

10 Variable Data and Blocks

PDFlib supports a template-driven PDF workflow for variable data processing. Using the concept of blocks, imported pages can be populated with variable amounts of single- or multi-line text, images, or PDF graphics which can be pulled from an external source.

This can be used to easily implement applications which require customized PDF documents, for example:

- ▶ mail merge
- ▶ flexible direct mailings
- ▶ transactional and statement processing
- ▶ business card personalization

Note Block processing requires the PDFlib Personalization Server (PPS). Although PPS is contained in all commercial PDFlib packages, you must purchase a license key for PPS; a PDFlib or PDFlib+PDI license key is not sufficient. The PDFlib Block plugin for Adobe Acrobat is required for creating blocks in PDF templates interactively.

10.1 Installing the PDFlib Block Plugin

The Block plugin and its sibling, the PDF form field conversion plugin, work with Acrobat 5, Acrobat 6/7 Standard and Acrobat 6/7 Professional on Windows and Mac, and Acrobat 8 Standard and Professional on Windows. The plugins don't work with Acrobat Elements or any version of Acrobat Reader/Adobe Reader.

Note The plugins contain multiple language versions of the user interface, and will automatically use the same interface language as the Acrobat application if possible. If the plugin does not support the native Acrobat language the English user interface will be used instead.

Installing the PDFlib Block plugins for Acrobat 5/6/7/8 on Windows. To install the PDFlib Block plugin and the PDF form field conversion plugin in Acrobat 5, 6, 7, or 8 the plugin files must be copied to a subdirectory of the Acrobat plugin folder. This is done automatically by the plugin installer, but can also be done manually. The plugin files are called *Block.api* and *AcroFormConversion.api*. A typical location of the plugin folder looks as follows:

```
C:\Program Files\Adobe\Acrobat 7.0\Acrobat\plug_ins\PDFlib Block Plugin
```

Installing the PDFlib Block plugins for Acrobat 6/7 on the Mac. With Acrobat 6/7 the plugin folder will not be visible in the Finder. Instead of dragging the plugin files to the plugin folder use the following steps (make sure that Acrobat is not running):

- ▶ Extract the plugin files to a folder by double-clicking the disk image.
- ▶ Locate the Acrobat application icon in the Finder. It is usually located in a folder which has a name similar to the following:

```
/Applications/Adobe Acrobat 7.0 Professional
```

- ▶ Single-click on the Acrobat application icon and select *File, Get Info*.
- ▶ In the window that pops up click the triangle next to *Plug-ins*.
- ▶ Click *Add...* and select the *PDFlib Block Plugin* folder (for Acrobat 7) or the *PDFlib Block Plugin Acro 5-6* folder (for Acrobat 6) from the folder which has been created in the

first step. Note that after installation this folder will not immediately show up in the list of plugins, but only when you open the info window next time.

Installing the PDFlib Block plugins for Acrobat 5 on the Mac. To install the plugins for Acrobat 5, start by double-clicking the disk image. Drag the folder *PDFlib Block Plugin Acro 5-6* to the Acrobat 5 plugin folder. A typical plugin folder name is as follows:

`/Applications/Adobe Acrobat 5.0/Plug-Ins`

Trouble-shooting. If the PDFlib Block plugin doesn't seem to work check the following:

- ▶ Make sure that in Edit, Preferences, [General...], Startup (Acrobat 6/7) or Options (Acrobat 5) the box *Use only certified plug-ins* is unchecked. The plugins will not be loaded if Acrobat is running in Certified Mode.
- ▶ Some PDF forms created with Adobe Designer may prevent the Block plugin as well as other Acrobat plugins from working properly since they interfere with PDF's internal security model. For this reason we suggest to avoid Designer's static PDF forms, and only use dynamic PDF forms as input for the Block plugin.

10.2 Overview of the PDFlib Block Concept

10.2.1 Complete Separation of Document Design and Program Code

PDFlib data blocks make it easy to place variable text, images, or graphics on imported pages. In contrast to simple PDF pages, pages containing data blocks intrinsically carry information about the required processing which will be performed later on the server side. The PDFlib block concept completely separates the following tasks:

- ▶ A designer creates the page layout, and specifies the location of variable text and image elements along with relevant properties such as font size, color, or image scaling. After creating the layout as a PDF document, the designer uses the PDFlib Block plugin for Acrobat to specify variable data blocks and their associated properties.
- ▶ A programmer writes code to connect the information contained in PDFlib blocks on imported PDF pages with dynamic information, e.g., database fields. The programmer doesn't need to know any details about a block (whether it contains a name or a ZIP code, the exact location on the page, its formatting, etc.) and is therefore independent from any layout changes. PDFlib will take care of all block-related details based on the block properties found in the file.

In other words, the code written by the programmer is »data-blind« – it is generic and does not depend on the particulars of any block. For example, the designer may decide to use the first name of the addressee in a mailing instead of the last name. The generic block handling code doesn't need to be changed, and will generate correct output once the designer changed the block properties with the Acrobat plugin to use the first name instead of the last name.

Example: adding variable text to a template. Adding dynamic text to a PDF template is a very common task. The following code fragment will open a page in an input PDF document (the template), place it on the output page, and fill some variable text into a text block called *firstname*:

```
doc = p.open_pdi_document(filename, "");
if (doc == -1)
    throw new Exception("Error: " + p.get_errmsg());

page = p.open_pdi_page(doc, pageno, "");
if (page == -1)
    throw new Exception("Error: " + p.get_errmsg());

p.begin_page_ext(width, height, "");
p.fit_pdi_page(page, 0.0, 0.0, "");
p.fill_textblock(page, "firstname", "Serge", "encoding=winansi");
p.close_pdi_page(page);
p.end_page_ext("");
p.close_pdi_document(doc);
```

Controlling the display order of imported page and blocks. The imported page must have been placed on the output page before using any of the block filling functions. This means that the original page will usually be placed below the block contents. However, in some situations it may be desirable to place the original page on top of the filled blocks. This can be achieved with the *blind* option of *fit_pdi_page()*:

```

/* Place the page in blind mode to prepare the blocks, without the page being visible */
p.fit_pdi_page(page, 0.0, 0.0, "blind");
p.fill_textblock(page, "firstname", "Serge", "encoding=winansi");
/* ... fill more blocks ... */

/* Place the page again, this time visible */
p.fit_pdi_page(page, 0.0, 0.0, "");

```

10.2.2 Block Properties

The behavior of blocks can be controlled with block properties. The properties are assigned to a block with the PDFlib Block plugin for Acrobat.

Standard block properties. PDFlib blocks are defined as rectangles on the page which are assigned a name, a type, and an open set of properties which will later be processed on the server side. The name is an arbitrary string which identifies the block, such as *firstname*, *lastname*, or *zipcode*. PDFlib supports the following kinds of blocks:

- ▶ Type *Text* means that the block will hold one or more lines of textual data. Multi-line text will be formatted with the Textflow feature. Textflow blocks can be linked so that one block holds the overflow text of the previous block (see Section 10.2.3, »Linking multiple Textflow Blocks«, page 217).
- ▶ Type *Image* means that the block will hold a raster image. This is similar to importing a TIFF or JPEG file in a DTP application.
- ▶ Type *PDF* means that the block will hold arbitrary PDF graphics imported from a page in another PDF document. This is similar to importing an EPS graphic in a DTP application.

A block may carry a number of standard properties depending on its type. For example, a text block may specify the font and size of the text, an image or PDF block may specify the scaling factor or rotation. For each type of block the PDFlib API offers a dedicated function for processing the block. These functions search an imported PDF page for a block by its name, analyze its properties, and place some client-supplied data (text, raster image, or PDF page) on the new page according to the corresponding block properties.

Custom block properties. Standard block properties make it possible to quickly implement variable data processing applications, but these are limited to the set of properties which are internally known to PDFlib and can automatically be processed. In order to provide more flexibility, the designer may also assign custom properties to a block. These can be used to extend the block concept in order to match the requirements of the most demanding variable data processing applications.

