**PLANETPRESS®**
**DESIGN**

# PlanetPress Design
# User Guide

www.objectiflune.com

# Copyright Information

Copyright © 1994-2011  Objectif Lune Inc. All Rights Reserved.

No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any other language or computer language in whole or in part, in any form or by any means, whether it be electronic, mechanical, magnetic, optical, manual or otherwise, without prior written consent of Objectif Lune Inc.

Objectif Lune Inc.disclaims all warranties as to this software, whether expressed or implied, including without limitation any implied warranties of merchantability, fitness for a particular purpose, functionality, data integrity or protection.

PlanetPress and PrintShop Mail are registered trademarks of Objectif Lune Inc.

PostScript and Acrobat are registered trademarks of Adobe Systems Inc.

Pentium is a registered trademark of Intel Corporation.

Windows is a registered trademark of Microsoft Corporation.

Adobe, Adobe PDF Library, Adobe Acrobat, Adobe Distiller, Adobe Reader, Adobe Illustrator, Adobe Photoshop, Optimized Postcript Stream, the Adobe logo, the Adobe PDF logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Trademarks of other companies mentioned in this documentation appear for identification purposes only and are the property of their respective companies.

| | |
|---|---|
| **Title** | PlanetPress Design User Guide |
| **Revision** | 2013-03-22 |

PLANETPRESS. SUITE

# ⬛ Table of Content

# Fonts and Styles

A style is a specific set of properties that describe how a font is used within a PlanetPress Design object. Styles are created at the document level and can be used in any object where text is displayed, including data selections, text boxes, human-readable barcode text and business graphics.

While creating your PlanetPress Design document, you can use multiple styles as well as unique fonts for each style. This chapter explains how to create and manage styles, how to install and manage fonts, as well as how to troubleshoot font issues.

## Create a Style

To create a style:

1. Do one of the following to display the Style properties dialog box:
   - In the Structure area, right-click on the Styles folder and choose **Style**.
   - Choose **Home** | **Style**.
     The Style properties dialog box appears.
2. If you want to preview the style as you work, click the **Preview** button to display the Style Preview. Click the Preview button a second time to hide the Style Preview. You can show or hide the Style Preview at any time as you work in the Style dialog. As you work the Style Preview updates to reflect any changes you make to the properties in the Style dialog. See To work in the Style Preview: for help using the Style Preview.
3. In the **Style** properties dialog box, click **Style properties,** enter a name for the style, indicate whether or not you want to set it as the default style, and set the font properties for the style.
   **Name**: Enter a name for the style. Although PlanetPress Design supplies a default name, it is recommended you choose a name that reflects the purpose of the style. A meaningful name makes it easier to distinguish one style from another in a document, and thus makes the document easier to design and maintain. Names cannot begin with a number, and can contain only the following ASCII characters: underscore, upper and lower case letters of the alphabet, all digits 0 through 9. If you use an underscore in the name, it should not appear as either the first or last character of the name as this may cause internal conflicts in the software. Names are case-insensitive and must be unique (no two elements in a document can have the same name). Names can be a maximum of 50 characters in length. Finally, PlanetPress Talk variable and command names are reserved words; you cannot use any of these reserved words as a name.
   **Default style**: Select to make this style the default style. The default style is the style PlanetPress Design associates by default with each new object it creates. It is the style that appears by default in the Style box in the Basic attributes properties of an object when you create that object. The current default style appears in bold in the Structure area.
   Font Definition
   **Font type**: Select the type of font for the style. PostScript fonts are strongly recommended to improve printer performance and reduce the file size of a document. The type you select determines the contents and availability of the remaining options. The Default font type option in the User Options dialog box determines the font type that appears here by default.
   **Refresh Fonts List button**: Click to refresh the list of available fonts. You use this button if you added fonts to, or removed fonts from, your PlanetPress Design installation after you began your current PlanetPress Design session. PlanetPress Design automatically refreshes the list of available fonts each time it starts.
   **Font name**: Select the font you want to use for the style. The Font type you selected determines the contents of this box. If the font you want to use is a Bold, Italic, or Bold Italic font, you should choose the regular version of the font in this box, and use the Bold and/or Italic buttons to adjust its Bold and Italic properties. For example, if you want to use the Helvetica Bold font, select Helvetica in this box, and click the Bold button. This increases the flexibility of the style; rather than create a new style for each version of the font, you create a single style and adjust the properties as necessary when you reference the style. Note that if the font you select exists in the PPD file of the document (i.e. is printer-resident), PlanetPress Design does not include the font when it performs a hard copy preview or installation of

the document. If you did not modify the Font type option, the font name that appears by default is the one set in the Default font name option in the User Options dialog box.

Default attributes

**Font size**: Select or type the point size for the font. Note that this is the default font size for the style. You can override the default as necessary in various places in PlanetPress Design, including the text in text/box objects, and in PlanetPress Talk using the setstyleext() command.

**Bold button**: Click to turn the Bold property of the font on or off. When you turn the Bold property on, the style uses the font selected in the Bold font name box, or, if the Italic property is also on, the font selected in the Bold italic font name box, of the Advanced Fonts dialog box. The property is on when the button appears recessed.

**Italic button**: Click to turn the Italic property of the font on or off. When you turn the Italic property on, the style uses the font you selected in the Italic font name box, or, if the Bold property is also on, the font selected in the Bold italic font name box, of the Advanced Fonts dialog box. The property is on when the button appears recessed.

**Underline button**: Click to turn the Underline property of the font on or off. When you turn the Underline property on, the style underlines the characters of its font. The property is on when the button appears recessed. Note that spaces may not appear underlined in PlanetPress Design, but that they will be when the document is used to generate output.

**Color box**: View the current color for the style. When you select a color in the Color Picker, this box updates to reflect the selected color.

**Color button**: Click to select a color for the style using the Color Picker. The Color box displays the selected color for the style.

**Font ratio**: Enter a percentage by which you want to shrink or stretch the font spacing. This value adjusts both the width of each glyph and the spacing between glyphs. This is in contrast to kerning, which modifies the spacing between characters without modifying the width of characters.

4. If necessary, click **Set advanced fonts** (the button to the right of the Font name box), and use the Advanced Fonts dialog box to specify the fonts to use when you set the Bold, Italic, and BoldItalic properties on this style.

**Bold font name**: Select the font to use when you set the Bold property on this style. You use the Bold button to set the Bold property. The Font type you selected determines the contents of this box, and the Font name you selected determines which font appears by default in this box. It is important to select a font from the same family as the font you selected for the style in the Font name box. For example, if you selected Helvetica in the Font name box, you should select a Helvetica font (for example, Helvetica Bold) in this box. This ensures all fonts the style references use the same encoding table, and thus prevents unpredictable results. Note that if the font you select exists in the PPD file of the document (i.e. is printer-resident), PlanetPress Design does not include the font when it performs a hard copy preview or an installation of the document.

**Italic font name**: Select the font to use when you set the Italic property on this style. You use the Italic button to set the Italic property. The Font type you selected determines the contents of this box, and the Font name you selected determines which font appears by default in this box. It is important to select a font from the same family as the font you selected for the style in the Font name box. For example, if you selected Helvetica in the Font name box, you should select a Helvetica font (for example, Helvetica Oblique) in this box. This ensures all fonts the style references use the same encoding table, and thus prevents unpredictable results. Note that if the font you select exists in the PPD file of the document (i.e. is printer-resident), PlanetPress Design does not include the font when it performs a hard copy preview or an installation of the document.

**Bold italic font name**: Select the font to use when you set both the Bold and Italic properties on this style. You use the Italic and Bold buttons to set these properties. The Font type you selected determines the contents of this box, and the Font name you selected determines which font appears by default in this box. It is important to select a font from the same family as the font you selected for the style in the Font name box. For example, if you selected Helvetica in the Font name box, you should select a Helvetica font (for example, Helvetica Bold Oblique) in this box. This ensures all fonts the style references use the same encoding table, and thus prevents unpredictable results. Note that if the font you select exists in the PPD file of the document (i.e. is printer-resident), PlanetPress Design does not include the font when it performs a hard copy preview or an installation of the document.

5. In the **Encoding** box, select the encoding table you want to use for the font.

6. Click **OK**.
   PlanetPress Design creates the style. The style appears in the Styles folder in the Structure area.

To work in the Style Preview:

1. If the Style Preview is not visible, in the Style dialog click the Preview button.
   The Style Preview appears, and displays the test string using the settings entered to date in the Style dialog.
2. In the Style Preview do any of the following:
   - *To change the test string*, in the Test string box either modify the currently selected string or select a previously entered string. The Preview updates to reflect the changes to the test string.
   - *To change the zoom level*, use the Zoom toolbar in the upper left of the Style Preview. Click in the Current zoom factor box and enter the new zoom (the zoom factor can be any value from 10 to 1000). Alternatively click the Zoom in or Zoom out tool to zoom in or out respectively, by the zoom factor set in the User Options dialog box. PlanetPress Design updates the Current zoom factor box to reflect the new zoom level.

## Apply a Style

The procedures here describe how to apply a style to one or more objects in a document. Note that when you first create an object or a group, PlanetPress Design assigns the default style to that object or group. You can subsequently change that style using the object's properties dialog box, the Object Inspector, or the procedures described here.

You can also change the style PlanetPress Design assigns to new objects by default. See "Set the Default Style for New Objects and Groups" (page 32).

To apply a style to a single object or group:

1. In the Structure area, drag the style to the object or group to which you want to apply it.
2. Release it.

To apply a style to one or more objects and/or groups in a document:

1. Select one or more objects and/or groups.
2. In the **Object Inspector**, select the style you want to associate with the selected objects and/or groups. Note that if you associate a style with a text object as a whole, that style overrides any others defined in the text object. PlanetPress Design applies the style to all text in the selected objects and/or groups. This style becomes the default style for any PlanetPress Talk code you add in the PlanetPress Talk properties of the objects/groups.

## View or Edit the Properties of a Style

You can view or edit the properties of a style using either the Object Inspector or the properties dialog box for the style. If the property you want to edit is the name of the style, also see "Change the Display Name of an Element in the Structure Area" (page n).

To view or edit properties using the Object Inspector:

1. In the Structure area, select the style whose properties you want to view or edit.
   The Object Inspector displays the properties of that style. If the Object Inspector is not visible, see "Show or Hide Areas of the Program Window" (page 81).
2. In the Object Inspector, make any necessary modifications to the properties. See "Use the Object Inspector" (page n) for help.
   PlanetPress Design updates the style. If you edited the name of the style, PlanetPress Design also updates the name of the style in all objects that reference it

To view or edit properties using the properties dialog box:

PLANETPRESS. SUITE

1. In the Structure area, do any of the following to display the properties dialog box for the style:
    - Double-click the style.
    - Select the style and press **ENTER**.
    - Select the style and, in the Object Inspector, double-click one of its properties.
2. Use the properties dialog box to edit the style properties, if necessary.
3. In the properties dialog box, click **OK**.
    PlanetPress Design updates the style. If you edited the name of the style, PlanetPress Design also updates the name of the style in all objects that reference it.

## Delete a Style

This procedure describes how to delete one or more styles. When you delete a style, you must define how you want PlanetPress Design to handle references to that style in existing document elements.

To delete one or more styles:

1. In the Structure area, select the styles you want to delete.
2. Do any of the following:
    - Select your style and go to **Home** | **Clipboard** | **Delete**.
    - In the Structure area, right-click one of the selected styles and choose **Delete**.
    - Press **DELETE**.
      If no objects or groups reference any of the selected styles, PlanetPress Design performs the deletion.
      If any objects or groups reference any of the selected styles, PlanetPress Design prompts you to define how you want to handle the deletion of each referenced style. More precisely, for each referenced style, it displays the Style Deletion dialog box. You use that dialog box to set the deletion options and proceed with the deletion. See To use the Style Deletion dialog box:.

To use the Style Deletion dialog box:

1. Adjust the options to reflect how you want PlanetPress Design to handle the deletion request. The name of the style you selected for deletion appears in the title bar of the Style Deletion dialog box, and the list of objects and groups that reference it appear on the right of the dialog box.
    **Replace reference by**: Select to delete the style and to replace all references to it with a reference to another of the styles in the document.
    **Styles available**: Select the style you want to use as the replacement reference. When you delete the style, PlanetPress Design replaces all references to the deleted style with a reference to the style you select here. You can use the Styles button to create a new style to add to this list.
    **Style button**: Click to create a new style. PlanetPress Design creates the new style, and selects it in the Styles available box.
    **Delete**: Select to delete the style and all objects and groups that reference it. All objects and groups that reference the style appear in the list on the right of the Style Deletion dialog box.
2. Click **OK**.
    PlanetPress Design deletes the style according to the selected option.
    If the style you deleted was the default style, PlanetPress Design makes the topmost style in the Structure area the new default style. Recall that the style whose name appears in bold in the Structure area is the default style.

## Set the Default Style for New Objects and Groups

You can set the default style PlanetPress Design uses for objects and groups you add to a document. The default style is the one that appears by default in the Style box in the Basic attributes properties of an object or a group. The style that appears in bold in the Structure area is the current default style.

To set the default style:

- Do either of the following:
    - In the Structure area, click the style you want to set as the default style. Then, in the Object Inspector, locate the property **IsDefaultStyle**, click **Press to Set as Default**, then click the button the appears on the right of the property.
    - Open the Style properties dialog box of the style you want to set as the default style (either double-click the style in the Structure area, or select it in the Structure area and press **ENTER**). In the **Style** properties dialog box, click **Style properties**, select **Defaultstyle**, then click **OK** to exit the dialog box.
    PlanetPress Design updates the default style to reflect the chosen style. The chosen style appears in bold in the Structure area.

## Create a MICR Style for Account Information on Cheques

Magnetic Ink Character Recognition (MICR) is a technology commonly used by banks to mark account information at the bottom of cheques. It uses a highly stylized font with precise dimensions. This procedure describes how to create a style that uses the MICR font.

To create a style that uses the MICR font:

1. Do one of the following to display the Style properties dialog box:
    - In the Structure area, right-click on the Styles folder and choose **Style**.
    - Choose **Home** | **Style**.
    The Style properties dialog box appears.
2. In the **Style** properties dialog box, click **Style properties** and enter a name for the style.
3. Set the font properties for the MICR style.
4. In the **Font type** box, select **TrueType host single byte**.
5. In the **Font name** box, select **MICRE 13B**.
6. In the **Font size** box, set the size of the font to **17** points.
7. Leave all other settings at their default values.
8. Click **OK**.
    PlanetPress Design creates the MICR style. The style appears in the Styles folder in the Structure area.

# Arabic Content in PlanetPress Design Documents

PlanetPress Suite provides you with two ways to add Arabic content to PlanetPress Design documents. Static text as well as variable data can be added using both Text objects and PlanetPress Talk objects.

You can use Text objects to display both static text and variable content, such as data selections, in the same object. Text objects also let you mix Arabic and non-Arabic styles within the same object, although not in the same paragraph.

Before you can create documents that contain Arabic text, you must have enabled the linguistic options required for Arabic text. Refer to Windows documentation for more information on regional settings and linguistic options.

To include objects that contain Arabic text, be it Text or PlanetPress Talk objects, your PlanetPress Design document must have at least one Arabic style. Arabic styles must be associated with the font type "TT, host, Arabic" and with a Unicode font, such as Andalus or Arial.

Note that in some cases, Arabic characters included in the same word may not be correctly attached.

### Arabic Support in the Data Selector

You can display Arabic text in the Data Pane as well as in the Data Selector itself. A box named Data encoding lists all the single and double-byte encodings supported by PlanetPress Design Just below this box is a new option that lets you specify

wether or not the data is already formatted as Arabic text. Finally, an option lets you turn contextual analysis of the data on or off.

# CID-Keyed Fonts

A CID-keyed font is a postscript (or Open Type) font designed to hold Chinese, Japanese and Korean characters efficiently. More accurately a CID font is a collection of several sub-fonts each with certain common features—one might hold all the latin letters, another all the kana, a third all the kanji. CID keyed fonts do not have an encoding built into the font, and the characters do not have names. Instead the font is associated with a character set and on each character set there are several character mappings defined. These mappings are similar to encodings but allow for a wider range of behaviors.

PlanetPress only accepts horizontal fonts and double-byte character set (DBCS) font encodings. In a PPD file, a typical DBCS font is defined as follows:

*Font HeiseiKakuGo-W5-90ms-RKSJ-H: RKSJ "()" 90ms ROM

where:

**Font** Defines a font

**HeiseiKakuGo-W5** Font Name

**90ms-RKSJ** Encoding type. In this example, the type is RKSJ Japanese. Other language options include: GBK EUC, ETen-B5, Chinese B5, and KSCms-UHC:Korean.

**H** Specifies whether it is a horizontal font. PlanetPress supports only horizontal fonts and DBCS encodings.

# Double-byte Character Sets

Does PlanetPress Design support double-byte character sets?

Double-byte character sets are supported in the PlanetPress Suite. It is important to note that double-byte character sets occupy twice the amount of space as single-byte character sets.

When working with Asian character sets and other non-Latin character sets that are double-byte, note that in the documentation, the length of strings is specified for single-byte character sets. For example, if the limit is specified as 256 characters and you are using a double-byte character set, the limit is actually 128 characters.

Supported double-byte character sets include PostScript and TrueType fonts. For optimum performance, PostScript fonts are recommended.

When previewing or printing PlanetPress Design documents that contain double-byte TrueType fonts, such as those used for some Asian languages, you must enable the Run locally option. This true whether the document is previewed or printed by PlanetPress Design or via PlanetPress Suite Workflow Tool.

# Encoding Tables

What is an encoding table and why is there more than one?

Encoding tables are tables computers use to map keystrokes to font glyphs. Your keystroke generates a numeric code and the computer consults an encoding table for the Helvetica font. The encoding table tells the computer which glyph is associated with that numeric code, in this case the glyph for the upper case 'A'. The computer displays the upper case 'A' glyph for the Helvetica font on the screen.

## Why have different encoding tables?

The obvious immediate strategy was to extend the ASCII character set. Each character in the standard ASCII character set fits in a single byte, but it uses only seven of the eight bits in the byte to represent characters. Using the full 8 bits of a byte to represent a character increased the number of characters you could represent from 128 to 255 and made it possible to represent many more languages.

Other strategies also developed for multi-byte character sets, such as those for the Chinese, Japanese and Korean languages.

## Encoding Tables in PlanetPress Design

Encoding tables can vary across platforms. When you create your documents in PlanetPress Design, you want to ensure that the input data the document receives maps to the correct glyphs in the output. You use encoding tables to make any necessary adjustments.

In PlanetPress Design you specify the encoding table you want a given style to use, or you define your own encoding table for that style. You can rearrange the glyphs in the encoding table, altering the glyph associated with a specific numeric code. You can also add glyphs to the encoding table from the list of all glyphs in the font. Not all glyphs in a font necessarily appear in an encoding table.

You also specify an encoding table for the font you select to display the sample data file in the Data Pane.

There are four key points to keep in mind as you work with encoding tables in PlanetPress Design:

1. A font usually contains more glyphs than an encoding table references.
2. Different fonts have different glyphs. If you use two different fonts, there may be differences in the glyphs available in each.
3. Different encoding tables reference different glyphs and/or may place the same glyphs in different positions. If you use the same font but a different encoding table, the glyph that represents a given input character may change.
4. You can edit the encoding table a style uses, and adjust both the glyphs the encoding table references and the positions of those glyphs within the encoding table. You cannot edit the encoding table for the font you use to display the sample data file. The output of the document always reflects what appears in the data selections on the document page.

## Edit the Encoding Table for a Style

The procedures here describe how to edit the encoding table for a style, and how to import an external encoding table for a style. Editing an encoding table means editing the individual characters that compose it. You set the encoding table that appears by default in the Style properties dialog box, in the User Options dialog box. See <u>"Set a Default Encoding Table" (page 38)</u>.

It is important to understand that when you edit the encoding table for a style, the edits are internal to that style and are not made to the encoding table itself. Thus the edits you make to an encoding table in one style do not appear in that encoding table when you view it in another style.

Note that PlanetPress Design saves all the encoding tables the styles in a document use in the PP7 file for that document.

To edit the encoding table for a style:

1. If you are not currently in the Style properties dialog box for the style whose encoding table you want to edit, do either of the following to display the dialog box:
   - In the Structure area, double-click the style.
   - In the Structure area, select the style and press **ENTER**.
2. In the Style properties dialog box, click **Style properties**.
3. In **Style properties**, click **Encoding** (the button to the right of the Encoding box).
   The Font Encoding dialog box appears. The Preview area displays a string that uses the currently selected encoding. Use the box just above the Preview area to select or enter the string you want to preview.
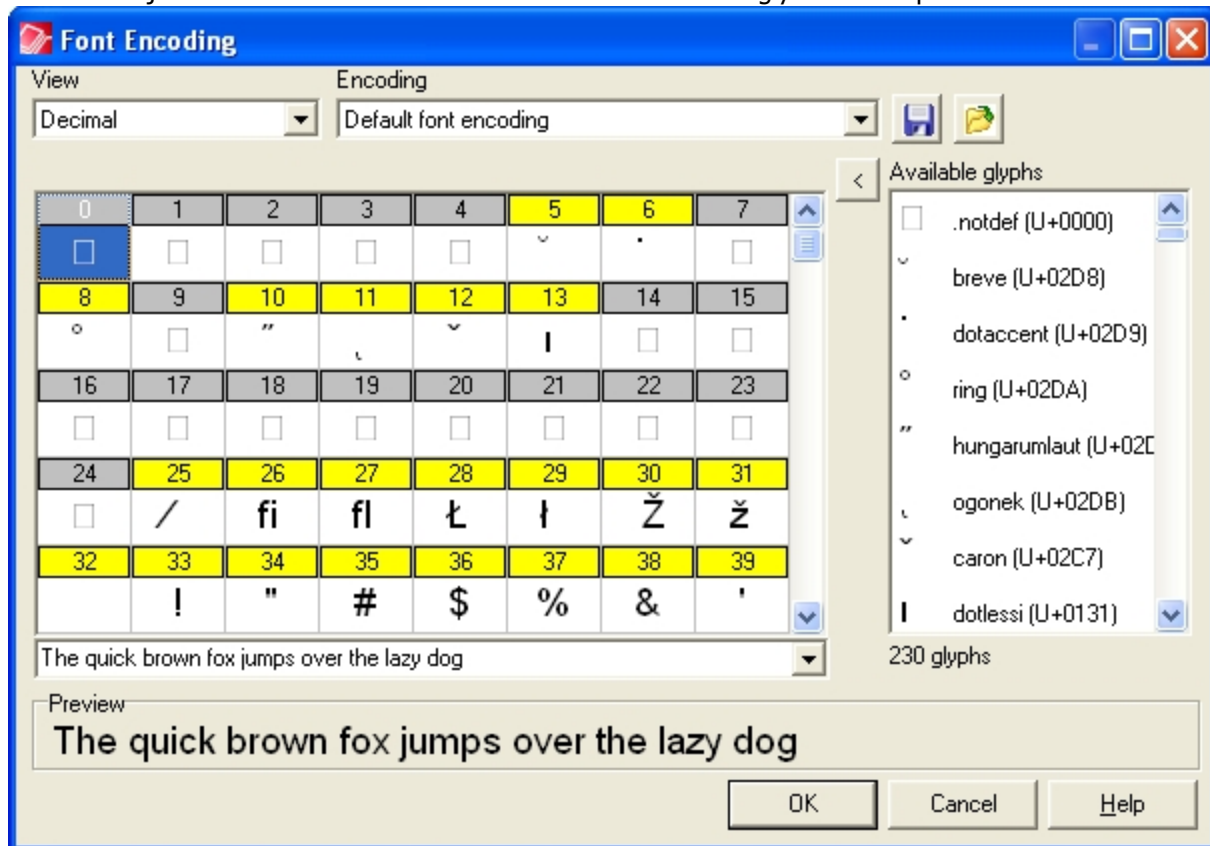


A. Encoding table B. Preview area

4. In the **View** box, select a view.
   **View**: Use to control the information that appears above each glyph in the table. The encoding table currently associated with the font determines the values associated with each glyph. Select Name to display the name of glyph. Select Decimal to display the position of each glyph in the encoding table as a decimal value. Select Hex to display the position of each glyph in the encoding table as a hexadecimal value. Select ANSI to display the ANSI equivalent of each glyph. Select Unicode to display the Unicode equivalent of each glyph. Select Width to display the width of each glyph.
5. Do either of the following:
   Import an Encoding Table
   See To import an encoding table: .
   Edit the glyphs
   In the **Encoding** box, select the encoding table you want to use as a starting point. Alternatively, import the encoding table you want to use as a starting point (see To import an encoding table: ). Note that the ISO Latin-9 encoding table includes the euro symbol. Edit the glyphs in the encoding table. See To edit the glyphs in an encoding table:.
6. If useful, use the Preview box just above the Preview area to select or enter a string to preview using the imported and/or edited encoding table.
7. Click **OK** to exit the Font Encoding dialog box and return to the Style properties dialog box.

