13 PPS and the PDFlib Block Plugin

The PDFlib Personalization Server (PPS) supports a template-driven PDF workflow for variable data processing. Using the Block concept, imported pages can be populated with variable amounts of single- or multi-line text, images, PDF pages or vector graphics. 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

You can create and edit Blocks interactively with the PDFlib Block Plugin, convert existing PDF form fields to PDFlib Blocks with the form field conversion Plugin. Blocks can be filled with PPS. The results of Block filling with PPS can be previewed in Acrobat since the Block Plugin contains an integrated version of PPS.

Note *Block processing requires the PDFlib Personalization Server (PPS). Although PPS is contained in all 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.*

Cookbook *Code samples regarding variable data and Blocks can be found in the* `blocks` *category of the PDFlib Cookbook.*

13.1 Installing the PDFlib Block Plugin

The Block Plugin works with the following Acrobat versions (it doesn't work with Adobe Reader):

- Windows: Acrobat 8/9/X/XI
- Windows: Acrobat Standard and Pro 2017/2020
- Windows: Acrobat Pro DC (32-bit and 64-bit)
- macOS: Acrobat Standard and Pro 2017/2020 (Intel)
- macOS: Acrobat DC (universal binary for Intel and M1)

Since Acrobat DC on Windows is available in 32-bit and 64-bit versions two different installers are available. It is important to use the appropriate installer which matches the installed Acrobat version.

Installing the PDFlib Block Plugin on Windows. To install the PDFlib Block Plugin and the PDF form field conversion Plugin in Acrobat, the plugin files must be placed in 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*.

The plugin folder for Acrobat 32-bit on 64-bit Windows typically looks as follows:

```
C:\Program Files (x86)\Adobe\Acrobat DC\Acrobat\plug_ins\PDFlib Block Plugin
```

The plugin folder for Acrobat 64-bit typically looks as follows:

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

Installing the PDFlib Block Plugin on macOS. Proceed as follows to install the Plugin for all users:

- Double-click the disk image to mount it. A folder with the Plugin files will be visible.
- Copy the Plugin folder to the following path in the system's *Library* folder (create the *Plug-Ins* folder if it doesn't yet exist):

```
/Library/Application Support/Adobe/Acrobat/XXX/Plug-ins
```

Alternatively you can install the Plugin only for a single user as follows:

- Click the desktop to make sure you're in the Finder, hold down the *Option* key, and choose *Go, Library* to open the user's *Library* folder.
- Copy the Plugin folder to the following path in the user's *Library* folder (create the *Plug-Ins* folder if it doesn't yet exist):

```
/Users/<username>/Library/Application Support/Adobe/Acrobat/XXX/Plug-ins
```

Multi-lingual Interface. The PDFlib Block Plugin supports multiple languages in the user interface. Depending on the application language of Acrobat, the Block Plugin chooses its interface language automatically. Currently English, German and Japanese interfaces are available. If Acrobat runs in any other language mode, the Block Plugin uses the English interface.

Sandbox Protection for Acrobat DC on Windows. Acrobat DC 2020 introduced a new security model called *Sandbox Protections* which can be activated via *Preferences, Security (Enhanced), Protected Mode* and *Protected View*. If it is enabled various operations are restricted and a yellow bar with a security message appears at the top of the document window. More information about Sandbox Protections can be found at:

```
helpx.adobe.com/acrobat/using/whats-new/2020-august.html
www.adobe.com/devnet-docs/acrobatetk/tools/AppSec/sandboxprotections.html
```

If Sandboxing is enabled it affects the Preview feature of the PDFlib Block Plugin. Protected View by default grants access to Acrobat's *AppData* directory, the temp directory and several other directories, but not to arbitrary user directories. The Block Plugin can only read from and write to directories which are included in the default directory list of Protected View or which have been configured (whitelisted) in a policy file at the following location (for 32-bit and 64-bit versions of Acrobat):

```
C:\Program Files (x86)\Adobe\Acrobat DC\Acrobat\PDFlibBlockCustomPolicies.txt
C:\Program Files\Adobe\Acrobat DC\Acrobat\PDFlibBlockCustomPolicies.txt
```

By default the policy file grants access to the following directories, but more directory names can be added by the Administrator:

```
; Protected Path Section
FILES_ALLOW_ANY = C:\Users\<username>\*.*
FILES_ALLOW_ANY = C:\Users\Public\*.*
```

If Protected Mode or Protected View is enabled and directories are used which are not whitelisted, some features of the Block Plugin including Preview and Block import/export may fail.

Troubleshooting. If the PDFlib Block Plugin doesn't seem to work check the following:

- Make sure that in *Edit*, *Preferences*, *[General...]*, *General* the box *Use only certified plug-ins* is unchecked. The plugins are not loaded if Acrobat runs in Certified Mode.
- Some PDF forms created with Adobe Designer or Adobe Experience Manager may prevent the Block Plugin as well as other Acrobat plugins from working properly since they interfere with Acrobat'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.

13.2 Overview of the Block Concept

13.2.1 Separation of Document Design and Program Code

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

- The designer creates the page layout and specifies the location of variable page 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.
- The 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. PPS 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 can move the Block with name of the addressee in a mailing to a different location on the page, or change the font size. 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.

As an intermediate step Block filling can be previewed in Acrobat to accelerate the development and test cycles. Block previews are based on default data (e.g. a string or an image file name) which is specified in the Block definitions.

13.2.2 Block Properties

The behavior of Blocks can be controlled with Block properties. Properties are assigned to a Block with the Block Plugin.

Predefined Block properties. 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 by PPS. The name is an arbitrary string which identifies the Block, such as *firstname*, *lastname*, or *zipcode*. PPS supports different kinds of Blocks:

- *Textline Blocks* hold a single line of textual data which will be processed with the Textline output method in PPS.
- *Textflow Blocks* hold one or more lines of textual data. Multi-line text will be formatted with the Textflow formatter in PPS. Textflow Blocks can be linked so that one Block holds the overflow text of the previous Block (see »Linking Textflow Blocks«, page 379).
- *Image Blocks* hold a raster image. This is similar to placing a TIFF or JPEG file in a DTP application.
- *PDF Blocks* hold arbitrary PDF contents imported from a page in another PDF document. This is similar to placing a PDF page in a DTP application.
- *Graphics Blocks* hold vector graphics. This is similar to placing an SVG file in a layout application.

Blocks can carry a number of predefined properties depending on their type. Properties can be created and modified with the Block Plugin (see Section 13.3.2, »Editing Block Properties«, page 364). A full list of predefined Block properties can be found in Section 13.7, »Block Properties«, page 382. For example, a text Block can specify the font and size of the text, an image or PDF Block can specify the scaling factor or rotation PPS offers dedicated functions for processing the Block types, e.g. *PDF_fill_textblock()*. These functions search a placed PDF page for a Block by its name, analyze its properties, and place client-supplied data (single- or multi-line text, raster image, PDF page, or vector graphics) on the new page according to the specified Block properties. The programmer can override Block properties by specifying the corresponding options in the Block filling functions.

Properties for default contents. Special Block properties can be defined which hold the default contents of a Block, i.e. the text, image, PDF, or graphics contents that will be placed in the Block if no variable data is supplied to the Block filling functions, or in situations where the Block contents are currently constant, but may change in the next print run.

Default properties are also used by the Preview feature of the Block Plugin (see Section 13.5, »Previewing Blocks in Acrobat«, page 372).

Custom Block properties. Predefined Block properties make it possible to quickly implement variable data processing applications, but they are restricted to the set of properties which are internally known to PPS and can automatically be processed. In order to provide more flexibility, the designer can also assign custom properties to a Block. These can be used to extend the Block concept in order to match the requirements of more advanced variable data processing applications.

There are no rules for custom properties since PPS will not process custom properties in any way, except making them available to the client. The client code can retrieve the value of custom properties and process it as appropriate. Based on a custom property of a Block the application 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.

13.2.3 Why not use PDF Form Fields?

Experienced Acrobat users may ask why we implemented a new Block concept instead of relying on the existing form field mechanism 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 combine PDF forms and PDFlib Blocks with the form field conversion plugin (see Section 13.4, »Converting PDF Form Fields to PDFlib Blocks«, page 369).

Although there are many parallels between both concepts, PDFlib Blocks offer several advantages over PDF form fields as detailed in Table 13.1.

Table 13.1 Comparison of PDF form fields and PDFlib Blocks

feature	*PDF form fields*	*PDFlib Blocks*
design objective	*for interactive use*	*for automated filling*
typographic features (beyond choice of font and font size)	*–*	*kerning, word and character spacing, underline/ overline/strikeout*
OpenType layout features	*–*	*dozens of OpenType layout features, e.g. ligatures, swash characters, oldstyle figures*
complex script support	*limited*	*shaping and bidirectional formatting, e.g. for Arabic and Devanagari*
font control	*font embedding*	*font embedding and subsetting, encoding*
text formatting controls	*left-, center-, right-aligned*	*left-, center-, right-aligned, justified; various formatting algorithms and controls; inline options can be used to control the appearance of text*
change font or other text attributes within text	*–*	*yes*
merged result is integral part of PDF page description	*–*	*yes*
users can edit merged field contents	*yes*	*no*
extensible set of properties	*–*	*yes (custom Block properties)*
use image files for filling	*–*	*BMP, CCITT, GIF, PNG, JPEG, JBIG2, JPEG 2000, TIFF*
use vector graphics for filling	*–*	*SVG*
color support	*RGB*	*grayscale, RGB, CMYK, Lab, spot color (HKS and Pantone spot colors integrated in the Block Plugin), DeviceN*
PDF standards	*–*	*PDF/A, PDF/X, PDF/VT, PDF/UA*
graphics and text properties can be overridden upon filling	*–*	*yes*
transparent contents	*–*	*yes*
Text Blocks can be linked	*–*	*yes*

13.3 Editing Blocks with the Block Plugin

13.3.1 Creating Blocks

Activating the Block tool. The 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 the following ways:

- By clicking the Block icon  which you can locate as follows in Acrobat DC: click *Tools, Advanced Editing*.
- Via the menu item *PDFlib Blocks, PDFlib Block Tool*.