There are no rules for custom properties since PDFlib will not process custom properties in any way, except making them available to the client. The client code can examine the custom properties and act in whatever way it deems appropriate. Based on some custom property of a block the code may make layout-related or data-gathering decisions. For example, a custom property for a scientific application could specify the number of digits for numerical output, or a database field name may be defined as a custom block property for retrieving the data corresponding to this block.

Overriding block properties. In certain situations the programmer would like to use only some of the properties provided in a block definition, but override other properties with custom values. This can be useful in various situations:

- ▶ The scaling factor for an image or PDF page will be calculated instead of taken from the block definition.
- ▶ Change the block coordinates programmatically, for example when generating an invoice with a variable number of data items.
- ▶ Individual spot color names could be supplied in order to match the requirements of individual customers in a print shop application.

Property overrides can be achieved by supplying property names and the corresponding values in the option list of all *fill_*block()* functions as follows:

```
p.fill_textblock(page, "firstname", "Serge", "fontsize=12");
```

This will override the block's internal *fontsize* property with the supplied value 12. Almost all names of general properties can be used as options, as well as those specific to a particular block type. For example, the *underline* option is only allowed for *fill_textblock()*, while the *scale* option is allowed for both *fill_imageblock()* and *fill_pdfblock()* since *scale* is a valid property for both image and PDF blocks.

Property overrides apply only to the respective function calls; they will not be stored in the block definition.

Coordinate systems. The coordinates describing a block reference the PDF default coordinate system. When the page containing the block is placed on the output page, several positioning and scaling options may be supplied to *fit_pdi_page()*. These parameters are taken into account when the block is being processed. This makes it possible to place a template page on the output page multiply, every time filling its blocks with data. For example, a business card template may be placed four times on an imposition sheet. The block functions will take care of the coordinate system transformations, and correctly place the text for all blocks in all invocations of the page. The only requirement is that the client must place the page and then process all blocks on the placed page. Then the page can be placed again at a different location on the output page, followed by more block processing operations referring to the new position, and so on.

Note The Block plugin will display the block coordinates differently from what is stored in the PDF file. The plugin uses Acrobat's convention which has the coordinate origin in the upper left corner of the page, while the internal coordinates (those stored in the block) use PDF's convention of having the origin at the lower left corner of the page.

10.2.3 Linking multiple Textflow Blocks

Textflow blocks can be linked so that one block holds the overflow text from a previous block. For example, if you have long variable text which may need to be continued on another page you can link two blocks and fill the text which is still available after filling the first block into the second block.

PPS internally creates a Textflow from the text provided to *fill_textblock()* and the block properties. For unlinked blocks this Textflow will be placed in the block and the corresponding Textflow handle will be deleted at the end of the call; overflow text will be lost.

With linked Textflow blocks the overflow text which remains after filling the first block can be filled into the next block. The remainder of the Textflow will be used as block contents instead of creating a new Textflow. Linking Textflow blocks works as follows:

- ▶ In the first call to `fill_textblock()` within a chain of linked blocks a value of -1 must be supplied for the `textflowhandle` option. The Textflow handle created internally will be returned by `fill_textblock()`, and must be stored by the user.
- ▶ In the next call the Textflow handle returned in the previous step can be supplied to the `textflowhandle` option (the text supplied in the `text` parameter will be ignored in this case, and should be empty). The block will be filled with the remainder of the Textflow.
- ▶ This process can be repeated with more Textflow blocks.
- ▶ The returned Textflow handle can be supplied to `info_textflow()` in order to determine the results of block filling, e.g. the end condition or the end position of the text.

Note that the `fitmethod` property should be set to `clip` (this is the default anyway if `textflowhandle` is supplied). The basic code skeleton for linking Textflow blocks looks as follows:

```
p.fit_pdi_page(page, 0.0, 0.0, "");
tf = -1;

for (i = 0; i < blockcount; i++)
{
    String optlist = "encoding=winansi textflowhandle=" + tf;
    tf = p.fill_textblock(page, blocknames[i], text, optlist);
    text = null;

    if (tf == -1)
        break;

    /* check result of most recent call to fit_textflow() */
    reason = (int) p.info_textflow(tf, "returnreason");
    result = p.get_parameter("string", (float) reason);

    /* end loop if all text was placed */
    if (result.equals("_stop"))
    {
        p.delete_textflow(tf);
        break;
    }
}
```

10.2.4 Why not use PDF Form Fields?

Experienced Acrobat users may ask why we implemented a new block concept for PDFlib, instead of relying on the established form field scheme available in PDF. The primary distinction is that PDF form fields are optimized for interactive filling, while PDFlib blocks are targeted at automated filling. Applications which need both interactive and automated filling can easily achieve this by using a feature which automatically converts form fields to blocks (see Section 10.3.4, »Converting PDF Form Fields to PDFlib Blocks«, page 225).

Although there are many parallels between both concepts, PDFlib blocks offer several advantages over PDF form fields as detailed in Table 10.1.


Table 10.1 Comparison of PDF form fields and PDFlib blocks

feature	PDF form fields	PDFlib blocks
<i>design objective</i>	<i>for interactive use</i>	<i>for automated filling</i>
<i>typographic features (beyond choice of font and font size)</i>	–	<i> Kerning, word and character spacing, underline/overline/strikeout</i>
<i>font control</i>	<i>font embedding</i>	<i>font embedding and subsetting, encoding</i>
<i>text formatting controls</i>	<i>left-, center-, right-aligned</i>	<i>left-, center-, right-aligned, justified; various formatting algorithms and controls; inline options can be used to control the appearance of text</i>
<i>change font or other text attributes within text</i>	–	<i>yes</i>
<i>merged result is integral part of PDF page description</i>	–	<i>yes</i>
<i>users can edit merged field contents</i>	<i>yes</i>	<i>no</i>
<i>extensible set of properties</i>	–	<i>yes (custom block properties)</i>
<i>use image files for filling</i>	–	<i>BMP, CCITT, GIF, PNG, JPEG, JPEG 2000, TIFF</i>
<i>color support</i>	<i>RGB</i>	<i>grayscale, RGB, CMYK, Lab, spot color (HKS and Pantone spot colors integrated in the Block plugin)</i>
<i>PDF/X and PDF/A compatible</i>	<i>PDF/X: no; PDF/A: restricted</i>	<i>yes (both template with blocks and merged results)</i>
<i>graphics and text properties can be overridden upon filling</i>	–	<i>yes</i>
<i>Text blocks can be linked</i>	–	<i>yes</i>

10.3 Creating PDFlib Blocks

10.3.1 Creating Blocks interactively with the PDFlib Block Plugin

Activating the PDFlib Block tool. The PDFlib Block plugin for creating PDFlib blocks is similar to the form tool in Acrobat. All blocks on the page will be visible when the block tool is active. When another Acrobat tool is selected the blocks will be hidden, but they are still present. You can activate the block tool in several ways:

- ▶ by clicking the block icon  in Acrobat's *Advanced Editing* toolbar (in Acrobat 5: *Editing* toolbar);
- ▶ via the menu item *PDFlib Blocks, PDFlib Block Tool*;
- ▶ by using the keyboard shortcut *P*; make sure to enable *Edit, Preferences, [General...], General, Use single key accelerators to access tools*, which is disabled by default (not required in Acrobat 5)

Creating and modifying blocks. Once you activated the block tool you can simply drag the cross-hair pointer to create a block at the desired position on the page and the desired size. Blocks will always be rectangular with edges parallel to the page edges. When you create a new block the block properties dialog appears where you can edit various properties of the block (see Section 10.3.2, »Editing Block Properties«, page 222). The block tool will automatically create a block name which can be changed in the properties dialog. Block names must be unique within a page. You can change the block type in the top area to one of Text, Image, or PDF. The *General* and *Custom* tabs will always be available, while only one of the *Text*, *Image*, and *PDF* tabs will be active at a time depending on the chosen block type. The *Textflow* tab will only be present for blocks of type text if the *textflow* property is *true*. Another tab labelled *Tabs* (tabulator positions) will only be available if the *hortabmethod* property in the *Textflow* tab has been set to *ruler*.

