF file. This greatly simplified the situation where you have to produce different reports that share some common pages which can now be produced and inserted by additional report classes. For a description of this feature please read the chapter Designing PDFDevice.

PDF/A Support (v3.0)

PDFDevice is capable of producing ISO 19005 PDF/A-1b compliant documents suitable for long-term archiving. As part of this feature, developers or users can specify their own RGB color profiles if so required. In addition, document information such as the Author and Title are now embedded as searchable XMP-compliant metadata.

Font Embedding

PDFDevice is capable of creating subsets of true type fonts and embeds their glyphs in the PDF document. To lessen the impact on the resulting file size, only glyphs required in the document are embedded. If size is of primary concern, this option can be disabled to further reduce the size of the file, but the resulting PDF output may not render accurately on other computers that do not have the correct fonts installed and some unicode characters may not render correctly.

Font substitution (v3.0)

PDFDevice provides an option to substitute non-true-type fonts with true-type fonts that have similar characteristics for the purpose of font embedding.

Font synthesising (v3.0)

Not all true-type fonts support the standard bold and italic typefaces. For example, the font "American Typewriter" does not support the italic typeface. However a developer or user may be able to select the italic typeface for this font and the system will do its best to produce the correct appearance. Equally, in such cases PDFDevice will simulate bold and italic using appropriate matrix settings that increase the glyphs' fatness or slant and in that way synthesises the font's

bold and italic appearance within PDF files.

Image Compression

Images can be compressed using the JPEG compression format. The compression ratio can be specified to control the amount of compression and the image quality. A high quality setting produces larger files. A setting of 75% produces good quality images with good compression.

If JPEG compression is disabled, the image's RGB values are compressed using standard deflate compression.

Watermarks

PDFDevice allows the placement of text based watermark on the pages. Watermarks can be placed in front of or behind the pages content and can be rotated at angles of up to 360 degrees anti-clockwise or clockwise. Watermarks are limited to 255 characters of single line text.

Background Image (v2.0)

The background image device options allow the placement and positioning of a background image on each page or just the first page of the PDF document.

Document Info

You may specify document information such as the document title, author and subject. This information will be stored in the PDF's document info section and can be viewed using Acrobat or other readers that can display the document info.

Viewer Options

The viewer options allow you to control how PDF viewers present the resultant PDF document. You can control the page layout, page mode and hide user interface controls. Please be aware that not all PDF viewers support these options.

Bookmarks / Document Outlines

PDFDevice is capable of producing document outline trees from your existing reports as long as your titles use consistent font, size and style for each level. You can specify the font name, size and style in the PDF options and PDFDevice will add bookmarks for text that matches the given criteria, to the document outline tree. You can create trees with numerous levels by specifying different criteria for each outline level.

Security (v2.0)

The security options allow the encryption of documents. The user can choose between standard 40 bit or more secure 128 bit encryption. Separate Owner and User passwords allow the control of user permissions such as printing, copying and modifying document content.

Advanced Options Window

The advanced options window is implemented as an Omnis window. The print destination dialog only displays the destination file name option. The "Advanced..." push button on this dialog can be linked to Omnis script that is specified with the `kDevPdfAdvanced` parameter constant. The example library implements a fully functional options window that you can copy to your own library. The startup task's `$construct` method installs this window as described above.

Page Content and Font File Compression

To reduce the overall size of PDF files, all page content streams and embedded font streams are compressed using “Deflate” compression.

Supported Platforms

Platforms currently supported are macOS and winOS.

Omnis Server Implementation (v4.0)

Please read the section Multi-threaded Server in the chapter “Designing PDFDevice” for guidance on implementing the thread-safe Brainy Data PDF server.

Limitations

External Report Objects

Some external report objects will be converted to images at the current screen resolution and using the current image compression options. This is due to the objects being of unknown type. Some external report objects, such as the HTML device objects and the report lists, add known report manager objects to a print job and are handled more efficiently.

Font Embedding

Designing PDFDevice

This chapter gives a brief description of what is involved to add PDFDevice to your libraries. For a more detailed description of the example classes and external component please read the chapters Examples Reference and External Reference.

Integrating PDFDevice

In its basic form, adding PDFDevice to your application is a simple process that can be achieved in just a few minutes.

Downloading and Installing PDFDevice

Please refer to the Welcome chapter for instructions.

PDF Options Window

As a minimum, you should provide an interface for the advanced PDF options. You can copy the classes wPDFOptions and wPDFPickStyle from the examples library. Feel free to change the windows if you wish. Once you have designed the interface you must tell PDFDevice about your options window. You do this by setting the device parameter kDevPdfAdvanced to the Omnis notation that will open your window. You must provide a full notational path to your window, starting at \$root, as PDFDevice is a global device and does not exist in the context of your library or any other context.

Example:

```
Do
$device.$setparam(kDevPdfAdvanced, "$root.$libs.PDFDevice.$windows.wPDFOptions.
$.openonce('PDFOptions', kWindowCenter) ")
```

View example code...

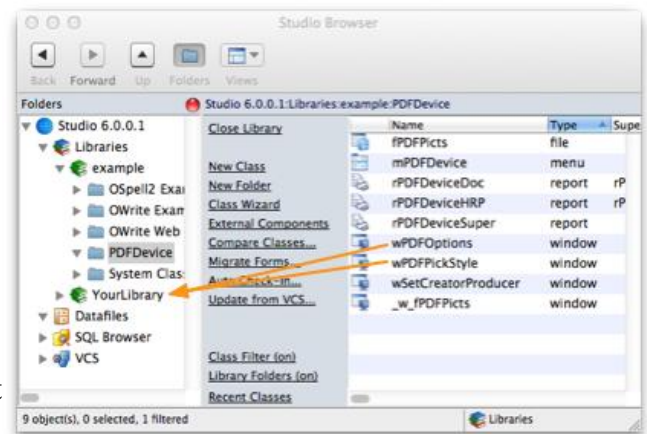
You can do this in the \$construct method of your startup task or any other appropriate time during startup. If your application opens several libraries, choose the one that provides the interface.

Note: The library that provides the interface must remain loaded at all times and the \$external property of the window must be set to kTrue.

External Device Parameters

PDFDevice has a large number of device parameters such as kDevPdfWatermark or kDevPdfPermissions, etc.

It is important to know that these options are global and that they are stored in the Omnis config file in the Studio folder of the Omnis tree. Once an option is set, the setting will persist between Omnis sessions. This configuration makes it easy for each individual client machine to have its own settings.

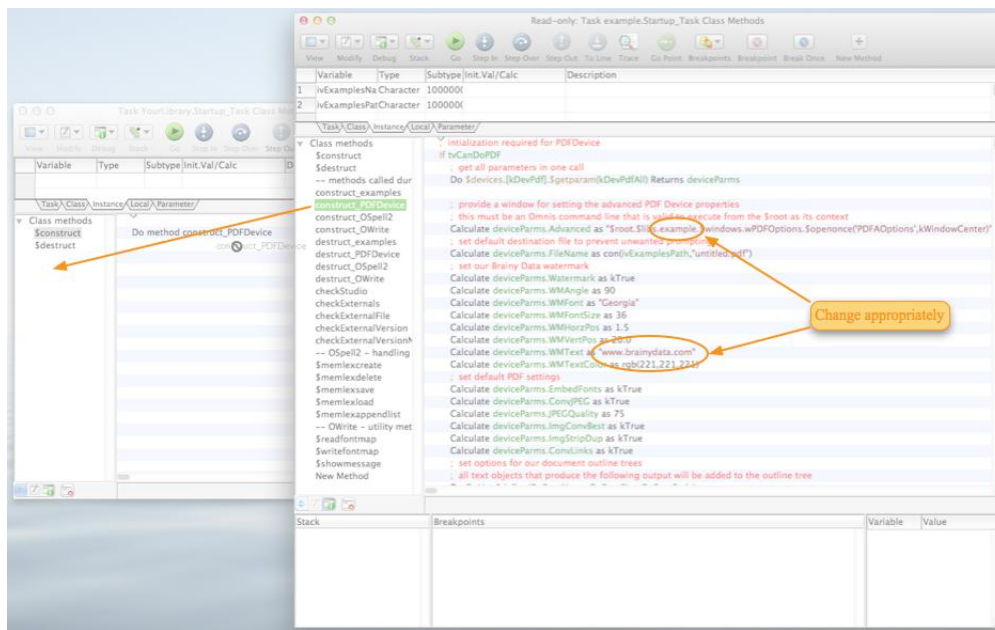


The side effect is that if you run an Omnis library that changes these settings, that Omnis installation will be changed permanently until the setting is changed by the user or code in your Omnis library(s). It also means that the output that is produced by a specific installation of Omnis may not be the same as the output that will be produced by another installation of Omnis, because the settings may not be the same.

These issues highlight the need to be disciplined about the default settings of your device parameters, especially for new client installations.

Ideally, after a new installation of your software you want the default device parameters set in a way which is most useful to your application. PDFDevice has its own defaults that are set the first time the external is loaded in a new Omnis installation. If you are happy with these defaults than there is nothing else for you to do. If you require different defaults, you have two choices.

1. You create an Omnis config file with the correct defaults and ship this config file with your runtimes.
2. You program the default settings and run this code after a new installation. You should not set the defaults for every Omnis session as this will prevent your users from changing these settings. You can copy a method from the examples and adopt it for this purpose.