Creating and modifying Blocks. When the Block tool is active you can drag the crosshair pointer to create a Block at the desired position on the page and with the desired size. Blocks are always rectangular with edges parallel to the page edges (use the *rotate* property for Block contents which are not parallel to the page edges). After dragging a Block rectangle the Block properties dialog appears where you can edit the properties of the Block (see Section 13.3.2, »Editing Block Properties«, page 364). The Block tool automatically creates a Block name which can be changed in the properties dialog. Block names must be unique on a page, but can be repeated on another page.

You can change the Block type in the top area to one of *Textline*, *Textflow*, *Image*, *PDF*, or *Graphics*. Different colors are used for representing the Block types (see Figure 13.1). The *Block Properties* dialog hierarchically organizes the properties in groups and subgroups depending on the Block type.

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.*

Selecting Blocks. Several Block operations, such as copying, moving, deleting, or editing Properties, work with one or more selected Blocks. You can select Blocks with the Block tool as follows:

- To select a single Block simply click on it.
- To select multiple Blocks hold down the Shift key while clicking on the second and subsequent Block.
- Press Ctrl-A (on Windows) or Cmd-A (on macOS) 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 macOS. 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.

If you right-click (or Ctrl-click on macOS) an area on the page where no Block is located the context menu contains entries for creating a Block Preview and for configuring the Preview feature.

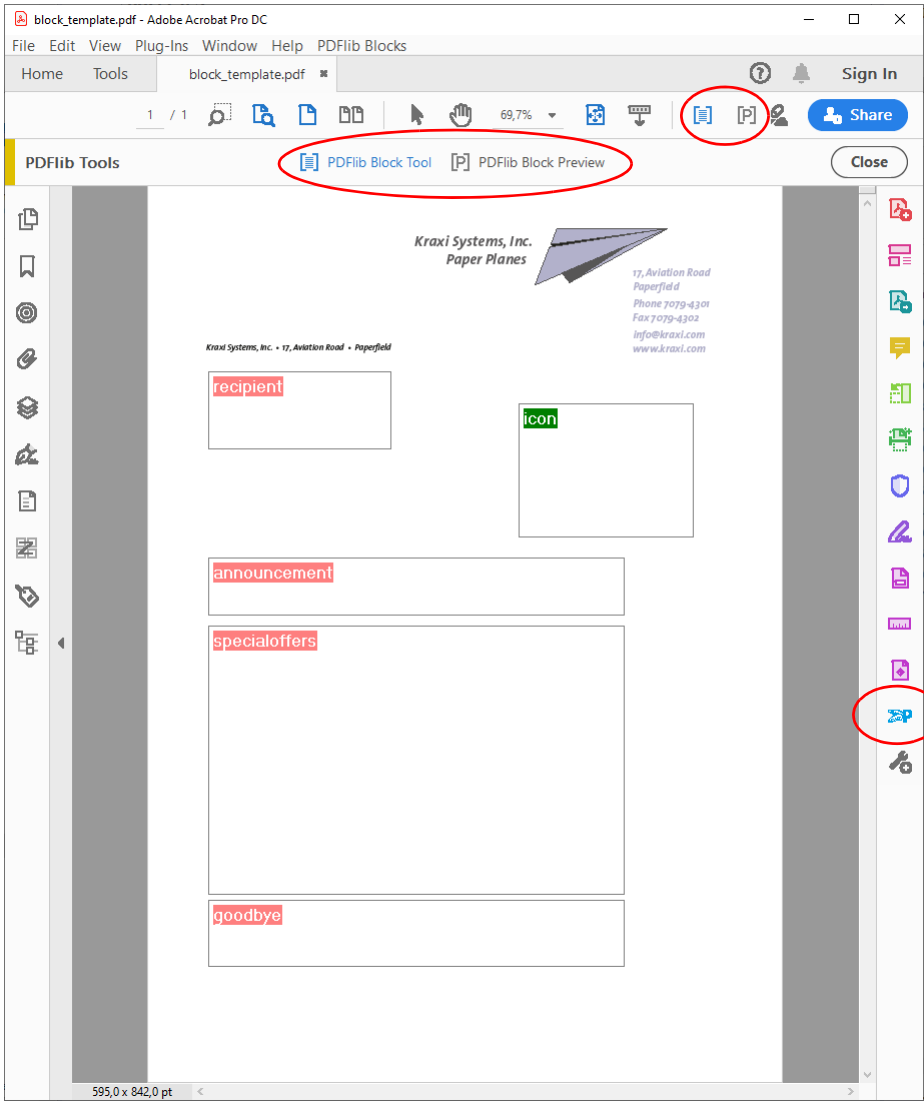

Fig. 13.1
Visualization of Blocks

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 a double 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 in small increments by using the arrow keys.

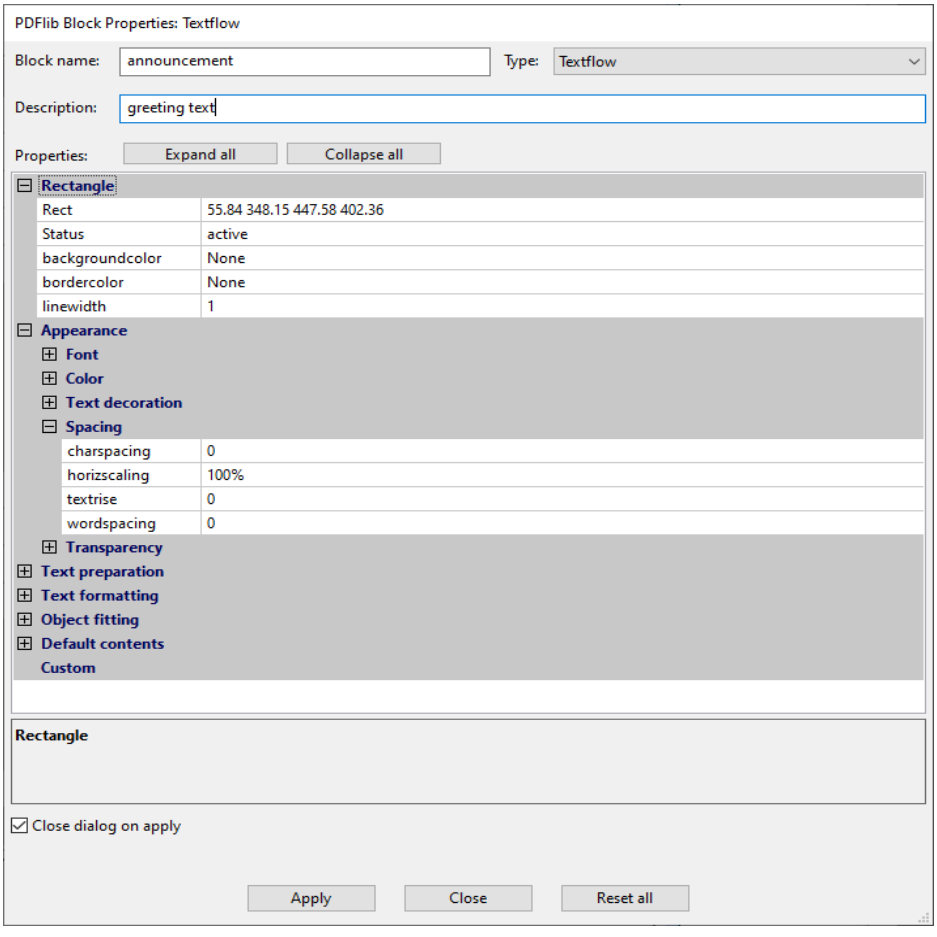

Fig. 13.2
The Block properties dialog

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 in Acrobat DC proceed as follows: go to *Edit, Preferences, [General...], Units & Guides, Page & Ruler Units* and choose one of Points, Inches, Millimeters, Picas, Centimeters.
- To display cursor coordinates use *View, Show/Hide, Cursor Coordinates*.

Note that the selected unit will only affect the *Rect* property, but not any other numerical properties (e.g. *fontsize)*.

Using a grid to position Blocks. You can take advantage of Acrobat's grid feature for precisely positioning and resizing Blocks:

- Display the grid: *View, Show/Hide, Rulers & Grids, Grid*;
- Enable grid snapping: *View, Show/Hide, Rulers & Grids, Snap to Grid*;
- Change the grid: go to *Edit, Preferences, [General...], Units & Guides*. Here you can change the spacing and position of the grid as well as the color of the grid lines.

If *Snap to Grid* is enabled the size and position of Blocks will be aligned with the configured grid. *Snap to Grid* affects newly generated Blocks as well as existing Blocks which are moved or resized with the Block tool.

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 same size and location as 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 aid.

The *Click Object* feature tries 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, not only individual lines.

Automatically detect font properties. The Block Plugin can analyze the underlying font which is present at the location where a Textline or Textflow 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 if 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. 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 edit its properties.

13.3.2 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 13.2). As detailed in Section 13.7, »Block Properties«, page 382, there are several groups of properties available, subject to the Block type. The *Apply* button will only be enabled if you changed one or more properties in the dialog. The *Apply* button will be inactive for locked Blocks.

Note *Some properties may be inactive depending on the Block type and certain property settings. For example, the property subgroup* `Ruler tabs for hortabmethod=ruler` *where you can edit tabulator settings is enabled only if the* `hortabmethod` *property in the group* `Text formatting`, `Tabs` *is set to* `ruler`.

Note If you enter text for a Block property you may experience character replacements, e.g. straight quotes are replaced by smart quotes. This substitution is done by the operation system and can be disabled via »System Preferences«, »Keyboard«, »Text«, »Use smart quotes and dashes«.