Note After you added blocks or made changes to existing blocks in a PDF, use Acrobat's »Save as...« Command (as opposed to »Save«) to achieve smaller file sizes.

Note When using the Acrobat plugin *Enfocus PitStop* to edit documents which contain PDFlib blocks you may see the message »This document contains *PieceInfo* from PDFlib. Press OK to continue editing or Cancel to abort.« This message can be ignored; it is safe to click OK in this situation.

Selecting blocks. Several block operations, such as copying or moving, work with selected blocks. You can select one or more blocks with the block tool as follows:

- ▶ To select a single block simply click on it with the mouse.
- ▶ Hold down the Shift key while clicking on another block to extend the selection.
- ▶ Press Ctrl-A (on Windows) or Cmd-A (on the Mac) or *Edit, Select All* to select all blocks on a page.

The context menu. When one or more blocks are selected you can open the context menu to quickly access block-related functions (which are also available in the PDFlib Blocks menu). To open the context menu, click on the selected block(s) with the right mouse button on Windows, or Ctrl-click the block(s) on the Mac.

For example, to delete a block, select it with the block tool and press the *Delete* key, or use *Edit, Delete* in the context menu.

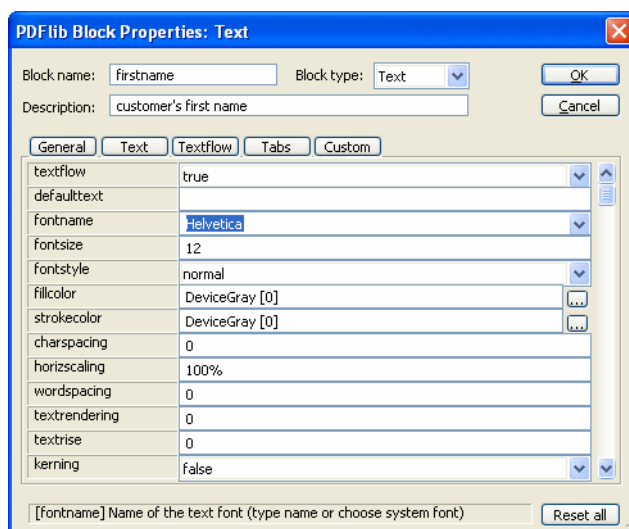


Fig. 10.1
 Editing block properties: the Textflow
 panel is only visible if textflow=true;
 the Tabs panel is only visible if
 hortabmethod=ruler

Fine-tuning block size and position. Using the block tool you can move one or more selected blocks to a different position. Hold down the Shift key while dragging a block to restrain the positioning to horizontal and vertical movements. This may be useful for exactly aligning blocks. When the pointer is located near a block corner, the pointer will change to an arrow and you can resize the block. To adjust the position or size of multiple blocks, select two or more blocks and use the *Align*, *Center*, *Distribute*, or *Size* commands from the *PDFlib Blocks* menu or the context menu. The position of one or more blocks can also be changed by using the arrow keys.

Alternatively, you can enter numerical block coordinates in the properties dialog. The origin of the coordinate system is in the upper left corner of the page. The coordinates will be displayed in the unit which is currently selected in Acrobat. To change the display units proceed as follows:

- ▶ In Acrobat 6/7 go to *Edit, Preferences, [General...], Units & Guides* [or *Unit, Page Units* in Acrobat 7 Standard] and choose one of Points, Inches, Millimeters, Picas, Centimeters. You can also go to *View, Navigation Tabs, Info* and select a unit from the *Options* menu.
- ▶ In Acrobat 5 go to *Edit, Preferences, General..., Display, Page Units* and choose one of Points, Inches, Millimeters. You can also go to *Window, Info* and select a unit from the *Info* menu.

Note that the chosen unit will only affect the *Rect* property, but not any other numerical properties.

Creating blocks by selecting an image or graphic. As an alternative to manually dragging block rectangles you can use existing page contents to define the block size. First, make sure that the menu item *PDFlib Blocks, Click Object to define Block* is enabled. Now you can use the block tool to click on an image on the page in order to create a block with the size of the image. You can also click on other graphical objects, and the block tool will try to select the surrounding graphic (e.g., a logo). The *Click Object* feature is intended as an aid for defining blocks. If you want to reposition or resize the block you can

do so afterwards without any restriction. The block will not be locked to the image or graphics object which was used as a positioning and sizing aid.

The *Click Object* feature will try to recognize which vector graphics and images form a logical element on the page. When some page content is clicked, its bounding box (the surrounding rectangle) will be selected unless the object is white or very large. In the next step other objects which are partially contained in the detected rectangle will be added to the selected area, and so on. The final area will be used as the basis for the generated block rectangle. The end result is that the *Click Object* feature will try to select complete graphics, and not only individual lines.

The *Click Object* feature isn't perfect: it will not always select what you want, depending on the nature of the page content. Keep in mind that this feature is only intended as a positioning aid for quickly creating block rectangles.

Automatically detecting font properties. The PDFlib Block plugin can analyze the underlying font which is present at the location where a block is positioned, and can automatically fill in the corresponding properties of the block:

fontname, fontsize, fillcolor, charspacing, horizscaling, wordspacing, textrendering, textrise

Since automatic detection of font properties can result in undesired behavior when the background shall be ignored, it can be activated or deactivated using *PDFlib Blocks, Detect underlying font and color*. By default this feature is turned off.

Locking blocks. Blocks can be locked to protect them against accidentally moving, resizing, or deleting them. With the block tool active, select the block and choose *Lock* from its context menu. While a block is locked you cannot move, resize, or delete it, nor display its properties dialog.

Using Blocks with PDF/X. Unlike PDF form fields, PDFlib blocks are PDF/X compatible. Both the input document containing blocks as well as the generated output PDF can be made PDF/X conforming. However, in preparing block files for a PDF/X workflow you may run into the following problem:

- ▶ PDF/X-1:2001, PDF/X-1a:2001, and PDF/X-3:2002 are based on Acrobat 4/PDF 1.3, and do not support Acrobat 5 files;
- ▶ The PDFlib Block plugin requires Acrobat 5 or above.

You can solve this problem by using Acrobat to convert the files to the required PDF version. See »Changing the PDF version of a document«, page 186, for details.

10.3.2 Editing Block Properties

When you create a new block, double-click an existing one, or choose *Properties* from a block's context menu, the properties dialog will appear where you can edit all settings related to the selected block (see Figure 10.1). As detailed in Section 10.4, »Standard Properties for Automated Processing«, page 228, there are several types of properties:

- ▶ Name, type, description, and the properties in the *General* tab apply to all blocks.
- ▶ Properties in the *Text*, *Image*, and *PDF* tabs apply only to the respective block type. Only the tab corresponding to the block's type will be active, while the other tabs are inactive.

- ▶ If a block of type *Text* has the *textflow* property set to *true*, another tab called *Textflow* will appear with Textflow-related settings.
- ▶ If a block of type *Text* has the *textflow* property set to *true*, and the *hortabmethod* property in the *Textflow* tab is set to *ruler*, still another panel called *Tabs* will appear where you can edit tabulator settings.
- ▶ Properties in the *Custom* tab can be defined by the user, and apply to any block type.

To change a property's value enter the desired number or string in the property's input area (e.g. *linewidth*), choose a value from a drop-down list (e.g. *fontname*, *fitmethod*), or select a color value or file name by clicking the »...« button at the right-hand side of the dialog (e.g. *backgroundcolor*). For the *fontname* property you can either choose from the list of fonts installed on the system, or type a custom font name. Regardless of the method for entering a font name, the font must be available on the system where the blocks will be filled with the PDFlib Personalization Server.

When you are done editing properties, click OK to close the properties dialog. The properties just defined will be stored in the PDF file as part of the block definition.

Stacked blocks. Overlapping blocks can be difficult to select since clicking an area with the mouse will always select the topmost block. In such a situation the *Choose Block* entry in the context menu can be used to select one of the blocks by name. As soon as a block has been selected the next action (e.g. double-click) within its area will not affect other blocks, but only the selected one. This way block properties can easily be edited even for blocks which are partially or completely covered by other blocks.