To edit the glyphs in an encoding table:

1. Do any of the following. Notice that as soon as you edit the first glyph, the Encoding box updates to User defined font encoding. You can only modify a user defined encoding table.
   Drag and Drop
   In the **Available glyphs** list, locate the glyph you want to use in a given position in the encoding table. Use the scroll-bar, or click in the list and use the **ARROW** keys to navigate in the table. Click on the glyph, then drag and drop it into the position in the **Encodingtable**.
   Double-click
   In the **Encodingtable**, click on the glyph you want to edit. Use the scrollbar, or click in the table and use the **ARROW** keys to navigate in the table. In the **Available glyphs** list, double-click the glyph with which you want to replace the glyph selected in the encoding table.
   Use The Transfer Button
   In the **Encodingtable**, click on the glyph you want to edit. Use the scrollbar, or click in the table and use the **ARROW** keys to navigate in the table. In the **Available glyphs** list, click the glyph then click the **Transfer** button.
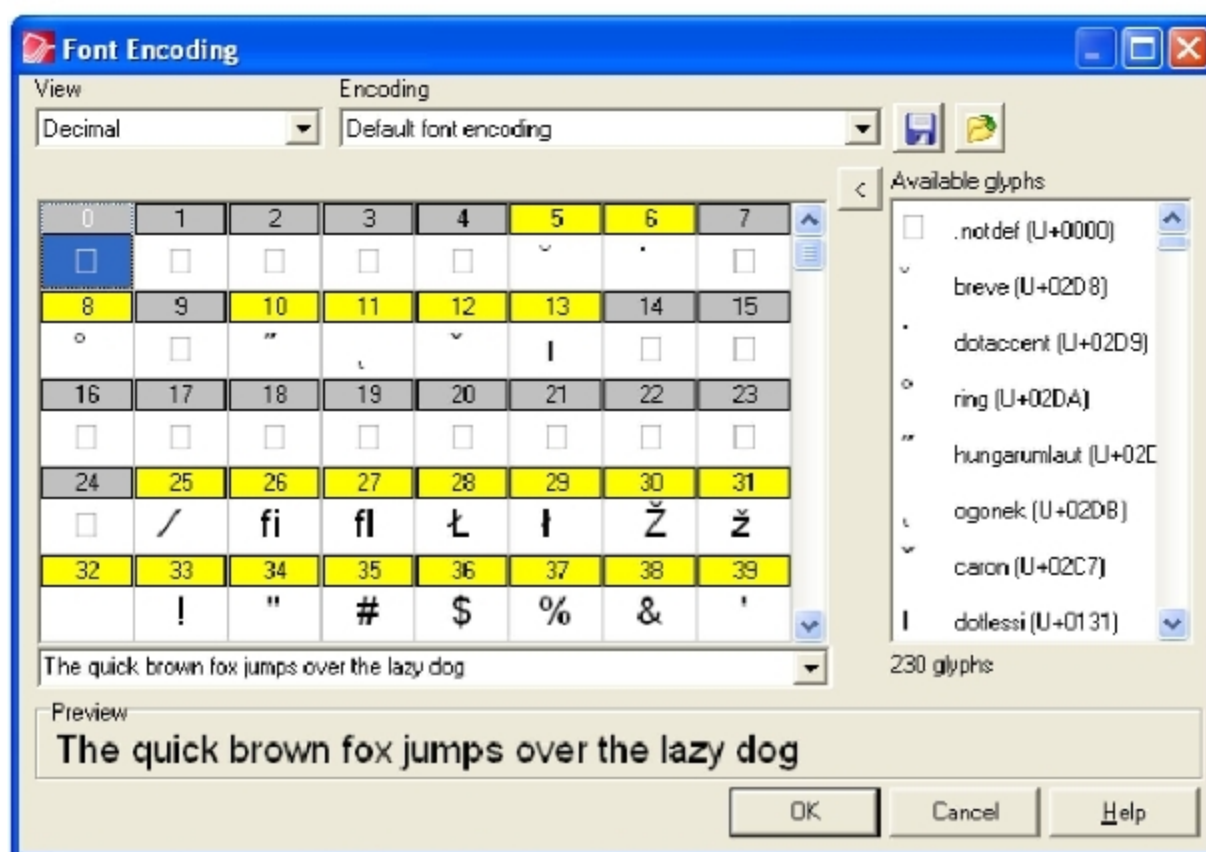


   A. Transfer button
2. Repeat step 1 for each glyph you want to edit.

To import an encoding table:

1. In the **Font Encoding** dialog box, click the **Browse** button ( ⬀ ).
   The Import an Encoding dialog box appears.
2. In the **Import an Encoding** dialog box, navigate to the file containing the encoding table you want to import and click **Open**. Encoding files have an **.enc** file name extension.
   PlanetPress Design imports the encoding table and returns you to the Font Encoding dialog box. The imported encoding table appears in the Font Encoding dialog box. The Encoding box displays User defined encoding.

---

## Set a Default Encoding Table

This procedure describes how to set the encoding table that appears by default in the Encoding box in the Style properties dialog box. This is useful if most of the styles you create use the same encoding table, and you do not want to select it each time you create a style.

To set a default encoding table:

1. From the **PlanetPress Design Button**, choose **Preferences** to display the Preferences dialog box.
2. In the User Options dialog box, click **Document and pages**, under **Document defaultvalues**.
3. In the **Default single byte font encoding** box, select the encoding table you want to have appear by default in the Encoding box of the Style properties dialog box.
4. Click **OK** to exit the User Options dialog box.

## Export an Encoding Table

This procedure describes how to export an encoding table. This is useful if you want to create an encoding table and re-use it in other styles you create in this or other variable content documents.

To export an encoding table:

1. If you are not currently in the Style properties dialog box for the style whose encoding table you want to export, do either of the following to display the dialog box:
   - In the Structure area, double-click the style.
   - In the Structure area, select the style and press **ENTER**.
2. In the Style properties dialog box, click **Style properties**.
3. In **Style properties**, click **Encoding** (the button to the right of the Encoding box).
   The Font Encoding dialog box appears.
4. Verify the encoding table that appears in the Font Encoding dialog box is the one you want to export.
5. Click the **Export** button (  ).
   The Export Encoding dialog box appears.
6. In the **Export Encoding** dialog box, specify the folder and file name for the file in which you want to save the encoding table, and click **Save**. Files that contain encoding tables have an **.enc** file name extension.
   PlanetPress Design exports the encoding table and returns to the Font Encoding dialog box.
7. Click **OK** to exit the Font Encoding dialog box.
8. Click **OK** to exit the Style properties dialog box.

## Refresh the Font Lists

If you update the set of fonts in the PlanetPress Design installation, you can refresh the contents of the relevant boxes in the interface either by restarting PlanetPress Design or by using the Refresh Fonts List button in the Style properties dialog box.

To refresh the list of fonts using the Refresh Fonts List button:

1. Do one of the following to display the Style properties dialog box:
   - In the Structure area, double-click an existing style.
   - In the Structure area, click an existing style and press **ENTER**.
   - In the Structure area, right-click on the Styles folder and choose **Style**.
   - In the Structure area or in the Page area, right-click and choose **Style**.
   - Choose **Home | Style**.
     The Style properties dialog box appears.
2. In the **Style** properties dialog box, click **Font properties**.

3. Click the **Refresh Fonts List** button located on the right of the **Font type** box.
4. PlanetPress Design refreshes all font lists in the interface.

## Install a PostScript Font in PlanetPress Design

The set of Postscript fonts available in PlanetPress Design is the combination of those available in the PlanetPress Design Fonts subfolder of the Windows Common Files folder (in a default installation, the path of this subfolder is C:\Documents and Settings\All Users\Application Data\Objectif Lune\PlanetPress Suite X\Fonts) and those available in the PPD file selected for the document.

**To install a PostScript font in PlanetPress Design:**

1. Verify the font is ready for installation. If the PostScript font files exist on a CD-ROM and are read-only, you must copy them to the hard disk and clear the read-only property of the hard disk copies before you can install the font in PlanetPress Design. You should provide the .pfb file for each PostScript font you install in PlanetPress Design.
2. Verify there is an appropriate TrueType font in Windows that you can use to represent the PostScript font on-screen. Ideally, you should install the TrueType version of the PostScript Type 1 font.
3. Choose Tools | Install PostScript Font. The Install a PostScript Font in PlanetPress Design dialog box appears.
4. Select the font you want to install.
   **Font name:** Select a font or click the Browse button to navigate to the file and select it.
   **Font name info:** Displays information about the font selected in the Font name box.
5. Select the TrueType font you want to associate with this PostScript font. Note that you can later edit this selection.
   **TrueType font to represent this font:** Select the TrueType font you want PlanetPress Design to use to represent this PostScript font on-screen. If the TrueType version of this PostScript font is not installed, select the TrueType font that most closely resembles it. When you select a TrueType font, the Glyphs available in TrueType font box updates to contain all the glyphs in the selected TrueType font. The Glyphs list also updates to display each of the glyphs in the PostScript font, with their corresponding glyph in the selected TrueType font. Note that this association between a True-Type font and a PostScript font is local to the computer on which PlanetPress Design is currently running and that the PlanetPress Suite Workflow Tool Configuration program does not let you make such associations between fonts.
6. If necessary or of interest, click Preview to display a PDF file of the complete character set of the selected PostScript font and its default encoding. Exit this file to return to the Install a PostScript Font in PlanetPress Design dialog box.
7. In the Glyphs list, verify the correct TrueType glyph is associated with each PostScript glyph, and use the Glyphs available in TrueType font box to edit any that are incorrect.
8. Click OK. PlanetPress Design copies the font files to the Fonts subfolder, and updates its font lists to make the font available from PlanetPress Design.

**To change the TrueType glyph associated with a PostScript glyph:**

1. In the Glyphs list, select the glyph correspondence you want to edit.
2. In the Glyphs available in TrueType font box, edit the glyph correspondence. Glyphs available in TrueType font: Select the TrueType glyph you want to associate with the PostScript glyph currently selected in the Glyphs list. When you select a glyph, PlanetPress Design immediately updates the Glyphs list to reflect the new correspondence.

**Things to Remember**

- Use PostScript fonts when possible to improve printer performance and reduce the file size of a document.
- PlanetPress Design must include in the document any PostScript fonts that the document uses and that do not appear in the PPD selected for the document.

PLANETPRESS. SUITE

# Overview

This documentation covers PlanetPress Suite **version 7.2**. To view the documentation of previous versions please refer to the PDF files available in the *Downloads* section of our website: http://www.objectiflune.com/OL/en-CA/Download/DownloadCenter.

## Icons used in this guide

Some icons are used throughout this guide in order to catch your attention to certain particular information.

**Notes**: This icon shows you something that complements the information around it. Understanding notes is not critical but may be helpful when using PlanetPress Design.

**Warnings**: This icon shows information that may be critical when using PlanetPress Design. It is important to pay attention to these warnings.

**Technical**: This icon shows technical information that may require some technical knowledge to understand.

## Other Documentation

For more online documentation on different PlanetPress Suite Products, refer to:

PlanetPress Design User Guide
PlanetPress Search User Guide
PlanetPress Talk Reference Guide
PlanetPress Trigger and Data Capture Guide

# 🎨 Getting Started

This section will help you get started with PlanetPress Design

## Environment Considerations

This page is intended to provide technical information about the environment in which PlanetPress Suite is intended to run.

### Terminal Server/Service

PlanetPress Suite does not support Terminal Server (or Terminal Service) environment as possible under Windows 2000, 2003 and 2008. This is to say, if Terminal Service is installed on the server where PlanetPress Suite is located, unexpected behaviours may occur and will not be supported by our company. Furthermore, using PlanetPress Suite in a Terminal Service environment is probably an infringement of our End-User License Agreement.

Terminal Service is not to be confused with "Remote Desktop" used in conjunction with the "Remote Desktop" options of all Windows operating systems. Remote Desktop, when used to connect to a PlanetPress Server in a non-simultaneous setting, is perfectly acceptable. However, it has been reported that PlanetPress may lock up when access through Remote Desktop connections.

### VMWare/ESX

PlanetPress Suite supports VMWare Workstation, VMWare Server, VMWare Player and VMWare ESX infrastructure environments as software installed on the Guest operating system. However, note that only PlanetPress version 7.1.3 and higher support VMotion under ESX. Also note that copying (duplicating) your VMWare Guest Machine and using it simultaneously constitutes an infringement of our End-User License Agreement.

Note that while VMWare ESX, Workstation, Server and Player (from VMWare, Inc.) are supported, other virtual environments (such as Microsoft Virtual PC, Parallels, Xen and others) are not supported at this time.

### 32-Bit or 64-Bit?

PlanetPress Suite version 7.1.3 and higher support 64-Bit operating system. However, our application remains 32-bits in this environment, which means that for all intents and purposes there is no difference between those two environments as far as PlanetPress Suite is concerned.

## Network Considerations

### Mapped Drives

> This section only applies when sending documents to the PlanetPress Suite Workflow Tools and when using mapped drives in your PlanetPress Talk scripts or in any variable property that refers to a mapped drive path.

Mapped drives (for example, drive X: leading to \\server\public\) are always user-specific and are created at logon. This means that mapped drives are typically not available by the PlanetPress Suite services when running a live configuration. Furthermore, while the mapped drives are not shared, they are still limited to one map per computer, meaning if one user maps the X: drive, a different user (or a service) will not be able to map it again.

This creates a limitation in the PlanetPress Suite Workflow Tools: if you create a mapped drive as a user, you will not have access to this mapped drive while running as a service unless you log off, and then have PlanetPress Workflow Tools map the drive using a Run Script action inside a Startup Process.

We strongly recommended that instead of using mapped drives, you use full UNC paths for your network drives.

# Activate Your Printers

The Activate a Printer dialog lists the existing activated printers on the system and lets you add new activations.

> Printer activations are normally given to you by the activations department electronically, including a file that will automatically add all your printers in this dialog.

**The printer list displays the following information**

- **License Number**: Reference number of the activation, linked to your customer account.
- **Magic Number**: The magic number generated by the printer. If the magic number is incorrect, your jobs will output with a watermark on that printer.
- **Activation Code**: The activation code generated by your license number and magic number. If the activation code is incorrect, your jobs will output with a watermark on that printer.
- **Printer Name** (Optional): Name and/or model of the printer.
- **Comments** (Optional): Comments about the printer.

The following buttons are available in this dialog:

- **Add**: Brings up the Printer Activation dialog. This dialog lets you enter the information for the printer (see previous section), then click OK to save the new activation.
- **Delete**: Removes the currently selected activation from the list.
- **Web Activation**: Click to access the online activation manager on our website.
- **OK**: Save changes and exit.
- **Cancel**: Exit without saving changes.

You can also double-click on any existing activation to edit it.

# The Nature of PlanetPress Design

PlanetPress Design allows you to create variable content documents from scratch or repurpose and enhance existing business documents. You can create documents for every need from a simple mail merge to complex multi-part business forms. All documents created with PlanetPress Design can have built-in conditions and rules to dynamically add barcodes, graphics & logos, page numbers, targeted marketing messages or envelope inserter controls, all based on the information found in the data or the original formatted document.

PlanetPress Design is included with the purchase of any of PlanetPress Suite software modules. It has an intuitive interface with various design features that are easy to use yet powerful enough to create simple to complex variable content documents. The possibilities are endless with PlanetPress Talk, a straightforward scripting language that allows you to fully customize your documents to answer highly complex or unique requirements.

# The PlanetPress Design Program

This chapter provides basic information about the PlanetPress Design user interface and how to use it.

## Start PlanetPress Design

To start the PlanetPress Design program:

- In the Windows **Start** menu, choose **Programs** | **PlanetPress Suite 7** | **PlanetPress Design**.

  The PlanetPress Design window appears and a blank document is created.

You can also double-click on any existing PlanetPress Design document (*.pp7) in order to open it in PlanetPress Design.

You can have multiple instances of PlanetPress Design open. Note however that each instance uses more memory and processing power, so only open the documents you need to work on and closed the unused ones.

## The PlanetPress Suite Button

The PlanetPress Design Button replaces the File menu from previous versions, and provides access to the File menu options.

The available menu options in the PlanetPress Suite Button are:

- **New**: Closes the PlanetPress Design Document that is currently opened and creates a new PlanetPress Design Document, with a single page and no objects.
- **Open**: Displays the dialog to open an existing PlanetPress Design Document. Supported documents are PlanetPress 4 to PlanetPress Design 7. See "Open a Document" (page 46).
- **Save**: Saves the current PlanetPress Design Document under the same name. If the file is new and has not been saved, the Save As dialog is displayed instead. See "Save a Document" (page 47)
- **Save As**: Saves the current PlanetPress Design Document under a new name. It does not overwrite any existing document, unless an existing file is selected and overwritten manually by the user.
- **Import**:
    - **PlanetPress 3 Documents**: PlanetPress 3 documents cannot be opened directly and need to be converted to the newer PlanetPress Design 7 format. Use this menu option to do this automatically.
    - **FSL...** : Import a Xerox Form Description Language (FSL) file.
- **Preview** : Opens the Preview dialog. See "Preview a Document On Screen" (page 661).
- **Print** : Opens the Print dialog. See "Print a Document Preview" (page 665).
- **Print using a Windows Driver** : Opens the standard Windows Print Dialog, which prints in non-optimized format but lets you print on any printer, whether postscript or not. See "Printing Using a Windows Driver" (page 668).
- **Create SoftProof** : Opens the Soft Proof dialog, which lets you create a PDF file of a maximum of 10 pages which bears no watermark whatever your activation status. See "Generate a Soft Proof" (page 668).
- **Batch Send To** : Opens the Batch Send To wizard, which lets you send multiple PlanetPress Design Documents to any number of printers, PlanetPress Suite Workflow Tools or iWatch servers. See "Perform a Batch Conversion and/or Installation" (page 674).
- **Send To** (see Install a Document) :
    - **Printer** : Sends your document to a PostScript printer.
    - **PlanetPress Suite Workflow** : Sends your document to PlanetPress Suite Workflow Tools.
    - **Host** : Sends your document to and iWatch or Codehost BrightQ server.

- **Set Password** : Protects the editing of the document with a password. The password is required when opening the document. Setting the password without entering any password will remove it and unprotect the document.
- **Close**: Closes the PlanetPress Design Document that is currently opened and creates a new document, with a single page and no objects. Closing the current document is the same as creating a one.
- **Recent Documents**: Displays a list of the 9 most recently opened PlanetPress Design documents. Click on any of them to open it.
- **Select Language**: Click to display the language selection dialog, which changes the PlanetPress Design interface language.
- **Preferences**: Displays the User Preferences dialog.
- **Exit**: Closes the PlanetPress Design program window. See .

## Open a Document

You can open the document from PlanetPress Design or from Windows. Documents created using PlanetPress Design6 as well as those created with versions 4 and 5 can be opened directly. Those documents that were created using PlanetPress Design 3 must be imported.

When opening PlanetPress 4 documents created with earlier releases of this software, some style changes may not be imported correctly. When this occurs, you can fix the problem by first opening the documents with a later release PlanetPress 4, by saving it and then by opening it again with PlanetPress Design6. In some cases you may also have to reapply the style changes before opening the document with PlanetPress Design 6.

To open a document edited with PlanetPress Design version 4 to 6:

1. From the **PlanetPress Design Button**, choose **Open**.
2. In the **Open Document** dialog box, navigate to the PlanetPress Design document you want to open and click **Open**. By default, PlanetPress Design looks for documents with the **PP6** extension. If the document was created with either version 4 or 5, select the corresponding extension in the File of type box.
3. If the document is password protected, enter the correct password in the dialog box displayed by PlanetPress Design.
4. If the sample data file associated with the document is more recent than the copy in the document, confirm whether or not to update the copy in the dialog box displayed by PlanetPress Design.
5. If any of the attachments associated with the document are more recent than the copy in the document, confirm whether or not to update the copy in the dialog box displayed by PlanetPress Design.
6. If any of the static image resources associated with the document are more recent than the copy in the document, confirm whether or not to update the copy in the dialog box displayed by PlanetPress Design.
   The first page of the document appears in the Page area of the program window. The sample data associated with the document appears in the Data Pane.

To open a document edited with PlanetPress Design version 3:

1. From the **PlanetPress Design Button**, choose **Import | PlanetPress Design 3 Document**.
2. In the **Import PlanetPress Design 3 Document** dialog box, navigate to the PlanetPress Design document you want to open and click **Open**.
3. If the document is password protected, enter the correct password in the dialog box displayed by PlanetPress Design.
4. If the sample data file associated with the document is more recent than the copy in the document, confirm whether or not to update the copy in the dialog box displayed by PlanetPress Design.
5. If any of the attachments associated with the document are more recent than the copy in the document, confirm whether or not to update the copy in the dialog box displayed by PlanetPress Design.
6. If any of the static image resources associated with the document are more recent than the copy in the document, confirm whether or not to update the copy in the dialog box displayed by PlanetPress Design.
   The first page of the document appears in the Page area of the program window. The sample data associated with the document appears in the Data Pane.