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. *fitmethod, orientate),* or select a font, color value or file name by clicking the »...« button at the right-hand side of the dialog (e.g. *backgroundcolor, defaultimage)*. 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 PPS.

Modified properties will in be displayed in bold face in the Block Properties dialog. If any of the properties in a Block has been modified, the suffix *(*)* will be appended to the displayed Block name. When you are done editing properties click the *Apply* button to update the Block. 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 will always select the topmost Block. In this situation the *Select 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 this way, the next action within its area will not affect other Blocks, but only the selected one. For example, press *Enter* to edit the selected Block's properties. This way Block properties can easily be edited even for Blocks which are partially or completely covered by other Blocks.

Using and restoring repeated values of Block properties. In order to save some amount of typing and clicking, the Block tool remembers 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 resets most Block properties to their respective default values. The following items remain unmodified:

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

Note Do not confuse the default values of predefined Block properties with the `defaulttext`, `defaultimage`, `defaultpdf`, *and* `defaultgraphics` *properties which hold placeholder data for generating previews (see »Default Block contents«, page 372).*

Editing multiple Blocks at once. Editing the properties of multiple Blocks at once is a big time saver. You can select multiple Blocks as follows:

- Activate the Block tool via the menu item *PDFlib Blocks, PDFlib Block Tool.*
- Click on the first Block to select it. The first selected Block is the master Block. Shift-click other Blocks to add them to the set of selected Blocks. Alternatively, click *Edit, Select All* to select all Blocks on the current page.
- Double-click on any of the Blocks to open the Block Properties dialog. The Block where you double-click will be the new master Block.
- Alternatively, you can click on a single Block to designate it as master Block, and then press the *Enter* key to open the Block Properties dialog.

The Properties dialog displays only the subset of properties which apply to all selected Blocks. The dialog is populated with property values taken from the master Block. Now you can apply properties to all selected Blocks as follows:

- If the checkbox *Apply all properties of the master Block* is unchecked: upon clicking *Apply* only the properties changed manually in the dialog (highlighted in black) are copied to all selected Blocks.
- If the checkbox *Apply all properties of the master Block* is checked: upon pressing *Apply* all current properties of the master Block as well as all properties changed manually in the dialog are copied to all selected Blocks. This can be used to copy all properties from a particular Block to one or more other Blocks.

The following predefined properties as well as custom properties can not be shared, i.e. they can not be edited for multiple Blocks at once:

```
Name, Description, Subtype, Type, Rect, Status
```

13.3.3 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 between 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 using standard copy/paste operations
- export Blocks to a new file (with empty pages) or to an existing document (apply the Blocks to existing pages)
- import Blocks from another document

In order to update the page contents while maintaining Block definitions you can replace the underlying page(s) while keeping the Blocks. Use *Tools, Organize Pages, Replace* for this purpose.

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 macOS). The mouse cursor will change while this key is pressed. A copied Block has the same properties as the original Block, with the exception of its name and position which will automatically be adjusted in the new Block.

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 macOS) or *Edit, Copy* to copy the selected Blocks to the clipboard.
- Navigate to the target page (if necessary).
- Make sure that the Block Tool is active, and use Ctrl-V (on Windows) or Cmd-V (on macOS) or *Edit, Paste* to paste the Blocks from the clipboard to the current page and document.

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 Blocks on this Page)* and the range of target pages to which you want to duplicate the selected 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 of the new PDF with 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 *Repeate 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.

13.3.4 Customizing the Block Plugin User Interface with XML

Some aspects of the Block Plugin user interface are stored/reloaded upon each Acrobat session, and can be controlled via an XML configuration file. A sample configuration file *factory settings.xml* is included in the distribution. If the configuration has been modified the new settings are stored in *user settings.xml*. The modified configuration will be loaded every time Acrobat is started and written when Acrobat is closed. The configuration file is stored in a location similar to the following (the names of system directories may be localized; replace *DC* with another Acrobat track name as appropriate):

```
Windows: C:\Users\<user>\AppData\Local\Adobe\Acrobat\DC\PDFlib\Block Plugin 5
macOS:   /Users\<user>/Library/Application Support/Adobe/Acrobat/DC/PDFlib/Block Plugin 5
```

The following XML elements can be used to modify the configuration manually:

- The element */Block_Plugin/MainDialog/CloseOnApply* controls the initial status of the *Close dialog on apply* checkbox in the Block properties dialog. This checkbox determines whether the Block Properties dialog will be kept open after creating a Block or modifying Block properties.
- The element */Block_Plugin/MainDialog/ApplyAllProps* controls the initial status of the *Apply all properties of the mastesr Block* checkbox in the Block properties dialog. This checkbox determines whether all properties of the master Block are copied to multiple selected Blocks or only those properties which have been modified in the dialog.

- The element */Block_Plugin/FontDialog/ShowBaseFonts* controls whether the base 14 fonts will be displayed in the font list of the Block Properties dialog (property group *Appearance*, property *fontname)* in addition to the fonts installed on the system.
- The element */Block_Plugin/Command/ControlByClick* controls the initial status of the menu item *PDFlib Blocks, Click object to define Block*.
- The element */Block_Plugin/Command/DetectFonts* controls the initial status of the menu item *PDFlib Blocks, Detect underlying font and color*.
- The element */Block_Plugin/Command/KeyAccelerator* with the possible values *control* (which designates the Ctrl key on Windows and the Command key on macOS), *shift* for the Shift key, *control+shift* or *none* specifies the accelerator key for the following keyboard shortcuts:

  ```
  A (select all), C (copy), I (Block Properties dialog), V (paste), X (cut)
  ```

  The change will be effective upon the next start of Acrobat since keyboard shortcuts cannot be changed at runtime. If this entry is absent, no accelerators are available. The default is *control*.
- The element *Configuration/Preferences/PreviewStatusMessage* controls whether a status message dialog *(e.g. »10 Block(s) processed: ...«)* is shown after each Preview operation.

13.4 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 complex PDF forms which you want to fill automatically with PPS 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 *Select Object* Tool via *Tools, Rich Media)* to select one or more 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 *Textline* or *Textflow*. 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 13.3.

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 369).

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 of 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 XFDF 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 records details about the original form field as custom properties when creating the corresponding Block. Table 13.2 lists the custom properties which can be used to reliably identify the Blocks; all properties have type *string*.

Table 13.2 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*

Table 13.3 Conversion of PDF form fields to PDFlib Blocks

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

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 plugin 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 Acrobat's menu functions or tools until the conversion is finished.

13.5 Previewing Blocks in Acrobat

Note You can try the Preview feature with the `block_template.pdf` *document in the PDFlib distribution. The required resources (e.g. font and image) are also included in the PDFlib distribution.*

PDFlib Blocks will be processed by PPS where the Block filling process can be customized regarding the data sources (e.g. text from a database, image files on disk) as well as visual and interactive aspects of the generated documents. This process is detailed in Section 13.6, »Filling Blocks with PPS«, page 377.

However, the Block Plugin contains an integrated version of PPS which can be used to generate Preview versions of the filled Blocks interactively in Acrobat without any programming. Although this Preview feature cannot offer the same flexibility as custom programming, it provides a quick overview of Block filling results. The Block Preview can be used for improving the position and size of Blocks as well as for checking the Block properties (e.g. font name and size). You can change the Blocks and create a new Preview until you are satisfied with the results shown in the Preview. Previews can be generated for the current page or the whole document.

The Preview will always be shown in a new PDF document. The original document (which contains the Blocks) will not be modified by generating a Preview. You can save or discard the generated Preview documents according to your requirements.

Default Block contents. Since the server-side data sources (e.g. a database) for the text, image, vector graphics or PDF contents of a Block are not available in the Plugin, the Preview feature always uses the Block's default contents, i.e. the values specified in the *defaulttext*, *defaultimage*, *defaultpdf*, or *defaultgraphics* properties. Usually, a sample data set will be used as default data which is representative for the real Block contents used with PPS. Blocks without any default contents are ignored when generating the Preview, as well as Blocks with *Status=ignoredefault*.

The default properties are empty for new Blocks. Before using the Preview feature you must fill the *defaulttext*, *defaultimage*, *defaultpdf*, or *defaultgraphics* properties (depending on the Block type) in the *Default contents* property group, or supply suitable values for the options of the same name in the *Advanced PPS options...* dialog.

Note Entering default text for symbolic fonts can be a bit tricky; see »Using symbolic fonts for default text«, page 375, for details.

Generating Block Previews. You can create Block Previews with one of the following methods:

- By clicking the PDFlib Block Preview icon  which you can locate as follows in Acrobat DC: click *Tools, Advanced Editing*.
- Via the menu item *PDFlib Blocks, Preview, Generate Preview*.
- If the Block tool is active you can right-click outside of any Block to bring up a context menu with the entries *Generate Preview* and *Preview Configuration*.

The Previews will be created based on the PDF file on disk. Any changes that you may have applied in Acrobat will only be reflected in the Preview if the Block PDF has been saved to disk using *File, Save* or *File, Save As...* . You can identify modified Blocks by the asterisk after the Block name. The Preview feature can be configured to save the Block PDF automatically before creating a Preview. This way you can make sure that interactive changes will immediately be reflected in the Preview.