Using and restoring default properties. In order to save some amount of typing and clicking, the block tool will remember the property values which have been entered into the previous block's properties dialog. These values will be reused when you create a new block. Of course you can override these values with different ones at any time.

Pressing the *Reset All* button in the properties dialog will reset most block properties to their respective defaults. However, the following items will remain unmodified:

- ▶ the *Name*, *Type*, *Rect*, and *Description* properties
- ▶ all custom properties.

Shared properties. By holding the Shift key and using the block tool to select multiple blocks you can select an arbitrary number of blocks on a page. Double-clicking one of the selected blocks or pressing the *Enter* key will display the properties dialog which now applies to all selected blocks. However, since not all properties can be shared among multiple blocks, only a subset of all properties will be available for editing. Section 10.4, »Standard Properties for Automated Processing«, page 228, details which properties can be shared among multiple blocks. Custom properties cannot be shared.

10.3.3 Copying Blocks between Pages and Documents

The Block plugin offers several methods for moving and copying blocks within the current page, the current document, or other documents:

- ▶ move or copy blocks by dragging them with the mouse, or pasting blocks to another page or open document
- ▶ duplicate blocks on one or more pages of the same document
- ▶ export blocks to a new file (with empty pages) or to an existing document (apply the blocks to existing pages)

- ▶ import blocks from other documents

In order to update the page contents while maintaining block definitions you can replace the underlying page(s) while keeping the blocks. Use *Document, Replace Pages...* (Acrobat 5 and 7) or *Document, Pages, Replace* (Acrobat 6).

Moving and copying blocks. You can relocate blocks or create copies of blocks by selecting one or more blocks and dragging them to a new location while pressing the Ctrl key (on Windows) or Alt key (on the Mac). The mouse cursor will change while the key is pressed. A copied block will have the same properties as the original block, with the exception of its name and position which will automatically be changed.

You can also use copy/paste to copy blocks to another location on the same page, to another page in the same document, or to another document which is currently open in Acrobat:

- ▶ Activate the block tool and select the blocks you want to copy.
- ▶ Use Ctrl-C (on Windows) or Cmd-C (on the Mac) or *Edit, Copy* to copy the selected blocks to the clipboard.
- ▶ Use Ctrl-V (on Windows) or Cmd-V (on the Mac) or *Edit, Paste* to paste the blocks which are currently in the clipboard.

Duplicating blocks on other pages. You can create duplicates of one or more blocks on an arbitrary number of pages in the current document simultaneously:

- ▶ Activate the block tool and select the blocks you want to duplicate.
- ▶ Choose *Import and Export, Duplicate...* from the *PDFlib Blocks* menu or the context menu.
- ▶ Choose which blocks to duplicate (selected blocks or all on the page) and the range of target pages where you want duplicates of the blocks.

Exporting and importing blocks. Using the export/import feature for blocks it is possible to share the block definitions on a single page or all blocks in a document among multiple PDF files. This is useful for updating the page contents while maintaining existing block definitions. To export block definitions to a separate file proceed as follows:

- ▶ Activate the block tool and Select the blocks you want to export.
- ▶ Choose *Import and Export, Export...* from the *PDFlib Blocks* menu or the context menu. Enter the page range and a file name for the file containing the block definitions.

You can import block definitions via *PDFlib Blocks, Import and Export, Import...*. Upon importing blocks you can choose whether to apply the imported blocks to all pages in the document, or only to a page range. If more than one page is selected the block definitions will be copied unmodified to the pages. If there are more pages in the target range than in the imported block definition file you can use the *Repeat Template* checkbox. If it is enabled the sequence of blocks in the imported file will be repeated in the current document until the end of the document is reached.

Copying blocks to another document upon export. When exporting blocks you can immediately apply them to the pages in another document, thereby propagating the blocks from one document to another. In order to do so choose an existing document to export the blocks to. If you activate the checkbox *Delete existing blocks* all blocks which may be present in the target document will be deleted before copying the new blocks into the document.

10.3.4 Converting PDF Form Fields to PDFlib Blocks

As an alternative to creating PDFlib blocks manually you can automatically convert PDF form fields to blocks. This is especially convenient if you have complicated PDF forms which you want to fill automatically with the PDFlib Personalization Server, or need to convert a large number of existing PDF forms for automated filling. In order to convert all form fields on a page to PDFlib blocks choose *PDFlib Blocks, Convert Form Fields, Current Page*. To convert all form fields in a document choose *All Pages* instead. Finally, you can convert only selected form fields (choose Acrobat's Form Tool or the Select Object Tool to select form fields) with *Selected Form Fields*.

Form field conversion details. Automatic form field conversion will convert form fields of the types selected in the *PDFlib Blocks, Convert Form Fields, Conversion Options...* dialog to blocks of type *Text*. By default all form field types will be converted. Attributes of the converted fields will be transformed to the corresponding block properties according to Table 10.2.

Table 10.2 Conversion of PDF form fields to PDFlib blocks

PDF form field attribute...	...will be converted to the PDFlib block property
all fields	
Position	General, Rect
Name	General, Name
Tooltip	General, Description
Appearance, Text, Font	Text, fontname
Appearance, Text, Font Size	Text, fontsize; »auto« font size will be converted to a fixed font size of 2/3 of the block height, and the fitmethod will be set to auto. For multi-line fields/blocks this combination will automatically result in a suitable font size which may be smaller than the initial value of 2/3 of the block height.
Appearance, Text, Text Color	Text, strokecolor; Text, fillcolor
Appearance, Border, Border Color	General, bordercolor
Appearance, Border, Fill Color	General, backgroundcolor
Appearance, Border, Line Thickness	General, linewidth: Thin=1, Medium=2, Thick=3
General, Common Properties, Form Field	General, Status: Visible=active, Hidden=ignore, Visible but doesn't print=ignore, Hidden but printable=active
General, Common Properties, Orientation	General, orientate: 0=north, 90=west, 180=south, 270=east
text fields	
Options, Default Value	Text, defaulttext
Options, Alignment	General, position: Left={left center}, Center={center center}, Right={right center}
Options, Multi-line	Text, textflow: checked=true, unchecked=false
radio buttons and check boxes	
If »Check box/Button is checked by default« is selected: Options, Check Box Style or Options, Button Style	Text, defaulttext: Check=4, Circle=l, Cross=8, Diamond=u, Square=n, Star=H (these characters represent the respective symbols in the ZapfDingbats font)

Table 10.2 Conversion of PDF form fields to PDFlib blocks

PDF form field attribute...	...will be converted to the PDFlib block property
list boxes and combo boxes	
Options, Selected (default) item	Text, defaulttext
buttons	
Options, Icon and Label, Label	Text, defaulttext

Multiple form fields with the same name. Multiple form fields on the same page are allowed to have the same name, while block names must be unique on a page. When converting form fields to blocks a numerical suffix will therefore be added to the name of generated blocks in order to create unique block names (see also »Associating form fields with corresponding blocks«, page 226).

Note that due to a problem in Acrobat the field attributes of form fields with the same names are not reported correctly. If multiple fields have the same name, but different attributes these differences will not be reflected in the generated blocks. The Conversion process will issue a warning in this case and provide the names of affected form fields. In this case you should carefully check the properties in the generated blocks.

Associating form fields with corresponding blocks. Since the form field names will be modified when converting multiple fields with the same name (e.g. radio buttons) it is difficult to reliably identify the block which corresponds to a particular form field. This is especially important when using an FDF or XPDF file as the source for filling blocks such that the final result resembles the filled form.

In order to solve this problem the AcroFormConversion plugin will record details about the original form field as custom properties when creating the corresponding block. Table 10.3 details the custom properties which can be used to reliably identify the blocks; all properties have type *string*.