View example code...

Warning: Every now and then we receive a report that PDFDevice adds the watermark “PDFDevice by Brainy Data” to the PDF files it produces. The developer typically believes that their component is a demo version although they have purchased a full license. This is not so.

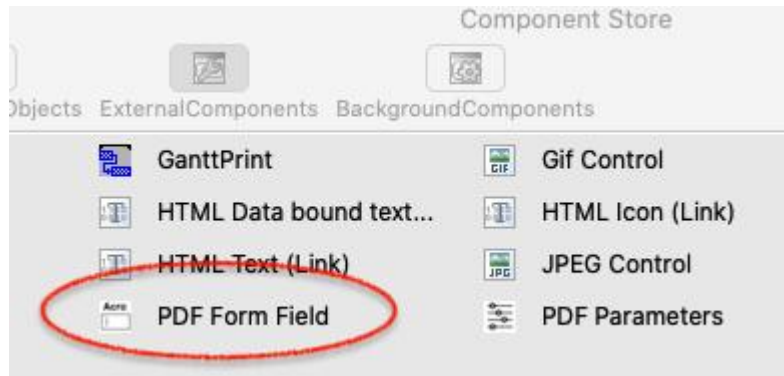
It is simply the case that at one point the PDFDevice examples had been opened by that Omnis and the examples turned on this watermark using the device parameter functions. Please see the method “construct_PDFDevice” in the startup task of the example library.

Beyond the basics

This section describes some of the additional features that you may wish to take advantage of which will require some additional coding.

Acroform Fields (v5.0)

PDFDevice version 5 implements support for Acroform fields in Omnis reports. To drop an Acroform field onto your report, select the 'External Components' tab in the 'Component Store' and drag the 'PDF Form Field' object to your report.



The form field's \$fieldtype property determines the type of the form field. PDFDevice supports all standard Acroform field types, except the radio button (use the choice field instead, see kDevPdfFFTChoice). Using these fields you can produce PDF files that can collect data from your clients by using the Adobe Reader to fill in the fields and

1. submit them to your server. See the PDFDevicePoDoFo examples for more details. The examples include a sample server script that will mail the data that was entered to the email address provided in the form.
2. return the filled in PDF form. You can then use the PDFDevice static function \$filereaddata() to extract the data from the form. Again, see the PDFDevicePoDoFo examples for more details.

The PDFDevicePoDoFo examples provide an Omnis sample report that demonstrates all of the supported field types.

The screenshot shows a PDF form titled "Acro Form Example" with a ruler at the top. The form contains the following elements:

- Subscription:** A dropdown menu with "Subscription" selected.
- Allow Notifications:** A checked checkbox.
- Categories:** A dropdown menu with "Categories" selected.
- Name:** A text input field with "Name" entered.
- E-Mail:** A text input field with "EMail" entered.
- Sign in Box:** A dashed box containing a "Signature" field and a "Printed Name" field.
- Important Note:** A yellow box with the text: "Important Note: The provided e-mail is used to send an email with the filled in form data when you click 'Submit'. We do not use the provided email for any other purpose. A copy of the server script that we use is included in this beta download."
- Buttons:** "Clear Form" and "Submit" buttons at the bottom right.

Combining Reports

PDFDevice allows you to print several reports to a single PDF file. In order to do this, you must open the device prior to printing your reports, and close the device when the final report has printed.

Example:

```
Do $cdevice.$assign(kDevPdf) ;; set the current destination
Do $cdevice.$open() ;; open the device

;; print your reports

Do $cdevice.$close() ;; close the device
```

Printing to memory (v4.0)

It is possible to direct output to memory by calling the method `$setmemoryoutput`, specifying the name of the binary variable that is to receive the PDF data.

Examples:

```
Do PDF Device.$setmemoryoutput(nam(variable))
```

View example code...

The variable must remain in scope while the report is printed or Omnis may crash. Therefore using an instance variable of the instance that is printing the report, or a task variable of the task that is active while the report is printed is recommended. However, using a local variable will work if the method that owns the variable also completes the printing.

Once the report is printed, the data in the binary variable can be saved to a file on disk, a database, an FTP server, or whatever is required.

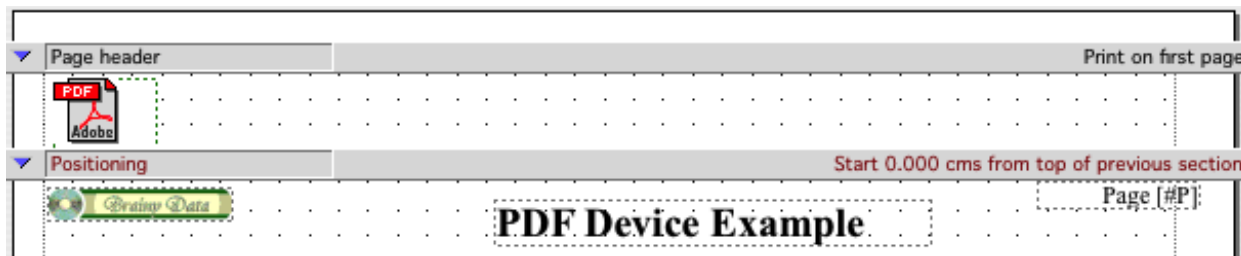
PDF Report Object (changed in v4.0)

It may sometimes be desirable to change the device parameters of an output device instance associated with the running print job. For example, you may wish to turn off watermarks after the first page has been printed.

Using the PDFDevice report object facilitates this feature. To assign parameters directly to the device instance belonging to the report instance follow these instructions.

Declare an instance variable of type Row. Place the report object in the page header section prior to any other content and set its \$dataname to the row variable. To insure the PDF object is printed prior to all other content you can use a positioning section in the page header section as shown below. When a new page is generated by Omnis, the PDF object is printed which simply sends the current device data in your row variable to the device.

Page header	componentctrl	PDF Parameters
	componentlib	PDF Device
	dataname	ivPDFParams
	height	1.339



Now you can change any of the device parameters as and when is needed. For example, in the \$print method of the header section you may place the following code to turn of the watermark after page 1:

```
If #P=2
  Calculate ivPDFParams.[kDevPdfWatermark] as kFalse
End if
Do default
```

If you simply want to use the report object to set device parameters without effecting the global parameters, you can set all parameters during the report's \$construct method, in which case you may place the report object in the Report header section, so it is only printed once during the print job (i.e. send the device parameters just the once to the device).

Version Numbers

It is advisable that you check the correctness of the version numbers of any third-party components to ensure you always distribute the correct versions. Please refer to technical note [TN0022](#) to find out how to programmatically check the version numbers of our software.

View example code...

Multi-threaded Server (version 4.0)

PDF Device version 4 implements some major improvements in the way you can use the device with the Omnis multi-threaded server. The following sections describe the key actions that you must perform when using the new Brainy Data PDF device server (hereafter referred to as BDPDF server) in conjunction with the Omnis server. The four key actions are summarized as follows:

- Preparing and starting the PDF server. See sub-section Library Startup
- Deciding and setting the output destination. See sub-section Setting the Destination
- Perform the printing. See sub-section Printing
- Return PDF to client. See sub-section Display PDF on Client

Note: This documentation is best read alongside studying the accompanying example library [PDFDeviceAndJSClient.lbs](#).

Library Startup

Wherever you normally execute the Omnis *Start server* and *Stop server* commands, you should now perform the following additional actions.

Preparing device settings

It is advisable to prepare the main device settings for use by the server, prior to starting the device server. Starting the device server will create copies of these settings for each Omnis server thread.

Recommendation: For performance reasons always turn off the option *kDevPdfImgConvBest* as this option substantially affects performance when reports contain images.

Starting the device server

The device server can be started by executing the function PDF Device.\$startserver(), which is available from the Catalog function tab. When executed, the external will create a device instance for each Omnis server thread and copy the device settings from the application's main device instance. From this moment onwards, whenever device notation such as \$setparam(), \$getparam(), \$open(), \$close() is executed, they will address the device instance associated with the current executing thread. Consequently, any actions are isolated from the other threads.

Consideration: Multiple remote form/task instances will share an execution thread, not at the same time, but sequentially, which means that changes to device settings can affect other form or task instances. See the section 'Managing Device Settings' for a detailed discussion on this subject.

Example:

```
; start the Omnis server
Start server
; prepare device settings for our threaded devices
Do $cdevice.$assign(kDevPdf)
Do $cdevice.$setparam(kDevPdfImgConvBest, kFalse, kDevJPEGQuality, 90)
; start the device server
Do PDF Device.$startserver()
```


Stopping the device server

Where you typically stop the Omnis Server, that is where you should also stop the device server, albeit after you have already stopped the Omnis server.

Example:

```
; stop the Omnis server first
Stop server
; now it should be save to stop the device server
Do PDF Device.$stopserver()
```

Setting the Destination

When receiving a print request from a client, one of the first actions will be setting the destination file or variable. Whether you use a file or variable output depends very much on your client implementation and needs. If you want to use the client commands “showpdf” or “assignpdf” you must use a PDF file. If you intend to embed the PDF in a HTML control or are responding to an ultra-thin request, you can use a variable and directly return the data as the result of the client request. Using a variable has various logistic and performance benefits.