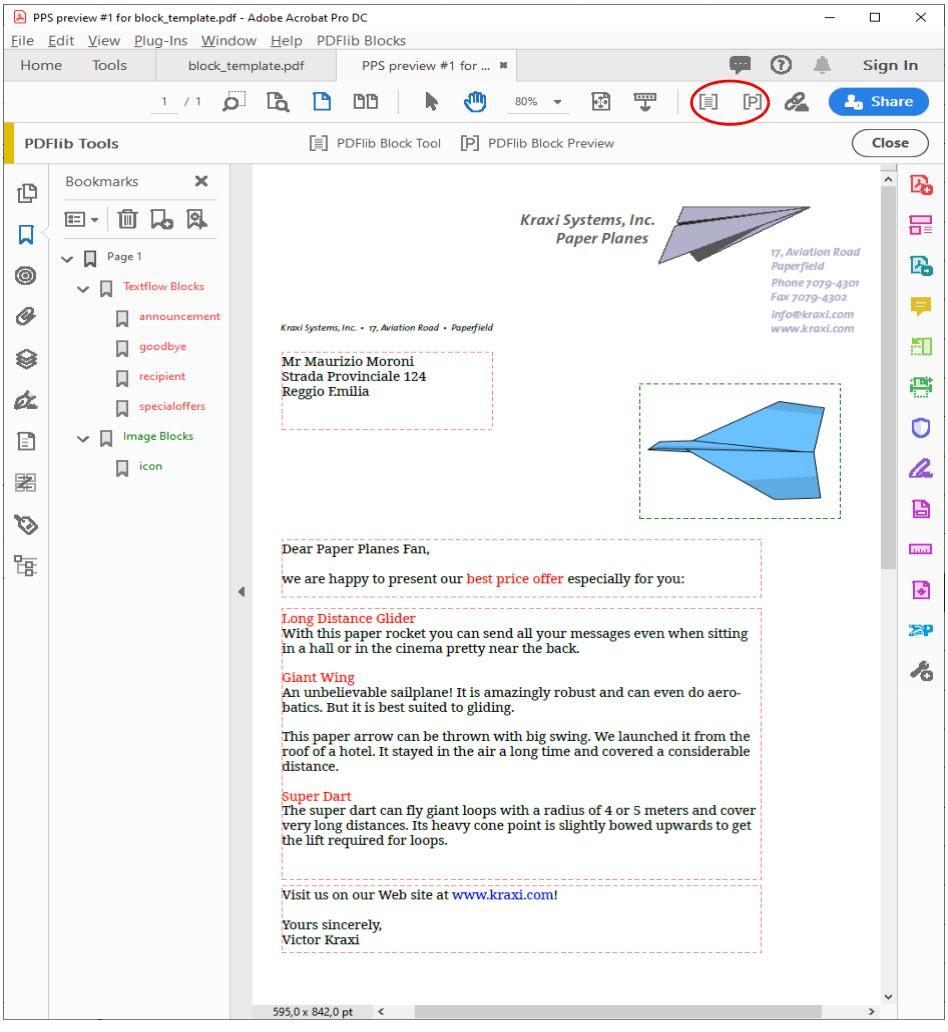

Fig. 13.3 Preview PDF for the document shown in Figure 13.1. It contains Block info layers and annotations

Configuring the Preview. Several aspects of Block Preview creation and the underlying PPS operation can be configured via *PDFlib Blocks, Preview, Preview Configuration...:*

- Preview for the current page or the full document;
- Output directory for the generated Preview documents;
- Automatically save the Block PDF before creating the Preview;
- Add Block info layers and annotations;
- Copy Blocks to the generated output;
- *Clone PDF/A, PDF/UA or PDF/X status of Block PDF:* since these standards restrict the use of layers and annotations the *Block info layers and annotations* option is mutually exclusive with this option.
- *Copy Blocks to Preview File* allows you copy the PDFlib Blocks to the generated Preview upon filling. All Blocks will be copied, regardless of whether or not they could successfully be filled.

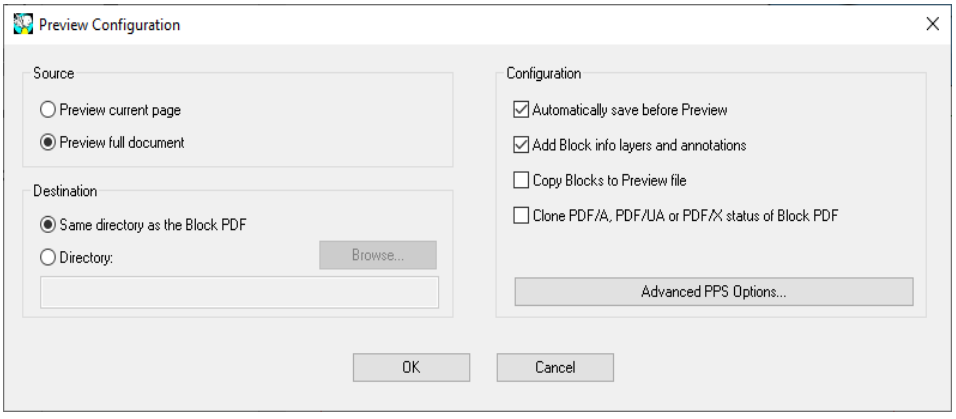

Fig. 13.4 Block Preview configuration

- The *Advanced PPS options* dialog can be used to specify additional option lists for PPS functions according to the PPS API. For example, the *searchpath* option for *PDF_set_option( )* can be used to specify a directory where fonts or images for Block filling are located. It is recommended to specify advanced options in cooperation with the developer of the PPS code.

Block ordering. When the document is stored with »Save as...« in Acrobat the Blocks are sorted in alphabetical order according to the Block name. This is also the ordering in which Blocks are processed by Preview and reported by pCOS. However, applications will typically fill Blocks based on their name (as opposed to the storage order on file), so the ordering in the PDF document is usually not relevant.

Information provided with the Preview. The generated Preview documents contain the original page contents (the background), the filled Blocks, and optionally various other pieces of information. This information can be useful for checking and improving Blocks and PPS configuration. The following items will be created for each active Block with default contents:

- *Error markers:* Blocks which could not be filled successfully are visualized by a crossed-out rectangle so that they can easily be identified. Error markers will always be created if a Block couldn't be processed.
- *Bookmarks:* The processed Blocks will be summarized in bookmarks which are structured according to the page number, the Block type, and possible errors. Bookmarks can be displayed via *View, Show/Hide, Navigation Panes, Bookmarks* (Acrobat X/XI/DC). Bookmarks will always be created.
- *Annotations:* For each processed Block an annotation will be created on the page in addition to the actual Block contents. The annotation rectangle visualizes the original Block boundary (depending on the default contents and filling mode this may differ from the boundary of the Block contents). The annotation contains the name of the Block and an error message if the Block couldn't be filled. Annotations are generated by default, but can be disabled in the Preview configuration. Since the use of annotations is restricted in the PDF/A and PDF/X standards, annotations are not created if the *Clone PDF/A, PDF/UA or PDF/X status of Block PDF* option is enabled.

- *Layers:* The page contents will be placed on layers to facilitate analysis and debugging. A separate layer will be created for the page background (i.e. the contents of the original page), each Block type, error Blocks which couldn't be filled, and the annotations with Block information. If a layer remains empty (e.g. no errors occurred) it will not be created. The layer list can be displayed via *View, Navigation Panels, Layers*. By default, all layers on the page will be displayed. In order to hide the contents of a layer click on the eye symbol to the left of the layer name. Layer creation can be disabled in the Preview configuration. Since the use of layers is restricted in the standards PDF/A-1 and PDF/X-3, layers are not created if the *Clone PDF/A, PDF/UA or PDF/X status of Block PDF* option is enabled.

Cloning PDF/A, PDF/UA or PDF/X status. The *Clone PDF/A , PDF/UA or PDF/X status of Block PDF* configuration is useful when PDF output according to one of these standards must be created. Clone mode can be enabled if the input conforms to one of the following standards:

```
PDF/A-1a:2005, PDF/A-1b:2005
PDF/A-2a, PDF/A-2b, PDF/A-2u
PDF/A-3a, PDF/A-3b, PDF/A-3u

PDF/UA-1

PDF/X-3:2003
PDF/X-4, PDF/X-4p
PDF/X-5n
```

When Previews are created in clone mode, PPS duplicates the following aspects of the Block PDF in the generated Preview:

- the PDF standard identification;
- output intent condition;
- page sizes including all page boxes;
- Tagged PDF: document language (if present);
- XMP document metadata.

When cloning standard-conforming PDF documents all Block filling operations must conform to the respective standard. For example, if no output intent is present RGB images without ICC profile can not be used. Similarly, all used fonts must be embedded. The full list of requirements can be found in Section 12.3, »PDF/A for Archiving«, page 319, and Section 12.4, »PDF/X for Print Production«, page 331. If a Block filling operation in PDF/A or PDF/X cloning mode would violate the selected standard (e.g. because a default image uses RGB color space, but the document does not contain a suitable output intent) an error message pops up and no Preview will be generated. This way users can catch potential standard violations very early in the workflow.

Using symbolic fonts for default text. Two methods are available to supply default text for Blocks with symbolic fonts:

- Working with 8-bit legacy codes, e.g. as shown in the Windows character map application: supply the 8-bit codes for the *defaulttext* either by entering the corresponding 8-bit character literally (e.g. by copy/pasting from the Windows character map) or as a numerical escape sequence. In this case you must keep the default value of the *charref* property in the *Text preparation* property group as *false* and can not work

with character references. For example, the following default text will produce the »smiley« glyph from the symbolic Wingdings font if *charref=false:*

```
J
\x4A
\112
```

- Working with the Unicode values or glyph names used in the font: set the *charref* property in the *Text preparation* property group to *true* and supply character references or glyph name references for the symbols (see Section 5.6.2, »Character References«, page 119). For example, the following default text will produce the »smiley« glyph from the symbolic Wingdings font if *charref=true:*

```
&#xF04A;
&.smileface;
```

Keep in mind that with both methods an alternate representation will be visible instead of the actual symbolic glyphs in the Block properties dialog.

13.6 Filling Blocks with PPS

In order to fill Blocks with PPS you must first place the page containing the Blocks on the output page with the *PDF_fit_pdi_page()* function. After placing the page its Blocks can be filled with the *PDF_fill_*block()* functions.