## Save a Document

To save a document:

- From the **PlanetPress Design Button**, choose **Save**.
  If the document has been saved at least once, PlanetPress Design copies the last save of the file to a file bearing the name of the document with the extension .**BAK**, and then saves the document.

To save a document under a new name:

1. From the **PlanetPress Design Button**, choose **Save As**.
2. Enter the new name under which you want to save the document, and then click **Save**.

## Save and Open a Document Template

To save a document as a PlanetPress Design template:

1. From the **PlanetPress Design Button**, choose **Save As**, and enter a name for the document.
2. In the Save as type field, select **PlanetPress Design 7 Template** and click **Save**.
   The document is saved as a template with the file extension *tpl*7.

To open a PlanetPress Design template:

1. From the **PlanetPress Design Button**, choose **Open**.
2. In the Files of type field, select **PlanetPress Design 7 Template (*.tlp7)**.
3. Browse to select the template of your choice, and click **Open**.

## Change the Interface Language

PlanetPress Design can be used in multiple languages, and the list of available languages grows as we translate the software. The first time you use PlanetPress Design, it starts in the language used for the installation. You may change this setting as often as you like, but you need to restart the application every time you do so.

To change the language used by PlanetPress Design program:

1. Click the **PlanetPress Suite Button**, then click **Select Language**.
   The **Select Language** dialog box appears. This box lists all the languages that can be used by PlanetPress Design as well as the "Use System Default Locale" checkbox,.
2. Select the desired language and option.
3. Click OK.

**Use System Default Locale**: Select to mirror your language settings, as defined in the Regional and Language Options of the Windows Control Panel. This option is typically used to enter and process information in non-European languages. It is only enabled when English is selected as the program language.

> If you plan to enter and process information in non-European languages, you should know that PlanetPress Suite uses codepages when storing and retrieving information (a codepage is a mapping used to convert back and forth the letters and numbers used by humans to the numeric characters used by computers). By default, codepage 1252 is used for Latin languages (good for Afrikaans, Basque, Catalan, Danish, Dutch, English, Faroese, Finnish, French, Galician, German, Icelandic, Indonesian, Italian, Malay, Norwegian, Portuguese, Spanish, Swahili and Swedish) and codepage 932 is used for Japanese.

## Exit PlanetPress Design

Once you are done working on your PlanetPress Design document, you can close the program. You may exit the PlanetPress Design in any of the following ways:

1. From the **PlanetPress Design Button**, choose **Exit**.
2. Click the **X** at the top-right corner of PlanetPress Design.
3. Press **ALT+F4** on your keyboard.
4. Right-click on the PlanetPress Design button in your task bar, and select **Close**.

If your document has been changed, a dialog appears asking you if you want to save the file. Choose one of the following options:

- **Yes**: Save all unsaved work and then exit. If the document is new, the Save As dialog appears to let you choose a file-name. If the document was existing, it saves under the same name.
- **No**: Exit without saving any unsaved work.
- **Cancel**: Cancels the exit request and returns to the document.
- **Always save before closing**: Check to have PlanetPress Design always save any unsaved work automatically before exiting. The option has no effect when clicking on Cancel. The option can be changed in the Settings windows. See "Notification Messages Preferences" (page 69).

# The Quick Access Toolbar

The PlanetPress Suite Workflow Tools Quick Access Toolbar is displayed, by default, on the right side of the PlanetPress Suite Button and provides one-click shortcuts to commonly used functions and features. You can add as many buttons as you want to the Quick Access Toolbar and remove them at will.

**To add a new button to the Quick Access Toolbar:**

1. Locate the button you want to add in one of the tabs of the Ribbon
2. Right-Click on the button
3. Select **Add to Quick Access toolbar**.

**To remove a button from the Quick Access Toolbar:**

1. Locate the button you want to remove in the Quick Access Toolbar.
2. Right-click on the button
3. Select **Remove From Quick Access toolbar**.

To move the Quick Access Toolbar below or above the Ribbon:

1. Right-click on the Quick Access Toolbar, or click on the downwards arrow at the rightmost end of the Quick Access Tool-bar.
2. Click on **Show Quick Access Toolbar Below the Ribbon** or **Show Quick Access Toolbar Above the Ribbon**, depending on where you want it.

> The Quick Access Toolbar buttons cannot be moved or re-ordered. If you wish to re-order them, you will need to remove all the buttons and re-add them in the desired order.

# The PlanetPress Design Ribbon

The PlanetPress Design Ribbon replaces the main menu and toolbars of previous versions, and centralizes commands, organizing them into a set of **Tabs**, each **Tab** containing **Groups** of **Controls**. Each tab on the Ribbon displays the commands that are most relevant to a given feature set. For instance, the **Objects** tab in PlanetPress Design is used to draw any of the supported objects.

You can minimize the Ribbon by right-clicking on it and selecting **Minimize the Ribbon** or by double-clicking on any of the tabs.

You can also customize the Ribbon's color scheme in the **User Options** window.

PlanetPress Design's **Ribbon** has five tabs: the **Home** tab, the **View** tab, the **Page Layout** tab, the **Tools** tab and the **Help** tab. Each one of these tabs contains a series of groups, each group holding a number of controls.

- The **Home** tab includes the **Tools**, **Clipboard**, **Document** and **Objects** groups.
    - The **Tools** group contains:
        - The **Select Tool** is used to select objects on the document page.
        - The **Hand Tool** is used to move the document page.
        - The **Zoom Tool** is used for zooming in and out of the Page area.
    - The **Clipboard** group contains the typical Windows-based editing controls: **Cut**, **Copy**, **Paste**, **Select All**, **Delete**.
    - The **Document** group contains the document objects controls, used to add objects to the document: **Page**, **Style**, **Condition**, **Global Variable**, **Global Function**, **Metadata Field**, **Image Resource** and **Attachment**.
    - The **Objects** group contains the page objects controls, used to add objects to the selected page: **Text**, **Data Selection**, **Picture**, **PlanetPress Talk, N-Up Printing**, **Address**, **Box**, **Shape**, **Barcode** and **Business Graphic**.
      Some tools in the **Objects** controls lead to dropdown menus that let you make selections among often-used versions of those objects. This is the case with the **Box**, **Shape**, **Business Graphics** and **Barcode** objects. These menus can be opened by clicking the arrowhead visible at the bottom of each control.

- The **View** tab includes the **Zoom**, **Navigate** and **Show/Hide** groups.
    - The **Zoom** group contains the zoom controls: **Zoom In**, **Zoom Out**, **Zoom Factor**, as well as **Fit Page Width** and **Fit in window**.
    - The **Navigate** group contains a data page box you can use to move forward and backward in the sample data file, and tools to move forward and backward in your document one page at a time.
    - The **Show/Hide** group contains four controls to display or hide any of the four panes; the **Document Structure** area, the **Object Inspector** pane, the **PlanetPress Talk Messages** pane and the **Data** Pane.
- The **Page Layout** tab includes the **Arrange**, **Lock/Unlock** and **Duplicate** groups.
    - The **Arrange** group contains the **Alignment** and **Order** controls, allowing to align and reorder objects on a document page.
    - The **Lock/Unlock** group contains controls to lock or unlock a single object on a document page, or every objects on a document page.
    - The **Duplicate** group contains controls to duplicate, and duplicate and pack objects on a document page.
- The **Tools** tab includes the **Data**, **Advanced**, **Managers**, **Application** and **PressTalk Messages** groups.
    - The **Data** group contains:
        - The **Add New Data** control allows you to associate multiple sample data files with your document.
        - The **Open Active Data** loads the active sample data in the Data selector.
        - The **Save Active** allows you to save a local copy of the active sample data file.

- The **Set as Background** control allows you to set a PDF sample data file as the background for the selected document page.
- The **Advanced** group contains:
    - The **Data Capture** control triggers the selected capture tool and allows to grab incoming data.
    - The **Convert to PlanetPress Talk** control converts the selected object to a PlanetPress Talk object, using PlanetPress Talk code.
    - The **Refresh Metadata** control reloads the metadata file associated with the active sample data file. (**Important Note**: When a user-defined emulation is used with metadata, results and behavior are unknown and unsupported. For instance, refreshing the metadata file may cause the document to crash and/or corrupt. For this reason, it is strongly advised to create backup copies of your documents before-hand.)
- The **Managers** group contains:
    - The **Printer Utilies** control displays the Printer information dialog box.
    - The **Virtual Drive Manager** control loads the PlanetPress Suite Virtual Drive.
    - The **Access Manager** control loads the Access Manager, allowing to grant/remove permissions to hosts.
    - The **Install PostScript Font** control allows to install a PostScript font into your PlanetPress Suite installation.
- The **Application** group contains:
    - The **Hex Viewer** control, used to load PlanetPress Suite's Hexadecimal Viewer.
    - The **Image Downloader** control, used to send image resources to a printer.
    - The **Check for updates** control, used to update the current PlanetPress Design version.
    - The **Launch Upgrade Wizard** control, used when migrating from a previous PlanetPress Design version.
    - The **Document Utilies** control, used to acces the Global Function Library Manager.
    - The **Graybar Wizard** control, used to generate grabar reports.
- The **PlanetPress Talk Messages** group contains the **Save Error Log** and the **Clear Messages** controls, used to interact with the **PlanetPress Talk Messages** pane.
- The **Help** tab includes the **Help** and **Activation** groups.
    - The **Help** group contains the **User Guide**, the **Reference Guide** and the **About** controls, used to access online documentation and version information.
    - The **Activation** group contains the **Software Activation** and the **Printer Activation** controls, used to enter activation codes for either the software or a given device.

## Undo and Redo Commands

You cannot undo any move of an object or group that were performed using the mouse or shortcuts. Undo/Redo functionality supports up to 100 levels of undo operations.

Avoid using Undo/Redo to undo or redo a database emulation. If you want to modify the database emulation, re- create the emulation to ensure database integrity and accurate results.

To undo a command or a sequence of commands:

1. From the **Quick Access Toolbar**, click **Undo**.
2. Repeat step 1 as many times as necessary to move backwards through the sequence of commands.

To reverse the effect of one or more undo commands:

1. From the **Quick Access Toolbar**, click **Redo**.
2. Repeat step 1 as many times as necessary to move backwards through the sequence of undo commands.

**Navigate Data Pages**

To move through the pages in a sample data file using a Data page box:

1. Locate a Data page box in the toolbar in the PlanetPress Design Program window.
2. Click in the Data page box and press **SHIFT**+**PAGE UP** to move forward one page or **SHIFT**+**PAGE DOWN** to move backward one page.

To move through the pages in a sample data file, in the Data Selector:

- In the Data Selector, press **SHIFT**+**PAGE UP** to move forward one page or **SHIFT**+**PAGE DOWN** to move backward one page.

# Document Structure Area

The Document Structure area is a hierarchical representation of all the elements in your document. You can use the expand/collapse buttons in the hierarchy to expand and collapse the top level folders of the hierarchy as well as the contents of each page, and the contents of any group of objects.

The Document Structure area contains all of the elements used in your document.

The following options are available in all contextual (right-click) menus, though some options may be disabled or have no effect on some elements of the Document Structure area:

- **Cut**: Cuts the object under the mouse location and places it in the clipboard.
- **Copy**: Places a copy of the object under the mouse location in the clipboard.
- **Paste** Removes the object located in the clipboard and places it on the page at the current mouse location. Has no effect if the clipboard does not contain an object.

> ⚠️ You should never try to paste a PostScript attachment. If you need multiple copies of a postscript attachment, simply rename the one you have an re-add it to your document.

- **Delete**: Deletes the object under the mouse location.
- **Rename**: Changes the display name of the object under the mouse location. Note that this changes the display name, not the PlanetPress Talk ID which is used to call the object from a PlanetPress Talk script.
- **Align**: Aligns multiple selected objects, either horizontally or vertically.
- **Order**: Changes the top-down order of the object, so it is either "on top" or "behind" other objects.
- **Group**: Groups multiple selected objects together.
- **Ungroup**: Removes the object under the mouse location from any group in which it is located. Has no effect on objects not in a group.
- **Add**: Displays a menu of any object or element you can add to your document, including objects, pages, conditions, variables and functions. This menu has the same effect as using the Home tab of the PlanetPress Design Ribbon to add these objects and document elements.
- **Properties**: Displays the properties of the object under the mouse location.

To expand or collapse elements in the Structure area:

- Click the expand/collapse button to the left of the element you want to expand or collapse.

## Selecting and moving elements

You can select objects in the Structure area, as well as drag and drop any element in the Structure area to reposition it in the hierarchy. Note that in the case of pages and objects, the position of the element within the hierarchy influences the order in which it executes:

- **Normal pages** are always processed from top to bottom, the first page on the top will print first, followed by the second one. This can be controlled using a runpage. For more information see ["Setting Up Pages" (page 117)](#).
- **Objects** on a page are processed from top to bottom and displayed on the page successively from the back towards the front. This means that the last object of the page will always appear on top of all the objects before it.
- **Conditions** are evaluated from top to bottom. A condition that depends on another one or multiple other conditions must be placed after the conditions on which it depends.
- **Metadata fields** are also evaluated from top to bottom. A metadata fields that contains another metadata field value must be placed after the field on which it depends.
- **Global Functions** are also evaluated from top to bottom. A function that calls another function or multiple other functions must be placed after the functions on which it depends.

To select elements in the Document structure area, do one of the following:

- Click an element to select it, and then **CTRL**+click each subsequent element you want to add to the selection. Note that you can only add elements **at the same level in the Structure area hierarchy**.
- Use **SHIFT**+**UPARROW** and **SHIFT**+**DOWN ARROW** to add the next element above or below the currently selected elements, to the selection. **CTRL**+click a selected element to remove it from the selection.
- **SHIFT**+click an element to select all elements **at the same level in the Structure area hierarchy** between it and the last element selected.
- Click one of the elements that appears in the Structure area at the level of the element(s) you want to include in the selection. Then click and drag to draw a marquee around the elements you want to move.

To select all elements at the same level in the Structure area hierarchy:

1. In the Structure area, click on one of the elements that occupies the level in the Structure area hierarchy whose elements you want to select.
2. Choose **Home** |**Clipboard** | **Select All** or press **CTRL**+**A**.
   All elements at that level in the Structure area hierarchy appear highlighted in the Structure area. If the elements are objects or groups, those objects and groups also appear highlighted in the Page area.

To move an element in the Structure area:

1. In the Structure area, select the element or elements you want to move.
2. Drag the elements. As you drag, a ghost image of the element(s) you are moving follows the pointer, and either a blue bar appears (if the pointer is outside the name of an element) or the element is highlighted in black (if the pointer is over the name of an element) to indicate the current drop target. The pointer changes to indicate whether the current drop target is legal (⬚) or illegal (🚫). Note that if you pause over a collapsed group or page as you drag, PlanetPress Design expands that group or page.
3. Release the elements at a legal drop target.
   PlanetPress Design moves the element(s) to the new position.

# Add a Metadata Field

To add a metadata field:

- Choose **Home** | **Document** | **Metadata Field**, or right-click on **Metadata Fields** in the *Document Structure Pane* and click **Metadata Field**.
- Enter the **PlanetPress Talk ID**, the unique identifier for your new metadata field (Note that Metadata objects do not possess a **Display Name** attribute).
- Choose the **Level** where your new field should be created in the metadata structure.
- Use the **Fields** list to add metadata fields, providing a **Value**, a **Condition (optional)** and a **Create action**.
- Click **OK**.

**Metadata Field Properties**

- **PlanetPress Talk ID**: A unique identifier for the metadata field. The name you choose should be both descriptive and unique, cannot begin with a number, and can contain only the following ASCII characters: underscore (_), upper and lower case letters of the alphabet, all digits 0 through 9.If you use an underscore in the name, it should not appear as either the first or last character of the name as this may cause internal conflicts in the software. Names are case-insensitive and must be unique; no two elements in a document can have the same name. Names can be a maximum of 50 characters in length. Finally, PlanetPress Talk variable and command names are reserved words; you cannot use any of these reserved words as a name.
- **Level**: The hierarchical level where the new field will be created.
- **Fields**: Attributes for the actual field(s) to be created:
    - **Value**: The actual value of the new field(s).
    - **Condition**: A global or local condition to determine whether or not the field will be created.
    - **Create action**: How the field(s) creation will behave if the currently added field already exists.
- **Add button**: Used to add a new element to the current metadata field, thus creating an array of metadata values within this field.
- **Delete button**: Used to remove a given value from the metadata field.
- **Move up/Move down buttons**: Used to modify the order of the values within the current metadata field.

## Add a Document Page

There are 3 ways to add a new blank page:

1. Go to the **Home** tab, then click the **Page** icon.
2. Right-click on the **Pages** folder in the **Document structure**, then click the **Page** icon.
3. Right-click on any element under the **Pages** folder of the **Document structure**, then click on **Add** then **Page** icon.

New pages will always appear at the end of the document structure

## Edit the Order of Pages

To edit the order of pages in a document:

- In the Structure area, select the page whose position you want to edit, drag it to its new position and release. As you drag, a blue bar appears indicating the new position the page will occupy if you release the mouse button at that point.

## Duplicate a Page

To duplicate one or more page(s):

1. In the Structure area, select the page(s) you want to duplicate.
2. Choose **Page Layout** | **Duplicate** or use the **Ctrl-D** keyboard shortcut to duplicate the selected page(s).

## Delete a Page

To delete one or more page(s):

1. In the Structure area, select the page(s) you want to delete.
2. Either use the **Delete** key or right-click and choose **Delete.**
   If no elements in the document reference any of the selected page(s), PlanetPress Design performs the deletion.
   If any elements in the document reference any of the selected pages, PlanetPress Design prompts you to define how you want to handle the deletion of each of the referenced pages.

To use the Page Deletion dialog box:

1. Adjust the options to reflect how you want PlanetPress Design to handle the deletion request. The name of the page you selected for deletion appears in the title bar of the Page Deletion dialog box, and the list of elements that reference it appear on the right of the dialog box.
   **Replace reference by**: Select to delete the page and to replace all references to it with a reference to another page in the document.
   **Pages available**: Select the page you want to use as the replacement reference. When you delete the page, PlanetPress Design replaces all references to the deleted page with a reference to the page you select here. You can use the Page button to create a new page to add to this list.
   **Page button**: Click to create a new page.
   **Delete**: Select to delete the page and all document elements that reference it.
2. Click **OK**.

## Object Layering Order

Is the order in which objects appear in the Structure area significant?

The order in which objects appear in the Structure area determines the order in which the document executes them, as well as which object appears on top when two objects overlap.

You can think of each object as existing on its own layer. As you add objects, you add layers. The most recently added object always occupies the foreground layer. In the Structure area, the topmost object occupies the background layer.

If two objects overlap, the one closest to the foreground appears on top. If you have several objects you want to place either on top of or under another object or group, it may be useful to group them into a single unit and then edit the layering order of the group.

To edit the layering order of objects:

- In the PlanetPress Design main menu, choose **Page Layout** | **Arrange** | **Order** and then choose any of the following:
  **Bring Forward:** Move the selected objects or groups forward one layer graphically. This also results in moving the selected objects or groups downward one spot in the Document Structure.
  **Send Backward:** Move the selected objects or groups backward one layer graphically. This also results in moving the selected objects or groups upward one spot in the Document Structure.
  **Bring to Front:** Move the selected objects or groups to the very front of all layers graphically. This also results in moving the selected objects or groups to the bottom of the Document Structure.
  **Send to Back:** Move the selected objects or groups to the very back of all layers graphically. This also results in moving the selected objects or groups to the top of the Document Structure.

## Group and Ungroup Objects and/or Groups

A group is a set of objects that you group together to treat as a single unit. A group may be composed of individual objects, groups, or both. There is no restriction on the type of object you can include in a group, or on the number of different types of objects you can include in a single group. However, any objects and groups you want to include in the group must exist on the same page of a document.

To create a group:

1. Select the objects and/or groups you want to include in the group.
2. Choose **Page Layout** | **Arrange** | **Group**.

To ungroup a group:

1. In the Structure area or the Page area, select the group you want to ungroup.
2. Choose **Page Layout** | **Arrange** | **Ungroup**.

# Data Pane

The Data Pane is a view on your input data, and one of the ways you can select data to include in your document. The Data Pane is a component of the Data Selector and its contents and appearance reflect the options set for it in the Data Selector.

You can drag and drop a data selection directly from the Data Pane into the Page area to quickly create a data selection object, a bar code object, or a business graphic object.

In the Document Structure area or in the Page area, when you select a data selection, bar code, or business graphic that uses a Contiguous data selection, PlanetPress Design highlights the data selection for that object in the Data Pane.

For more information on the data pane and data selections, see the chapter on Data in PlanetPress Design.

# Object Inspector

The Object Inspector displays the properties of the element currently selected in the Document structure area or in the Page area. If you select several elements, the Object Inspector displays values only for those properties that are common to all the selected elements. If you select an image resource, the Object Inspector also displays the image.

The number of elements currently selected appears in the upper left of the Object Inspector, and you use the Object Inspector to view and edit properties.

**To edit properties Using the Object Inspector:**

1. Select the element (object, page, document, condition, style, attachment) you want to change. To change multiple elements, do a multiple selection (**CTRL**-click).
2. In the Object Inspector, select the property you want to edit. If you selected multiple elements, only those properties common to all of the selected elements appear in the Object Inspector. If necessary use the scroll bar and/or the expand/collapse button.
3. Edit the property using the appropriate method, as described below:
   - If the property lets you enter any value, type in the new value and press **ENTER**.
   - If the property is associated with a list of possible values, select the new value from the list.

- If the property lets you select any value, click the selection button located to the right of the value field. For example, if the button is for a PlanetPress Talk before, PlanetPress Talk after, or PlanetPress Talk code property, click it to launch the PlanetPress Talk Editor.

**To expand or collapse a group:**

- In the Object Inspector, click the expand/collapse button for the group.

# Workspace Area

The Worskpace always displays the currently selected, or active, page of your document. It displays the result of what your page should look like when it previews or print. The data shown in the Workspace is that of the current data page, and it will only show objects that are not hidden through a condition.

The Workspace is separated in a few important elements:

- The white area is your actual page, and is always the size of the page as set in the page's properties (for example, Letter or A4). If you change the page size, the white space in the Workspace will also be updated to reflect the new size.
- The light gray line inside the page indicates the outside margin, where the printer drive or PPD has determined that data will never be printed. If you place objects within this margin, it may not print since it is outside of the printer-defined margins.
- The dark gray area around the page is never printed as it is outside of physical page area. However, you can still place objects in this area and they will be executed. For example, you could place statistics you would like to see when opening the document, or run some PlanetPress Talk code without being seen.
- The Rulers (top and left) are used to help guide where you place objects. While you move your mouse in the Workspace, lines appear in the rules to display the horizontal and vertical position on the page. You can also use the rule to place guides on your page.