Using temporary files

For this purpose, version 4 implements the function PDF Device.\$settempfilename. Using this method, you can specify a time limit in minutes in parameter one. If you specify zero, the file is permanent, i.e. it is not removed after a period of time. The method \$settempfilename will return the temp destination folder and unique file name in parameters two and three, unless these parameters are not empty to begin with. Please see the discussion in the reference chapter.

Example of use:

```
; set the destination
Do PDF Device.$settempfilename(1,lvPath,lvName)
; print report(s)
...
; return PDF to client
...
```

Using permanent files

As indicated above, the function \$settempfilename() can also be used to send output to a PDF file that will not be deleted by the device server (i.e. by specifying zero for parameter one). You can of course continue to use \$setparam(kDevPdfFileName,...) to send output to a permanent file but you will have to calculate the destination path and unique file name yourself.

Using a variable

Output can directly be send to an Omnis binary variable. Sending output to a local, instance or task variable assumes that your implementation wishes to directly return PDF mime content or base64 data to the client or to store the pure PDF data in a database. If the former, it also assumes that the action to return the PDF data was sent either by:

1. An ultra-thin request using a special URL requesting the PDF mime data of the output to be returned. In this case, parameter two must specify a download file name which causes the PDF data to be prefixed by a mime header containing the specified file name and other details. Example 3 in the example library [PDFDeviceAndJSClient.lbs](#) demonstrates this scenario.
2. The returned base64 data is to be embedded in an HTML object using the embed tag with a 'data' URL. Example 1 in the [aforementioned library](#) demonstrates how this can be achieved.

A simple example of both cases is documented in the section Examples at the end of this chapter.

WARNING! The variable that you specify must be in scope when the report finishes printing (or the device is closed using \$close() when printing multiple reports). In other words, if you are using a local variable, the print job must be completed prior to the method going out of scope. The PDF device server cannot tell if a variable that was specified still exists by the time it comes to write the PDF data. **If the variable is allowed to go out of scope prior to the print job completing, Omnis may crash.**

Example of use:

```
; set the destination
Do PDF Device.$setmemoryoutput (nam(lvPdfOutput), "report.pdf")
; print report(s)
...
; return PDF to client
```

Printing

Once the destination has been set, the server thread can either just print a single report or multiple reports to the same destination.

Single Report

Printing a single report to PDF is as simple as setting the report name and executing the Omnis *Print* command. By the time the *Print* command returns control to you, the PDF data will have been written to your file or variable.

Multiple reports

To print multiple reports, you use the device notation \$open() to open the PDF device output stream, prior to printing your reports. **When using PDFDevice version 4 it is now save to do this in the Omnis multi-threaded server.** After you have printed all the reports to be included, you must call the device notation \$close() to complete the PDF output.

Example of use:

```
; open the device for multi-report output
Do $cdevice.$open()
; print the reports
...
; close the device
; ;; this action writes the PDF content to your file or variable
Do $cdevice.$close()
```

Display PDF on Client

The final action is to return the PDF file or mime data to the client.

Using Omnis client commands 'showpdf' or 'assignpdf'

If you have generated an output file, you may use the client commands 'showpdf' or 'assignpdf'. You must concatenate the path and name as returned by \$settempfilename() when using these commands.

Example:

```
$cinst.$clientcommand('showpdf', row(con(lvPath,lvName),1,lvName))
```

or

```
$cinst.$clientcommand('assignpdf', row('ctrlEmbedPDF', 'toolbar=1',  
con(lvPath,lvName), 1, lvName ))
```

Returning PDF mime data

If the remote task was called via an ultra-thin request and you sent the output to a memory variable, you can simply return the PDF mime data in your variable using the Quit method command. See section 'Using a memory variable' above.

Example:

```
If binlength(lvPdfOutput)  
; we can just return the data as is because its prefixed with a mime  
; header by the external device (see $setmemoryoutput)  
Quit method lvPdfOutput  
End If
```

Examples

This section lists basic examples that demonstrate: choosing a destination, printing, and returning PDF to the client in various scenarios.

Using a temp temp file

```
# set the temp file destination with a 3 minute life-span
Calculate lvPath as ""
Calculate lvName as "myprefix"
Do PDF Device.$settempfilename(3,lvPath,lvName)
# on return from $settempfilename
# - lvPath will point to the Omnis temp folder
# - lvName will be set to 'myprefixYYMMDDhhmmn.pdf'

# now we print the report
Do $cdevice.$assign(kDevPdf)
Set report name eX_Report
Print report

# show the report in our html control using 'assignpdf'
# - lvName specifies the document name for when the client saves the file
Calculate lvRow as
    row('ctrlEmbedPDF','toolbar=1',con(lvPath,lvName),1,lvName)
Do $cinst.$clientcommand('assignpdf',lvRow)
```

Using a permanent file

```
# set the temp file destination with a 3 minute life-span
Calculate lvPath as "c:\temp"
Calculate lvName as "myprefix"
Do PDF Device.$settempfilename(3,lvPath,lvName)
# on return from $settempfilename
# - lvPath will point to "c:\temp"
# - lvName will be set to 'myprefixYYMMDDhhmmn.pdf'

# now we print the report
Do $cdevice.$assign(kDevPdf)
Set report name eX_Report
Print report

# show the report in our html control using 'assignpdf'
# - lvName specifies the document name for when the client saves the file
Calculate lvRow as row(con(lvPath,lvName),1,lvName)
Do $cinst.$clientcommand('assignpdf',lvRow)
```

Using a binary variable for embedding

Requires two methods. One for producing and returning PDF on server, the other to receive the PDF data on return which embeds it.

```
Server: RemoteForm.$printForEmbed =====
# print for embedding in our ctrlEmbedPDF control 2
# 1. prepare our output device □
Do $cdevice.$assign(kDevPdf)
# 1.a ## apply instance's device settings to external device
```

```

Do $cdevice.$setparam(kDevPdfAll,ivDeviceParams)
# 1.b ## tell PDFDevice to send output to our local variable
Do PDF Device.$setmemoryoutput(nam(lvPdfData))

# 2. print our report
Do method $cinst.$printDocument

# 3. return result for display on client
If binlength(lvPdfData)
  # ## we can just return the data as is because its prefixed with a
  mime header (which we requested when calling $setmemoryoutput)
  Quit method lvPdfData
End If

Client: RemoteForm.$printForEmbed_return =====
# assign PDF data returned by the server to our html control using the
embed tag
Calculate $cinst.$objs.ctrlEmbedPDF.$html as con('<embed
style="border: 1px solid rgb(128,128,128)" width="100%" height="100%"
src="data:application/pdf;base64,',pPdfData,'">')

```

Using a binary variable for ultra-thin requests

```

Client: HTML =====
<!DOCTYPE html>
<html><head><style>body {background-color:
rgb(255,255,221);}</style></head><body leftmargin="50">
<h1>PDFDevice ultra-thin client Example</h1>
<p>The submit button below executes an ultra thin call and opens the returned
content in a new browser window:</p>
<form action="http://127.0.0.1:61098/ultra" target="_blank">
  <input type="hidden" id="OmnisServer" name="OmnisServer"
value="'127.0.0.1:61098'">
  <input type="hidden" id="OmnisLibrary" name="OmnisLibrary"
value="PDFDeviceAndJSClient">
  <input type="hidden" id="OmnisClass" name="OmnisClass"
value="e3_RemoteTask">
<h3>File Options</h3>
  <label for="FileName">PDF File Name:</label>
  <input type="text" id="FileName" name="FileName"
value="UltraThinLetter.pdf">
  <input type="checkbox" id="PrintBlurp" name="PrintBlurp" value="1">
  <label for="PrintBlurp">Print Blurp</label><br><br><br>
<h3>Report Data</h3>
<blockquote>
  <label for="Title">Title:</label><br>
  <input type="text" id="Title" name="Title" value="Mr."><br><br>
  <label for="FirstName">First name:</label><br>
  <input type="text" id="FirstName" name="FirstName" value="John"><br><br>
  <label for="LastName">Last name:</label><br>
  <input type="text" id="LastName" name="LastName" value="Doe"><br><br><br>
  <input type="submit" value="Submit">
</blockquote>
</form>
</body></html>

```

```

; Sever: RemoteTask.$construct =====
# ultra-thin call to print report and return PDF
# 1. prepare our output device
Do $cdevice.$assign(kDevPdf)
# ## for ultra-thin connects restore device settings to those from main
# application thread so we do not inherit
# ## some other client's settings who has left them lying about in this
# server-thread
Do PDF Device.$initparams()
# 1.b ## tell PDFDevice to send output to our local variable
# ## IMPORTANT: we can only use a local var if this function completes the
# print-job
Do PDF Device.$setmemoryoutput(nam(lvPdfData),pParams.FileName)

# 2. print our report
Set report name eX_Report
Print report * (pParams) ## pass form data to report instance

# 3. return result for display on client
Quit method lvPdfData

```

Printing multiple reports

```

# set the destination with a 1 minute life-time for the file
Calculate lvPath as ""
Calculate lvName as "myprefix"
Do PDF Device.$settempfilename(1,lvPath,lvName)
# on return
# - lvPath will point to the Omnis temp folder
# - lvName will be set to 'myprefixYYMMDDhhmmn.pdf'

# print the reports
Do $cdevice.$assign(kDevPdf)
Do $cdevice.$open()
Set report name eX_Report
Print report
Set report name eX_ReportPDFBlurb
Print report
Set report name eX_ReportPDFKeyActions
Print report
Do $cdevice.$close()

# show the report in our html control using 'assignpdf'
# - lvName specifies the document name for when the user saves the file
Calculate lvRow as row('ctrlEmbedPDF','toolbar=1',con(lvPath,lvName))
Do $cinst.$clientcommand('assignpdf',lvRow,1,lvName)

```

Managing Device Settings

As was mentioned in the section ‘Library Startup’, the device server will create a single device instance for each server thread. This guarantees that any device settings that are changed, are only changed for the thread in which the change occurs. However, each thread is not uniquely used by a single remote form or task instance which means that if task A makes a change on thread 3, when another task’s method (i.e. task B) executes on thread 3 later on, this method will inherit any changes task A applied previously. There are four possible scenarios that must be considered.