Simple example: add variable text to a template. Adding dynamic text to a PDF template is a very common task. The following code fragment opens a page in an input PDF document (the template or Block container), places it on the output page, and fills 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, "");
/* Place the imported page */
p.fit_pdi_page(page, 0.0, 0.0, "");

/* Fill a single Block on the placed page */
p.fill_textblock(page, "firstname", "Serge", "encoding=winansi");

p.close_pdi_page(page);
p.end_page_ext("");
p.close_pdi_document(doc);
```

Cookbook *A full code sample can be found in the Cookbook topic* `blocks/starter_block`.

Overriding Block properties. In certain situations the programmer wants 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:

- Business logic may decide to enforce certain overrides.
- The scaling factor for an image or PDF page will be calculated by the application 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 customer requirements in a print shop application.

Property overrides can be achieved by supplying property names and the corresponding values in the option list of the *PDF_fill_*block()* functions, e.g.

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

This will override the Block's internal *fontsize* property with the supplied value 12. Almost all property names can be used as options.

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

Moving Textflow Blocks while filling. The fixed size of a Textflow Block may not match its varying textual contents. If there is only few text a gap between two Blocks may arise; if there is too much text it may not fit into the Block rectangle. In this situation you can query the results of Textflow fitting to adjust the position of the next Block:

- The default *fitmethod* is *auto*, i.e. the text is forced to fit into the Block rectangle. To allow excess text to overflow the Block you must set the *fitmethod* to *nofit*. This can be specified in the Block properties in the Block template at design time or by supplying the *fitmethod* option to *PDF_fill_textblock( )*.
- Supply the dummy option *textflowhandle=-1* (in PHP: *textflowhandle=0)* to *PDF_fill_textblock( )* so that this method returns a Textflow handle for the contents of the Block.
- The returned Textflow handle is supplied to *PDF_info_textflow( )* to query the end position of the text using the keyword *textendy*.
- Query the Block's lower vertical position with *PDF_pcos_get_number( )* and the pCOS path *pages[..]/blocks/<blockname>/Rect[1]*.
- Calculate the difference between both values. If this offset is positive the Textflow didn't completely fill the Block; if it is negative the Textflow overflowed the Block. In both cases you can move the next Block up or down by this offset. This can be achieved with the *refpoint* option of *PDF_fill_textblock( )* which overrides the *Rect* property. Since this option requires absolute coordinates you must query the vertical position of the Block (see previous step) and supply the sum of the original position and the offset to the *refpoint* option.
- You can apply this method to an arbitrary number of Textflow Blocks by accumulating the per-Block offsets. Depending on the contents each Block will move successive Blocks up or down by an appropriate amount.

Cookbook A full code sample can be found in the `starter_block` *sample.*

Placing the imported page on top of the filled 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 by placing the page once with the *blind* option of *PDF_fit_pdi_page( )* in order to make its Blocks and their position known to PPS, and place it again after filling the Blocks in order to actually show the page contents:

```
/* 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");

/* Fill the Blocks */
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, "");
```

Cookbook A full code sample can be found in the Cookbook topic `blocks/block_below_contents`.

Ignoring the container page when filling Blocks. Imported Blocks can also be useful as placeholders without any reference to the underlying contents of the Block's page. You

can place a container page with Blocks in blind mode on one or more pages, i.e. with the *blind* option of *PDF_fit_pdi_page( )*, and subsequently fill its Blocks. This way you can take advantage of the Block and its properties without placing the container page on the output page, and can duplicate Blocks on multiple pages (or even on the same output page).

Cookbook *A full code sample can be found in the Cookbook topic* `blocks/duplicate_block`.

Linking Textflow Blocks. Textflow Blocks can be linked so that one Block holds the overflow text of 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 remaining text of the first Block into the second Block.

PPS internally creates a Textflow from the text provided to *PDF_fill_textblock( )* and the Block properties. For unlinked Blocks this Textflow is placed in the Block and the corresponding Textflow handle is deleted at the end of the call; overflow text is lost in this case.

With linked Textflow Blocks the overflow text of the first Block can be filled into the next Block. The remainder of the first Textflow is used as Block contents instead of creating a new Textflow. Linking Textflow Blocks works as follows:

- In the first call to *PDF_fill_textblock( )* within a chain of linked Textflow Blocks the value -1 (in PHP: 0) must be supplied for the *textflowhandle* option. The Textflow handle created internally is returned by *PDF_fill_textblock( )*, and must be stored by the application.
- In the next call to *PDF_fill_textblock( )* the Textflow handle returned in the previous step can be supplied to the *textflowhandle* option (the text supplied in the *text* parameter is ignored in this case, and should be empty). The Block is filled with the remainder of the Textflow.
- This process can be repeated with more Textflow Blocks.
- The returned Textflow handle can be supplied to *PDF_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 fragment 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;
        int reason;
        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_string(reason, "");
```

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

Cookbook *A full code sample can be found in the Cookbook topic* `blocks/linked_textblocks`.

Block filling order. The Block functions *PDF_fill_*block()* process properties and Block contents in the following order:

- Background: if the *backgroundcolor* property is present and contains a color space keyword different from *None*, the Block area will be filled with the specified color.
- Border: if the *bordercolor* property is present and contains a color space keyword different from *None*, the Block border will be stroked with the specified color and linewidth.
- Contents: the supplied Block contents and all other properties except *bordercolor* and *linewidth* will be processed.
- Textline and Textflow 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.

Nested Blocks. Before Blocks can be filled the page containing the Blocks must have been placed on the output page before (since otherwise PPS wouldn't know the location of the Blocks after scaling, rotating, and translating the page). If the page only serves as a Block container without bringing static content to the new page you can place the imported page with the *blind* option.

For successful Block filling it doesn't matter how the imported page was placed on the output page:

- The page can be placed directly with *PDF_fit_pdi_page()*.
- The page can be placed indirectly in a table cell with *PDF_fit_table()*.
- The page can be placed as contents of a another PDF Block with *PDF_fill_pdfblock()*.

The third method, i.e. filling a PDF Block with another page containing Blocks, allows nested Block containers. This allows simple implementations of interesting use cases. For example, you can implement both imposition and personalization with a two-step Block filling process:

- The first-level Block container page contains several large PDF Blocks which indicate the major areas on the paper to be printed on. The arrangement of PDF Blocks reflects the intended post-processing of the paper (e.g. folding or cutting).
- Each of the first-level PDF Blocks is then filled with a second-level container PDF page which contains Text, Image, PDF, or Graphics Blocks to be filled with variable text for personalization.

With this method Block containers can be nested. Although Block nesting works to an arbitrary level, a nesting level of three or more will only rarely be required.

The second-level Block containers (e.g. a template page for a letter) may be identical or different for each imposed page. If they are identical the Blocks on the letter template must be filled before placing the letter template itself in the next first-level Block since PPS always uses the location of the most recent placement of the template page.

Cookbook A full code sample can be found in the Cookbook topic `blocks/nested_blocks`.

Block coordinates. The Rectangle coordinates of a Block refer to the PDF default coordinate system. When the page containing the Block is placed on the output page with PPS, several positioning and scaling options can be supplied to *PDF_fit_pdi_page( )*. These options 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.

The Block Plugin displays 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. The coordinate display in the Properties dialog is also subject to the units specified in Acrobat (see »Block size and position«, page 362).

Spot colors in Block properties. 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 PPS 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 PPS application using *PDF_makespotcolor( )* or a suitable color option list. Otherwise Block filling will fail.

13.7 Block Properties

PPS and the Block Plugin support general properties which can be assigned to any type of Block. In addition there are properties which are specific to the Block types *Textline*, *Textflow*, *Image*, *PDF*, and *Graphics*.

Properties support the same data types as option lists except handles and action lists. The names of Block properties are generally identical to options for API functions such as *PDF_fit_textline( )*, *PDF_fit_image( )* (e.g., *fitmethod*, *charspacing*). In these cases the behavior is exactly the same as the one documented for the respective option.

13.7.1 Administrative Properties

Administrative properties apply to all Block types. Required entries will automatically be generated by the Block Plugin. Table 13.4 lists the administrative Block properties.

Table 13.4 Administrative properties

keyword	***possible values and explanation***
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 by PPS.*
Locked	*(Boolean) If* `true`*, the Block and its properties can not be edited with the Block Plugin. This property will be ignored by PPS. Default:* `false`
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.*
Subtype	*(Keyword; required) Depending on the Block type, one of* `Text`, `Image`, `PDF`, *or* `Graphics`. *Note that Textline and Textflow Blocks both have Subtype* `Text`*, but are distinguished by the* `textflow` *property.*
textflow	*(Boolean) Controls single- or multiline processing. This property is not available explicitly in the user interface of the Block Plugin, but will be mapped to Textline or Textflow Blocks, respectively (Default:* `false`*):* ***false*** *Textline Block: text spans a single line and will be processed with PDF_fit_textline( ).* ***true*** *Textflow Block: text can span multiple lines and will be processed with PDF_fit_textflow( ). In addition to the standard text properties Textflow-related properties can be specified (see Table 13.9).*
Type	*(Keyword; required) Always* `Block`

13.7.2 Rectangle Properties

Rectangle properties apply to all Block types. They describe the appearance of the Block rectangle itself. Required entries will automatically be generated by the Block Plugin. Table 13.5 lists the rectangle properties.

Table 13.5 Rectangle properties

keyword	***possible values and explanation***
background-color	*(Color) 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) 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; must be greater than 0) Stroke width of the line used to draw the Block rectangle; only used if* `bordercolor` *is set. Default: 1*
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 displays 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 by PPS and the Preview feature (default:* `active`*):* ***active*** *The Block will be fully processed according to its properties.* ***ignore*** *The Block will be ignored.* ***ignoredefault*** *Like* `active`*, except that the* `defaulttext/image/pdf/graphics` *properties and options are ignored, i.e. the Block remains empty if no variable contents are available (especially in the Preview). This may be useful to make sure that the Block's default contents are not used for filling Blocks on the server side although the Block may contain default contents for generating Previews. It can also be used to disable the default contents for previewing a Block without removing the default contents from the Block properties.* ***static*** *No variable contents will be placed; instead, the Block's default text, image, PDF, or graphics contents will be used if available.*

13.7.3 Appearance Properties

Appearance properties specify formatting details:

- Table 13.6 lists transparency appearance properties for all Block types.
- Table 13.7 lists text appearance properties for Textline and Textflow Blocks.