## Using the Select Tool in the Workspace

The Select Tool is the default tool that is used when opening or creating a PlanetPress Design document, and is most likely to be the one you will use most of the time. The Select Tool lets you select and move objects around in the Workspace.

To select any object in the Workspace:

- Place your mouse over the object and click on it.

To select multiple objects in the Workspace, use either methods:

- Create a box around all the objects you want to select by using drag & drop from an empty area on the page.
- Click on the first object you want to select and, while holding down the CTRL button, click on any number of other objects to add to your selection.
  Selected objects have a red border around them, as well as resize handles in the corner and edges.

To move an object or objects in the Workspace:

- First, select one or more objects to move.
- Drag the object or objects to the new location.

To resize an object or objects in the Workspace:

- First, select one or more objects to resize.
- Using any of the resize handles (squares at the corners or edges of your selection), click and move the handle to resize. If multiple objects are selected, they will all be resized together.

## Zoom in and out of the Workspace

You can use the zoom tools to zoom in and out of the Workspace, in order to see more of your document (for precise editing and placement) or less (to see a broader picture). The zoom factor can be any value from 10 to 1000 percent (%).

**To set a specific zoom level:**

- In the **PlanetPress Design Ribbon**, go to the **View** tab.
- In the **Zoom** group, click in the **Current zoom factor** box and enter the new zoom.
  The Workspace will zoom at the top-left corner, not in the center.

**To zoom in or out using the Zoom toolbar:**

- In the **PlanetPress Design Ribbon**, go to the **View** tab.
- In the **Zoom** group, click **Zoom In** or **Zoom Out**.
  The Workspace will zoom at the top-left corner, not in the center.

**To zoom in or out using the Zoom tool pointer:**

1. In the **PlanetPress Design Ribbon**, go to the **Home** tab.
2. In the **Tools** group, click on **Zoom Tool**.
3. To zoom in, **click** anywhere in the Workspace.
4. To zoom out, **right-click** anywhere in the Workspace.
   The Workspace will zoom in (or out) at the location where your mouse is located.

**To zoom in our out using the mouse scroll wheel (when available):**

1. Place your mouse in the Workspace
2. Hold down the **CTRL** key on your keyboard.
3. To zoom in, scroll the **mouse wheel up** (away from you).
4. To zoom out, scroll the **mouse wheel down** (towards you).
   The Workspace will zoom in (or out) at the location where your mouse is located.

## Pan the Workspace

The Hand tool can be used to move the Workspace left, right, up and down, in order to see a different area of the Workspace. This is especially useful when you are zoomed in close on your page and simply want to move either way slightly.

To pan the Workspace using the Hand Tool:

1. In the **PlanetPress Design Ribbon**, go to the **Home** tab.
2. In the **Tools** group, click on **Hand Tool**.
3. Click and hold your mouse anywhere in the Workspace, and move it in any direction to move the Workspace.

To pan the Workspace up and down using the mouse scroll wheel:

1. Place your mouse in the Workspace
2. To scroll up, scroll the **mouse wheel up** (away from you).
3. To scroll down, scroll the **mouse wheel down** (towards you).

To pan the Workspace left and right using the mouse scroll wheel:

1. Place your mouse in the Workspace
2. Hold down the **SHIFT** key on your keyboard.
3. To scroll right, scroll the **mouse wheel up** (away from you).
4. To scroll left, scroll the **mouse wheel down** (towards you).

## Use Guides

Guides are vertical or horizontal reference lines that you place on a page to help you accurately align and position elements. You can set a guide for an individual page or for all pages of a document. Guides cannot print and thus never appear on the output of a document.

You can select the Snap to guides option in the User Options dialog to have the edge of an object or group automatically snap to a guide when it approaches a guide. See "Miscellaneous Preferences" (page 72).

To create a guide:

- Click in the top ruler (to create a vertical guide) or the left ruler (to create a horizontal guide) at the point at which you want to create the guide.
  A dark blue triangle appears in the ruler, and the guide appears in the document. By default the guide appears only on the document page on which you create it. If you want to have it appear on all document pages, see To edit the properties of a guide:.
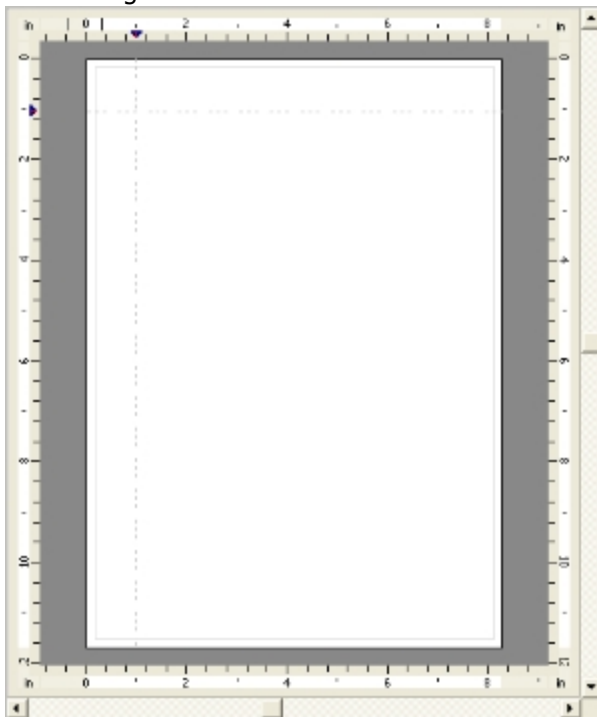


Document page with one vertical and one horizontal guide

To reposition a guide:

- In the ruler, click and drag the blue triangle of the guide to the new position.

To edit the properties of a guide:

1. In the ruler, double-click the blue triangle of the guide you want to edit.
   The Guide dialog box appears.
2. Edit the properties of the guide.
   **Position**: Enter the ruler position for the guide. Units are as set in the User Options dialog.

**Display on all pages**: Select to have this guide display on all pages in the document. Clear to have the guide display only on the page on which you created it.

3. Click **OK** to exit the Guide dialog box.
   PlanetPress Design updates the guide to reflect the new settings.

To delete a guide:

- Click and drag the black triangle of the guide inside the Page area, and release.
  PlanetPress Design deletes the guide.

## Right-Click Menu in the Workspace

You can use the right-click menu in the workspace to add, remove, move and edit objects on your page.

The right-click menu contains the following menu actions:

- **Cut**: Cuts the object under the mouse location and places it in the clipboard. Has no effect if right-clicking on the page itself or outside of the page.
- **Copy**: Places a copy of the object under the mouse location in the clipboard. Has no effect if right-clicking on the page itself or outside of the page.
- **Paste**: Removes the object located in the clipboard and places it on the page at the current mouse location. Has no effect if the clipboard does not contain an object.
- **Delete**: Deletes the object under the mouse location. Has no effect if right-clicking on the page itself or outside of the page.
- **Rename**: Changes the display name of the object under the mouse location. If right-clicking on the page, will rename the page. This is equivalent to changing the name of the object in its Basic Attributes properties.
- **Align**: Aligns multiple selected objects, either horizontally or vertically. Has no effect when selecting a single object or the page itself.
- **Order**: Changes the top-down order of the object, so it is either "on top" or "behind" other objects. If right-clicking on the page, will change the order of the page itself.
- **Group**: Groups multiple selected objects together. Has no effect when selecting a single object or the page itself.
- **Ungroup**: Removes the object under the mouse location from any group in which it is located. Has no effect on objects not in a group. If right-clicking on the page, will remove the page from any group in which it is located in the Document Structure.
- **Add**: Displays a menu of any object or element you can add to your document, including objects, pages, conditions, variables and functions. This menu has the same effect as using the Home tab of the PlanetPress Design Ribbon to add these objects and document elements.
- **Properties**: Displays the properties of the object under the mouse location. If right-clicking on the page, will display the properties of the page.

## Detailed Directions

This section includes the following procedures:

## Debug an Object or Group

You can use the Messages area in either the Program window or the Object Preview to help you debug an object. The PlanetPress Talk Editor also offers a number of features for debugging any PlanetPress Talk code you enter in a document.

## Select Objects and/or Groups

To select all objects and groups at the same level in the Structure area hierarchy:

1. In the Structure area or in the Page area, click on one of the objects or groups that occupies the level in the Structure area hierarchy whose objects and groups you want to select.
2. Choose **Home** | **Clipboard** | **Select All**.

To select one or more objects and/or groups in the Structure area:

1. In the Structure area, if necessary, expand the page containing the objects and/or groups you want to select.
2. Click an object or group, and then **CTRL**+click each subsequent object or group you want to include in the selection. **SHIFT**+click to select all objects between the currently selected object or group and the last object or group selected. **CTRL**+click an object or group a second time to remove it from the selection.

To select one or more objects or groups in the Page area:

- In the Page area, click the object or group you want to select.

## Lock and Unlock Objects and Groups

To lock an individual object/group:

- Select the object or group, and, in the Object Inspector, set the **Selectable** property to False. You can set it to False either by selecting False in its list or by double-clicking its value to toggle between True and False.

To unlock an individual object/group:

- Select the object or group, and, in the Object Inspector, set the **Selectable** property to True. You can set it to True either by selecting True in its list or by double-clicking its value to toggle between True and False.

To lock one or more objects/groups in a single operation:

1. If necessary, in the Structure area, select the page containing the object(s) and/or group(s) whose Selectable property you want to clear.
2. Do either of the following:
   - **To lock only a selected set of objects/groups:** Verify that all the objects you want to lock are selected, then choose **Page Layout** | **Lock/Unlock** | **Lock Selected Objects**.
   - **To lock all objects/groups:** Choose **Page Layout** | **Lock/Unlock** | **Lock All Objects on Page**.

To unlock one or more objects/groups in a single operation:

1. If necessary, in the Structure area, select the page containing the object(s) and/or group(s) you want to unlock.
2. Do either of the following:
   - **To unlock only a selected set of objects/groups:** In the Structure area, select each object you want to unlock, then choose **Page Layout** | **Lock/Unlock** | **Unlock Selected Objects**.
   - **To unlock all objects/groups:** Choose **Page Layout** | **Lock/Unlock** | **Unlock All Objects on Page**.

## Reposition Objects and/or Groups

To reposition objects and/or groups:

1. Select the objects and/or groups you want to reposition.
   A red rectangle appears around the selected objects and groups. It contains eight resize handles, one on each corner, and one in the middle of each edge of the rectangle.

2. Press **CTRL**+**ARROW** to move the objects and/or groups. You set the magnitude of the move that occurs with each press of the **ARROW** key in the User Options dialog. If you are repositioning an object/group within a group, after you select it in the Structure area you must click it in the Page area to use the **CTRL**+**ARROW** shortcut.

## Resize Objects and/or Groups

The following describes how to resize text objects, picture objects, and data selection objects.

### Text Objects

If you resize a text object that uses a background box, it is important to understand how the background box resizes with respect to the text in the object. First, the margins and indents remain constant when you resize a text object. Thus the top edge of the text is always the same distance from the top edge of the background box, and the left edge of the text is always the same distance from the left edge of the background box. What changes are the right and bottom edges of the text relative to the right and bottom edges of the background box. The change that occurs depends on whether word wrap is on or off, and whether the Dynamic height box is selected in the Basic attributes of the text object.

| Word wrap: | Dynamic height: | Resize behavior: |
|---|---|---|
| on | cleared | Background box resizes horizontally to maintain the relationship between the right indent and the right edge of the background box. It resizes vertically without regard for the bottom edge of the text. Thus in this case text may extend below the bottom edge of the background box. |
| on | selected | PlanetPress Design dynamically adjusts the height of the background box to accommodate all of the text. In this case you cannot manually adjust the height of the text object; you can only resize the width. |
| off | cleared | Background box resizes without regard for either the right or bottom edges of the text. Text may extend beyond either or both of the right and bottom edges of the background box. |

### Picture Objects

The ability to resize a picture object, and the effect of the resize on the image that object references and on the height and width of the picture object, depends on the Fit setting selected for the image.
If you selected a Fit setting of Constant resolution, you cannot resize the picture object.
In the case of any of the other Fit settings (Constant height, Constant width, and Best fit), PlanetPress Design carries out the resize in accordance with the Fit setting. For example, if you selected Constant height, when you resize the picture object, PlanetPress Design scales the image such that its height reflects the new height of the picture object. It then adjusts the width of the picture object to the new width of the image. Note that when you resize a dynamic image, you are resizing a single image. If the dynamic image references more than one image, and the images are different sizes, you should consider the effect the resize has on all of the images referenced by the dynamic image.

### Data Selection Objects

If you defined the data selection in a data selection object using PlanetPress Talk expressions, you cannot resize the object along any of the edges defined by those expressions.
You cannot resize a data selection object that contains a Contiguous data selection in database emulation, along the horizontal axis. A Contiguous data selection in a database emulation cannot span more than one field.
To resize one or more objects and/or groups:

1. Select the object(s) and/or group(s) you want to resize.
2. Press SHIFT+DOWN ARROW or SHIFT+UP ARROW to respectively increase or decrease the size of the object or group along its bottom edge. You set the magnitude of the resize that occurs with each press of the ARROW key in the Preferences dialog.

## Delete Objects and/or Groups

To delete one or more objects and/or groups:

1. Select the objects and/or groups you want to delete.
2. Choose **Home** | **Clipboard** | **Delete**.
   If no elements in the document reference any of the selected objects/groups, PlanetPress Design performs the deletion.
   If any elements in the document reference any of the selected objects/groups, PlanetPress Design prompts you to define how you want to handle the deletion of each of the referenced objects/groups. More precisely, for each referenced object/group, it displays the Object Deletion dialog box. You use that dialog box to set the deletion options and perform the deletion.

To use the Object Deletion dialog box:

1. Adjust the options to reflect how you want PlanetPress Design to handle the deletion request. The name of the object/group you selected for deletion appears in the title bar of the Object Deletion dialog box, and the list of elements that reference it appear on the right of the dialog box.
   **Replace reference by**: Select to delete the object/group and to replace all references to it with a reference to another of the objects/groups in the document.
   **Objects available**: Select the object/group you want to use as the replacement reference. When you delete the object/group, PlanetPress Design replaces all references to the deleted object/group with a reference to the object/group you select here. You can use the Objects button to create a new object to add to this list.
   **Objects button**: Use to create a new object. Click and choose the object you want to create. PlanetPress Design creates the new object and selects it in the Objects available list.
   **Delete**: Select to delete the object/group, and all document elements that reference it. All document elements that ref-

2. Click **OK**.

## Align Objects and/or Groups

To align objects:

1. Select the reference object or group and then each of the objects and/or groups you want to align with it.
2. Use one of the following to align the objects and/or groups:
   Alignment toolbar
   Click the appropriate horizontal and/or vertical alignment in the **Alignment** toolbar.
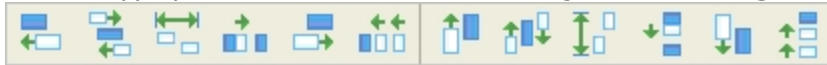
   Alignment dialog box
   Choose **Edit** | **Align** to display **t**he Alignment dialog box. In the Alignment dialog box, select a horizontal and vertical alignment option and click **OK**.
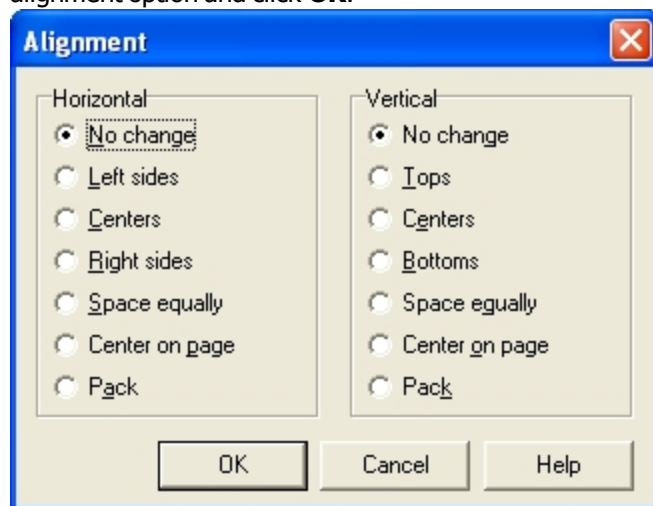
## Rotate Objects and/or Groups

You rotate an object or group in PlanetPress Design by adjusting the Angle property in the Basic attributes properties of the object/group.

## Duplicate Objects and/or Groups

To duplicate one or more objects and/or groups:

1. Select the objects and/or groups you want to duplicate.
2. Choose **Page Layout** | **Duplicate** | **Duplicate**  to have PlanetPress Design duplicate the selected objects and/or groups using the displacement and data page offsets set in the User Options dialog box.

## Snap or Unsnap Objects and/or Groups

To snap one object or group to another using the properties dialog boxes:

1. Determine the snapping point you want to use for each object and/or group, and whether or not you require an offset. In other words, determine the relationship you want to establish between the two.
2. In the Structure area, examine the relationship between the two and if necessary, edit their positions in the Structure area.

3. If, in the object/group of the two that is lower in the Structure area hierarchy, the **Snap to previous object** property is already selected and the proper offset set, proceed to step 9.

4. In the Structure area or the Page area, double-click that object or group to display the properties dialog box for the object/group that is **lower** in the Structure area hierarchy. This is the object/group PlanetPress Design moves when it performs the snap.

5. In the properties dialog box, click **Snapping points**.
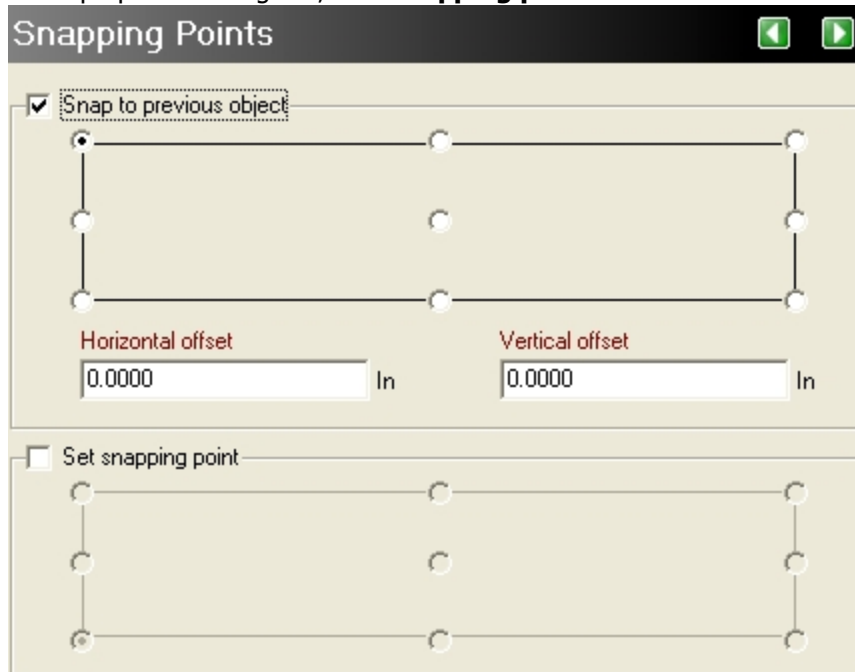


6. Select **Snap to previous object** and click the snapping point you want this object/group to use when it snaps to an object/group that precedes it in the Structure area. A dot in the center of the snapping point indicates it is selected.
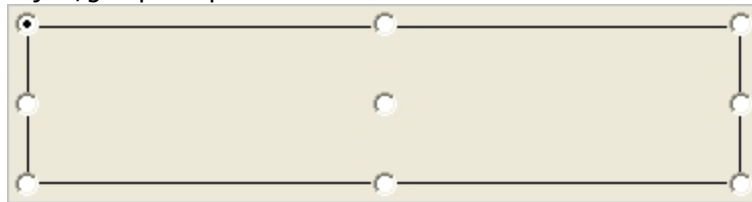


A. Selected snapping point

7. If necessary, set offsets for the snapping point.
   **Horizontal offset**: Enter a horizontal offset for the snapping point.
   **Vertical offset**: Enter a vertical offset for the snapping point.

8. In the properties dialog box, click **OK**.
   If the **Set snapping point** property is already set in the object/group of the two that is higher in the Structure area hierarchy, you have completed this procedure. Otherwise, proceed to step 9.

9. In the Structure area or the Page area, double-click that object or group to display the properties dialog box for the second object/group. This object/group is the one **higher** in the Structure area hierarchy.

10. In the properties dialog box, click **Snapping points**.

11. Select **Set snapping point** and click the snapping point you want this object/group to use for all objects that snap to it.

12. In the properties dialog box, click **OK**.

To snap one object or group to another using the Object Inspector:

1. Determine the snapping point you want to use for each object and/or group, and whether or not you require an offset. In other words, determine the relationship you want to establish between the two.

2. In the Structure area, examine the relationship between the two and if necessary, edit their positions in the Structure area.
3. If, in the object/group of the two that is lower in the Structure area hierarchy, the **Snap to previous object** property is already selected and the proper offset set, proceed to step 6.
4. Select the object/group that is **lower** in the Structure area hierarchy. This is the object/group PlanetPress Design moves when it performs the snap.
   The Object Inspector displays the properties of that object/group.
5. In the **Object Inspector**, set the following snapping points properties. If you have set the Object Inspector to use groups, all of these properties appear in the Snapping points group.
   - Set **Snap to previous object** to True. If an object/group that precedes it in the Structure area has its **Set snapping point** property set, PlanetPress Design snaps this object/group to it.
   - Set **Snapping point before** to the snapping point you want this object/group to use when it snaps to an object/group that precedes it in the Structure area (Top left, Top middle, Top right, for example).
   - Set **Snapping point horizontal offset** to the horizontal offset you want to use for the snapping point. This is useful when you want a horizontal offset between this object/group and the one to which you are snapping it. Units are as set in the User Options dialog box.
   - Set **Snapping point vertical offset** to the vertical offset you want to use for the snapping point. This is useful when you want a vertical offset between this object/group and the one to which you are snapping it.
     If, in the object/group of the two that is higher in the Structure area hierarchy, the **Set snapping point** property is already selected, you have completed this procedure. Otherwise, proceed to step 6.
6. Select the object/group that is higher in the Structure area hierarchy.
7. In the **Object Inspector**, set the following snapping points properties for this object/group:
   - Set **Snapping point after** to the snapping point you want this object/group to use for all objects/groups that snap to it.
   - Set **Snapping point after** to True.