1. The changed settings don’t matter

It may be that the device settings that are changed by a remote task are limited to setting the destination, which is carried out by every method that intends to print. In this case there is nothing else to consider.

2. The changed settings do matter

In this case, prior to printing and applying new settings that are specific to this job, the function `PDF Device.$initparams()` can be called to first restore the device settings to those of the main application thread.

3. Device settings are to be maintained for each user between requests

You may allow client’s to modify the device parameters as part of their session. This means that device settings for the user must be maintained between server requests. In this case, the following sub-scenarios must be considered:

- i. Client connects for the first time. Use `PDF Device.$initparams(ivParams)` to initialise the device to the application default and at the same time receive these settings in your binary instance variable that maintains the client settings between requests.
- ii. Client edits settings or prints. Use `$cdevice.$setparam(kDevPdfAll,ivParams)` to load the client’s settings into the current execution thread. If the client changes settings via your JS interface, use your binary instance variable to directly manipulate the device settings on the client. The example library [PDFDeviceAndJSClient.lbs](#) includes the remote form `rfPDFOptions` and various support classes that implement such an interface. This interface is used by Examples 1 and 2 in the example library.

4. Device settings are to be maintained for each user between sessions

A form instance may have a log-in so that the user is known and clients have their own device settings that the server maintains between sessions. In this case, you must consider sub-scenarios 3.i and 3.ii above and in addition, within each of these sub-scenarios, store the device settings in `ivParams` with the client’s other data for maintaining the settings between sessions.

Further Reading

There are a number of technical notes available on our [website](#) that you should read before you integrate PDF Device into your application(s). These are

- [TN0014](#) ‘External Device Parameters’ explains issues surrounding Omnis device parameters.
- [TN0015](#) ‘Performance’ explains the impact various device options can have on performance.
- [TN0017](#) ‘Font Embedding’ gives a detailed description about the importance of font embedding and potential issues. It also explains how to monitor font related errors and warnings to fend off potential problems with documents.
- [TN0021](#) ‘Omnis Studio V6 Compatibility’ is relevant to developers who have implemented PDFDevice prior to version 3 and need to upgrade to Omnis Studio version 6.
- [TN0022](#) ‘External Component Version Numbers’ explains how to programmatically check the version numbers of our software.

External Reference

This chapter documents the constants for getting and setting device parameters, it's methods and properties. Additional descriptions for the generic device methods and properties can be found in the Omnis help.

Contents

Device Parameters - for setting and queering device properties using the methods \$setparam and \$getparam. Device parameters are organised into the following groups; Basic Parameters, General Parameters, Watermark Options, Document Info, Viewer Options, Bookmarks, Compatibility Options, Background Image, Security Options.

Constants - for specifying values for device parameters and static methods.

Device Properties - lists the standard Omnis device instance properties and describes how they apply to PDFDevice.

Device Methods - lists the standard Omnis device instance methods and describes how they apply to PDFDevice

Static Methods - non-standard methods provided by PDFDevice.

Report Objects - lists the Acroform and PDF parameters report objects and their properties.

Form Field Constants - lists the constants for use with the Acroform field properties.

Device Parameters

You set and get device properties using the methods \$setparam and \$getparam together with the constants listed below.

Example:

```
Calculate $cdevice as kDevPdf
Do $cdevice.$setparam(kDevPdfEmbedFonts,kTrue,kDevPdfEmbedLicFonts,kFalse)
Calculate fname as $cdevice.$getparam(kDevPdfFileName)
```

Note: Device parameters are stored in the Omnis configuration file (omnis.cfg). When Omnis starts all device parameters are automatically loaded.

Basic Parameters

Device parameters that can not be described within a well defined group.

Name	Value	Description
kDevPdf	varies	The unique registration ID of the PDF device (the value of this constant is assigned by Omnis during startup). You can use this constant to refer to or set the current device. Example: Calculate \$cdevice as kDevPdf
kDevPdfAdvanced	3	The Omnis command to be executed when the "Advanced..." button is clicked. Please read the Chapter Designing PDFDevice for a full description.
kDevPdfAll	255	This parameter is used to fetch and set all device parameters in a row variable. The column names are the constant names without the prefix kDevPdf. You may not alter the order of the columns as PDFDevice uses the column numbers to identify the parameters.
kDevPdfFileName	2	Sets the destination file name or notation to a method when printing to memory. Refer to the chapter Designing PDFDevice for further details.
kDevPdfTempFileTime (v4.0)	56	Time in minutes after which to delete the output file that was set by kDevPdfFileName. This parameter must be set after setting kDevPdfFileName as kDevPdfFileName resets the time to zero. Zero means do not delete the file.

General Parameters		
Device parameters that manage the general options such as Image and Font handling.		
Name	Value	Description
kDevPdfConvJPEG	4	If set to kTrue, Images will be converted to JPEG using the current JPEG quality setting. Converting images to JPEG may reduce the quality. If this option is not selected, 24 bit RGB values are compressed using standard deflate compression with no loss of quality, but possible loss of compression.
kDevPdfConvLinks	8	If set to kTrue, this option converts any occurrences of the text “http://” and the text “www.” to Link annotations. Links to files on disk can also be established by using standard URI paths beginning with “file://”.
kDevPdfEmbedFonts	6	If set to kTrue, this options will subset and embed true type fonts for better cross platform results. See also kDevPdfSubstituteFonts.
kDevPdfEmbedLicFonts	7	If set to kTrue, this option embeds copyrighted fonts. Please make sure that you have a license that permits the embedding of these fonts. If you are unsure, do not select this option.
kDevPdfImgConvBest (v1.3)	34	If set to kTrue and kDevPdfConvJPEG is true, PDFDevice will choose between the specified JPEG compression and the default RGB deflate compression for each image. Some images will yield better compression ratios when using RGB deflate. Enabling this option may reduce performance slightly, as PDF device has to apply both compressions to each image.
kDevPdfImgStripDup (v1.3)	35	If set to kTrue, duplicate images will be stripped from the document. This can reduce the final file size substantially if you are repeatedly using the same image throughout a document.
kDevPdfJPEGQuality	5	The quality of the images when kDevPdfConvJPEG is true. The valid range is 0 to 100, where 100 means high quality with low compression, and 0 means low quality with high compression. A value of 75 generally produces a good rate of compression without compromising too much on quality.

www.brainydata.com

kDevPdfPictMetaDPI	31	This option controls the quality of Macintosh PICT and Windows Meta picture conversion. Valid range is 72 to 600. Default is 150. Increasing the resolution to improve quality will increase the size of the PDF file.
kDevPdfSubstituteFonts (v3.0)	51	Substitute fonts that cannot be embedded with true-type fonts that can be embedded. (see technical note TN0017)

Watermark Options

PDFDevice allows the placement of text based watermark on the pages. Watermarks can be placed in front of or behind, the pages content and can be rotated at angles of up to 360 degrees anti-clockwise or clockwise. Watermarks are limited to 255 characters of single line text. The following constants are used to set watermark specific parameters.

Name	Value	Description
kDevPdfWatermark	9	If set to kTrue, watermarks are placed on every page.
kDevPdfWMAngle	14	The rotation of the text in degrees anti-clockwise. Valid range is from 0 (no rotation) to 359.
kDevPdfWMFont	11	The font name for the watermark. This must be a font name that exists in the operating system.
kDevPdfWMFontSize	12	The font size in points. Valid range is 4 to 128.
kDevPdfWMHorzPos	16	The horizontal starting position from the left of the paper edge, measured in centimeters or inches. The library preference \$usecms determines the unit of measure.
kDevPdfWMInFront	15	If set to kTrue, the watermark will appear in front of the page content.
kDevPdfWMText	10	The text to be displayed. This parameter is limited to 255 characters of single line text.
kDevPdfWMTextColor	13	The RGB value of the text color. You can use the Omnis rgb() function to specify the value.
kDevPdfWMVertPos	17	The vertical starting position from the top of the paper edge, measured in centimeters or inches. The library preference \$usecms determines the unit of measure.

Document Info

Device parameters that manage the document info. Document info is stored in the PDF's document info dictionary and can be viewed using Acrobat or other readers that can display document info.

Name	Value	Description
kDevPdfInfoAuthor	19	The name of the person who created the document.
kDevPdfInfoKeywords	21	Keywords associated with the document.
kDevPdfInfoSubject	20	The subject of the document.
kDevPdfInfoTitle	18	The document's title.

Viewer Options

Device parameters that manage the PDF viewer options. Please note that not all PDF viewer applications implement these options.