Table 13.6 Transparency appearance properties for all Block types

keyword	***possible values and explanation***
blendmode	*(Keyword list; if used in PDF/A-1 mode it must have the value* `Normal`*) Name of the blend mode:* `None`, `Color`, `ColorDodge`, `ColorBurn`, `Darken`, `Difference`, `Exclusion`, `HardLight`, `Hue`, `Lighten`, `Luminosity`, `Multiply`, `None`, `Normal`, `Overlay`, `Saturation`, `Screen`, `SoftLight`. *Default:* `None`
opacityfill	*(Float; if used in PDF/A mode it must have the value 1) Opacity for fill operations in the range 0..1. The value 0 means fully transparent; 1 means fully opaque.*
opacitystroke	*(Float; if used in PDF/A mode it must have the value 1) Opacity for stroke operations in the range 0..1. The value 0 means fully transparent; 1 means fully opaque.*

Table 13.7 Text appearance properties for Textline and Textflow Blocks

keyword	possible values and explanation
charspacing	*(Float or percentage) Character spacing. Percentages are based on fontsize. Default: 0*
decoration-above	*(Boolean) If* `true`*, the text decoration enabled with the* `underline`*,* `strikeout`*, and* `overline` *options will be drawn above the text, otherwise below the text. Changing the drawing order affects visibility of the decoration lines. Default:* `false`
fillcolor	*(Color) Fill color of the text. Default:* `gray 0` *(=black)*
fontname[1]	*(String) Name of the font as required by PDF_load_font( ). The Block plugin will present a list of fonts available in the system. However, these font names may not be portable between macOS, Windows, and Unix systems. If* `fontname` *starts with an '@' character the font will be applied in vertical writing mode. The encoding for the text must be specified as an option for PDF_fill_textblock( ) when filling the Block unless the* `font` *option has been supplied.*
fontsize[1]	*(Float) Size of the font in points*
horizscaling	*(Float or percentage) Horizontal text scaling. Default: 100%*
italicangle	*(Float) Italic angle of text in degrees. Default: 0*
kerning	*(Boolean) Kerning behavior. Default:* `false`
overline	*(Boolean) Overline mode. Default:* `false`
shadow	*(Composite) Create a shadow effect (default: no shadow). The following subproperties are available:* ***fillcolor*** *(Color) Color of the shadow. Default:* {`gray 0.8`} ***offset*** *(List of 2 floats or percentages) The shadow's offset from the reference point of the text in user coordinates or as a percentage of the font size. Default:* {`5% -5%`}
strikeout	*(Boolean) Strikeout mode. Default:* `false`
strokecolor	*(Color) Stroke color of the text. Default:* `gray 0` *(=black)*
strokewidth	*(Float, percentage, or keyword; only effective if* `textrendering` *is set to stroke text) Line width for outline text (in user coordinates or as a percentage of the* `fontsize`*). The keyword* `auto` *or the equivalent value 0 uses a built-in default. Default:* `auto`
textrendering	*(Integer) Text rendering mode. Only the value 3 has an effect on Type 3 fonts (default: 0):* **0**  *fill text* **1**  *stroke text (outline)* **2**  *fill and stroke text* **3** *invisible text* **4**  *fill text and add it to the clipping path* **5**  *stroke text and add it to the clipping path* **6**  *fill and stroke text and add it to the clipping path* **7**  *add text to the clipping path (not for Blocks)*
textrise	*(Float pr 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. This property is required in Textline and Textflow Blocks; it will be enforced by the Block Plugin.

13.7.4 Text Preparation Properties

Text preparation properties specify preprocessing steps for Textline and Textflow Blocks. Table 13.8 lists text preparation properties for Textline and Textflow Blocks.

Table 13.8 Text preparation properties for Textline and Textflow Blocks

keyword	possible values and explanation
charref	*(Boolean) If* true*, enable substitution of numeric and character entity references and glyph name references. Default: the global* charref *option*
escape-sequence	*(Boolean) If* true*, enable substitution of escape sequences in content strings, hypertext strings, and name strings. Default: the global* escapesequence *option*
features	*(List of keywords) Specifies which typographic features of an OpenType font will be applied to the text, subject to the* script *and* language *options. Keywords for features which are not present in the font will silently be ignored. The following keywords can be supplied:* ***_none*** *Apply none of the features in the font. As an exception, the* vert *feature must explicitly be disabled with the* novert *keyword.* ***<name>*** *Enable a feature by supplying its four-character OpenType tag name. Some common feature names are* liga*,* ital*,* tnum*,* smcp*,* swsh*,* zero*. The full list with the names and descriptions of all supported features can be found in Section 7.3.1, »Supported OpenType Layout Features«, page 164.* ***no<name>*** *The prefix* no *in front of a feature name (e.g.* noliga*) disables this feature.* *Default:* _none *for horizontal writing mode. In vertical writing mode* vert *will automatically be applied. The* readfeatures *option in PDF_load_font( ) is required for OpenType feature support.*
language	*(Keyword; only relevant if* script *is supplied) The text will be processed according to the specified language, which is relevant for the* features *and* shaping *options. A full list of keywords can be found in Section 7.4.2, »Script and Language«, page 172, e.g.* ARA *(Arabic),* JAN *(Japanese),* HIN *(Hindi). Default:* _none *(undefined language)*
script	*(Keyword; required if* shaping=true*) The text will be processed according to the specified script, which is relevant for the* features*,* shaping*, and* advancedlinebreaking *options. The most common keywords for scripts are the following:* _none *(undefined script),* latn*,* grek*,* cyrl*,* armn*,* hebr*,* arab*,* deva*,* beng*,* guru*,* gujr*,* orya*,* taml*,* thai*,* laoo*,* tibt*,* hang*,* kana*,* han*. The keyword* _auto *selects the script to which the majority of characters in the text belong, where* _latn *and* _none *are ignored. A full list of keywords can be found in Section 7.4.2, »Script and Language«, page 172. Default:* _none
shaping	*(Boolean) If* true*, the text will be formatted (shaped) according to the* script *and* language *options. The* script *option must have a value different from* _none *and the required shaping tables must be available in the font. Default:* false

13.7.5 Text Formatting Properties

Table 13.9 lists properties which can only be used for Textflow Blocks, with the exception of the *stamp* property which can also be used for Textline Blocks. They will be used to construct the initial option list for processing the Textflow (corresponding to the *optlist* parameter of *PDF_create_textflow( )*). Inline option lists for Textflows 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 *PDF_fill_textblock( )*, or in the Block's *defaulttext* property.

Table 13.9 Text formatting properties (mostly for Textflow Blocks)

keyword	possible values and explanation
adjustmethod	*(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* `minspacing` *and* `maxspacing` *options (default:* `auto`*):*
	auto *The following methods are applied in order:* `shrink`, `spread`, `nofit`, `split`.
	clip *Same as* `nofit`*, except that the long part at the right edge of the fit box (taking into account the* `rightindent` *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* `nofitlimit` *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* `shrinklimit`*. If it still doesn't fit the nofit 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* `spreadlimit`*. If justification still cannot be achieved the* `nofit` *method will be applied.*
advancedlinebreak	*(Boolean) Enable the advanced line breaking algorithm which is required for complex scripts. This is required for linebreaking in scripts which do not use space characters for designating word boundaries, e.g. Thai. The options* `locale` *and* `script` *will be honored. Default:* `false`
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*
avoidemptybegin	*(Boolean) If* `true`*, empty lines at the beginning of a fitbox will be deleted. Default:* `false`
fixedleading	*(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:* `false`
hortabmethod	*(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:* `relative`*):*
	relative *The position will be advanced by the amount specified in* `hortabsize`*.*
	typewriter *The position will be advanced to the next multiple of* `hortabsize`*.*
	ruler *The position will be advanced to the n-th tab value in the ruler option, where n is the number of tabs found in the line so far. If n is larger than the number of tab positions the relative method will be applied.*
hortabsize	*(Float or percentage) Width of a horizontal tab[1]. The interpretation depends on the* `hortabmethod` *option. Default: 7.5%*