To unsnap one object or group from another using the properties dialog box:

1. In the Structure area or the Page area, double-click that object or group to display the properties dialog box for the object/group (of the two snapped together) that is **lower** in the Structure area hierarchy. This is the object/group PlanetPress Design moved when it performed the snap.
2. In the properties dialog box, click **Snapping points**.
3. Clear **Set snapping point before**.
4. In the properties dialog box, click **OK**.
   If the object/group you just unsnapped was not the only one snapped to the other object/group, you have completed this procedure. Otherwise, proceed to step 5 to clear the **Set snapping point** property of the second of the two objects/groups.
5. In the Structure area or the Page area, double-click that object or group to display the properties dialog box for the second object/group. Of the two objects that were snapped together, it is the one **higher** in the Structure area hierarchy.
6. In the properties dialog box, click **Snapping points**.
7. Clear **Set snapping point**.
8. In the properties dialog box, click **OK**.

To unsnap one object or group from another using the Object Inspector:

1. Select the object/group that is **lower** in the Structure area hierarchy. This is the object/group PlanetPress Design moved when it performed the snap.
2. In the **Object Inspector**, set the **Snap to previous object** property to False.
   If the object/group you just unsnapped was not the only one snapped to the other object/group, you have completed this procedure. Otherwise, proceed to step 3 to clear the **Set snapping point** property of the second of the two objects/groups.

3. Select the object/group that is **higher** in the Structure area hierarchy.
4. In the **Object Inspector**, set the **Set snapping point after**property to False.

## Copy Values of Properties between Objects and/or Groups

To copy values of properties between objects:

1. Select the reference object or group. This is the object/group that contains the properties whose values you want to copy.
2. In the Structure area or the Page area, **CTRL**+click each object and/or group to which you want to copy the property value.
3. In the **Object Inspector**, click the property whose value you want to copy.
4. If necessary, edit the value of the property.
5. Repeat step 3 through step 4 for each property whose value you want to copy.
6. Click anywhere outside the Object Inspector to enter the last edit you made.

To copy values using the cut and paste features:

1. Select an object in the document.
2. Choose **Edit** | **Copy**.
3. Select the second object.
4. Choose **Object** | **PasteProperty**.
5. Select the desired properties and click **OK**.

## Adding image resources to your document

1. Do any of the following:
   - **To add a single static image** In the Structure area, click on the image resource and drag it into the Page area. Alternatively, if the image resource is already selected and its image is visible in the Object Inspector, click the image in the Object Inspector and drag it into the Page area.
   - **To add a single static image that references a page of a multi-page PDF** In the Structure area, click on the multi-page PDF image resource, and then, in the Object Inspector, in the Page box, navigate to the page of the PDF you want to use as the static image. Finally, in the Object Inspector, click on the image and drag it into the Page area.
   - **To add one or more static images** In the Structure area, select the image resources and then click and drag them into the Page area. You can do any of the following to select image resources in the Structure area. Click on the first image resource, then **CTRL**+click each additional image resource, or **SHIFT**+click to select all image resources between the one you click and the last one selected. You can also click and drag a marquee around the image resources. To remove an item from the selection, **CTRL**+click it a second time.
   - **To add a static image using a clipped region**: With a PDF sample data file loaded in the Data pane, select the region to clip and create an image object with, then **right**-click and drag the selected region to the selected page. Release the right mouse button and choose **Insert Clipped Region** to insert the selected PDF region.
2. Release the image resource(s) in the Page area.
   PlanetPress Design creates a new picture object for each image resource you dragged into the Page area. Each new picture object appears in the Page area and its name appears in the Structure area hierarchy. The height and width of the picture object reflect the height and width of the image resource.
3. If necessary, adjust the properties of each new picture object, as described in step 3 through step 11 of To add a static image:.

# Messages Area

The Messages area displays messages from the PlanetPress Talk Converter, and is useful when you add PlanetPress Talk objects, or objects that include PlanetPress Talk statements to your document. Any errors the PlanetPress Talk Converter encounters in your code, it displays in the Messages area. These include converter errors, run error messages, and any debugging strings you instructed your code to output using the PlanetPress Talk **outputdebugstring()** command.

PlanetPress Design also uses the Messages area to report any problems it had carrying out certain operations and how it resolved those problems (for example, opening a document that references unavailable fonts, or importing a PlanetPress Design 3 document that used the same name for two different elements).

The type of message appears to the right of the message itself, and you can set a distinct color for converter errors, for run errors, and for debugging strings. This makes it easier to quickly distinguish one message type from another. Notification messages always appear in black.

You can double-click a message to have PlanetPress Design display the source of the error.

PlanetPress Design clears the Messages area automatically when you open an existing document, or import an FSL form or PlanetPress Design 3 document. You can also clear the area manually.

To find the source of an error:

- In the Messages area of the PlanetPress Design Program window, double-click the error.

To clear some or all of the messages in the Messages area:

- Right-click in the Messages area and choose either **Clear all messages** or a given message category (Converter Error, Run Error, OutputDebugString, or Notification).

To save the contents of the Messages area to a file:

1. Right-click in the Messages area and choose **Save PlanetPress Talk Messages**.
2. In the **Save PlanetPress Talk Messages** dialog, navigate to the folder in which you want to save the file, enter a file name for the saved file, and click **Save**.

## Adding External Resources

You can drag and drop one or more files from Windows directly into any of the following areas of the PlanetPress Design Program window: the Structure area, the Data Pane, the Page area, or the Object Inspector (when the Object Inspector is displaying an image resource).

You can drag and drop any of the following file types. PlanetPress Design determines the type of file from the file name extension, and accepts specific file formats for each file type.

| File type: | PlanetPress Design accepts files with file name extension: |
|---|---|
| PlanetPress Design document (versions 3 and up) | PP3, PP4, PP5, PP6 and PP7 |
| Image | BMP, EPS, JPEG or JPG, PDF, PNG, TIF or TIFF |
| Attachment | PRN, PS |
| Sample data file | CSV, DAT, DB, DBF, MDB, PDF, TXT, XML |

If you drag and drop a PlanetPress Design6 document, PlanetPress Design opens that document. If you drag and drop a PlanetPress Design document from an earlier version, PlanetPress Design imports that document.

For all other file types, the type of file (image, attachment, or sample data file) and the area of the program window in which you drop it determine what PlanetPress Design does with the file. Whether you drag and drop a single file or multiple files can also have an impact on how PlanetPress Design treats the file or files. The procedure here describes the behavior of each area of the PlanetPress Design Program window when you drag and drop image, attachment, or sample data files into it.

To drag and drop files into PlanetPress Design:

1. Select the file that you want to drag and drop into PlanetPress Design, and then drag it over the appropriate area of the program window.
2. When pointer is over an area where a drop is permitted, release the mouse button.
   dragging Image Files
   - **Into the Structure area:** PlanetPress Design creates an image resource for each image file.
   - **Into the Page area:** If you dropped multiple image files, PlanetPress Design creates an image resource for each image file. If you dropped a single image file, PlanetPress Design also creates a picture object containing that image on the current page. If you dropped a single multi-page PDF, PlanetPress Design creates a single image resource for the PDF, a new document page for each page of the PDF, and, on each new document page, a picture object that contains a page of the PDF.
   - **Into the Data pane:** PlanetPress Design creates an image resource for each image file.
   - **Into the Object inspector:** If you dropped multiple image files, PlanetPress Design creates an image resource for each image file. If you dropped a single image file, PlanetPress Design replaces the image resource currently displaying in the Object Inspector with the one you dragged and dropped.
   dragging attachments
   - **Into the Structure or Page area or into the Data pane:** PlanetPress Design creates an attachment resource for each attachment file.
   dragging sample Data files
   - **Into the Structure or Page area:** PlanetPress Design replaces the sample data file currently associated with the document with the one you dropped in the Structure or Page area, and it updates the Data Pane to reflect the contents of the new sample data file. Note that if you drop several sample data files in the Structure or Page area, PlanetPress Design adds each in the order in which it receives them, each subsequent file replacing the previous one as the sample data file. The last one it adds is the one that becomes the sample data file associated with the document. There is no way to control the order in which PlanetPress Design receives multiple sample data files. If the sample data file you drag and drop does not have a filename extension PlanetPress Design recognizes, PlanetPress Design opens the Data Selector and displays the contents of the file using a line printer emulation. You can then select a different emulation if necessary.
   - **Into the Data pane:** PlanetPress Design replaces the sample data file currently associated with the document with the one you dropped in the Data Pane, and it updates the Data Pane to reflect the contents of the new sample data file. Note that if you drop several sample data files in the Data Pane, PlanetPress Design adds each in the order in which it receives them, each subsequent file replacing the previous one as the sample data file. The last one it adds is the one that becomes the sample data file associated with the document. There is no way to control the order in which PlanetPress Design receives multiple sample data files. If the sample data file you drag and drop does not have a filename extension PlanetPress Design recognizes, PlanetPress Design opens the Data Selector and displays the contents of the file using a line printer emulation. You can then select a different emulation if necessary.

## PlanetPress Design Preferences

The PlanetPress Design program lets you configure a variety of options to customize how the application looks, and what default values are used when creating a new document and importing images.

The preferences are located in the PlanetPress Design Preferences window, accessible through the Preferences button in the PlanetPress Design button. Those preferences are:

- Behavior
  - "Notification Messages Preferences" (page 69)
  - "Image Resources Preferences" (page 70)
  - "Color Preferences" (page 71)
  - "Object Duplication Preferences" (page 71)
  - "Miscellaneous Preferences" (page 72)
- Editor
  - "Editor Preferences" (page 73)
  - "Display Preferences" (page 74)
  - "Color Preferences" (page 75)
- Appearance
  - General Preferences
  - "Object Inspector Preferences" (page 75)
  - "Document Structure Area Preferences" (page 76)
  - "Rulers Preferences" (page 76)
  - "Form Pages Preferences" (page 77)
  - "Compiler Messages Preferences" (page 77)
- Document default values
  - "Document and Pages Preferences" (page 77)
  - "Image Resources Preferences" (page 78)

## Notification Messages Preferences

To set the Notification Messages options:

1. From the **PlanetPress Design Button**, choose **Preferences**.
2. If necessary, expand **Behavior**, and click **Notification messages**.
   **Invalid Undo/Redo action warning**: When enabled, issues a message when an Undo or Redo operation is not possible.
   **Text object's tab support warning for right to left text**: When enabled, issues a message when tabulation with right to left text is not supported in text fields.
   **Text object's PlanetPress Talk before/after paragraph support warning for right to left text**: When enabled, issues a message when PlanetPress Talk before/after paragraph with right to left text is not supported in text fields.
   **Text object's UTF8 style warning**: When enabled, issues a message when a UTF8 style is not supported in text fields.
   **Text object's indent support warning for right to left text**: When enabled, issues a message when indent support with right to left text is not supported in text fields.
   **PDF version warning**: When enabled, issues a message when PDF version is greater than 1.5 (Adobe® Acrobat® 6) and identifies PDF files with which PlanetPress Design may have compatibility issues, such as PDF's that contain transparent objects. PlanetPress Design supports PDF 1.3 (Adobe® Acrobat® 4), 1.4 (Adobe® Acrobat® 5) and 1.5 (Adobe® Acrobat® 6) formats without transparent objects. PlanetPress Design does not support PDF 1.6 format (Adobe® Acrobat® 7).
   **Send to Host notification of success**: When enabled, a notification message appears indicating the document transfer is successful. If an error occurs, a notification message appears whether or not this option is enabled.
3. Set the Notification Messages options.

**Invalid name**: Select to have PlanetPress Design display an error message when you enter an invalid name for an element. Clear to prevent the display of the error message.

**Invalid added resources**: Select to have PlanetPress Design display a message reporting whether it added all selected resources successfully when you add resources to a document.

**Save before closing**: Select to have PlanetPress Design prompt for confirmation to save an unsaved document before closing it. This option applies only to documents that have been saved at least once during a session.

**New version of attachments**: Select to have PlanetPress Design monitor all attachment resources and prompt for confirmation to update its copy of an attachment resource when the original file changes. If the original attachment resource file changes between sessions, PlanetPress prompts for confirmation to update the document's copy the next time you open the document.

**New version of data file**: Select to have PlanetPress Design monitor the sample data file and prompt for confirmation to update its copy when the original data file change. PlanetPress prompts for confirmation either when you return to PlanetPress after making the changes to the data file, or when the focus changes to an element that references the data.

**New version of picture resource**: Select to have PlanetPress Design monitor all image resources and prompt for confirmation to update its copy of an image resource when the original file changes. If the original image resource file changes between sessions, PlanetPress prompts for confirmation to update the document's copy the next time you open the document. Note that if you edit an image resource from PlanetPress Design, the edits apply only to the copy of the image internal to the document, and have no effect on the original, external image file.

**Invalid PPD notification**: Select to have PlanetPress Design display an error message when you attempt to add a PostScript Printer Description (PPD) file that is either not a valid PPD, or is a PPD that does not use Level 2 PostScript or higher.

**Document name too long for host**: Select to have PlanetPress Design issue a warning when you try using file names that contain more characters than the host computer can support.

**PostScript Language Level 3 warning**: Select to have PlanetPress Design display a warning when you select Line art in the Image quality box. The warning reminds you that Line art works only with PostScript Language Level 3 printers.

**ASCII emulation warning**: Select to have PlanetPress Design warn you if your document uses an ASCII emulation and the settings of the Optimized PostScript Stream option and the Read in binary mode option may cause discrepancies between the visual appearance of the preview and the visual appearance of the document in the Page area of the PlanetPress Design Program window. This warning may occur for both hard copy and on-screen previews.

**Invalid Undo/Redo action warning**: When enabled, issues a message when an Undo or Redo operation is not possible.

**Double-byte text tab support warning**: When enabled, issues a message when tabulation with double-byte fonts is not supported in text fields.

**PDF version warning**: When enabled, issues a message when PDF version is different and identifies PDF files with which PlanetPress Design may have compatibility issues, such as PDF versions 1.4 and 1.5 formats. PlanetPress Design supports only PDF 1.3 format (Adobe® Acrobat® 4) and PDF 1.4 format without transparent objects. PlanetPress Design does not support PDF 1.5 format (Acrobat 6). PDF 1.4 (Acrobat 5) format files that contain transparent objects are not supported.

**Send to Host notification of success**: When enabled, a notification message appears indicating the document transfer is successful. If an error occurs, a notification message appears whether or not this option is enabled.

4. Click **OK**.

## Image Resources Preferences

To set the Image resources options:

1. From the **PlanetPress Design Button**, choose **Preferences**.
2. If necessary, expand **Behavior,** and click **Image resources**.

3. Specify the cache size PlanetPress Design uses for image resources.
   **Picture cache size**: Specify the size of the cache, in megabytes, that PlanetPress Design uses for image resources. The minimum and maximum values you can enter are 64 megabytes and 1000 megabytes respectively. The image resource cache is an amount of RAM set aside to hold image resources that the document uses. The cache improves performance by providing faster access to images as you work.
   **Image editor (bitmap)**: Specify the image editing application you want to use to edit bitmapped image resources (image resources in GIF, TIFF, PNG or BMP format) in PlanetPress Design. Enter the path name to the application, or click the Browse button to the right of the box to browse and select the application. If you do not define an application, PlanetPress Design launches the default image editor defined in Windows. Note that for image resources in either PDF or EPS format, PlanetPress Design launches the default editor defined for those formats in Windows.
4. Specify the image editor you want to use to edit bitmapped image resources.
5. Click **OK**.

## Color Preferences

To set the Color options:

1. From the **PlanetPress Design Button**, choose **Preferences**.
2. If necessary, expand **Behavior,** and click **Color**.
3. Adjust the Color type option.
   **Color type**: Select the color model you want PlanetPress Design to use when it displays the numerical value of a color. This is also the model the Color Picker displays by default when you open it.
   **Color management active**: Select to enable color management in PlanetPress Design. You can select Color management active only if you selected a color profile in both Monitor profile and Printer profile. When you work with color in PlanetPress Design, PlanetPress Design assumes the color is in the color space of the selected printer profile. To represent the color on-screen, it converts the color from the color space of the selected printer profile to the color space of the selected monitor profile. This ensures the on-screen color closely represents the one the printer prints.
   **Monitor profile**: Select the device color profile for the monitor of the computer on which you are running PlanetPress Design. The profiles available are the ICC-compliant monitor profiles PlanetPress Design finds on the system. This option has an effect only when you select Color management active.
   **Printer profile**: Select the device color profile for the printer on which you intend to execute the document. The profiles available are the ICC-compliant printer profiles PlanetPress Design finds on the system. This option has an effect only when you select Color management active.
4. Adjust the color management options.
5. Click **OK**.

> Color profiles are not specific to PlanetPress Design or your document. They are installed on your operating system and are generally downloadable from your hardware manufacturer's website when they are not provided with the hardware's drivers.

## Object Duplication Preferences

To set the Object Duplication options:

1. From the **PlanetPress Design Button**, choose **Preferences**.
2. If necessary, expand **Behavior**, and click **Object duplication**.
3. Adjust the Object Duplication options.
   **Duplicate style**: Select where you want PlanetPress Design to place the copy of any object, group, or selection of objects and/or groups you duplicate. Select Pack vertically to have PlanetPress Design align the copy along the Y axis, under and flush with the most recent copy. Select Pack horizontally to have PlanetPress Design align the copy along the X axis, to the right of and flush with the most recent copy. Select Relative displacement to have PlanetPress Design

align the copy using the specified horizontal and vertical offset values. If you select this option, you must enter values for the vertical and horizontal displacements.

The menu items **Edit | Duplicate and Pack Horizontally** and **Edit | Duplicate and Pack Vertically,**override this setting.

**Vertical relative displacement value**: Enter the displacement for the copy along the Y axis. Units are as set in the User Options dialog. This option is available only when you select Relative displacement as the duplicate style.

**Horizontal relative displacement value**: Enter the displacement for the copy along the X axis. Units are as set in the User Options dialog. This option is available only when you select Relative displacement as the duplicate style.

Data SELECTION Offsets

**Row/child record**: Set the number of lines you want to advance in the data page with each duplication of an object or group. In the case of a database emulation, this is the number of records you want to advance in the record set with each duplication of an object or group. This permits each copy to display a distinct selection of data. Note that this offset does not work with objects that use custom data selections.

**Column**: Set the number of columns you want to advance in the data page with each duplication of an object or group. This permits each copy to display a distinct selection of data. Note that this offset does not work with objects that use custom data selections. This offset has no effect in database emulation.

4. Click **OK**.

## Miscellaneous Preferences

The Miscellaneous options include zoom factors, the unit of measure, the object selection mode, a snap to guides option, and options that determine object dialog behavior.

1. From the **PlanetPress Design Button**, choose **Preferences**.
2. If necessary, expand **Behavior**, and click **Miscellaneous**.
3. Adjust the object selection mode.
   **Object selection mode**: Use to determine where you must position the pointer on an object or group in order to select it. This is particularly important when you are selecting overlapping objects and/or groups. Select bounding box to select an object or group when the pointer is anywhere inside or along the perimeter of the bounding box of the object. Select Pixel to select an object or group only when the pointer is over the perimeter of the object.
4. Adjust how objects and groups behave with respect to guidelines.
   **Snap to guidelines**: Select to have objects and groups automatically snap to the nearest guide (or guides) when you move or resize those objects and groups on the page. More precisely, when you move or resize an existing object or group, as the edge of the object or group approaches a guide, the edge jumps to lie flush with that guide. When you select this option, existing objects and groups in the document remain in their current position. Only if you subsequently move or resize them, will they snap to the guides.
5. Adjust the zoom factors. PlanetPress Design uses these values to determine the new zoom when you zoom in or out on the document page.
   **Zoom factor**: Set the zoom factor PlanetPress Design uses when you zoom in or out on the document page using the Zoom tool in the **Objects** toolbar, the Zoom in and Zoom out tools in the **Zoom** toolbar, and the plus (+) and minus (-) keys on the numeric keypad. Values can range from 10 to 1000.
   **Fine zoom factor**: Set the fine zoom factor PlanetPress Design uses when you zoom in using **SHIFT**+ the plus key (+) on the numeric keypad, or zoom out using **SHIFT**+ the minus key (-) on the numeric keypad.
6. Adjust the unit of measure you want to use in PlanetPress Design.
   **Unit of measure**: Set the unit of measure. This determines the units PlanetPress Design uses for the rulers and for all measurements outside of PlanetPress Talk. Units can be centimeters or inches.
7. Adjust the nudge factor.
   **Nudge factor**: Set the magnitude of the resize that occurs when you resize an object or group using keyboard shortcuts. Units are as set in the unit of measure.
8. Adjust the dialog box options.

**Remember last dialog box position**: Select to have PlanetPress Design remember the last screen position of each type of dialog box, and, if the dialog box is resizable, the last size of the dialog box. Resizable dialog boxes include the Text/Box properties dialog box and the Data Selector.

**Remember last dialog box page**: Use this to control the area that is displayed when you open a dialog box. Select to have PlanetPress display the area that was visible when you last closed a dialog box of that type.

9. Click **OK**.

## Editor Preferences

To set the Editor options:

1. From the **PlanetPress Design Button**, choose **Preferences**.
2. If necessary, expand **PressTalk Editor,** and click **Editor**.

**Auto indent mode**: Select to automatically position the insertion pointer under the first non-blank character of the preceding line when you press ENTER.

**Insert mode**: Select to use Insert mode and clear to use Overwrite mode. When you enter text in Insert mode, the existing text shifts to accommodate it. In Overwrite mode, the text you enter overwrites the existing text. You can also press INSERT to toggle between Insert and Overwrite mode.

**Use tab character**: Select to use the tab character instead of spaces to represent tabs in the program file. Clear to use spaces to represent the tabs. You must clear the Smart tab option to use this option.

**Smart tab**: Select to use smart tabs. A smart tab advances with reference to the preceding line. More precisely, it advances to align with the first non-blank character it encounters on the preceding line, from its current position forward. You must clear the Use tab character option to use Smart tabs.

**Optimal fill**: Select to optimize the indent of every auto-indented line by minimizing the number of space and/or tab characters it uses. You must select both Auto indent mode and Use tab character to use this option.

**Backspace unindents**: Select to move the insertion pointer to the previous indentation level when you press BACKSPACE.

**Cursor through tabs**: Select to move one by one through the spaces of tabs using the left or right arrow keys. Clear to have the arrow keys treat the tab as a single character. You must select Use tab character to use this option.

**Group undo**: Select to set the undo feature of the Editor to undo the last group of editing commands entered. An editing command is defined as a mouse click, a press on ENTER, or a press on any other key. A group of editing commands is a sequence of a single type of editing command. Clear to set the undo feature to undo only the last command entered.

**Cursor beyond EOF**: Select to make it possible to position the pointer beyond the end of the last line of code in the program. This is useful if you prefer to enter code by clicking the pointer at any point in the Editor, rather than by manually entering carriage returns or lines of code to advance to that point. For example, you might know you need a for() loop near the middle of your code. If you select Cursor beyond EOF, you can click in the middle of the Editor and enter the first line of the for() loop, without having to enter carriage returns to arrive at that point. Clear to prevent being able to position the pointer beyond the end of the last line entered to date in the Editor.