Name	Value	Description
kDevPdfCenterWindow	28	If set to kTrue, position the document's window in the center of the screen.
kDevPdfDisplayTitle	29	If set to kTrue, position the document's window in the center of the screen.
kDevPdfFitWindow	27	If set to kTrue, the PDF viewer application will resize the document window to fit the size of the first page.
kDevPdfHideMenubar	25	If set to kTrue, hide the viewer application's menu bar when the document is active.
kDevPdfHideToolbar	24	If set to kTrue, hide the viewer application's tool bar when the document is active.
kDevPdfHideUI	26	If set to kTrue, hide user interface elements in the document window, such as scroll bars and navigation controls.

kDevPdfPageLayout	22	<p>This parameter specifies the page layout to be used when the document is opened. The parameter takes a string as its value. Possible values are.</p> <p>SinglePage Display one page at a time.</p> <p>OneColumn Display the pages in one column.</p> <p>TwoColumnLeft Display the pages in two columns, with odd numbered pages on the left.</p> <p>TwoColumnRight Display the pages in two columns, with odd numbered pages on the right.</p>
kDevPdfPageMode	23	<p>This parameter specifies how the document should be displayed when opened. The parameter takes a string as its value. Possible values are.</p> <p>UseNone Neither document outline nor thumbnail images visible.</p> <p>UseOutlines Document outline visible.</p> <p>UseThumbs Thumbnail images visible.</p> <p>FullScreen Full-screen mode, with no menu bar, window controls, or any other window visible</p>

Bookmarks

PDFDevice is capable of producing document bookmarks (outline trees) from your existing reports as long as your titles use consistent font, size and style for each level. You can specify the font name, size and style in the PDF options and PDFDevice will add bookmarks for text that matches the given criteria, to the document outline tree. You can create trees with numerous levels by specifying different criteria for each outline level.

The outline font matching criteria is stored in an Omnis list with three columns named FontName, FontSize and FontStyle. Each row in the list creates a new outline level in the final document. The first list row is the root level.

Name	Value	Description
kDevPdfOutlines	30	Sets or gets the outline font matching criteria.

Compatibility Options

The following are compatibility options. At some stage functional changes were made that may effect your output. To counteract any unwanted side effects you can change the compatibility settings to turn off the new behaviour.

Name	Value	Description
kDevPDF72DPILines (v1.1)	31	In version 1.0, a lines thickness was calculated based on the platforms screen resolution. That meant that lines were drawn slightly thicker on windows platforms. In version 1.1, turning this option on will ensure that a lines thickness is based on a 72DPI resolution on all platforms.
kDevPDFIgnoreFontStyle	36	OBSOLETE in version 3 (see technical note TN0017). Prior to version 3, PDFDevice will not embed fonts if the font file does not contain glyph data for the requested style. Turning on this option forces PDF device to embed the available glyph data even if the styles are not supported by the font.
kDevPDFPgCntPerReport (v1.2)	32	In version 1.1 and earlier, page count objects would display a single page range, even when printing multiple reports to the same PDF file. It was recognised that this is a difference in behaviour from when printing to other devices. Nevertheless, it is considered usefull and as such an option was added to enable or disable this feature. When individual page ranges are required for each print job, set this device option to kTrue.
kDevPdfaEnabled (v3.0)	52	Enables PDF/A-b1 support based on the ISO 19005 standard. Enabling this option will generate PDF/A compliant documents. See technical note TN0023 for full details.
kDevPdfaOutputInfo (v3.0)	53	Information string for custom output intent color profile provided by kDevPdfaOutputProfile. See technical note TN0023 for full details.
kDevPdfaOutputProfile (v3.0)	54	Output intent color profile data (*.icc). See technical note TN0023 for full details.
kDevPdfPrintScale (v3.1)	55	Sets the scaling factor for the document (valid range 25% to 400%) Note: In versions prior to version 3.1.0, PDFDevice would retrieve intended scaling from the Omnis \$scale property.
<p>Background Image (v2.0)</p> <p>The following device parameters control the background image properties. Background images are placed relative to the top left of the page. There are various parameters that control the positioning and size of the image.</p>		

www.brainydata.com

Name	Value	Description
kDevPdfBkgImgData	38	Image data or notation that returns the image data.
kDevPdfBkgImgDPI	44	Specifies the DPI at which image is drawn. Used when kDevPdfBkgImgScale is turned off.
kDevPdfBkgImgFirstPageOnly	45	If true, background image is only added to first page.
kDevPdfBkgImgHPos	42	Horizontal offset of image from paper edge. Used when kDevPdfBkgImgScale is turned off.
kDevPdfBkgImgKeepAspect	41	If true, aspect ratio is maintained when image is scaled.
kDevPdfBkgImgOn	37	If true and kDevPdfBkgImgData is not empty, PDFDevice will add a background image to every page.
kDevPdfBkgImgScale	39	If true, background image is scaled to fit the entire page minus the amount specified by kDevPdfBkgImgScaleBorder.
kDevPdfBkgImgScaleBorder	40	Specifies the gap between image and paper edge when kDevPdfBkgImgScale is enabled.
kDevPdfBkgImgVPos	43	Vertical offset of image from paper edge. Used when kDevPdfBkgImgScale is turned off.

Security Options (v2.0)

These device parameters specify the security options and user permissions for the final PDF document. PDFDevice will encrypt documents if either one of the Owner or User passwords is supplied. If both passwords are empty, no encryption takes place.

Note: If a user-password is specified but the owner-password is empty, PDF viewers may only provide user access permissions to the resulting PDF document. They may not provide a way for entering an empty password for full Owner permissions.

Note: If a owner-password is specified but the user-password is empty, a user will not be prompted to enter a password when opening the document and operations will be limited according to the permission flags. Some viewers allow the subsequent entering of the owner password for full access permissions.

Name	Value	Description
kDevPdfOwnerPassword	46	The owner password grants full access to the resulting PDF documents.

kDevPdfPermissions	49	Specifies user permissions. PDFDevice provides a set of constants for setting and querying the various options. You can use the Omnis functions bitor(), bitxor() and bitand() to set, clear and query the individual permissions.
kDevPdfUse128BitEncryption	48	By default, data is encrypted using 40 bit encryption keys. If this option is set, 128 bit encryption keys are used to achieve a higher level of encryption.
kDevPdfUserPassword	47	The user password grants restricted access to the resulting PDF documents as specified by kDevPdfPermissions.

kDevPdfPermissions Examples:

```
;; get the permission flags (lPerms must be a long integer)
Do $cdevice.$assign(kDevPdf)
Do $cdevice.$getparam(kDevPdfPermissions) Returns lPerms

;; add print and copy permissions
Calculate lPerms as bitor(lPermissions,kDevPdfPermPrint+kDevPdfPermCopy)

;; remove the copy permission
Calculate lPerms as bitxor(lPermissions,kDevPdfPermCopy)

;; test if either or both the print or copy permissions are enabled
If bitand(lPermissions,kDevPdfPermPrint+kDevPdfPermCopy)

;; set the permissions
Do $cdevice.$setparam(kDevPdfPermissions,lPerms)
```

Constants

PDFDevice provides a set of additional constants for its device parameters and static methods.

kDevPdfEncrypt... (v5.0)

Group of constants for specifying encryption when calling the function \$fileencrypt().

Name	Value	Description
kDevPdfEncryptRC4V1	1	use 40 bit RC4 encryption, outputs PDF v1.4
kDevPdfEncryptRC4V2	2	use 40-128 bit RC4 encryption, outputs PDF v1.4
kDevPdfEncryptAESV2	4	use 128 bit AES encryption, outputs PDF v1.5
kDevPdfEncryptAESV3	8	use 256 bit AES encryption, outputs PDF v1.7

kDevPdfPerm... (v2.0)

Group of constants for setting user permissions. See kDevPdfPermissions.

Name	Value	Description
kDevPdfPermAssemble	1024 0x0400	the user can assemble documents (insert, rotate, or delete pages and create bookmarks or thumbnail images)
kDevPdfPermCopy	16 0x0010	the user can copy or otherwise extract text and graphics from the document by operations other than that controlled by kDevPdfPermExtract.
kDevPdfPermExtract	512 0x0200	the user can extract text and graphics (in support of accessibility to disabled users or for other purposes)
kDevPdfPermFillFields	256 0x0100	the user can fill in existing interactive form fields (including signature fields)
kDevPdfPermFillFieldsComments	32 0x0020	the user can add or modify text annotations, fill in interactive form fields, and, if kDevPdfPermModify is checked, create or modify interactive form fields (including signature fields).
kDevPdfPermModify	8 0x0008	the user can modify the contents of the document by operations other than those controlled by kDevPdfPermFillFieldsComments, kDevPdfPermFillFields and kDevPdfPermAssemble.
kDevPdfPermPrint	4 0x0004	the user can print the document.

kDevPdfPermPrintFull	2048 0x0800	the user can print the document to a representation from which a faithful digital copy of the PDF content could be generated. When this option is off, printing is limited to a low-level representation of the appearance, possibly degraded quality. Requires kDevPdfPermPrint to be enabled.
----------------------	----------------	---

Device Properties

Below are the device properties and their initial values. Properties in red are read-only and cannot be altered. For a more detailed description of these properties, please consult the Omnis documentation.