Table 13.9 Text formatting properties (mostly for Textflow Blocks)

keyword	possible values and explanation
lastalignment	*(Keyword) Formatting for the last line in a paragraph. All keywords of the alignment option are supported, plus the following (default:* `auto`*):* ***auto*** *Use the value of the alignment option unless it is justify. In the latter case* `left` *will be used.*
leading	*(Float or percentage) Distance between adjacent text baselines in user coordinates, or as a percentage of the font size. Default: 100%*
locale	*(Keyword) The locale which will be used for localized linebreaking methods if* `advancedlinebreak=true`*. The keywords consists of one or more components, where the optional components are separated by an underscore character '_' (the syntax slightly differs from NLS/POSIX locale IDs):* ► *A required two- or three-letter lowercase language code according to ISO 639-2 (see* `www.loc.gov/standards/iso639-2`*), e.g.* `en`*, (English),* `de` *(German),* `ja` *(Japanese). This differs from the* `language` *option.* ► *An optional four-letter script code according to ISO 15924 e.g.* `Hira` *(Hiragana),* `Hebr` *(Hebrew),* `Arab` *(Arabic),* `Thai` *(Thai).* ► *An optional two-letter uppercase country code according to ISO 3166, e.g.* `DE` *(Germany),* `CH` *(Switzerland),* `GB` *(United Kingdom)* *Specifying a locale is not required for advanced line breaking: the keyword* `_none` *specifies that no locale-specific processing will be done. Default:* `_none` *Examples:* `de_DE`, `en_US`, `en_GB`
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* `wordspacing` *option will still be added). Defaults:* `minspacing=50%`*,* `maxspacing=500%`
minlinecount	*(Integer) Minimum number of lines in the last paragraph of the fitbox. If there are fewer lines they will be placed in the next fitbox. The value 2 can be used to prevent single lines of a paragraph at the end of a fitbox (»orphans«). Default: 1*
nofitlimit	*(Float or percentage) Lower limit for the length of a line with the* `nofit` *method (in user coordinates or as a percentage of the width of the fitbox). Default: 75%*
parindent	*(Float or percentage) Left indent of the first line of a paragraph[1]. The amount will be added to* `leftindent`*. Specifying this option within a line will act like a tab. Default: 0*
rightindent **leftindent**	*(Float or percentage) Right or left indent of all text lines[1]. If* `leftindent` *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: 0*
ruler[2]	*(List of floats or percentages) List of absolute tab positions for* `hortabmethod=ruler`*[1]. The list may contain up to 32 non-negative entries in ascending order. Default: integer multiples of* `hortabsize`
shrinklimit	*(Percentage) Lower limit for compressing text with the* `shrink` *method; the calculated shrinking factor is limited by the provided value, but will be multiplied with the value of the* `horizscaling` *option. Default: 85%*
spreadlimit	*(Float or percentage) Upper limit for the distance between two characters for the* `spread` *method (in user coordinates or as a percentage of the font size); the calculated character distance will be added to the value of the* `charspacing` *option. Default: 0*
stamp	*(Keyword; Textline and Textflow Blocks) This option can be used to create a diagonal stamp within the Block rectangle. The text comprising the stamp will be as large as possible. The options* `position`*,* `fitmethod`*, and* `orientate` *(only* `north` *and* `south`*) will be honored when placing the stamp text in the box. Default:* `none`*.* ***ll2ur*** *The stamp will run diagonally from the lower left corner to the upper right corner.* ***ul2lr*** *The stamp will run diagonally from the upper left corner to the lower right corner.* ***none*** *No stamp will be created.*

Table 13.9 Text formatting properties (mostly for Textflow Blocks)

keyword	***possible values and explanation***
tabalignchar	*(Unichar) Unicode value of the character at which decimal tabs will be aligned. Default: the period character '.' (U+002E)*
tabalignment[2]	*(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* `tabalignchar` *will be left-aligned at the tab position. If no* `tabalignchar` *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.*

1. In user coordinates, or as a percentage of the width of the fit box
2. Tab settings can be edited in the property subgroup `Ruler Tabs for hortabmethod=ruler` *in the Block properties dialog.*

13.7.6 Object Fitting Properties

Fitting properties are available for all Block types, although some properties are specific to a certain Block type. They control how the contents will be placed in the Block:

- Table 13.10 lists fitting properties for Textline, Image, PDF, and Graphics Blocks
- Table 13.11 lists fitting properties for Textflow Blocks (mostly related to aspects of vertical fitting).

The object fitting algorithm uses the Block rectangle as fitbox. Except for *fitmethod=clip* there will be no clipping; if you want to make sure that the Block contents do not exceed the Block rectangle avoid *fitmethod=nofit*.

Table 13.10 Fitting properties for Textline, Image, PDF, and Graphics Blocks

keyword	possible values and explanation
alignchar	*(Unichar or keyword; only for Textline Blocks) If the specified character is found in the text, its lower left corner will be aligned at the lower left corner of the Block rectangle. For horizontal text with* `orientate=north` *or* `south` *the first value supplied in the* `position` *option defines the position. For horizontal text with* `orientate=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 0 and the keyword* `none` *suppress alignment characters. The specified* `fitmethod` *will be applied, although the text cannot be placed within the Block rectangle because of the forced positioning of* `alignchar`**.** *Default:* `none`
dpi	*(Float list; only for image Blocks) 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*
fitmethod	*(Keyword) Strategy to use if the supplied content doesn't fit into the Block rectangle:* `auto`, `clip`, `entire`, `meet`, `nofit` *or* `slice` *(default:* `meet`).
margin	*(Float list; only for Textline Blocks) One or two float values describing additional horizontal and vertical reduction of the Block rectangle. Default: 0*
orientate	*(Keyword) Specifies the desired orientation of the content when it is placed. Possible values are* `north`, `east`, `south`, `west`. *Default:* `north`
position	*(Float list) One or two values specifying the position of the reference point within the content. The position is specified as a percentage within the Block. Only for Textline Blocks: the keyword* `auto` *can be used for the first value in the list. It indicates* `right` *if the writing direction of the text is from right to left (e.g. for Arabic and Hebrew text), and* `left` *otherwise (e.g. for Latin text).* *Default: {0 0}, i.e. the lower left corner*
rotate	*(Float) 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*
scale	*(Float list; only for image, PDF, and Graphics Blocks) 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*
shrinklimit	*(Float or percentage; only for Textline Blocks) The lower limit of the shrinkage factor which will be applied to fit text with* `fitmethod=auto`. *Default: 0.75*

Table 13.11 Fitting properties for Textflow Blocks

keyword	possible values and explanation
firstlinedist	*(Float, percentage, or keyword) The distance between the top of the Block rectangle 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* `fixedleading=true`*, and the maximum of all font sizes in the line otherwise), or as a keyword (default:* `leading`*):* **leading** *The leading value determined for the first line; typical diacritical characters such as À will touch the top of the fitbox.* **ascender** *The ascender value determined for the first line; typical characters with larger ascenders, such as d and h will touch the top of the fitbox.* **capheight** *The capheight value determined for the first line; typical capital uppercase characters such as H will touch the top of the fitbox.* **xheight** *The xheight value determined for the first line; typical lowercase characters such as x will touch the top of the fitbox.* *If* `fixedleading=false` *the maximum of all* `leading`*,* `ascender`*,* `xheight`*, or* `capheight` *values found in the first line will be used.*
fitmethod	*(Keyword) Strategy to use if the Block is too small for the Textflow:* **auto** `fontsize` *and* `leading` *are decreased until the text fits.* **clip** *Text is clipped at the Block margin (useful for linking Textflow Blocks).* **nofit** *Text runs beyond the bottom margin of the Block (useful for moving Blocks).* *Default:* `clip` *if the* `textflowhandle` *option is supplied, otherwise* `auto`
lastlinedist	*(Float, percentage, or keyword) Will be ignored for* `fitmethod=nofit`*) 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* `fixedleading= true`*, and the maximum of all font sizes in the line otherwise), or as a keyword. Default: 0, i.e. the bottom of the* `fitbox` *will be used as baseline, and typical descenders will extend below the Block rectangle.* **descender** *The descender value determined for the last line; typical characters with descenders, such as* `g` *and* `j` *will touch the bottom of the fitbox.* *If* `fixedleading=false` *the maximum of all descender values found in the last line will be used.*
linespread-limit	*(Float or percentage; only for* `verticalalign=justify`*) 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 PDF_fit_textflow( ) will return the string* `_boxfull`*.*
minfontsize	*(Float or percentage) Minimum allowed font size when text is scaled down to fit into the Block rectangle with* `fitmethod=auto` *when* `shrinklimit` *is exceeded. The limit is specified in user coordinates or as a percentage of the height of the Block. If the limit is reached the text will be created with the specified* `minfontsize` *as fontsize. Default: 0.1%*
orientate	*(Keyword) Specifies the desired orientation of the text when it is placed. Possible values are* `north`*,* `east`*,* `south`*,* `west`*. Default:* `north`
rotate	*(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: 0*

Table 13.11 Fitting properties for Textflow Blocks

keyword	possible values and explanation	
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* `linespreadlimit`*. The height of the first line will only be increased if* `firstlinedist=leading`*.*

13.7.7 Properties for default Contents

Properties for default contents specify how to fill the Block if no specific contents are provided. They are especially useful for the Preview feature since it will fill the Blocks with their default contents. Table 13.12 lists properties for default contents.

Table 13.12 Properties for default contents

keyword	possible values and explanation
default-graphics	*(String; only for graphics Blocks) Path name of a graphics file which will be used if no graphics is supplied by the client application.*[1]
defaultimage	*(String; only for image Blocks) Path name of an image which will be used if no image is supplied by the client application.*[1]
defaultpdf	*(String; only for PDF Blocks) Path name of a PDF document which will be used if no substitution PDF is supplied by the client application.*[1]
default-pdfpage	*(Integer; only for PDF Blocks) Page number of the page in the default PDF document. Default: 1*
defaulttext	*(String; only for Textline and Textflow Blocks) Text which will be used if no variable text is supplied by the client application*[2]

1. It is recommended to use file names without absolute paths, and use the `SearchPath` *feature in the PPS client application. This makes Block processing independent from platform and file system details.*
2. The text will be interpreted in `winansi` *encoding or Unicode.*

13.7.8 Custom Properties

Custom properties apply to Blocks of any type of Block, and will be ignored by PPS and the Preview feature. Table 13.13 lists the naming rules for custom properties.

Table 13.13 Custom Block properties for all Block types

keyword	possible values and explanation
any name not containing the three characters `[ ] /`	*(String, name, float, or float list) The interpretation of the values of custom properties is completely up to the client application; they will be ignored by PPS.*

13.8 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.

Cookbook *A full code sample for querying the properties of Blocks contained in an imported PDF can be found in the Cookbook topic* `blocks/query_block_properties`.