**Cursor beyond EOL**: Select to make it possible to position the pointer beyond the last character entered to date on a line of code. This is useful if you prefer to enter code by clicking the pointer at any point on the line, rather than by manually entering characters or spaces to advance to that point. Clear to restrict the places on the line where you can position the pointer, to only the characters entered to date on that line.

**Keep trailing spaces**: Select to preserve any trailing spaces that occur at the end of a line.

**Persistent blocks**: Select to have any text you enter immediately after selecting a block of code appended to that block of code as part of the selection. When you select this option, you can also use the arrow keys to move within the code without affecting the selected code. You must select the Enable selection option to use the Persistent blocks option.

**Overwrite blocks**: Select to have any text you enter immediately after selecting a block of code replace that block of code. You must clear Persistent blocks and select Enable selection for this option to have an effect.

**Enable selection**: Select to permit the creation of selections in the Code area. You can create a selection by clicking and dragging the pointer over a portion of code or by double-clicking to highlight the word or line that appears under the pointer. You can cut, copy, paste, and print selections. If you also select Enable dragging, you can drag selections to reposition them in the code.

**Enable dragging**: Select to permit dragging and dropping a selection to reposition it in the program. This option works only if you also select Enable selection.

**Enable search highlight**: Select to highlight the search term match found in the code when you perform a search. Clear to prevent the highlighting. In both cases, the pointer appears just after the last character of the search term match.

**Double click line**: Select to highlight the complete line of code when you double-click that line. Clear to highlight only the word that appears under the pointer.

**Find text at cursor**: Use to set the behavior of the Find dialog box. Select to automatically copy the word under the pointer into the Text to find box when you open the Find dialog box. Clear to prevent the copy. If no previous search terms appear in the Text to find box, the Editor performs the copy regardless of whether this option is selected or cleared.

**Block indent**: Enter the number of spaces to jump for each block indent. The default is 2 and the maximum is 16. It is common to make the Block indent agree with the tab stops you enter in the Tab stops box. You perform a block indent by selecting a region of code and then pressing **CTRL**+**SHIFT**+**I** (to indent the code to the right) or **CTRL**+**SHIFT**+**U** (to move the code to the left).

**Tab stops**: Use either to set the number of spaces to advance when you enter a tab character or to set a series of tab stops. Enter a single integer to set the number of spaces to advance with each tab. Enter a sequence of two or more integers, each separated by a space, to specify tab stops. The sequence must be in ascending order.

3. Click **OK**.

## Display Preferences

To set the PlanetPress Talk Editor Display options:

1. From the **PlanetPress Design Button**, choose **Preferences**.
2. If necessary, expand **PressTalk Editor,** and click **Display**.
3. Adjust the Display options.
   Display options
   **Editor font**: Use to select the font the Editor uses to display the program code. Select the Use monospace fonts only option to restrict the fonts available to fixed width fonts.
   **Size**: Use to select the font size the Editor uses to display the program code.
   **Use monospace fonts only**: Select to display only fixed width fonts in the Editor font list.
   **Sample**: Displays a preview of the font selected in the Editor font list, at the size selected in the Size list.
   Margin and gutter
   **Right margin**: Select to display a vertical gray bar as a right margin indicator. Use the Right margin position list to set the position of this indicator. This indicator is an on-screen visual reference only. It does not print and does not enforce word wrap on lines that exceed the number of characters set for it.
   **Right margin position**: Enter the position of the right margin indicator, in number of characters, relative to the left margin.
   **Gutter**: Select to have the Editor display a gutter between the Commands and Code areas. Use the Gutter width option to set the width of the gutter. Select the Line numbers on gutter option to display line numbers in this area.
   **Gutter width**: Enter the width, in pixels, of the gutter. Use the list to select a previously entered gutter width.
   **Line numbers on page**: Select to display code line numbers at the left edge of the Code area.
   **Line numbers on gutter**: Select to display code line numbers in the gutter between the Commands and Code areas. Selecting this option has effect only if you selected the Gutter option.
4. Click **OK**.

## Color Preferences

To set PlanetPress Talk Editor Color options:

1. From the **PlanetPress Design Button**, choose **Preferences**.
2. If necessary, expand **PressTalk Editor,** and click **Color**.
   **Element list**: Displays the list of code elements. Use to select the element whose color and/or style attributes you want to adjust. Select None to set the color for all elements that do not appear in the list. To set a color and style attribute for an element, select that element in the Element list, then use the Mapping, Background, and Foreground lists, and the Attributes check boxes to set the color and style attributes for that element. Repeat for each element whose color and style you want to adjust.
   **Mapping**: Use to select a color palette. The color palette determines the colors available in the Foreground and Background color lists.
   **Foreground**: Use to select the foreground color for the selected element.
   **Background**: Use to select the background color for the selected element.
   **Attributes**: Use to set the style attributes for the selected element. You can select any combination of bold, italic, and underlined.
   **PlanetPress Talk label color**: Select the label color for boxes that accept PlanetPress Talk code. The label color you select for these boxes serves to distinguish them from those which do not accept PlanetPress Talk code.
   **PlanetPress Talk background color**: Select the background color for boxes that accept PlanetPress Talk code. The background color you select for these boxes serves to distinguish them from those which do not accept PlanetPress Talk code.
3. Click **OK**.

## Ribbon Preferences

The Ribbon Preferences window can be used to change the color scheme used by PlanetPress Design between three preset themes: Blue, Silver and Black.

Click on any of the themes to select it. You will need to click OK for the change to become apparent.

## Object Inspector Preferences

To set the Object Inspector options:

1. From the **PlanetPress Design Button**, choose **Preferences**.
2. If necessary, expand **Appearance**, and click **Object Inspector**.
3. Adjust the Object Inspector options.
   **Colors**: Use to set the colors of individual Object Inspector components. To set a color, in the Colors list, click the component whose color you want to change.
   **Vertical line 3D**: Select to display the vertical line between property names and their values using a three-dimensional effect.
   **Use groups**: Select to organize the display of properties into groups. Clear to display properties in alphabetical order. When the Object Inspector displays properties in groups, it displays an expand/collapse button to the left of the name of the group that you use to expand or collapse the group.
   **Sunken active property**: Select to use a recessed effect to display the currently selected property.
   **Border active property**: Select to display a border around the currently selected property.
   **Show lines**: Select to display lines between elements.
   **Line Style**: Select a style for the lines.
4. To reset the Object Inspector options to their default values, click **Reset to Default**.
5. Click **OK**.

## Document Structure Area Preferences

You use these options to modify the appearance of the Document Structure area of the PlanetPress Design Program window. The first procedure describes how to set these options and includes a description of each option. The second describes how to reset the options to their default values.

To set the Document Structure Area options:

1. From the **PlanetPress Design Button**, choose **Preferences**.
   The Preferences dialog box appears.
2. If necessary, expand **Appearance**, and click **Document Structure** to display the Document Structure Area options.
3. Adjust the Document Structure area options as necessary.
   **Colors**: Use to set the colors of individual Structure area components. To set a color, in the Colors list, click the component whose color you want to change, and then choose a color from the list below the Colors list. The Structure area updates immediately to reflect the change.
   **Line style**: Select a line style for tree lines and grid lines. Use Show tree lines and Show grid lines to show or hide the lines.
   **Selection rectangle**: Select a style for the selection rectangle that appears when you click and drag inside the Structure area to select one or more elements.
   **Button style**: Select a style for the Structure area expand/collapse buttons.
   **Show tree lines**: Select to display lines that represent the hierarchical relationship between elements. Use the Line style list to select a style for the lines.
   **Show grid lines**: Select to display lines between elements. Use the Line style list to select a style for the lines.
   **Hot track**: Select to have the Structure area underline an element when you pass the mouse over it.
4. Click **OK**.
   PlanetPress Design exits the User Options dialog box and updates to reflect the new settings.

To reset the Structure area to its default appearance:

1. Color, option, planetPress talk editor, Set, user, user options
   The User Options dialog box appears.
2. If necessary, expand **Appearance**, and click **Document structure** to display the Document Structure Area options.
3. Click **Reset to Default**.
4. Click **OK**.
   PlanetPress Design exits the User Options dialog box and resets the Structure area to its default appearance.

## Rulers Preferences

To configure the appearance of the rulers:

1. From the **PlanetPress Design Button**, choose **Preferences**.
2. If necessary, expand **Appearance**, and click **Rulers**.
3. Adjust the Rulers options.
   **Colors**: Use to set the colors of individual ruler components.
   **Flat style**: Select to display the rulers without a bevelled edge.
   **Show rulers all around**: Select to display a ruler along each edge of the page area. Clear to display a ruler only along the left and top edges of the page area.
   **Show minus signs**: Select to display minus signs in front of negative ruler values.
   **Show hairlines**: Select to display hairlines in the ruler that indicate the current position of the pointer on the page.

4.  To reset the rulers to their default appearance, click **Reset to Default**.
5.  Click **OK**.

## Form Pages Preferences

To set the Document Page area options:

1.  From the **PlanetPress Design Button**, choose **Preferences**.
2.  If necessary, expand **Appearance**, and click **Document pages**.
3.  Adjust the border option for document pages.
    **Show imageable area**: Select to display a border around the printable area of the page. This helps ensure you do not inadvertently place an object or part of an object in an area of the page that does not print. The PPD you select determines the printable area of the page. Not all printers can print to the edge of the page.
4.  Adjust the colors for overlays.
    **Overlay/underlay color**: Select the color for overlay pages. The overlay page appears in this color in the Page area.
    **Overlay/underlay border color**: Select the color for the border around overlay pages. The border around the overlay page appears in this color in the Page area.
5.  Adjust the colors for imposed pages.
    **Imposed page color**: Select the color for virtual pages. The virtual page appears in this color in the Page area.
    **Imposed page border color**: Select the color for the border around virtual pages. The border around the virtual page appears in this color in the Page area.
6.  Click **OK**.

## Compiler Messages Preferences

To set Converter Messages options:

1.  From the **PlanetPress Design Button**, choose **Preferences**.
2.  If necessary, expand **Appearance**, and click **Compiler messages**.
3.  Adjust the color you want to use for each type of message.
    **Compile error:** Select the color for any compilation error messages that appear in the Messages area of the Program window or of the Object Preview.
    **Run error:** Select the color for any run error messages that appear in the Messages area of the Program window or of the Object Preview.
    **Warning:** Select the color for any warning messages that appear in the Messages area of the Program window or of the Object Preview.
    **Output Debug String**: Select the color for any output debug string messages that appear in the Messages area of the Program window or of the Object Preview. Consult the *PlanetPress Talk Language Reference* for help with the **outputdebugstring()** command that produces these messages.
4.  Set the behavior of the Messages area when a new message arrives.
    **Show Messages area on new message**: Select to have PlanetPress Design make the Messages area, if it is currently hidden, visible when a new message arrives. Note that if you select this option, and the undocked Messages area appears over the page in the Page area, you cannot close the Messages area if any of the objects or groups on the page issues a converter message when it executes. Each time you attempt to close it, PlanetPress Design redraws the page, executing each of the objects or groups on the page; the compiler messages that result from the execution cause the Messages area to become visible again. In this case you must move the Messages area outside the page to close it.
5.  Click **OK**.

## Document and Pages Preferences

To set the Document and pages options:

1. From the **PlanetPress Design Button**, choose **Preferences**.
2. If necessary, expand **Document default values,** and click **Document and pages**.
3. Adjust the default PPD and form cache default values.
   **Default printer**: Select the PPD file that appears by default in the Designed for property box in the Document properties dialog box.
4. Adjust the default paper size and orientation for the pages of the document.
   **Default paper size**: Select the page size that appears by default in the page size box in the page properties dialog box.
   **Default page orientation**: Select the paper size that appears by default in the Paper orientation box in the Page properties dialog box.
5. Adjust the default options for the Style properties dialog box.
   **Default font type**: Select the type of font that appears by default in the Style properties dialog box when you create a new style. Double-byte or CID-keyed fonts are required for Asian text and data. Also bear in mind that you should use Unicode fonts (UTF8) for Arabic text and data. PostScript fonts are recommended to improve printer performance and reduce file size. The type you select determines the contents and availability of the Default font name list. *Note that using double-byte TrueType fonts for data only works if the Optimized PostScript Stream printing option is turned selected.*
   **Default font name**: Select the name of the font that appears by default in the Style properties dialog box when you create a new style.
   **Default single byte font encoding**: Select the encoding table that appears by default in the Encoding list in the Style properties dialog box.
6. Adjust the default options for the Compilation options dialog box.
   **Default max form item**: Enter the value that appears by default in the Max page item box in the Document properties dialog box.
   **Default max form cache**: Enter the value that appears by default in the Max page item box in the Document properties dialog box.
   **PostScript level**: Select the PostScript level for the converted document (recall that a variable content document is a PostScript program). Select 2 to use PostScript Level 2, 3 to use PostScript Level 3, and PPD if you want PlanetPress Design to determine the level from the PPD selected for the document.
7. Click **OK**.

## Image Resources Preferences

To set the Image resources options:

1. From the **PlanetPress Design Button**, choose **Preferences**.
2. If necessary, expand **Default values** and click **Pictures**.
3. Adjust the options that appear by default in the Picture properties dialog box.
   **Color (DPI)**: Select the resolution that appears by default in the Color resolution property of picture objects.
   **Grayscale (DPI)**: Select the resolution that appears by default in the Grayscale resolution property of picture objects.
   **Monochrome (DPI)**: Select the resolution that appears by default in the Monochrome resolution property of picture objects.
4. Adjust the options that appear by default in the Document properties dialog box.
   **Convert to monochrome**: Select the option that appears by default in the Convert to monochrome box in the Document properties dialog box.
   **Scanline orientation**: Select the option that appears by default in the Scanline orientation box in the Document properties dialog box.
   **Picture quality**: Select the option that appears by default in the Image quality box in the Document properties dialog box.

**Picture compression ratio**: Select the value that appears by default in the Picture compression ratio box in the Document properties dialog box. The value that appears here when you start PlanetPress Design for the very first time indicates a 70% compression of image resources.

5.   Click **OK**.

## Data Selector Display Preferences

The Data Selector Preferences are accessible through the "Sector Options" tab in the "Data" tab of the Data Selector. It controls how text-based data files (such as Line Printer, ASCII and Channel Skip) are displayed in the data selector.
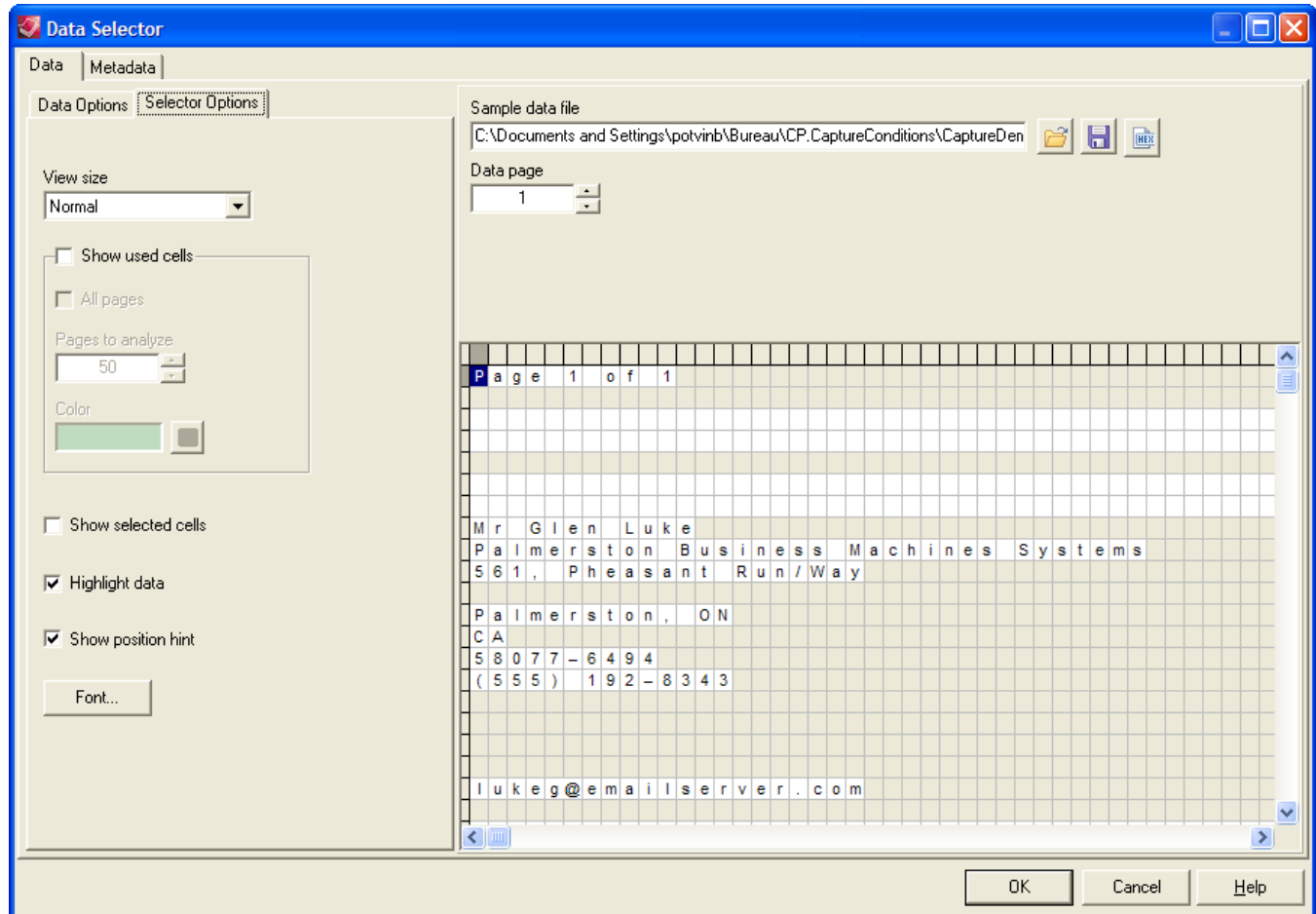


**To adjust the content and appearance of the Data Pane for all emulations except XML and PDF:**

1.   In the Data Selector, click the **Selector Options** tab.
2.   Change the options that modify the appearance and behavior of the Data Pane:
     - **View size**: Use to adjust the size of each cell in the Data Pane, and the amount of visible data that is visible.
     - **Show used cells:** Select this to display in green all cells that contain data. When you select this option, and your document uses any emulation other than database, you use either the All pages or Pages to analyze option to specify the number of data pages to which you want to apply the Show used cells option.
     - **All pages**: Select to apply the Show used cells option to all pages in the sample data file. This option is not available in database emulation.
     - **Pages to analyze**: Use this box to limit the number of data pages to which PlanetPress Design applies the Show used cells option. Enter the number of pages to which you want PlanetPress Design to apply the option, or use the spin buttons to adjust the value. This option is not available in database emulation.

- **Show selected cells:** Select this to display in gray all cells that your document currently references.
- **Highlight data:** Select to have the Data Selector highlight only those cells (or fields) that contain data from the sample data file.
- **Show position hint**: Select to have PlanetPress Design display information about the current mouse position in the Data Pane, under and to the right of the pointer as you move it in the Data Pane. If the mouse is over a current data selection, or is dragging to create a data selection, PlanetPress Design displays the line and column numbers that define the selection, or, in the case of a database emulation, the positions within the record set of the first and last records in the selection. If the mouse is not over a data selection, PlanetPress Design displays the line and column coordinates of the current mouse position), or, in the case of a database emulation, the position of the record within the record set.

3. If necessary, adjust the font the Data Selector uses to display data in the Data Panel for all emulations except XML and PDF.
4. Click **OK**.

**To select the color the Show used cells option will use:**

- Click on the **Select Color** button .

**To set the font the Data Selector uses for all emulations except XML and PDF:**

1. In the **Data Selector**, click the **Selector Options** tab.
2. Click **Select Font**.
3. In the **Font** dialog box, set the font you want PlanetPress Design to use to display the sample data file in the Data Pane.
   - **Font**: Select the font you want to use to display the sample data file in the Data Pane.
   - **Font style**: Select a weight for the font.
   - **Size**: Select the point size for the font.
   - **Sample**: Displays a preview of the font selected in the Font box.
   - **Script**: Select the system-level encoding table you want to use for the font selected in the Font box. The encoding tables available here are those available on the system on which you are running PlanetPress Design, and are distinct from those available when you create a style. While you can edit the encoding table a style uses, you cannot edit the system-level encoding table. If you see discrepancies between the glyphs that represent your sample data file in the Data Pane and those that appear in the data selections on the document page, the source of the discrepancy may be the encoding tables.
4. Click **OK**.

## Dock and Undock Areas of the Program Window

To undock an area:

- Double-click the title bar of a docked area or group (tabbed or stacked areas). In the case of a group, you can undock a single area in the group by double-clicking its tab.

To dock a floating area:

- Double-click the title bar of a docked area or group (tabbed or stacked areas). PlanetPress Design docks the area in its most recent docked position.

To show an area within a tabbed group:

- Click the tab of the area you want to show. If the tab is not visible, use the navigation buttons located to the right of the tabs.

To expand or restore an area within a stacked group:

- Click the area's expand (  ) or restore (  ) button.

To reset all areas to their default docking positions:

- Press **CTRL** when you start PlanetPress Design. Note that this also resets the toolbars to their default position.

## Minimize and Customize the Ribbon

To minimize the **Ribbon**:

1. Right-click anywhere on the Ribbon and choose **Minimize the Ribbon**.


To customize the Ribbon:

1. From the **PlanetPress Design Button**, choose **Preferences**.
2. If necessary, expand **Appearance**, and click **Ribbon**.
3. Select your **Ribbon Color Scheme**.


## Show or Hide Areas of the Program Window

To show or hide a Program window area:

- Choose **View** and then the area you want to show or hide. PlanetPress Design updates the Program window to reflect the requested show/hide.

## Resize the Program Window Area

To resize a Program window area:

- Move the pointer to the edge of an area you want to resize to display the resize pointer and click and drag to resize the area.

# Data in PlanetPress Design

This chapter explains how to work with Data, Sample Data Files, Data Selections and Emulation within your PlanetPress Design documents.

## Sample Data File

A Sample Data file is a file that, while it follows the same structure as the one you will use when actually processing your document, only contains part of the actual data. It is used to create your document while having an example of the data file, and can even be fake, computer-generated data in some cases.

The following are the two criteria for a reliable sample data file:

1. It includes all possible variations on the data that the document may encounter when it executes.
   Things to check for variation include field lengths, the location of decimal points in numeric data, and whether or not a field always contains data.
2. It exactly represents the input data at the moment that data arrives at the printer or a PlanetPress Suite Workflow Tool process. A difference of a single character can result in a document that does not produce accurate results. If your sample data file does not meet this criteria, you end up creating a document that executes with a different input data structure than the one for which you designed it.

The way a sample data file is displayed and how its data pages are separated depends on the emulation selected when opening the sample data file with the Data Selector.

# Capturing Data

To create your document, you need a reliable sample of the variable data you intend to use with the document.