Table 10.3 Custom properties for identifying the original form field corresponding to the block

custom property	meaning
PDFlib:field:name	Fully qualified name of the form field
PDFlib:field:pagenumber	Page number (as a string) in the original document where the form field was located
PDFlib:field:type	Type of the form field; one of pushbutton, checkbox, radiobutton, listbox, combobox, textfield, signature
PDFlib:field:value	(Only for type=checkbox) Export value of the form field

Binding blocks to the corresponding form fields. In order to keep PDF form fields and the generated PDFlib blocks synchronized, the generated blocks can be bound to the corresponding form fields. This means that the block tool will internally maintain the relationship of form fields and blocks. When the conversion process is activated again, bound blocks will be updated to reflect the attributes of the corresponding PDF form fields. Bound blocks are useful to avoid duplicate work: when a form is updated for interactive use, the corresponding blocks can automatically be updated, too.

If you do not want to keep the converted form fields after blocks have been generated you can choose the option *Delete converted Form Fields* in the *PDFlib Blocks, Convert*

Form Fields, Conversion Options... dialog. This option will permanently remove the form fields after the conversion process. Any actions (e.g., JavaScript) associated with the affected fields will also be removed from the document.

Batch conversion. If you have many PDF documents with form fields that you want to convert to PDFlib blocks you can automatically process an arbitrary number of documents using the batch conversion feature. The batch processing dialog is available via *PDFlib Blocks, Convert Form Fields, Batch conversion...*:

- ▶ The input files can be selected individually; alternatively the full contents of a folder can be processed.
- ▶ The output files can be written to the same folder where the input files are, or to a different folder. The output files can receive a prefix to their name in order to distinguish them from the input files.
- ▶ When processing a large number of documents it is recommended to specify a log file. After the conversion it will contain a full list of processed files as well as details regarding the result of each conversion along with possible error messages.

During the conversion process the converted PDF documents will be visible in Acrobat, but you cannot use any of Acrobat's menu functions or tools.

10.4 Standard Properties for Automated Processing

PDFlib supports general properties which can be assigned to any type of block. In addition there are properties which are specific to the block types *Text*, *Image*, and *PDF*. Some properties are *shared*, which means that they can be assigned to multiple blocks at once using the Block plugin.

Properties support the same data types as option lists except handles and action lists.

Many block properties have the same name as options for *fit_image()* (e.g., *fitmethod*) and other functions, or as PDFlib parameters (e.g., *charspacing*). In these cases the behavior is exactly the same as the one documented for the respective option or parameter.

Property processing in PDFlib. The PDFlib Block functions *fill_*block()* will process block properties in the following order:

- ▶ If the *backgroundcolor* property is present and contains a color space keyword different from *None*, the block rectangle will be filled with the specified color.
- ▶ All other properties except *bordercolor* and *linewidth* will be processed.
- ▶ If the *bordercolor* property is present and contains a color space keyword different from *None*, the block rectangle will be stroked with the specified color and linewidth.
- ▶ Text blocks: if neither text nor default text has been supplied, there won't be any output at all, not even background color or block border.

There will be no clipping; if you want to make sure that the block contents do not exceed the block rectangle avoid *fitmethod nofit*.

To use a separation (spot) color in a block property you can click the »...« button which will present a list of all HKS and Pantone spot colors. These color names are built into PDFlib (see Section 3.3.2, »Spot Colors«, page 55) and can be used without further preparations. For custom spot colors an alternate color can be defined in the Block plugin. If no alternate color is specified in the Block properties, the custom spot color must have been defined earlier in the PDFlib application using *makespotcolor()*. Otherwise the block functions will fail.

10.4.1 General Properties

General properties apply to all kinds of blocks (*Text*, *Image*, *PDF*). They are required for block administration, describe the appearance of the block rectangle itself, and manage how the contents will be placed within the block. Required entries will automatically be generated by the PDFlib Block Plugin. Table 10.4 lists the general properties.

Table 10.4 General block properties

keyword	possible values and explanation
Block administration	
Name	(String; required) Name of the block. Block names must be unique within a page, but not within a document. The three characters [] / are not allowed in block names. Block names are restricted to a maximum of 125 characters.
Description	(String) Human-readable description of the block's function, coded in PDFDocEncoding or Unicode (in the latter case starting with a BOM). This property is for user information only, and will be ignored when processing the block.

Table 10.4 General block properties

keyword	possible values and explanation
Locked	(Boolean; shareable) If true, the block and its properties can not be edited with the Block plugin. This property will be ignored when processing the block. Default: false.
Rect	(Rectangle; required) The block coordinates. The origin of the coordinate system is in the lower left corner of the page. However, the Block plugin will display the coordinates in Acrobat's notation, i.e., with the origin in the upper left corner of the page. The coordinates will be displayed in the unit which is currently selected in Acrobat, but will always be stored in points in the PDF file.
Status	(Keyword) Describes how the block will be processed. Default: active. active The block will be fully processed according to its properties. ignore The block will be ignored. static No variable contents will be placed; instead, the block's default text, image, or PDF contents will be used if available.
Subtype	(Keyword; required) Depending on the block type, one of Text, Image, or PDF
Type	(Keyword; required) Always Block
Block appearance	
background-color	(Color; shareable) If this property is present and contains a color space keyword different from None, a rectangle will be drawn and filled with the supplied color. This may be useful to cover existing page contents. Default: None
bordercolor	(Color; shareable) If this property is present and contains a color space keyword different from None, a rectangle will be drawn and stroked with the supplied color. Default: None
linewidth	(Float; shareable; must be greater than 0) Stroke width of the line used to draw the block rectangle; only used if bordercolor is set. Default: 1
Content placing	
fitmethod	(Keyword; shareable) Strategy to use if the supplied content doesn't fit into the box. Possible values are auto, nofit, clip, meet ¹ , slice ¹ , and entire ¹ . For simple text blocks, image, and PDF blocks this property will be interpreted according to the standard interpretation. Default: auto. For Textflow blocks where the block is too small for the text the interpretation is as follows: auto fontsize and leading will be decreased until the text fits. nofit Text will run beyond the bottom margin of the block. clip Text will be clipped at the block margin.
orientate	(Keyword; shareable) Specifies the desired orientation of the content when it is placed. Possible values are north, east, south, west. Default: north
position¹	(Float list; shareable) One or two values specifying the position of the reference point within the content. The position is specified as a percentage within the block. Default: {0 0}, i.e. the lower left corner
rotate	(Float; shareable) Rotation angle in degrees by which the block will be rotated counter-clockwise before processing begins. The reference point is center of the rotation. Default: 0

1. This keyword or property is not supported for Textflow blocks (text blocks with textflow=true).

10.4.2 Text Properties

Text-related properties apply to blocks of type *Text* (in addition to general properties). All text-related properties can be shared.

Properties for all text blocks. Text blocks can contain a single line or multiple lines, depending on the *textflow* property. Table 10.5 lists the text-related properties which apply to both types.

Table 10.5 Text block properties

keyword	possible values and explanation
alignchar	(Unichar or keyword) If the specified character is found in the text, its lower left corner will be aligned at the reference point. For horizontal text with <i>orientate</i> =north or south the first value supplied in the position option defines the position. For horizontal text with <i>orientate</i> =west or east the second value supplied in the position option defines the position. This option will be ignored if the specified alignment character is not present in the text. The value <i>o</i> and the keyword <i>none</i> suppress alignment characters. The specified <i>fitmethod</i> will be applied, although the text cannot be placed within the fitbox because of the forced positioning of <i>alignchar</i> . Default: <i>none</i>
charspacing	(Float or percentage) Character spacing. Percentages are based on <i>fontsize</i> . Default: <i>o</i>
defaulttext	(String) Text which will be used if no substitution text is supplied by the client ¹
escape-sequence	(Boolean) If <i>true</i> , enable substitution of escape sequences in strings. Default: the global <i>escapesequences</i> parameter
fillcolor	(Color) Fill color of the text. Default: <i>gray o (=black)</i>
fontname²	(String) Name of the font as required by <i>load_font()</i> . The PDFlib Block plugin will present a list of system-installed fonts. However, these font names may not be portable between Mac, Windows, and Unix systems. The encoding for the text must be specified as an option for <i>fill_textblock()</i> when filling the block unless the font option has been supplied.
fontsize²	(Float) Size of the font in points
fontstyle	(Keyword) Font style, must be one of <i>normal</i> , <i>bold</i> , <i>italic</i> , or <i>bolditalic</i>
horizscaling	(Float or percentage) Horizontal text scaling. Default: <i>100%</i>
italicangle	(Float) Italic angle of text in degrees. Default: <i>o</i>
 Kerning	(Boolean) Kerning behavior. Default: <i>false</i>
margin	(Float list) One or two float values describing additional horizontal and vertical extensions of the text box. Default: <i>o</i>
monospace	(Integer: 1...2048) Forces the same width for all characters in the font. Default: <i>absent</i> (metrics from the font will be used)
overline	(Boolean) Overline mode. Default: <i>false</i>
stamp	(Keyword; will be ignored if <i>boxsize</i> is not specified) This option can be used to create a diagonal stamp within the box specified in the <i>boxsize</i> option. The text comprising the stamp will be as large as possible. The options <i>position</i> , <i>fitmethod</i> , and <i>orientate</i> (only north and south) will be honored when placing the stamp text in the box. Default: <i>none</i> . <i>llzur</i> The stamp will run diagonally from the lower left corner to the upper right corner. <i>ulzlr</i> The stamp will run diagonally from the upper left corner to the lower right corner. <i>none</i> No stamp will be created.
strikeout	(Boolean) Strikeout mode. Default: <i>false</i>