Name	Default	Description
\$cangeneratepages	kTrue	PDF device generates paged reports
\$scankeepopen	kFalse	PDF device should not be left open for prolonged periods. Opening PDF device using \$open will create and open the destination file.
\$iconid	empty	The alternative icon to be displayed in the destination dialog.
\$ident	varies	The unique ID of the PDF device. The ID is assigned during startup of Omnis.
\$isopen	kFalse	Returns true if the device is currently open.
\$istextbased	kFalse	PDF Device is a graphic based output device and can handle all Omnis report objects.
\$name	“PDF”	The fixed name of the device.
\$title	“PDF”	The name to be displayed in the destination dialog.
\$visible	kTrue	If set to kTrue, the device will appear in the destination dialog.

Device Methods

Below are listed the standard device methods. For a more detailed description of these methods, please consult the Omnis documentation.

\$scanclose()

Syntax: `$devices.PDF.$scanclose(bQuit)`

Version: 1.0

Call this method to determine if the device can be closed. You should never have a need to call this method.

Parameter	Description
bQuit	Tells the device if the close is due to a quit operation.
returns	kTrue if the device can be closed.

\$close()

Syntax: `$devices.PDF.$close()`

Version: 1.0

Closes the device. You must have called `$open` prior to calling `$close()`.

Parameter	Description
returns	kTrue if the device was open and has been closed.

\$flush()

Syntax: `$devices.PDF.$flush()`

Version:

PDFDevice does not implement this method. Writing of the final file output is handled during the closing of this device. During printing all output is buffered to an intermediate file. This method is generally only used when writing data directly to the device using `$senddata()`. PDFDevice does not support `$senddata()`.

Parameter	Description
returns	returns kFalse

\$getparam()

Syntax: `$devices.PDF.$getparam(nParamNumber)`

Version: 1.0

Use this method to get the values of the device parameters.

nParamNumber	Specifies the parameter number. Use one of the kDevPdf... constants.
returns	the value of the parameter.
<p>\$open()</p> <p>Syntax: \$devices.PDF.\$open()</p> <p>Version: 1.0</p> <p>Opens the device for output. PDFDevice will create and open the destination file. If the destination file parameter is empty, the user will be prompted to specify a valid file name. You only open the device manually if you intend to print more than one report to the same PDF file. When you have finished printing, you must call \$close(), so the destination file is completed and closed.</p>	
Parameter	Description
returns	kTrue if the device was opened.
<p>\$senddata()</p> <p>Syntax: \$devices.PDF.\$senddata()</p> <p>Version:</p> <p>PDFDevice does not support this method.</p>	
Parameter	Description
returns	kFalse
<p>\$sendtext()</p> <p>Syntax: \$devices.PDF.\$sendtext()</p> <p>Version:</p> <p>PDFDevice does not support this method.</p>	
Parameter	Description
returns	kFalse
<p>\$setparam()</p> <p>Syntax: \$devices.PDF.\$setparam(nParamNumber,value[nParamNumber,value,...])</p> <p>Version: 1.0</p> <p>Use this method to set the values of device parameters. You can set several parameters with a single call to \$setparam.</p> <p>Note: Device parameters are stored in the Omnis configuration file (omnis.cfg). When Omnis starts all device parameters are automatically loaded.</p>	

www.brainydata.com

Parameter	Description
nParamNumber	Specifies the parameter number. Use one of the kDevPdf... constants.
value	The new value for the parameter.
returns	kTrue if successful.

Static Methods

Non-standard methods provided by PDFDevice. These methods are available from the Function tab of the Omnis Catalogue in the PDF Device group.

\$fileencrypt()

Syntax: PDF Device.\$fileencrypt(cInputFile, cOutputFile, cUserPassword, cOwnerPassword, iEncryption, iPermissions) Returns err

Version: 5.0.0.0

Encrypt the specified PDF file.

Parameter	Description
cInputFile	The PDF file to be encrypted.
cOutputFile	The output PDF file name.
cUserPassword	The user password that grants access to the files according to the permissions that are set via iPermissions. The user password may be empty in which case anyone can open the file with the limitations as set by iPermissions.
cOwnerPassword	The owner password that gives full access to the PDF file.
iEncryption	The encryption algorithm that will be used to encrypt the file. This can be one of the kDevPdfEncrypt... constants.
iPermissions	The permission for the user password. This can be any combination of the kDevPdfPerm... constants. Note: If the PDF file is unlocked with the owner password, full permissions are granted.
returns	Returns 1 if successful, 0 or a negative error code on failure. the function failed you can call \$filegetlasterror() to retrieve additional error information.

\$filemerge()

Syntax: PDF Device.\$filemerge(cInputFile1, cInputFile2, cOutputFile) Returns err

Version: 5.0.0.0

Merges two PDF files appending cInputFile2 to cInputFile1.

Parameter	Description
cInputFile1	The PDF file that is to be appended to.
cInputFile2	The PDF file that is to be appended at the end of cInputFile1.

cOutputFile	The output PDF file name.
returns	Returns 1 if successful, 0 or a negative error code on failure. the function failed you can call \$filegetlasterror() to retrieve additional error information.

\$filereaddata()

Syntax: PDF Device.\$filereaddata(cInputFile, cOutputListName) Returns err

Version: 5.0.0.0

Read the data from form fields within the specified PDF file and return their names and values in a list.

Parameter	Description
cInputFile1	The PDF file that is to be read.
cOutputListName	<p>The list name that is to receive the data from the form in the PDF file. On return, the list will be populated with the following columns if it had none to begin with:</p> <ul style="list-style-type: none"> FieldName: The internal name of the field AltName: The readable name for messages MapName: The name to be used for submits Value: The field's value <p>If you define the list before calling this method, you must ensure it contains at least four columns or an error is returned. You may use the same list to read multiple files, in which case the fields of subsequent files are appended at the end of the list.</p>
returns	Returns 1 if successful, 0 or a negative error code on failure. the function failed you can call \$filegetlasterror() to retrieve additional error information.

\$filesign()

Syntax: PDF Device.\$filesign(cInputFile, cOutputFile, cCertificateFile, cKeyFile, cKeyFilePassword [, cSignReason, bSigPrintable=kFalse, bSigUseExisting=kTrue, cSigFieldName, wAnnotationAtts]) Returns err

Version: 5.0.0.0

Sign the specified PDF file.

Parameter	Description
cInputFile	The PDF file to be signed.
cOutputFile	The output PDF file name.

cCertificateFile	The certificate file, either PEM (*.pem) or P12 (*.p12) format. When cCertificateFile file is a P12 file that includes both the certificate and private key, cKeyFile can be left empty but cKeyFilePassword must be specified as it is needed to extract the information from the P12 file.
cKeyFile	The path name to the private key file in PEM format (*.pem).
cKeyFilePassword	The file password for the private key file specified by cKeyfile, or for the certificate file specified by cCertificateFile, if that file is a .p12 file that combines the certificate and private key data.
cSignReason	The reason for the signature (i.e. "I agree" or "Signing on behalf of...", etc)
bSigPrintable	If true, the signature can be printed.
bSigUseExisting	CURRENTLY NOT SUPPORTED
cSigFieldName	The internal name of form field that will be created as part of the signature.
wAnnotationAtts	<p>A row variable that specifies annotation attributes via an additional subset of row variables, the total having the nested structure as follows:</p> <pre> ## annotation attributes - all measurements are in cms or inch depending on \$prefs.\$usecms row(## annotation box location row(PageNumber, Left, Top, Width, Height), ## image file path and location within annotation box row(Left, Top, Width, Height, ImageFilePath) ## text to be displayed within annotation box row(Left, Top, Text)) </pre>

	<pre>## font information for annotation text - for the font name use the text value of one of the kDevPdfffSF... constants, i.e. use con(kDevPdfffSFHelvetica) to extract the font name row(FontName, FontSize, TextColor))</pre>
returns	Returns 1 if successful, 0 or a negative error code on failure. the function failed you can call \$filegetlasterror() to retrieve additional error information.

\$getfonterrors()
 Syntax: PDF Device.\$getfonterrors(&IFontErrors)
 Version: 2.6.0 (updated for version 3.0.0)
 Returns a list of errors and warnings related to font embedding since the last call to \$getfonterrors. Calling this method will clear the errors.

Parameter	Description
IFontErrors	Specifies the list variable that is to be populated with the errors. The list is populated with three columns. <ol style="list-style-type: none"> 1. ErrorNumber - a number in the range 12 to 16 2. ErrorText - a description of the error 3. FontName - the postscript name of the effected font The possible errors are <ol style="list-style-type: none"> 12. Font embedding error. Font is not a true type font. 13. Font embedding error. Font is not licensed for sub-setting or embedding. 14. OBOSOLETE in version 3 15. Font embedding warning. Font licensing ignored. 16. Font embedding warning. Font style setting is not supported with this font and was synthesised. 17. Font substitution warning. Font has been substituted. 18. Font character warning. Font cannot be embedded and mapping to 8bit OS character failed. Please refer to technical note TN0017 for full details.
returns	kTrue if successful.

\$getmemoryoutput(iOutputId,&xOutputVariable)
OBSOLETE: since version 4.0 use \$setmemoryoutput instead

www.brainydata.com

\$sinitparams()

Syntax: PDF Device.\$sinitparams([&rOutParams])

Version: 4.0

Restores the device settings of the executing server thread's main device to the global settings from the main device of the application thread and may also return these settings in the optional parameter.