This section describes what a sample data file is and the two criteria that determine its reliability. It also provides general procedures for capturing a reliable sample data file. Consult the **Trigger and Data Capture Guide** for platform-specific data capture procedures.

The data capture tool lets you capture real data to be used as sample data. It can capture data sent to a Windows printer queue or a Serial, LPD or Telnet connection. The captured data can then be saved to a file and used immediately by PlanetPress Design to design or troubleshoot a document. You must correctly configure the output queue on the server if you use the data capture tool. Also, if you want to use the data capture tool, you must correctly set a proper queue on the server to sent an LPR to the IP address of the PC which is running PlanetPress Design as well as for the data capture tool. This rule is true for all possible channels.

## Capture Sample Data Using the Data Capture Tool

Use this procedure to momentarily capture data sent to an input for the purpose of generating a sample data file, as well as to capture data for a document you install on a printer or in PlanetPress Suite Workflow Tool

To capture sample data using the Data Capture tool:

1. In the PlanetPress Design Ribbon, go to the **Tools** tab, then click **Capture Data**.
2. From the dropdown list, select one of the following data inputs ports: LPD, Serial, Telnet, or Windows Queue.
3. Set the options for the capture based on the selected input.
4. Click **Capture** to actually capture the port.
   When PlanetPress Design is ready to receive the data on the selected port, a message box is displayed.
5. Send your job to the appropriate input.
6. To stop the capture, click Stop.
7. To see the log files, click the **Open Log** button. To view the captured data, double-click the corresponding log file.
   Log files are located in *C:\Documents and Settings\[User account name]\Application Data\PlanetPress Suite 7\PlanetPress Design\DataCap.* They are named according to the capture date.
8. Click OK.

## LPD Input

Retrieves data using LPD/LPR. LPD/LPR is a printer protocol that uses TCP/IP to establish connections between printers and workstations on a network. PlanetPress Design provides the LPD service for Windows to receive print jobs through a TCP/IP network without requiring a Window's driver.

**General tab**

- **LPD queue name**: Enter the queue name for the LPD server. This is the queue name you use when sending jobs through an LPR client.

**LPD Input Options tab**

- **Log all Winsock and network messages**: Select to have PlanetPress Design keep a log of all Winsock and other network messages that occur through the LPD service. These are messages related to jobs being sent

from other systems through LPR, and being received by PlanetPress Design via LPD. Since these messages can accumulate, you have the option of not logging them. Logs are kept in a Log folder relative to your install folder. They are named lpddate.log, where date is the current date in the *yyyymmdd* numerical format.

- **No source port range restriction (recommended)**: Select to remove any restrictions on the port of the LPR client computer that PlanetPress accepts data files from. Clear to have PlanetPress only accept data files sent from ports ranging between 721 and 731 on the LPR client computer.
- **Strict RFC 1179 control file**: Select to disable control file extensions the LPD service implements for some flavors of UNIX and LPR. This enforces the basic Line Printer Daemon protocol.
- **Enable BSD compatibility mode**: Select to have the LPD service emulate a BSD UNIX server. Although RFC 1179 is supposed to describe the BSD LPD/LPR protocol, and the LPD input in PlanetPress Design is RFC1179-compliant, there are some incompatibilities between the RFC and the BSD implementation. This option compensates for some of these incompatibilities. If you are not sure about the source of your output, clear this option.
- **Time-out**: Set the time in seconds the PlanetPress Design process waits for the transfer of bytes in the data file before ending the transfer of this file. On a time-out, partially received data files are not passed to the rest of the process; the LPD input resets and is ready to receive further data files.

## Serial Input

Retrieves data using a serial input service to PlanetPress Design for processing data input files. Only one port is used at a time by PlanetPress Design, and as such, only one configuration for the serial connection is allowed. The Serial service can be independently controlled, and its logs checked in real-time as it processes files, using the PlanetPress Suite Service Console.

- **Serial port**: Select the port of the computer where the Serial input is connected to (COM1 through COM8).
- **Baud rate**: Select the baud rate of the Serial input.
- **Data bits**: Select the number of data bits defining the incoming data file on this serial port. The majority of serial ports use between five and eight data bits. Binary data is typically transmitted as eight bits, while text-based data is transmitted as seven bits or eight bits.
- **Parity**: Select the type of parity used for error detection. The parity transfers through the serial connection as a single bit. It is used to verify that each set of data bits transfers correctly. It is then stripped away before the data file passes through the rest of the PlanetPress Design process. Select None to ignore all parity bits; no error detection occurs.
- **Stop bits**: Since most serial ports operate asynchronously, the transmitted byte must be identified by start and stop bits.
- **Time-out**: Set the time in seconds the PlanetPress Design process waits for the transfer of bytes in the data file before ending the transfer of this file. On a time-out, partially received data files are not passed to the rest of the process; the Serial input resets, ready to receive further data files.
- **Job delimiters**: Enter the strings that tell PlanetPress Design the data file being retrieved through the Serial input is complete. Each line in the Job delimiters text box is a different delimiter. You can enter as many delimiters as you want, one per line.
- **Log**: Select to keep a log of errors and other information related to the Serial input. Since these messages can accumulate, you have the option of not logging them and are kept in a Log folder relative to your install folder. They are named serdate.log, where date is the current date in the *yyyymmdd* numerical format.

## Telnet Input

The Telnet program runs on your computer and connects your PC to a server on the network. You can then enter commands through the Telnet program and they will be executed as if you were entering them

directly on the server console. This enables you to control the server and communicate with other servers on the network.

**General tab**

- **Port**: Enter the port number the Telnet input uses to receive data. The default telnet port is 9100.
- **Description**: This field displays the description of the port number, as given by Windows. It is not editable.

**Telnet Options tab**

- **Log all Winsock and network messages**: Select to have PlanetPress keep a log of all Winsock and other network messages that occur from the Telnet input. These messages are related to files sent from other systems using a telnet connection. Since these messages can accumulate, you have the option of not logging them, and are kept in a Log folder relative to your install folder. They are named teldate.log, where date is the current date in *yyyymmdd* numerical format.
- **Job delimiters**: Enter the strings that tell PlanetPress Design the data file being retrieved through the Serial input is complete. Each line in the Job delimiters text box is a different delimiter. You can enter as many delimiters as you want, one per line.

# Windows Queue Input

The Windows Queue Input (also called **WinQueue input** for short) is used to capture print jobs sent to a Windows printer queue on your system.

In order to capture a job with a WinQueue input however, the printer must be in a paused state, configured in either EMF or RAW format, and spooling must be disabled. While it is possible for PlanetPress to enable those options automatically, it is much easier to create a printer that uses the Objectif Lune Printer Driver (see "Objectif Lune Printer Driver (PS)" (page 679)), since these options are by default.

Available capture options:

- **Printer queue**: Select the Windows printer queue where PlanetPress Design obtains the input data file. The drop-down list includes printers installed on the local host computer.
- **New...**: Click this button to create a new Windows Printer Queue using the new **Objectif Lune Printer Driver (PS)**. For more on this driver, see "Objectif Lune Printer Driver (PS)" (page 679).
- **Advanced**: Click this button to toggle showing/hiding the Windows Queue advanced options:
    - **Spool Print Jobs in EMF Format (Advanced Printing Features):** Select to create EMF files for Windows Print Converter action tasks (see Windows Print Converter Action Task Properties in the PlanetPress Workflow Tools User Guide (English Only)). Note that this option must not be selected when capturing generic text type data.
    - **Spool Print Jobs in RAW Format**: Select to generate PostScript data files when the **Create PDF (with Metadata)** checkbox is *unchecked.*
    - **Create PDF(with Metadatea)**: Check box to insure metadata file creation when capturing data from the selected queue. This option will generate a PDF data file when *checked*. Note that this option is only available with PlanetPress Office or PlanetPress Production.
    - **Optimize resulting PDF**: Check box to optimize the PDF created by the capture. Optimizing the PDF will remove duplicate resources such as images and fonts, reducing the size of the PDF greatly if a large number of duplicate resources are in the document.

# The Data Selector

The Data Selector is the tool you use to choose your sample data and metadata files, to select the appropriate emulation, make data selections, and to stabilize your data.

The Data Selector is divided in two tabs: Data and Metadata. The Data tab contains the Data Options, which let you select your emulation, and the Selector Options, which lets you personalize the data selector's display options (see "Data Selector Display Preferences" (page 79))
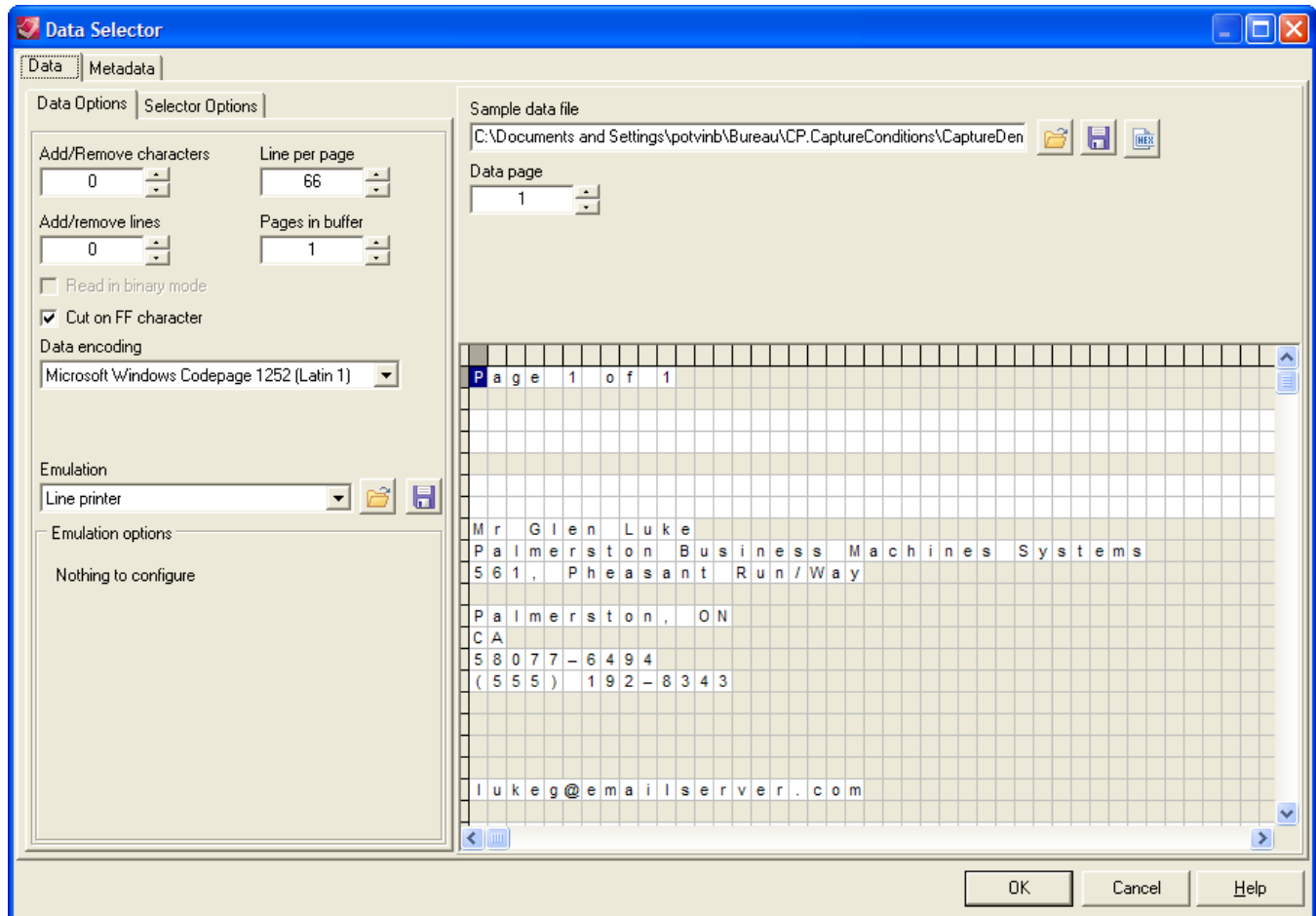


Depending on the chosen emulation and data file, the options in the data selector, the Sample data file section and the Data Pane itself may change to accommodate your choice. The Line Printer, Ascii, Channel Skip and User-Defined emulations will display the default options (see the "Emulation" (page 90) section) and a grid-like display of each character on each line. The following emulations however, will be slightly different.

**Database Emulation**

- The Database emulation changes the Browse button ( ) for the Database Emulation Configuration button ( ), which displays the Database Emulation Configuration (see "Database Emulation" (page 93)).
- Once a database has been opened and query entered, the Data Pane displays the results of the SQL Query in a grid format, which each line representing a single returned row from the database. Each column represents a field returned by the query, with its field name as a row header.

**XML Emulation**

- XML data is represented in a tree structure which corresponds to the data in the XML file. Each node of the XML can be expanded to see the nodes under it. See "XML Emulation" (page 96).

**PDF Emulation**

- If you use a PDF emulation, the Data Pane displays the data as you would see it in any PDF reader.
- A new **zoom** dropdown list is displayed to let you set the zoom in percentage or fit the PDF to the window or the width of the window.
- A new status bar, displaying the (Left, Top) and (Right, Bottom) coordinate pairs, is shown under the Data Pane.

## Metadata tab



The **Metadata** tab allows users to either generate the metadata file for their active sample data file, or to associate an existing metadata file to their document.

The **Sample Metadata Filename** is the path to the metadata file describing the current sample data file. Buttons on the right can be used to load metadata from a file or to save the current metadata to a file.

The **Generated PressTalk Expression** is a PlanetPress Talk command corresponding to the current attribute or field being selected. Its value is editable, which allows the user to customize the string returned by the metadata selector.

The **Search options** defines how to retrieve the value of a given metadata element (attribute or field) when it is not present at the current metadata level. The possible search options are:

---

- Search from a specific location only.
- Search from level X to Job, where X can be any metadata level (Job, Group, Document, Datapage, Page). With this search option, if the selected metadata element does not exist at the specified level, then it will be searched for, starting at the lowest metadata level as specified in the search option, then one level up until the element is found.

The **Raise an error if the field does not exist** option allows to control what to do when a given metadata element is not found, regardless of the search option.

The **Data page** box lets the user choose which data page metadata elements to be displayed.

The **Metadata level** is a treeview allowing users to select the metadata level from which to display or select metadata elements.

The **Attributes** list displays all metadata attributes describing the current metadata level, as selected in the *Metadata Level* treeview, for the current data page, as selected in the *Data Page* control.

The **Production information** list displays all metadata fields describing the current metadata level, as selected in the *Metadata Level* treeview, for the current data page, as selected in the *Data page* box.


# Data Page

A data page is the amount of data that your data file, through your emulation, is passing through a single document. This can mean one or more lines of a CSV file, a PDF page, or a certain number of lines of a text file.

A data page can ignored and print no page at all, or it can be used to generate multiple pages when printing. Unless you are using an N-Up object however, you cannot use multiple data pages to generate a single physical page.

To configure what appears in a data page, you use the Data Selector to specify emulation parameters.


# Emulation

The emulation defines how your document receives and processes its input data. It is basically a method with which PlanetPress Design will read the data and display it on screen. The way data selections are made and data pages are created depends on the emulation, so it is always set at the beginning of your document creation, as you select your data file(s).

> ⟩⟩⟩ While you can have multiple sample data files in your document, all of them have to use the same emulation, since data selections and data pages rely on the emulation to work properly.

Various emulation specific options can be set for most emulations, with the exception of the line printer and database emulations. All emulations, except the database and XML emulations, also let you perform operations on the data to stabilize it, such as add and remove characters or lines.

The sample data file you associate with the document, the emulation you select and the PPD you associate with the document define how your document handles its input data, and consequently determine the accuracy of the output the document produces with the data stream it receives at runtime. If you alter any of these settings, you should verify the change does not compromise the output accuracy of the output.

This section explains what an emulation is, the different types of emulations available in PlanetPress Design, and how to fine-tune the emulation to your input data.

# Text-Based Emulation Properties

Text-Based emulations display your data in plain text in the data selector and the Data Pane, one line at a time, up to the limit you specify in the emulation properties (by default, 66 lines). This is especially useful for legacy systems (such as AS/400 computers) that send data as text meant for older line printers using pre-printed forms. The emulation options are used to make sure your data is stable.

Stabilizing data is the process of defining the size of the data page and where the first data page occurs in the data stream. A stable data page is critical to obtain accurate results. When you stabilize your data, you also need to consider the internal structure of each data page. The internal structure of each data page must also be stable to make the data selections you use in your document reliable. Ideally, a given piece of data occupies the same position across all data pages, or provides some stable characteristic that makes it possible to locate it on every data page.

The following properties are available for the text-based emulations (Line Printer, ASCII, Channel Skip and CSV) to help stabilize the data:

- **Add/remove characters**: Enter the number of characters to add to, or remove from, the head of the data stream, or use the spin buttons to increment or decrement the value. Positive values add characters while negative values remove characters. Further note that if you remove characters in a CSV emulation, you should ensure that you do not inadvertently remove field or text delimiters.
- **Add/remove lines**: Enter the number of lines to add to, or remove from, the head of the data stream, or use the spin buttons to increment or decrement the value. Positive values add lines while negative values remove lines.
- **Lines per page:** Enter the number of lines each data page contains, or use the spin buttons to increment or decrement the value. A higher value means more lines will be displayed on each data page. *Note that increasing the value for this setting increases the amount of RAM used by the application and may exceed the system's capacity. Since the Show used cells option also uses up some RAM, consider removing this option (see "Data Selector Display Preferences" (page 79)) to reduce system load.*
- **Pages in buffer:** Enter the number of data pages you want the data page buffer to contain, or use the spin buttons to increment or decrement the value. Putting more pages in the buffer multiples the lines shown and is only useful in specific cases. You should also consider using the N-Up Object if you want to display multiple data pages.
- **Cut on FF character:** Select to have the document start a new data page when it encounters a form feed character in the data stream. If you select Cut on FF character, you have two conditions that signal the end of a data page: the form feed character and the number of lines set in the Lines per page box. Note that the **Cut on FF character** takes precedence on the **lines per page** option.
- **Read in binary mode:** Select this option to force the printer to read the incoming data in binary mode. Use this option with the ASCII emulation to fix problems related to line spacing caused by LFCR character pairs found within the data. Use it with the ASCII emulation and with the **Tab on carriage return** option to fix problems related to data formatting caused by isolated CR characters found within the data. This option can only be used with the ASCII and User defined emulations. Note that you cannot select this option if the document is to be installed on a printer that cannot run in binary mode.
- **Data encoding:** Select the appropriate encoding for the sample data file. You may look at the data in the Data pane (non-English characters especially, if any) to see how the your selection affects the data.

## Line Printer Emulation

Line printer emulation tells the document to treat the input data as data destined for a line printer.

In this emulation, a form feed signals the end of a data page. If no form feed occurs in the data stream, the emulation adds lines to the data page buffer until the buffer is full.

Line printer emulation offers the best overall performance of all the emulations.

**Line Printer Emulation options**

The line printer emulation does not have any options other than the general text-based emulation options (see "Text-Based Emulation Properties" (page 91))

## ASCII Emulation

ASCII emulation tells the document to treat the input data as a stream of ASCII characters. The document reads the data stream one character at a time, constructs a line, and adds that line to the data page buffer.

In this emulation, you can define how the document handles carriage returns that are not followed by line feeds and how it handles tabs. You can also define whether you want the document to remove any Hewlett Packard Printer Control Language (HP PCL) escape sequences it encounters.

If you use an ASCII emulation, you need to know if your printer supports binary mode as this is the recommended mode for ASCII emulation. On printers that support binary mode, you can switch the printer to binary mode using the printer keypad or by sending the appropriate PostScript code to the printer.

In binary mode, the printer reads the end of line characters (carriage return [CR], line feed [LF], and carriage return followed by a line feed [CRLF]) as they appear in the data stream and does not perform any substitution. A printer that does not support binary mode or is not running in binary mode replaces any CR, LF, or CRLF that appears at the end of a line of data with a LF.

A form feed signals the end of a data page in ASCII emulation. If no form feed occurs in the data stream, the emulation adds data to the data page buffer until the buffer is full.

### ASCII emulation options

- **Tab on carriage return**:Select this option to fix formatting problems caused by isolated CR characters found within the data. When this option is selected, isolated CR characters are spaces, as defined in the **Number of spaces in the tab** box below. Note that this option is available only when the **Read in binary mode** option is selected.
    - **Number of spaces in the tab**: Enter the number of spaces you want the application to use when an isolated carriage return character is found within the data. This number typically corresponds to the maximum column number. If your data is formatted so as to occupy a maximum of 120 characters on each line, enter a value of 120 in this box, so when an isolated CR character is found, the data following the CR character will appear starting from column 121. Note that this option is available only when the **Tab on carriage return** option is selected
- **Number of spaces per tab**: Enter the number of spaces you want to use when actual TAB characters are found within the data.
- **Remove HP PCL escapes**: Select if you want all Hewlett Packard Printer Control Language escape sequences to be removed from the data.

## Comma Separated Value (CSV) Emulation

CSV emulation tells the document to read the input data one line at a time and to treat each line as a database record. It also specifies the field delimiter the document uses to distinguish the different fields of a record. The document reads the data stream one line at a time and puts each field of the database record on a separate line in the data page buffer.

In CSV emulation, the emulation adds lines to the data page buffer until the buffer is full. You can force a new data page for each record when you set up the emulation.

Note that a double text delimiter within a field is not considered a normal character when not using the Optimized PostScript Stream option or when printing using a Windows printer driver.

**CSV emulation options**

- **Text delimiter**: Enter the character used to mark the beginning and end of each field within the data. Text delimiters are required if the character entered in the **Delimiter** box is present within the data itself. If the fields are separated using commas, and if the data itself contains commas, for example, then individual fields may be split into multiple ones. Using text delimiters ensures that actual commas within the data will not be interpreted as delimiters. If backslashes (\) are used in the data as text delimiters, enter double backslashes (\\) in this box. You can also specify ASCII characters using octal values preceded by a single backslash (for example, \041 for an exclamation mark).
- **Force one record per page**: Select to prevent splitting records across pages. If this option is not selected, when a document is printed, it may happen that the last record at the bottom of a page may be split between two pages.
- **Field delimiter**: Enter the character used to separate each field within the data. If backslashes (\) are used in the data as delimiters, enter double backslashes (\\) in this box. You can also specify ASCII characters using octal values preceded by a single backslash (for example, \041 for an exclamation mark).
- **Set tab as field delimiter**: Select if tabs used to separate each field within the data. Selecting this option overrides any value entered in the **Field delimiter** box.

## Channel Skip Emulation

Channel skip emulation is a variant of line printer emulation. It tells the document to read the data stream one line at a time, and to treat the first character of each line as a code that indicates how to position the line of data in the data page buffer.

By default, in channel skip emulation, the integer 1 signals the end of a data page. You can change this default when you set up the emulation.