Table 10.5 Text block properties

keyword	possible values and explanation
strokecolor	(Color) Stroke color of the text. Default: gray 0 (=black)
textflow	(Boolean) Controls single- or multiline processing (Default: false): false Text can span a single line and will be processed with <code>fit_text()</code> . true Text can span multiple lines and will be processed with <code>fit_textflow()</code> . The general property position will be ignored. In addition to the standard text properties all Textflow-related properties can be specified (see Table 10.6).
textrendering	(Integer) Text rendering mode. Default: 0
textrise	(Float or percentage) Text rise parameter. Percentages are based on fontsize. Default: 0
underline	(Boolean) Underline mode. Default: false
underline-position	(Float, percentage, or keyword) Position of the stroked line for underlined text relative to the baseline. Percentages are based on fontsize. Default: auto
underline-width	(Float, percentage, or keyword) Line width for underlined text. Percentages are based on fontsize. Default: auto
wordspacing	(Float or percentage) Word spacing. Percentages are based on fontsize. Default: 0

1. The text will be interpreted in winansi encoding or Unicode.

2. This property is required in a text block; it will automatically be enforced by the PDFlib Block plugin.

Properties for Textflow blocks. Textflow-related properties apply to blocks of type *Text* where the *textflow* property is *true*. The text-related properties will be used to construct the initial option list for processing the Textflow (corresponding to the *optlist* parameter of *create_textflow()*). Inline option lists can not be specified with the plugin, but they can be supplied on the server as part of the text contents when filling the block with *fill_textblock()*. All Textflow-related properties can be shared. Table 10.6 lists the Textflow-related properties.

Table 10.6 Textflow block properties

keyword	possible values and explanation
property for text semantics	
tabalignchar	(Integer) Unicode value of the character at which decimal tabs will be aligned. Default: the ' ' character (U+0020)
properties for controlling the text layout	
alignment	(Keyword) Specifies formatting for lines in a paragraph. Default: left. left left-aligned, starting at leftindent center centered between leftindent and rightindent right right-aligned, ending at rightindent justify left- and right-aligned

Table 10.6 Textflow block properties

keyword	possible values and explanation
firstlinedist	<p>(Float, percentage, or keyword) The distance between the top of the fitbox and the baseline for the first line of text, specified in user coordinates, as a percentage of the relevant font size (the first font size in the line if <code>fixedleading=true</code>, and the maximum of all font sizes in the line otherwise), or as a keyword. Default: <code>leading</code>.</p> <p>leading The leading value determined for the first line; typical diacritical characters such as <code>À</code> will touch the top of the fitbox.</p> <p>ascender The ascender value determined for the first line; typical characters with larger ascenders, such as <code>d</code> and <code>h</code> will touch the top of the fitbox.</p> <p>capheight The capheight value determined for the first line; typical capital uppercase characters such as <code>H</code> will touch the top of the fitbox.</p> <p>xheight The xheight value determined for the first line; typical lowercase characters such as <code>x</code> will touch the top of the fitbox.</p> <p>If <code>fixedleading=false</code> the maximum of all <code>leading</code>, <code>ascender</code>, <code>xheight</code>, or <code>capheight</code> values found in the first line will be used.</p>
fixedleading	<p>(Boolean) If true, the first leading value found in each line will be used. Otherwise the maximum of all leading values in the line will be used. Default: <code>false</code></p>
hortabsize	<p>(Float or percentage) Width of a horizontal tab¹. The interpretation depends on the <code>hortabmethod</code> option. Default: <code>7.5%</code></p>
hortab-method	<p>(Keyword) Treatment of horizontal tabs in the text. If the calculated position is to the left of the current text position, the tab will be ignored. Default: <code>relative</code>.</p> <p>relative The position will be advanced by the amount specified in <code>hortabsize</code>.</p> <p>typewriter The position will be advanced to the next multiple of <code>hortabsize</code>.</p> <p>ruler The position will be advanced to the <code>n</code>-th tab value in the <code>ruler</code> option, where <code>n</code> is the number of tabs found in the line so far. If <code>n</code> is larger than the number of tab positions the relative method will be applied.</p>
lastalignment	<p>(Keyword) Formatting for the last line in a paragraph. All keywords of the alignment option are supported, plus the following. Default: <code>auto</code>.</p> <p>auto Use the value of the alignment option unless it is <code>justify</code>. In the latter case left will be used.</p>
lastlinedist	<p>(Float, percentage, or keyword) Will be ignored for <code>fitmethod=nofit</code>) The minimum distance between the baseline for the last line of text and the bottom of the fitbox, specified in user coordinates, as a percentage of the font size (the first font size in the line if <code>fixedleading=true</code>, and the maximum of all font sizes in the line otherwise), or as a keyword. Default: <code>o</code>, i.e. the bottom of the fitbox will be used as baseline, and typical descenders will extend below the fitbox.</p> <p>descender The descender value determined for the last line; typical characters with descenders, such as <code>g</code> and <code>j</code> will touch the bottom of the fitbox.</p> <p>If <code>fixedleading=false</code> the maximum of all descender values found in the last line will be used.</p>
leading	<p>(Float or percentage) Distance between adjacent text baselines in user coordinates, or as a percentage of the font size. Default: <code>100%</code></p>
parindent	<p>(Float or percentage) Left indent of the first line of a paragraph¹. The amount will be added to <code>leftindent</code>. Specifying this option within a line will act like a tab. Default: <code>o</code></p>
rightindent leftindent	<p>(Float or percentage) Right or left indent of all text lines¹. If <code>leftindent</code> is specified within a line and the determined position is to the left of the current text position, this option will be ignored for the current line. Default: <code>o</code></p>
rotate	<p>(Float) Rotate the coordinate system, using the lower left corner of the fitbox as center and the specified value as rotation angle in degrees. This results in the box and the text being rotated. The rotation will be reset when the text has been placed. Default: <code>o</code></p>