Important Note: If this functions is executed from the application thread, it does nothing.

See also: \$startserver()

Parameter	Description
rOutParams	Optional row variable that is to receive the device parameters
returns	always returns zero

\$setcreatorandproducer()

Syntax: PDF Device.\$setcreatorandproducer(cApplicationName,cCompanyName)

Version: 1.5

Sets the creator and producer text that is embedded in the PDF document. This information is displayed by some PDF viewers.

Important Note: The ‘(’ and ‘)’ characters are reserved and may not be used. There may also be other characters that cannot be used with PDF creator and producer names and you must ensure the strings that you provide do not cause PDF readers any issues. We recommend you test your PDF files with different versions of the Adobe readers.

Parameter	Description
cApplicationName	Your software’s application/product name.
cCompanyName	Your company name.
returns	kTrue if successful.

\$setmemoryoutput()

Syntax: PDF Device.\$setmemoryoutput(cOutputVariableName[,cMimeFileName])

Version: 4.0

Sets the output to a memory variable which must be a variable within context at the time of this call and it must still be alive by the time the device is closed after printing. If cMimeFileName is specified output will be prefixed with a mime header.

Important Note: The specified variable must still exist by the time the report is printed or Omnis may crash.

See also: \$settempfilename()

Parameter	Description
cOutputVariableName	The name of the output variable. This can be a task variable of the remote task, or a instance variable of the remote form. It could also be a local variable in a method, as long as the report is printed by the same method.
cMimeTypeName	If a file name is specified, the output data is prefixed with the following mime-header: content-type: application/pdf content-length:[the byte length of the data] content-disposition: filename=[the provided file name]

\$settempfilename()

Syntax: PDF Device.\$settempfilename(iMinutes,&cOutPath,&cOutName)

Version: 4.0

Generates and sets the destination file name (kDevPdfFileName) to a unique name in the default Omnis PDF folder and returns the path and name. If iMinutes is > zero the file is stored in the Omnis temp folder and is deleted after the specified minutes have expired.

Important Note: Please note that to use the Omnis default PDF folders, you will need to add a “getpdfFolders” entry to the Omnis config.json file so that Omnis allows PDF files, which have not been produced by OmnisPDF, to be downloaded to the client using the client commands “showpdf” or “assignpdf”. Please contact Omnis software support if you have problems with this.

Further Note: The unique portion of the generated file name is made up of an encoded date as in YYMMDDHHNN plus a unique counter (within each one minute interval). This counter is protected by a mutex lock to ensure absolute uniqueness amongst the server threads.

See also: \$setmemoryoutput()

Parameter	Description
iMinutes	Specifies the number of minutes after which the file is deleted by the BDPDF server.
cOutPath	Returns the server path to the temp file name if cOutPath is empty. If a path is specified, PDFDevice will use the specified path.
cOutName	Returns the generated unique name of the output file. You may specify a full name with the extension “.pdf” in which case PDFDevice will use the given name for the output file. You may also specify a prefix (excluding the .pdf extension), in which case PDFDevice will generate the unique portion that is to follow the prefix. If cOutName is completely empty, PDFDevice will use the prefix “bdpdf” followed by a unique portion of the name.

returns	The function returns 1 if successful, 0 if it failed to generate a unique file name, or -102 if the incorrect number of parameters were passed.
---------	---

\$startserver()

Syntax: PDF Device.\$startserver()

Version: 4.0

When executed, the external will create a device instance for each Omnis server thread (see \$prefs.\$serverstacks) and copy the device settings from the application's main device instance. Once executed, device notation such as \$setparam(), \$getparam(), \$open(), \$close() will address the device instance associated with the current execution thread. Consequently, any device parameter actions are isolated from the other threads.

Important Note: This function must be called after executing the Omnis *Start server* command.

Recommendation: Prepare device parameters on the main application thread prior to calling this function so the server thread devices are initialised with the appropriate settings.

See also: \$stopserver(), \$initparams()

Parameter	Description
returns	The server stack count. A return value of zero indicates an error, i.e. the Omnis <i>Start server</i> command has not yet been executed.

\$stopserver()

Syntax: PDF Device.\$stopserver()

Version: 4.0

When executed, the device instances that were created when calling \$startserver() will be deleted.

Important Note: This function should be called prior to executing the Omnis *Stop server* command.

See also: \$startserver()

Parameter	Description
returns	always returns zero

www.brainydata.com

Report Objects

This section documents the PDF report objects and their properties.

Form Field (v5.0)

This object is used for embedding Acroform fields in PDF files. Supported Acroform field types are push-button, check-box, text, choice and signature fields. Radio button fields are currently not supported, but the choice field can be used instead.

Name	Type	Description
\$dataname	Character	<p>PDF Form Object's initial value (if blank, \$defaultvalue is used instead, otherwise data replaces default value)</p> <p>For push-buttons, the data or default value specifies the action of which there are currently three types supported</p> <p>URL~ When the value begins with the phrase "URL~" the subsequent data specifies an URL for submitting the form data using a standard html POST message. Example: URL~https://demos.brainydata.com/test.php</p> <p>Reset~ Generates a simple reset form action for the button and no additional data follows the phrase "Reset~"</p> <p>Script~ The data that follows the phrase "Script~" is Acrobat java script. For more details see the Adobe Acrobat java script reference at https://opensource.adobe.com/dc-acrobat-sdk-docs/acrobatsdk/pdfs/acrobatsdk_jsapiref.pdf</p>
\$defaultvalue	Character	<p>PDF Form Object's default value (initial and when form is reset). See \$dataname for details relating to push-buttons.</p>
\$displayname	Character	<p>The name to be displayed in the user interface. This maps to the 'TU' entry in the field dictionary in PDF. See the adobe PDF reference for more details.</p>

www.brainydata.com

\$fieldflags	Integer	Set of flags for field. Which flags are relevant depend on \$fieldtype (one of the kDevPdffff... constants). This maps to the 'Ff' entry in the field dictionary in PDF. See the adobe PDF reference for more details.
\$fieldname	Character	The internal name of form field which can be used by java script to identify and manipulate form fields during actions. This maps to the 'T' entry in the field dictionary in PDF. See the adobe PDF reference for more details.
\$fieldtype	Constant	PDF Form Object's type (Push-button, Check-box, Text, Choice, Signature). One of the kDevPdffffT... constants. Note: The Radio type is currently not supported
\$fillcolor	Omnis Color	Field additional fill adornment: Specifies the RGB fill color. Use an Omnis color constant or the rgb() function to specify the color value.
\$fntencoding	Constant	Specifies the font's encoding. One of the kDevPdffffE... constants.
\$fntname	Constant	Specifies Standard PDF font name. Use one of the kDevPdffffSF... constants.
\$fntsize	Integer	The font size in points.
\$strokecolor	Integer	Field additional border adornment: Specifies the RGB stroke color.
\$strokeradius	Integer	Field additional border adornment: Specifies the radius for rendering rounded corners of the border.
\$strokestyle	Constant	Field additional border adornment: Specifies the stroke style, one of the kDevPdffffB... constants.
\$strokewidth	Integer	Field additional border adornment: Specifies the stroke width in pixels.
\$submitname	Character	the name to be used when the data is submitted. This maps to the 'TM' entry in the field dictionary in PDF. See the adobe PDF reference for more details.
\$txtcolor	Omnis Color	The field's text color. Use an Omnis color constant or the rgb() function to specify the color value.

PDF Parameters

This object is used to change device parameters for the current report instance. To change device parameters only once at the start of the print job, place the object in the report header section. To change parameters for individual pages, place the object in the page header section.

Name	Type	Description
\$dataname	Row	Specifies the Omnis field (Row Variable) that contains the device parameters. Use <code>\$getparam(kDevPdfAll)</code> to fetch the current device parameters and then change the individual columns as required.

Form Field Constants (v5.0)

This section documents the constants for use with the form field properties.

kDevPdfFFB...

Constants that specify the **Form Field Border** styles for the `$strokestyle` property. These constants are exclusive, so you may only assign a single constant.

Name	Dec. Value	Description
kDevPdfFFBSolid	0	Solid line
kDevPdfFFBDash	1	Dashed line
kDevPdfFFBDot	2	Dotted line
kDevPdfFFBDashDot	3	Dash-Dot line
kDevPdfFFBDashDotDot	4	Dash-Dot-Dot line

kDevPdfFFF...

Constants that specify the **Form Field Flags** for use with `$fieldflags`. These constants are bit based and multiple constants can be specified using the `+` operator.