Finding the number and names of Blocks. The client code must not even know the names or number of 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 + "]/blocks");
```

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

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

The returned Block name can subsequently be used to query the Block's properties or fill the Block with text, image, PDF or graphics contents. 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 (keep in mind that the Block count will be one higher than the highest possible index since array indexing starts at 0).

Checking for the presence of a Block. For additional flexibility of the client application code you can check whether for the presence of a Block before attempting to fill it. This way the designer can move Blocks between pages without breaking the application which fills the Blocks.

The following code checks whether a Block with the name *foo* is present on a page:

```
/* pCOS object type "dictionary" means that the Block is present */
if (pcos_get_string(doc, "type:pages[" + pagenum + "]/blocks/" + "foo").equals("dict"))
{
        /* Block "foo" is present on the page */
}
```

Addressing Blocks by number or name. In the pCOS path syntax for addressing Block properties the following expressions are equivalent, assuming that the Block with number 6 has its *Name* property set to *foo:*

```
pages[...]/blocks[6]
pages[...]/blocks/foo
```

Querying 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 + "]/blocks/foo/rect[0]");
lly = p.pcos_get_number(doc, "pages[" + pagenum + "]/blocks/foo/rect[1]");
urx = p.pcos_get_number(doc, "pages[" + pagenum + "]/blocks/foo/rect[2]");
ury = p.pcos_get_number(doc, "pages[" + pagenum + "]/blocks/foo/rect[3]");
```

Note that these coordinates are provided in the default 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. The values queried with the pCOS pseudo object *rect* (all lowercase) take into account any relevant *CropBox/MediaBox* and *Rotate* entries and normalize the order of the coordinates. In contrast, the values queried with the native PDF key *Rect* cannot be directly used as new coordinates if a CropBox is present.

Note that the *topdown* option 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 + "]/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 + "]/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 predefined 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 PPS property names (as defined in Section 13.7, »Block Properties«, page 382) will never conflict with custom property names.

13.9 Creating and Importing Blocks programmatically

13.9.1 Creating PDFlib Blocks with POCA

PDFlib Blocks can be created programmatically with the POCA interface which is included in PPS. Using POCA the required PDF data structures for Block can be prepared and then supplied to the *blocks* option of *PDF_begin/end_page_ext()*. When creating the Block definitions the requirements in Section 13.10, »PDFlib Block Specification«, page 398, must be obeyed. The Block properties must be created according to the data types listed in Section 13.7, »Block Properties«, page 382.

Cookbook *A code sample for creating PDFlib Blocks with PPS can be found in the category* `blocks` *of the PDFlib Cookbook.*

The PDFlib Block specification contains an unfortunate redundancy in that the name of a Block is recorded twice: once in the main *Blocks* dictionary of a page, and again in the *Name* entry within a particular Block dictionary. These two names must be identical in order to avoid problems when filling the Block with PPS or previewing the Block with the Block Plugin. *PDF_begin/end_page_ext()* will therefore throw an exception if the dictionary provided with the *blocks* option contains a block definition which violates the »same block name« rule. The corresponding pairs are highlighted in blue in the code sample below.

The following code fragment demonstrates the use of POCA functions for creating the Block definition shown in Section , »Block dictionary keys«, page 399:

```
/* Create the Block dictionary */
blockdict = p.poca_new("containertype=dict usage=blocks");

/* ------------------------------------------------------
 * Create a Text Block
 * ------------------------------------------------------
 */
textblock = p.poca_new("containertype=dict usage=blocks type=name key=Type value=Block");

container1 = p.poca_new("containertype=array usage=blocks " +
        "type=integer values={70 640 300 700}");

p.poca_insert(textblock, "type=array key=Rect value=" + container1);
p.poca_insert(textblock, "type=name  key=Name value=job_title");
p.poca_insert(textblock, "type=name  key=Subtype value=Text");
p.poca_insert(textblock, "type=name  key=fitmethod value=auto");
p.poca_insert(textblock, "type=string  key=fontname value=Helvetica");
p.poca_insert(textblock, "type=float key=fontsize value=12");

/* Hook up the Block in the page's Block dictionary */
p.poca_insert(blockdict, "type=dict key=job_title direct=false value=" + textblock);

/* ------------------------------------------------------
 * Create an Image Block
 * ------------------------------------------------------
 */
imageblock = p.poca_new("containertype=dict usage=blocks " +
        "type=name key=Type value=Block");

container2 = p.poca_new("containertype=array usage=blocks " +
```

```
        "type=integer values={70 440 300 600}");

p.poca_insert(imageblock, "type=array key=Rect value=" + container2);
p.poca_insert(imageblock, "type=name  key=Name value=logo");
p.poca_insert(imageblock, "type=name  key=Subtype value=Image");
p.poca_insert(imageblock, "type=name  key=fitmethod value=auto");

/* Hook up the Block in the page's Block dictionary */
p.poca_insert(blockdict, "type=dict key=logo direct=false value=" + imageblock);

/* ------------------------------------------------------
 * Create a PDF Block
 * ------------------------------------------------------
 */
pdfblock = p.poca_new("containertype=dict usage=blocks " +
        "type=name key=Type value=Block");

container3 = p.poca_new("containertype=array usage=blocks " +
        "type=integer values={70 240 300 400}");

p.poca_insert(pdfblock, "type=array key=Rect value=" + container3);
p.poca_insert(pdfblock, "type=name key=Name value=pdflogo");
p.poca_insert(pdfblock, "type=name key=Subtype value=PDF");
p.poca_insert(pdfblock, "type=name key=fitmethod value=meet");

/* Hook up the Block in the page's Block dictionary */
p.poca_insert(blockdict, "type=dict key=pdflogo direct=false " + "value=" + pdfblock);


/* ------------------------------------------------------
 * Hook up the Block dictionary in the current page
 * ------------------------------------------------------
 */
p.end_page_ext("blocks=" + blockdict);


/* Clean up */
p.poca_delete(blockdict, "recursive");
```

13.9.2 Importing PDFlib Blocks

You can copy one ore more PDFlib Blocks from the input document to the current output page with *PDF_process_pdi()* and *action=copyallblocks* or *action=copyblock* as follows:

```
if (p.process_pdi(p, doc, 0, "action=copyallblocks block={pagenumber=1}") != 1)
{
        /* Error */
}
```

This way you can implement multi-level Block filling workflows. Keep in mind that Block names must be unique on each page, i.e. you cannot import multiple Blocks with the same name to the same page. Use the *outputblockname* suboption to rename Blocks upon copying.

13.10 PDFlib Block Specification

The Block syntax conforms to 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 description of the PDFlib Block syntax is provided here. Users who create Blocks with the Block Plugin or PDFlib don't need this information.

PDF object structure for PDFlib Blocks. The page dictionary contains a *PieceInfo* entry which has another dictionary as value. The page dictionary should also contain the key *LastModified* which contains a time stamp for the creation or last modification of the Block structures. This dictionary contains the key *PDFlib* with an application data dictionary as value. The application data dictionary contains two standard keys listed in Table 13.14.

Table 13.14 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. This entry will be created by PDFlib when creating Blocks with the POCA interface.*
Private	*(Dictionary; required) A Block list (see Table 13.15)*

A Block list is a dictionary containing general information about Block processing, plus a list of all Blocks on the page. Table 13.15 lists the keys in a Block list dictionary.

Table 13.15 Entries in a Block list dictionary

key	value
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 13.17). The value of the* `Name` *key in the Block dictionary must be identical to the Block's name in this dictionary.*
BlockProducer[1]	*(String) Name of the software used to create the Blocks programmatically. This entry will be created by PDFlib when creating Blocks with the POCA interface.*
PluginVersion[1]	*(String) A string containing a version identification of the Block plugin used to create the Blocks.*
pdfmark[1]	*(Boolean) Must be* `true` *if the Block list has been generated by use of pdfmarks.*
Version	*(Number; required) The version number of the Block specification to which the file complies. This document describes version 10 of the Block specification.*

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

Data types for Block properties. Properties support the same data types as option lists except handles and specialized lists such as action lists. Table 13.16 details how these types are mapped to PDF data types.

Table 13.16 Data types for Block properties

Data type	PDF type and remarks
boolean	*(Boolean)*
string	*(String)*

Table 13.16 Data types for Block properties

Data type	PDF type and remarks
keyword (name)	*(Name) It is an error to provide keywords outside the list of keywords supported by a particular property.*
float, integer	*(Number) While option lists support point and comma as decimal separators, PDF numbers require 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.*
list	*(Array)*
color	*(Array with two or three elements) The first element in the array specifies a color space, and the second element specifies a color value. To specify the absence of color the respective property must be omitted. The following entries are supported for the first element in the array:* ***/DeviceGray*** *The second element is a single gray value.* ***/DeviceRGB*** *The second element is an array of three RGB values.* ***/DeviceCMYK*** *The second element is an array of four CMYK values.* ***[/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 PPS, or which has been defined by the application at runtime.* ***[/Lab]*** *The first element is an array containing the keyword* `Lab`*. The second element is an array of three Lab values.*
unichar	*(Text string) Unicode string in* `utf16be` *format, starting with the BOM U+FEFF*

Block dictionary keys. Block dictionaries may contain the keys in Table 13.17.

Table 13.17 Entries in a Block dictionary

property group	values
administrative properties	*(Some keys are required) Administrative properties according to Table 13.4*
rectangle properties	*(Some keys are required) Rectangle properties according to Table 13.5*
appearance properties	*(Some keys are required) Appearance properties for all Block types according to Table 13.6 and text appearance properties according to Table 13.7 for Textline and Textflow Blocks*
text preparation properties	*(Optional) Text preparation properties for Textline and Textflow Blocks according to Table 13.8*
text formatting properties	*(Optional) Text formatting properties for Textline and Textflow Blocks according to Table 13.9*
object fitting properties	*(Optional) Object fitting properties for Textline, Image, PDF, and Graphics Blocks according to Table 13.10, and fitting properties for Textflow Blocks according to Table 13.11*

Table 13.17 Entries in a Block dictionary

property group	***values***
properties for default contents	*(Optional) Properties for default contents according to Table 13.12*
Custom	*(Dictionary; optional) A dictionary containing key/value pairs for custom properties according to Table 13.13.*