Table 10.6 Textflow block properties

keyword	possible values and explanation
ruler²	(List of floats or percentages) List of absolute tab positions for <code>horthabmethod=ruler¹</code> . The list may contain up to 32 non-negative entries in ascending order. Default: integer multiples of <code>horthabsize</code>
tabalignment	(List of keywords) Alignment for tab stops. Each entry in the list defines the alignment for the corresponding entry in the ruler option. Default: left. center Text will be centered at the tab position. decimal The first instance of <code>tabalignchar</code> will be left-aligned at the tab position. If no <code>tabalignchar</code> is found, right alignment will be used instead. left Text will be left-aligned at the tab position. right Text will be right-aligned at the tab position.
verticalalign	(Keyword) Vertical alignment of the text in the fitbox. Default: top. top Formatting will start at the first line, and continue downwards. If the text doesn't fill the fitbox there may be whitespace below the text. center The text will be vertically centered in the fitbox. If the text doesn't fill the fitbox there may be whitespace both above and below the text. bottom Formatting will start at the last line, and continue upwards. If the text doesn't fill the fitbox there may be whitespace above the text. justify The text will be aligned with top and bottom of the fitbox. In order to achieve this the leading will be increased up to the limit specified by <code>linespreadlimit</code> . The height of the first line will only be increased if <code>firstlinedist=leading</code> .
properties for controlling the line-breaking algorithm	
adjust-method	(Keyword) Method used to adjust a line when a text portion doesn't fit into a line after compressing or expanding the distance between words subject to the limits specified by the <code>minspacing</code> and <code>maxspacing</code> options. Default: auto. auto The following methods are applied in order: shrink, spread, nofit, split. clip Same as <code>nofit</code> , except that the long part at the right edge of the fit box (taking into account the <code>rightindent</code> option) will be clipped. nofit The last word will be moved to the next line provided the remaining (short) line will not be shorter than the percentage specified in the <code>nofitlimit</code> option. Even justified paragraphs may look slightly ragged. shrink If a word doesn't fit in the line the text will be compressed subject to <code>shrinklimit</code> . If it still doesn't fit the <code>nofit</code> method will be applied. split The last word will not be moved to the next line, but will forcefully be hyphenated. For text fonts a hyphen character will be inserted, but not for symbol fonts. spread The last word will be moved to the next line and the remaining (short) line will be justified by increasing the distance between characters in a word, subject to <code>spreadlimit</code> . If justification still cannot be achieved the <code>nofit</code> method will be applied.
linespread-limit	(Float or percentage; only for <code>verticalalign=justify</code>) Maximum amount in user coordinates or as percentage of the leading for increasing the leading for vertical justification. Default: 200%
maxlines	(Integer or keyword) The maximum number of lines in the fitbox, or the keyword auto which means that as many lines as possible will be placed in the fitbox. When the maximum number of lines has been placed <code>fit_textflow()</code> will return the string <code>_boxfull</code> .
maxspacing minspacing	(Float or percentage) The maximum or minimum distance between words (in user coordinates, or as a percentage of the width of the space character). The calculated word spacing is limited by the provided values (but the <code>wordspacing</code> option will still be added). Defaults: <code>minspacing=50%</code> , <code>maxspacing=500%</code>
nofitlimit	(Float or percentage) Lower limit for the length of a line with the <code>nofit</code> method (in user coordinates or as a percentage of the width of the fitbox). Default: 75%.

Table 10.6 Textflow block properties

keyword	possible values and explanation
shrinklimit	(Percentage) Lower limit for compressing text with the <i>shrink</i> method; the calculated shrinking factor is limited by the provided value, but will be multiplied with the value of the <i>horizscaling</i> option. Default: 85%
spreadlimit	(Float or percentage) Upper limit for the distance between two characters for the <i>spread</i> method (in user coordinates or as a percentage of the font size); the calculated character distance will be added to the value of the <i>charspacing</i> option. Default: 0

1. In user coordinates, or as a percentage of the width of the fit box
2. Rulers can be edited in the »Tabs« section of the Block properties dialog.

10.4.3 Image Properties

Image-related properties apply to blocks of type *Image* (in addition to general properties). All image-related properties can be shared. Table 10.7 lists image-related properties.

Table 10.7 Image block properties

keyword	possible values and explanation
defaultimage	(String) Path name of an image which will be used if no substitution image is supplied by the client. It is recommended to use file names without absolute paths, and use the SearchPath feature in the PPS client application. This will make block processing independent from platform and file system details.
dpi	(Float list) One or two values specifying the desired image resolution in pixels per inch in horizontal and vertical direction. With the value 0 the image's internal resolution will be used if available, or 72 dpi otherwise. This property will be ignored if the fitmethod property has been supplied with one of the keywords auto, meet, slice, or entire. Default: 0
scale	(Float list) One or two values specifying the desired scaling factor(s) in horizontal and vertical direction. This option will be ignored if the fitmethod property has been supplied with one of the keywords auto, meet, slice, or entire. Default: 1

10.4.4 PDF Properties

PDF-related properties apply to blocks of type *PDF* (in addition to general properties). All PDF-related properties can be shared. Table 10.8 lists PDF-related properties.

Table 10.8 PDF block properties

keyword	possible values and explanation
defaultpdf	(String) Path name of a PDF document which will be used if no substitution PDF is supplied by the client. It is recommended to use file names without absolute paths, and use the SearchPath feature in the PPS client application. This will make block processing independent from platform and file system details.
default-pdfpage	(Integer) Page number of the page in the default PDF document. Default: 1
scale	(Float list) One or two values specifying the desired scaling factor(s) in horizontal and vertical direction. This option will be ignored if the fitmethod property has been supplied with one of the keywords auto, meet, slice, or entire. Default: 1
pdieusebox	(Keyword; possible values: media, crop, bleed, trim, art) Use the placed page's MediaBox, CropBox, BleedBox, TrimBox, or ArtBox for determining its size. Default: crop

10.4.5 Custom Properties

Custom properties apply to blocks of any type of block (in addition to general and type-specific properties). Custom properties are optional, and can not be shared. Table 10.9 lists the naming rules for custom properties.

Table 10.9 Custom block properties

keyword	possible values and explanation
any name not containing the three characters [] /	(String, name, float, or float list) The interpretation of the values corresponding to custom properties is completely up to the client application.

10.5 Querying Block Names and Properties with pCOS

In addition to automatic block processing with PPS, the integrated pCOS facility can be used to enumerate block names and query standard or custom properties.

Finding the numbers and names of blocks. The client code must not even know the names or numbers of the blocks on an imported page since these can also be queried. The following statement returns the number of blocks on page with number *pagenum*:

```
blockcount = (int) p.pcos_get_number(doc,
    "length:pages[" + pagenum + "]/PieceInfo/PDFlib/Private/Blocks");
```

The following statement returns the name of block number *pagenum* on page *pagenum* (block and page counting start at 0):

```
blockname = p.pcos_get_string(doc,
    "pages[" + pagenum + "]/PieceInfo/PDFlib/Private/Blocks[" + blocknum + "]/Name");
```

The returned block name can subsequently be used to query the block's properties or populate the block with text, image, or PDF content. If the specified block doesn't exist an exception will be thrown. You can avoid this by using the *length* prefix to determine the number of blocks and therefore the maximum index in the *Blocks* array (remember that the block count will be 1 higher than the highest possible index since array indexing starts at 0).

In the path syntax for addressing block properties the following expressions are equivalent, assuming that the block with the sequential *<number>* has its *Name* property set to *<blockname>*:

```
pages[...]/PieceInfo/PDFlib/Private/Blocks[<number>]
pages[...]/PieceInfo/PDFlib/Private/Blocks/<blockname>
```

Finding block coordinates. The two coordinate pairs (*llx*, *lly*) and (*urx*, *ury*) describing the lower left and upper right corner of a block named *foo* can be queried as follows:

```
llx = p.pcos_get_number(doc,
    "pages[" + pagenum + "]/PieceInfo/PDFlib/Private/Blocks/foo/Rect[0]");
lly = p.pcos_get_number(doc,
    "pages[" + pagenum + "]/PieceInfo/PDFlib/Private/Blocks/foo/Rect[1]");
urxx = p.pcos_get_number(doc,
    "pages[" + pagenum + "]/PieceInfo/PDFlib/Private/Blocks/foo/Rect[2]");
ury = p.pcos_get_number(doc,
    "pages[" + pagenum + "]/PieceInfo/PDFlib/Private/Blocks/foo/Rect[3]");
```

Note that these coordinates are provided in the default user coordinate system (with the origin in the bottom left corner, possibly modified by the page's CropBox), while the Block plugin displays the coordinates according to Acrobat's user interface coordinate system with an origin in the upper left corner of the page. Since the *Rect* option for overriding block coordinates does not take into account any modifications applied by the CropBox entry, the coordinates queried from the original block cannot be directly used as new coordinates if a CropBox is present. As a workaround you can use the *refpoint* and *boxsize* options.