Example:

```
.$fieldflags.$assign(kDevPdfFFFReadOnly+kDevPdfFFFRequired)
```

Name	Hex Value	Description
kDevPdfFFFReadOnly	00000001	Form button field (Push-button, Check-box or Radio-button, see kDevPdfFFF... constants) Note: Radio-button is currently not supported.

kDevPdfFFFRequired	00000002	If set, the field must have a value at the time it is exported by a submit-form action.
kDevPdfFFFNoExport	00000004	If set, the field must not be exported by a submit-form action.
kDevPdfFFFMultiline	00001000	If set, the field can contain multiple lines of text; if clear, the field's text is restricted to a single line.(text fields only)
kDevPdfFFFPassword	00002000	If set, the field is intended for entering a secure password that should not be echoed visibly to the screen.(text fields only)
kDevPdfFFFPushbutton	00010000	If set, the field is a pushbutton that does not retain a permanent value. (button fields only)
kDevPdfFFFCombo	00020000	If set, the field is a combo box; if clear, the field is a list box. (list fields only)
kDevPdfFFFComboEdit	00040000	If set, the combo box includes an editable text box as well as a drop- down list; if clear, it includes only a drop-down list. (list fields only)
kDevPdfFFFSort	00080000	If set, the field's option items should be sorted alphabetically. (list fields only)
kDevPdfFFFMultiSelect	00200000	If set, more than one of the field's option items may be selected simultaneously. (list fields only)
kDevPdfFFFFileSelect	00100000	If set, the text entered in the field represents the pathname of a file whose contents are to be submitted as the value of the field. (text fields only)
kDevPdfFFFDoNotSpellCheck	00400000	If set, text entered in the field is not spell-checked. (text/combo fields only)
kDevPdfFFFDoNotScroll	00800000	If set, the field does not scroll (horizontally for single-line fields, vertically for multiple-line fields) to accommodate more text than fits within its annotation rectangle. (text fields only)
kDevPdfFFFRichText	02000000	If set, the value of this field should be represented as a rich text string, i.e. fully-formed XML. (text fields only)
kDevPdfFFFCommitOnSelChange	04000000	If set, the new value is committed as soon as a selection is made with the pointing device.(list fields only)

kDevPdfFFFE...

Constants that specify the **Form Field Font Encoding** for the \$fntencoding property. These constants are exclusive, so you may only assign a single constant.

Name	Dec. Value	Description
kDevPdfFFFEplatform	0	Assign according to platform, either kDevPdfFFFEmacRoman or kDevPdfFFFEwinAnsi.
kDevPdfFFFEMacRoman	1	Mac standard encoding
kDevPdfFFFEWinAnsi	2	Windows standard encoding
kDevPdfFFFESymbol	3	Symbol font encoding
kDevPdfFFFEZapfDingbats	4	ZapfDingbats font encoding

kDevPdfFFSF...

Constants that specify the **Form Field Standard Font** for use with \$fntname. These constants are exclusive, so you may only assign a single constant.

Name	Dec. Value	Character Value
kDevPdfFFSFHelvetica	0	Helvetica
kDevPdfFFSFHelveticaBold	1	Helvetica-Bold
kDevPdfFFSFHelveticaOblique	2	Helvetica-Oblique
kDevPdfFFSFHelveticaBoldOblique	3	Helvetica-BoldOblique
kDevPdfFFSFTimesRoman	4	Times-Roman
kDevPdfFFSFTimesBold	5	Times-Bold
kDevPdfFFSFTimesItalic	6	Times-Italic
kDevPdfFFSFTimesBoldItalic	7	Times-BoldItalic
kDevPdfFFSFCourier	8	Courier
kDevPdfFFSFCourierBold	9	Courier-Bold
kDevPdfFFSFCourierOblique	10	Courier-Oblique
kDevPdfFFSFCourierBoldOblique	11	Courier-BoldOblique
kDevPdfFFSFSymbol	12	Symbol
kDevPdfFFSFZapfDingbats	13	ZapfDingbats

kDevPdfFFT...

Constants that specify the **Form Field Type** for use with \$fieldtype. These constants are exclusive, so you may only assign a single constant.

Name	Dec. Value	Description
kDevPdfFFTButton	0	Form button field (Push-button, Check-box or Radio-button, see kDevPdfFFF... constants) Note: Radio-button is currently not supported.
kDevPdfFFTText	1	Form text field (single-line/multi-line/password, etc, see kDevPdfFFF... constants)
kDevPdfFFTChoice	2	Form choice list (list-box or combo, see kDevPdfFFF... constants)
kDevPdfFTSignature	3	Form signature field

Examples Reference

This reference serves as a guide to the most important classes in the example libraries OWriteDocumentManager.lbs and PDFDeviceAndJSClient.lbs. This chapter only documents methods and functionality of special interest. You will find additional comments in the methods of the various classes.

OWrite Document manager library

This library encompasses the OWrite, OSpell2 and PDFDevice examples. Within this section we will only concern our self with the classes that exclusively demonstrate the use of PDFDevice. Although the OWrite examples use PDFDevice in many interesting ways, it is matter for the OWrite documentation to elucidate on this further.



wPDFOptions

Implements the advanced PDF options interface for setting the PDF options. Th library installs this window into the device parameters and it is opened when you click the "Advanced..." button on the "Parameters" pane of the Studio destination dialog when viewing the PDF Device properties.

Other required classes: wPDFPickStyle



wPDFPickStyle

Implements an interface for selecting styles from a series of check boxes. This window is opened like a pop up selection dialog when you click the style combo button in the document outlines pane of the wPDFOptions window.

Other required classes: wPDFOptions



wSetCreatorProducer

This example window demonstrates the \$setcreatorandproducer() feature of PDFDevice.

Other required classes: mPDFDevice



rPDFDeviceSuper

This report super-class implements the PDF parameter report object and concerns itself with loading the main device parameters into a instance row variable for manipulation of the device parameters associated with the current print job.



rPDFDeviceDoc & rPDFDeviceHRP

These classes demonstrate the printing of basic reports to PDFDevice.

Other required classes: wPDFOptions, mPDFDevice



mPDFDevice

This menu is installed as a sub-menu in the libraries main example menu “mExample”.

PDFDeviceAndJSClient library

This library implements three different client implementations and their different use of PDFDevice. The classes for each example are organised into their own library folders and any classes common to all examples are stored in a folder of common classes.



e1_RemoteForm

This class demonstrates how to print to memory and return the PDF data to the client for embedding in a HTML control or for display in a browser window. It also utilises the remote PDF options form so that clients can directly change device parameters.

Other required classes: e1_RemoteTask, rfPDFOptions



e1_RemoteTask

This class is required by e1_Remoteform for returning PDF data on an evPost event, when the form displays the PDF in a browser window. It inherits the class rtPDFOptions.

Other required classes: rtPDFOptions



e2_RemoteForm

This class demonstrates how to use the client commands “showpdf” and “assignpdf” in conjunction with using temp files on the Omnis server. Just as e1_RemoteForm does, this class utilises the remote PDF options form so that clients can directly change device parameters.

Other required classes: e1_RemoteTask, rfPDFOptions



e2_RemoteTask

This class is required by e2_Remoteform for handling the PDF device parameters between requests. It inherits the class rtPDFOptions.

Other required classes: rtPDFOptions



e3_UltraThinCode

This code class implements a single method for producing an HTML file and opening it in the default browser. The HTML contains a form that sends an ultra-thin request to the remote task e3_RemoteTask that returns the PDF for display in a browser window.

Other required classes: e3_RemoteTask



e3_RemoteTask

This remote task implements the code to handle an ultra-thin request and prints and return the PDF data with a mime header, using a binary local variable as the output destination.



rfPDFOptions

This form implements the client interface for setting device options. It looks similar to the fat-client window class wPDFOptions and works in very similar ways. It directly manipulates the instance row variable that contain the complete set of device parameters. It requires a number of additional classes for the implementation of its interface.

Other required classes: rtPDFOptions, rmFontName, rmFontSize, rmFontStyles



rtPDFOptions

This remote class is the super-class for the example remote tasks and is the associated (design) remote task for the rfPDFOptions remote form. It mainly provides the storage and implements the maintenance of the instance row variable ivDeviceParams for maintaining device settings between different server requests for its sub-classes and associated remote form instances.



eX_Report, eX_ReportPDFBlurb, and eX_ReportPDFKeyActions

These reports are the example reports used in all of the above example forms. The class eX_Report is populated with data (Title, First Name and Last Name) provided by the client for printing a titled dummy letter. The other reports are used to add additional PDF blurb while being useful in demonstrating how to print multiple reports to the same PDF file.



wPDFOptions and wPDFPickStyle

These classes are the desktop windows which are mainly used by us (Brainy Data Engineers) to configure the initial device parameters for the examples. The class wPDFOptions is a modified version of the same windows in the OWrite Document Manager example. This class is modified in as much as the it saves a copy of the device parameters to the file *PDFDeviceAndJSClient.prefs* by calling the method *Startup_Task.\$saveDeviceParams()*.

Other required classes: Startup_Task



Startup_Task

The startup task implements the preparation and startup of the Omnis Server and BDPDF server for the purpose of these examples. It also checks the version number of the installed PDF Device and initialises the global PDF device settings from the file *PDFDeviceAndJSClient.prefs*.

Other required classes: wStartup

 **wStartup**

This window is opened by Startup_Task and displays the main navigational interface for the examples.

 **wConfigError**

This error dialog is opened by wStartup when attempting to run example 3 and the Omnis configuration file is not configured to take advantage of the Omnis PDF temp folders for producing and displaying PDF files using the client commands ‘assignpdf’ and ‘showpdf’.

PDFDevicePoDoFo Library (v5.0)

This library demonstrates the new to version 5 Acroform and file handling features.

 **objFileOps**

Implements generic file handling functions for for the main examples. Used by all the new file handling examples.

 **repAcroForm**

Implements the report for generating the Acroform PDF file. Used by the \$filereaddata() example to generate the PDF form that can be filled in using Adobe Reader and then read back using the static method \$filereaddata().

 **winMain**

This window implements the interface for all the version 5 examples.