Also note that the *topdown* parameter is not taken into account when querying block coordinates.

Querying custom properties. Custom properties can be queried as in the following example, where the property *zipcode* is queried from a block named *b1* on page *pagenum*:

```
zip = p.pcos_get_string(doc,
    "pages[" + pagenum + "]/PieceInfo/PDFlib/Private/Blocks/b1/Custom/zipcode");
```

If you don't know which custom properties are actually present in a block, you can determine the names at runtime. In order to find the name of the first custom property in a block named *b1* use the following:

```
propname = p.pcos_get_string(doc,
    "pages[" + pagenum + "]/PieceInfo/PDFlib/Private/Blocks/b1/Custom[0].key");
```

Use increasing indexes instead of 0 in order to determine the names of all custom properties. Use the *length* prefix to determine the number of custom properties.

Non-existing block properties and default values. Use the *type* prefix to determine whether a block or property is actually present. If the type for a path is 0 or *null* the respective object is not present in the PDF document. Note that for standard properties this means that the default value of the property will be used.

Name space for custom properties. In order to avoid confusion when PDF documents from different sources are exchanged, it is recommended to use an Internet domain name as a company-specific prefix in all custom property names, followed by a colon ':' and the actual property name. For example, ACME corporation would use the following property names:

```
acme.com:digits
acme.com:refnumber
```

Since standard and custom properties are stored differently in the block, standard PDFlib property names (as defined in Section 10.4, »Standard Properties for Automated Processing«, page 228) will never conflict with custom property names.

10.6 PDFlib Block Specification

The PDFlib Block syntax is fully compliant with the PDF Reference, which specifies an extension mechanism that allows applications to store private data attached to the data structures comprising a PDF page. A detailed description of the PDFlib block syntax is provided here for the benefit of users who wish to create PDFlib blocks by other means than the PDFlib block plugin. Plugin users can safely skip this section.

10.6.1 PDF Object Structure for PDFlib Blocks

The page dictionary contains a */PieceInfo* entry, which has another dictionary as value. This dictionary contains the key */PDFlib* with an application data dictionary as value. The application data dictionary contains two standard keys listed in Table 10.10.

Table 10.10 Entries in a PDFlib application data dictionary

key	value
LastModified	(Data string; required) The date and time when the blocks on the page were created or most recently modified.
Private	(Dictionary; required) A block list (see Table 10.11)

A Block list is a dictionary containing general information about block processing, plus a list of all blocks on the page. Table 10.11 lists the keys in a block list dictionary.

Table 10.11 Entries in a block list dictionary

key	value
Version	(Number; required) The version number of the block specification to which the file complies. This document describes version 7 of the block specification.
Blocks	(Dictionary; required) Each key is a name object containing the name of a block; the corresponding value is the block dictionary for this block (see Table 10.13). The <i>/Name</i> key in the block dictionary must be identical to the block's name in this dictionary.
PluginVersion	(String; required unless the <i>pdfmark</i> key is present ¹) A string containing a version identification of the PDFlib Block plugin which has been used to create the blocks.
pdfmark	(Boolean; required unless the <i>PluginVersion</i> key is present ¹) Must be true if the block list has been generated by use of <i>pdfmarks</i> .

1. Exactly one of the *PluginVersion* and *pdfmark* keys must be present.

Data types for block properties. Properties support the same data types as option lists except handles and action lists. Table 10.12 details how these types are mapped to PDF data types.

Table 10.12 Data types for block properties

block type	PDF type and remarks
boolean	(Boolean)
string	(String)
keyword	(Name) It is an error to provide keywords outside the list of keywords supported by a particular property.

Table 10.12 Data types for block properties

block type	PDF type and remarks
float, integer	(Number) While option lists support both point and comma as decimal separators, PDF numbers support only point.
percentage	(Array with two elements) The first element in the array is the number, the second element is a string containing a percent character.
color	<p>(Array with two or three elements) The first element in the array specifies a color space, and the second element specifies a color value as follows. The following entries are supported for the first element in the array:</p> <p>/DeviceGray The second element is a single gray value.</p> <p>/DeviceRGB The second element is an array of three RGB values.</p> <p>/DeviceCMYK The second element is an array of four CMYK values.</p> <p>[/Separation/spotname] The first element is an array containing the keyword /Separation and a spot color name. The second element is a tint value. The optional third element in the array specifies an alternate color for the spot color, which is itself a color array in one of the /DeviceGray, /DeviceRGB, /DeviceCMYK, or /Lab color spaces. If the alternate color is missing, the spot color name must either refer to a color which is known internally to PDFlib, or which has been defined by the application at runtime.</p> <p>[/Lab] The first element is an array containing the keyword /Lab. The second element is an array of three Lab values.</p> <p>To specify the absence of color the respective property must be omitted.</p>
unicar	(Text string) Unicode strings in utf16be format, starting with the U+FEFF BOM

Block dictionary keys. Block dictionaries may contain the keys in Table 10.13. Only keys from one of the Text, Image or PDF groups may be present depending on the /Subtype key in the General group (see Table 10.4).

Table 10.13 Entries in a block dictionary

key	value
general properties	(Some keys are required) General properties according to Table 10.4
text properties	(Optional) Text and Textflow properties according to Table 10.5 and Table 10.6
image properties	(Optional) Image properties according to Table 10.7
PDF properties	(Optional) PDF properties according to Table 10.8
Custom	(Dictionary; optional) A dictionary containing key/value pairs for custom properties according to Table 10.9.
Internal	(Dictionary; optional) This key is reserved for private use, and applications should not depend on its presence or specific behavior. Currently it is used for maintaining the relationship between converted form fields and corresponding blocks.

Example. The following fragment shows the PDF code for two blocks, a text block called *job_title* and an image block called *logo*. The text block contains a custom property called *format*:

```

<<
  /Contents 12 0 R
  /Type /Page
  /Parent 1 0 R
  /MediaBox [ 0 0 595 842 ]
  /PieceInfo << /PDFlib 13 0 R >>
>>

13 0 obj
<<
  /Private <<
    /Blocks <<
      /job_title 14 0 R
      /logo 15 0 R
    >>
    /Version 7
    /PluginVersion (3.0)
  >>
  /LastModified (D:20060813200730)
>>
endobj

14 0 obj
<<
  /Type /Block
  /Rect [ 70 740 200 800 ]
  /Name /job_title
  /Subtype /Text
  /fitmethod /auto
  /fontname (Helvetica)
  /fontsize 12
  /Custom << /format 5 >>
>>
endobj

15 0 obj
<<
  /Type /Block
  /Rect [ 250 700 400 800 ]
  /Name /logo
  /Subtype /Image
  /fitmethod /auto
>>

```

10.6.2 Generating PDFlib Blocks with pdfmarks

As an alternative to creating PDFlib blocks with the plugin, blocks can be created by inserting appropriate *pdfmark* commands into a PostScript stream, and distilling it to PDF. Details of the *pdfmark* operator are discussed in the Acrobat documentation. The following fragment shows *pdfmark* operators which can be used to generate the block definition in the preceding section:

```

% ----- Setup for the blocks on a page -----
[/_objdef {B1} /type /dict /OBJ pdfmark          % Blocks dict

[{ThisPage} <<
  /PieceInfo <<
    /PDFlib <<

```

```

        /LastModified (D:20060813200730)
        /Private <<
            /Version 7
            /pdfmark true
            /Blocks {B1}
        >>
    >>
>> /PUT pdfmark

% ----- text block -----
[ {B1} <<
    /job_title <<
        /Type /Block
        /Name /job_title
        /Subtype /Text
        /Rect [ 70 740 200 800 ]
        /fitmethod /auto
        /fontsize 12
        /fontname (Helvetica)
        /Custom << /format 5 >>
    >>
>> /PUT pdfmark

% ----- image block -----
[ {B1} <<
    /logo <<
        /Type /Block
        /Name /logo
        /Subtype /Image
        /Rect [ 250 700 400 800 ]
        /fitmethod /auto
    >>
>> /PUT pdfmark

```