Note that if a given value is used for multiple channels, the result may be different at design time, or when the document is previewed or printed.

Also note that Split on FormFeed (FF) is not supported with the Channel Skip emulation in Optimized PostScript Stream mode or when printing using a Windows driver.

**Channel skip emulation options**

- **Skip page**: Enter the code used within the data to mark the beginning of each page (the number 1 in standard channel skip emulation). Note that if the standard code is used within the data as the skip page code, it is likely that the other codes are also standard, and that you only need to make minor changes to the other codes, if any.
- **No line feed**: Enter the code used within the data to indicate that the next line feed character should be ignored. This causes the next line to print over the current line, and is a technique impact printers use to print a line, or elements of a line, in bold or with underlining.
- **Skip [x] lines**: Enter the code used within the data to indicate that the corresponding number of lines must be skipped.
- **Char** and **Skip to line**: Enter the code used within the data to mark a jump to a different line in the **Char** box, and enter the corresponding line number in the **Skip to line** box.
- **Char** and **Go to column**: Enter the code used within the data to mark a jump to a different column in the **Char** box, and enter the corresponding column number in the **Go to column** box.

## Database Emulation

This emulation differs from other emulations in regards to PlanetPress Suite applications. With other emulations, data is pushed either to PlanetPress Design documents residing on printers or to PlanetPress Suite Workflow Tool processes running on servers. But in the case of the database emulation, data must be pulled from the data source.

Like with every other emulation, it is possible to send a Design Document set up to use the database emulation to a printer. But contrary to documents that use the other emulations, you cannot send a raw data file to the document and expect the

document and data to merge and print automatically. In this case someone or something must query the database and extract the data that will populate the Design Document.

We can imagine two basic scenarios. In the first one, we can imagine someone in a print shop who needs to use data from a database to print a bunch of personalized letters using PlanetPress Design. That person opens a Design document and uses the Data Selector to select a database. By making a connection to the database, its structure can be accessed and it becomes possible to determine how data is to be pulled into PlanetPress Design. The process actually pulls data into PlanetPress Design and lets the print shop employee visualize and print the data on the personalized letters.

The second scenario involves PlanetPress Suite Workflow Tool. In this case, PlanetPress Database action task takes the place of the print shop employee and performs the database query automatically. The task generates a PlanetPress Design compatible data file that it passes to the following task, be it another action task, or any output task.

Bear the following in mind:

- The person or plugin performing the query must have full access to the database.
- The data is extracted at the time of the query. A new query must be performed whenever the data needs to be updated.
- Any changes to the structure of the database may have an impact on automated data querying tasks.
- You must have the proper ODBC driver installed to use this emulation.

Database emulation supports SQL ANSI 92 or higher, and supports the following data types: string, integer, floating point, all date formats, and text-only MEMO. It does not support any binary data types such as Binary Large Object (BLOB), images, sound files, and MEMO data that includes binary data.

Database emulation requires version 2.5 or higher of Microsoft Data Access Components (MDAC), including JET 4.0, and you can save database emulation configurations to a file.

**To set up a database emulation:**

1. Choose **Tools** | **Open Active Data**.
2. In the **Data Selector**, locate the **Emulation** box and select **Database**.
3. Click the **Database Emulation Configuration** button.
4. Associate a database.
   Microsoft Access Database or dBase file
   **Database**: Enter the path of the Microsoft Access database or dBase file, or click the Browse button to the right of the box to navigate to, the database file. Recall that a Microsoft Access database file bears the extension **.mdb**, and a dBase file bears the extension **.dbf**. If the file is a dBase file, you must specify the folder that contains the **.dbf** file. The folder in this case is considered to be the database, and the individual **.dbf** file a table in the database. Once you enter the path, the Table/query name box updates to reflect the tables and queries available in the selected database.
   ODBC Data Source
   **ODBC Data Source**: Click to connect to an ODBC Data Source. Use the Select Data Source dialog box that appears to select an existing Data Source or set up a new one. When you exit the Select Data Source dialog box, the Database box updates to display the connection string it uses to connect to the database, and the Table/query name box updates to reflect the tables and queries available in the selected database.
5. Click **Edit SQL** to create the SQL query by hand to define the SQL query that retrieves the data your document requires.
6. Set the properties that define a record set.
   **Condition**: Select the condition that signals the end of a record set. Three possibilities exist: create a new record set for each record, create a new record set after every x records, or create a new record set when the value of a specific field changes.
   **Sort on condition field**: Select this if the condition you set is to create a new record set when the value of a specific field changes, and you want to sort the records before applying that condition.

**Maximum records per record set**: Set either the number of records in each record set, or the maximum number of records in a record set. An individual record set can contain a maximum of 4000 records.

7. Set the number of records you want to include in the sample data file. The number of records you set should provide a reliable sample to ensure your document executes properly with any of the data it may encounter at runtime.
**All**: Select to include all records in the database in the sample data file.
**Records**: Select to define the range of records you want to include in the sample data file. Use the box that

**To enter an SQL query:**

1. In the **Database Connection** dialog box, click **Edit SQL**.
2. If necessary, click **Show Tables** to display, in the Tables area, a list of the tables available in the database.
3. In the **SQL Query Entry** area, enter the SQL query. The following two sample queries both retrieve all the fields in the Orders table. The second sorts the resulting records on the Date field.

```
SELECT * FROM [Orders]
SELECT * FROM [Orders] ORDER BY [Date]
```

4. Click **Test SQL** to verify the query you entered is a valid SQL query.
5. Define whether you want PlanetPress Design to automatically enclose table names and field names in square brackets.
**Alternate syntax(not recommended)**: Select to prevent PlanetPress Design from automatically enclosing the names of any database tables and fields that appear in the SQL query in square brackets when it exits the advanced SQL Statement dialog box.
6. **Client side cursor**: Select to download result sets to client computer running the SQL query.
7. Click **OK** to return to the **Database Connection** dialog box.

## Export or Import a Database Emulation Configuration

You use the export and import procedures when you intend to use the same configuration in several documents. You also export a database emulation configuration when you intend to execute the document in PlanetPress Workflow Tools, and want to simplify the process of configuring the database plug-in.

The exported configuration file is in XML format and bears the file name extension *.cfg*.

To export a database emulation configuration:

1. Open the document that uses the database emulation configuration you want to export.
2. Choose **Tools** | **Open Active Data**.
3. In the **Data Selector**, click the **Database Emulation Configuration** button to display the Database Connection dialog box.
4. In the **Database Connection** dialog box, set the password option.**Include password on export**: Select to include the password required to access the database, in the exported database emulation configuration.
5. In the **Database Connection** dialog box, click **Export**.
6. In the Export Database Configuration dialog box, navigate to the folder in which you want to save the configuration, enter a name for the exported file, and click **Save**.
PlanetPress Design exports the configuration and returns the focus to the Database Connection dialog. If you selected Include password on export, the exported configuration file contains the password required to access the database.
7. Click **OK** to exit the Database Connection dialog.
8. Click **OK**.

To import a database emulation configuration:

1. Open the document in which you want to import a database emulation configuration.
2. Choose **Tools** | **Open Active Data**.

3. In the **Data Selector**, click the **Database Emulation Configuration** button to display the Database Connection dialog box.
4. In the **Database Connection** dialog box, click **Import**.
5. In the Open dialog box, navigate to the folder containing the configuration file you want to import, select the configuration file, and click **Open**.
6. If necessary, adjust the database emulation configuration options in the Database Connection dialog.
7. Click **OK**.
8. Click **OK**.

## XML Emulation

XML data emulations allow you to capture data emanating from web databases, E-mail fulfillment, ecommerce, and general XML database engines. In XML emulation, the data elements in markup language format are organized in a folder view with a root node and sub-level nodes. Depending on the document configuration, a data page can be associated with a sublevel element contained in an XML data file much in the way a data page can be associated with an individual record in a CSV emulation. When you set-up an XML emulation, you define whether to separate the data by the root or the second level element.

Note that when XML data is merged with PlanetPress Design documents on a printer DOCTYPE and ENTITY tags are ignored.

Also note that characters referenced using the &#999; syntax are limited to values ranging from 000 (&#000;) to 256 (&#256;).

### XML emulation options

- **Root element (entire file):** Select this option to associate all the data within the XML file with a single data page.
- **Second Element:** Select this option to associate each second level element within the data file with with a different data pages.

## PDF Emulation

PDF Emulations allow you to capture data from fully composed documents in a PDF format.

PDF Emulation slightly differs from other PlanetPress Suite emulations: with other emulations, data is read either one line at a time or one character at a time, while PDF emulation processes the input data from the PDF file in such a fashion that every PDF page becomes a full data page. Each PDF page is thus graphically represented in the PlanetPress Design Data Pane as one data page.

Note that protected PDF and PDF of versions above 1.7 are not supported by PlanetPress Suite 7.

The PDF Emulation does not have any options - that is, there is nothing to set up when opening a PDF data file. PlanetPress Design simply reads each PDF file as a unique data page.

## User-Defined Emulation

In user-defined emulation, you use PlanetPress Talk commands to define how you want the document to treat the input data. You use this emulation when the structure of your input data prevents you from using any of the other emulations. You must ensure the emulation you create handles any variations in the data properly and under all circumstances.

In user-defined emulation, the document reads the data stream one line at a time. After it reads a line, it places all the characters in that line in a string variable. You use PlanetPress Talk commands to specify how the document handles the contents of this variable.

Note that when a user-defined emulation is used, whenever you request a data page that is passed the last data page, the last data page will be displayed.

**Create a User-Defined Emulation**

Since User-Defined emulations are based on PlanetPress Talk commands, please see the chapter on PlanetPress Talk for more information no creating user-defined emulations.

## Associate a Sample Data File with a Document

This procedure describes how to associate a sample data file with your document using either the Data Selector or the Data Pane of the Program window. Note that this does not apply to Database Emulations, which are configured using a database connection (see Database Emulation)

To associate a sample data file with your document, do any of the following:

- In the Data Selector, click the **Browse** button to the right of the Sample data file box to browse and select a sample data file.
- In the Data Selector, in the Sample data file box, enter the path of the new sample data file and either press **ENTER**, or click outside the Sample data file box.
- In Windows Explorer, select the sample data file, and drag and drop it into any of the following areas of the PlanetPress Design Program window: the Structure area, the Page area, the Data Pane, or, if the Object Inspector is displaying an image resource, the Object Inspector.

PlanetPress Design makes a copy of the sample data file and stores it within the document. The Sample data file box displays the path of the sample data file and the first data page of the sample data file appears in the Data Pane. It also automatically selects an emulation that corresponds to the file name extension of the sample data file you select, using the following table:

| File name extension: | Emulation: |
|---|---|
| *csv* | CSV |
| *dat*, *txt* | Line printer |
| *db*, *dbf*, *mdb* | Database |
| *pdf* | PDF |

You can set the User Options to have PlanetPress Design monitor the original sample data file and prompt for confirmation to update its internal copy if it detects changes in the original. See "Notification Messages Preferences" (page 69).

# Metadata

Simply put, metadata is data about data or, in other words, information tagged to data. Metadata includes information about the data file itself, the document, page properties, page counts and custom user fields.

> Applications or plug-ins created in PlanetPress Suite 6 and using Metadata will need to be updated for use in version 7. No backward compatibility mode is available.

> When a user-defined emulation is used with metadata, results and behavior are unknown and unsupported. For instance, refreshing the metadata file may cause the document to crash and/or corrupt. For this reason, it is strongly advised to create backup copies of your documents beforehand.

**Metadata structure**

Metadata in PlanetPress Suite Version 7 introduces the following concepts for adding information to a job:

- **Page**: 1 side of a physical paper sheet.
- **Datapage**: 1 atomic unit of content that produces zero, one or more pages.
- **Document**: group of 1 or more ordered datapages intended to the same recipient from the same source (ex: invoice).
- **Group**: a logical and ordered group of documents (ex: all invoices for a specific customer number; all documents going to the same address, etc.)
- **Job**: file that contains 1 or more groups.

When Metadata is produced for a given job, a hierarchical (i.e. tree-like) structure is created, composed of the above elements in the following order: Job->Group(s)->Document(s)->Datapage(s)->Page(s). Any operation that modifies the data with regards to this structure (ex: remove pages, alter the data, etc.) makes the metadata obsolete and so it must be recreated or refreshed.

As an example, consider the typical case of a PlanetPress Design document which uses a Line Printer datafile of transactional data in order to generate PDF invoices for a series of clients. By using the Metadata tools available in PlanetPress Suite version 7, we can add the following information to the datafile:

- The job contains only invoices for clients located in Montreal.
- Since more than one invoice can go to the same recipient, invoices are grouped by customer.
- Each invoice is a document resulting from the execution of a PlanetPress Design document over one or more datapages, which results in zero or more physical pages being output.
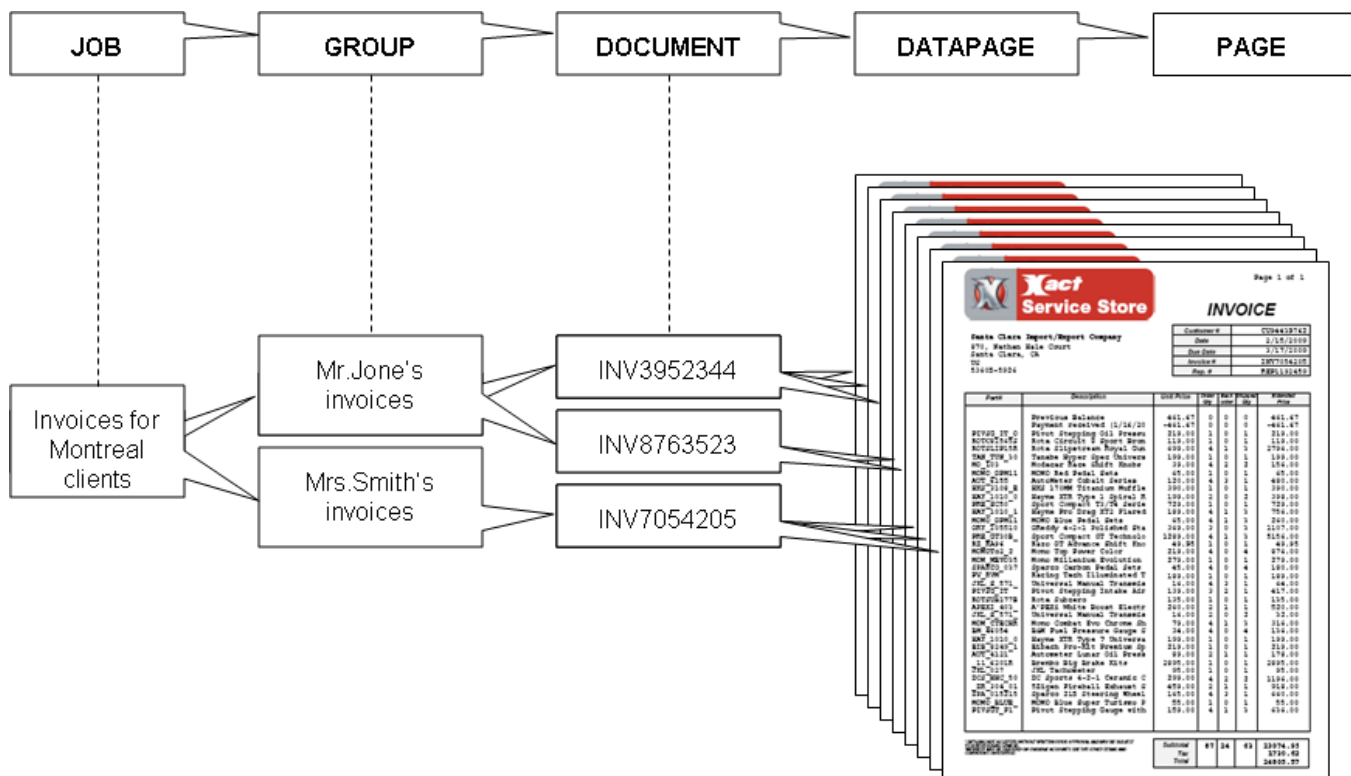


A single JOB can be composed of GROUPS of DOCUMENTS, which themselves are composed of physical PAGES produced by executing a PlanetPress Design document on one or more DATAPAGES.

**Metadata Elements**

Each metadata node (i.e. Job, Group, Document, etc.) is described with a series of elements, that is, system-defined attributes or user-defined fields holding static or dynamic information about the node they are attached to. Each element has a name and a value. More specifically, here is a definition of these 2 types of elements:

Attribute: A read-only, system-defined element which holds a certain information about a certain node from the Metadata structure. This information can be static (e.g. the size of a physical page) or evaluated on-the-fly (e.g. the number of documents in a group). Attributes are non-repetitive (i.e. name is unique) and does not persist through metadata recreation.

Field: A read-write, user-defined element which hold custom information about a certain node from the metadata structure. Fields are repetitive (i.e. the same field may appear multiple times) and persist through metadata recreation.

In addition to attributes and fields, each node of type group, document or datapage have a boolean property called selected that indicates whether or not to produce the pages under that node. By default, this property is set to true for all nodes.


## Metadata Attributes Reference

Here is a description of the Metadata attributes. The attributes are categorized as either Production, Finishing or Index/Count.

Production attributes describe the production of the job and/or metadata (e.g. path and name of the datafile, date at which metadata was created, etc.)

Finishing attributes describe the finishing intent (e.g. page dimensions, page orientation, duplex mode, etc.). Note that the presence of some finishing attributes depends on the PlanetPress Design document and target device used when producing the job.

Index/Count attributes are not part of the original metadata file. They are evaluated live based on the content of the metadata.

In the following table, the last 5 columns indicate at which level the corresponding attribute is available.

| Attribute | Description | Category | Job | Group | Docume |
|---|---|---|---|---|---|
| DataEncoding | (optional) Name of the character encoding. | Production | X | X | X |
| DataFile | (optional) Path and name of the data file used by the PlanetPress Design Document. | Production | X | X | X |
| Date | Date the metadata was created in ISO format. | Production | X | X | X |
| Time | Time the metadata was created in ISO format. | Production | X | X | X |
| Title | Title of the source document. | Production | X | X | X |
| Producer | Name of the sofware that created the metadata. | Production | X | X | X |
| Creator | Name of the software that created the source of the metadata. | Production | X | X | X |
| TargetDevice | Name of the device for which the metadata and associated data is intended. | Production | X | X | X |
| Dimension | Two floats separated by a colon indicating the media size in typographical points (ex: 612:792). | Finishing | X | X | X |
| Orientation | "Rotate0", "Rotate90", "Rotate180" or "Rotate270", indicating respectively portrait, landscape, rotated portrait and rotated landscape. | Finishing | X | X | X |
| Side | "Front" or "Back"; indicate whether the page is on the front or the back of the paper sheet. This attribute is a "best effort" and is device-dependent. | Finishing | | | |
| Duplex | "None", "DuplexTumble" or "DuplexNoTumble"; indicate a change of the duplex status. | Finishing | X | X | X |

| | | | | | |
|---|---|---|---|---|---|
| **InputSlot** | Device-dependent identifier of the media source. | Finishing | X | X | X |
| **OutputBin** | Device-dependent identifier of the media destination. | Finishing | X | X | X |
| **Weight** | Device-dependent weight of the media. | Finishing | X | X | X |
| **MediaColor** | Device-depedent color of the media. | Finishing | X | X | X |
| **MediaType** | Device-dependent type of the media. | Finishing | X | X | X |
| **Index** | | Index/Count | | X | X |
| **IndexInDocument** | Returns the Absolute index of the node within all the node under the parent Document. | Index/Count | | | |
| **IndexInGroup** | Returns the Absolute index of the node within all the node under the parent Group. | Index/Count | | | X |
| **IndexInJob** | Returns the Absolute index of the node within all the node under the parent Job. | Index/Count | | X | X |
| **Count** | | Index/Count | X | X | X |
| **DocumentCount** | | Index/Count | X | | |
| **DatapageCount** | | Index/Count | X | X | |
| **PageCount** | | Index/Count | X | X | X |
| **SelectedCount** | | Index/Count | X | X | X |
| **SelectedDocumentCount** | | Index/Count | X | | |
| **SelectedDatapageCount** | | Index/Count | X | X | |
| **SelectedPageCount** | | Index/Count | X | X | X |
| **SelectedIndexInDocument** | Returns the Absolute index of the node within all the selected node under the parent Document. | Index/Count | | | |
| **SelectedIndexInGroup** | Returns the Absolute index of the node within all the selected node under the parent Group. | Index/Count | | | X |
| **SelectedIndexInJob** | Returns the Absolute index of the node within all the selected node under the parent Job. | Index/Count | | X | X |
| **NumCopies** | Indicates how many times the job is set to execute, as set when printing using a Windows driver. | Index/Count | X | | |
| **Author** | Name of the user who printed the job initially, as available in the spool file, and as the first job info of the Windows capture input. | Production | X | | |

## Metadata Tools

PlanetPress Suite version 7 includes a complete set of metadata-related functionality, which can be referred to as Metadata Tools. These tools can be used to generate metadata, retrieve or define metadata elements, and build the metadata structure.

### PlanetPress Design Metadata Tools

Using PlanetPress Design version 7, one can:

- Generate metadata for any given sample datafile.
- Graphically retrieve the value of a metadata attribute or field for use in any design object.
- Define documents and groups using any condition.
- Define custom metadata fields.
- Manipulate Metadata with PlanetPress Talk commands.

Following is a description of the PlanetPress Design Metadata tools which allows to perform the above tasks.

Metadata Generation using Data Capture with PlanetPress Printer

The Objectif Lune Printer Driver (PS) allows end-users to print directly to PlanetPress Design from any Windows application, by using the familiar File|Print option. At the other end, PlanetPress Design can capture the incoming stream and convert it internally into a PDF file along with its metadata. By default, capturing a document input using a PlanetPress Printer will generate a PDF along with its metadata.

Metadata Generation and Refresh without using PlanetPress Printer

It is possible to generate or refresh metadata for any given sample datafile by using the Refresh Metadata option available when right-clicking on the Metadata Fields folder found in the Document Structure Window. For example, metadata can be generated this way for a Line Printer sample datafile captured using an LPD Queue Input.

**Metadata Selector**

PlanetPress Design's Data Selector window is accessible by double clicking inside the Sample Data window or by clicking on the Open Active Data button available in the ribbon. The Data Selector is equipped with a new tab labeled Metadata.

Firstly, two buttons at the top right corner of this tab allows to load or save a metadata file generated for the current sample datafile.

Secondly, the metadata tab graphically displays all elements (i.e. attributes and fields) available at the current level (i.e. Page, Datapage, Document, etc.). More importantly, these elements are graphically selectable, like any other part of the sample datafile when using the 'Select Data' option inside a Text object, for example.

The Metadata Selector allows to view and select metadata elements.

## Metadata in Document Properties



Metadata tab in the properties of a PlanetPress Design document allows to easily define documents or groups.
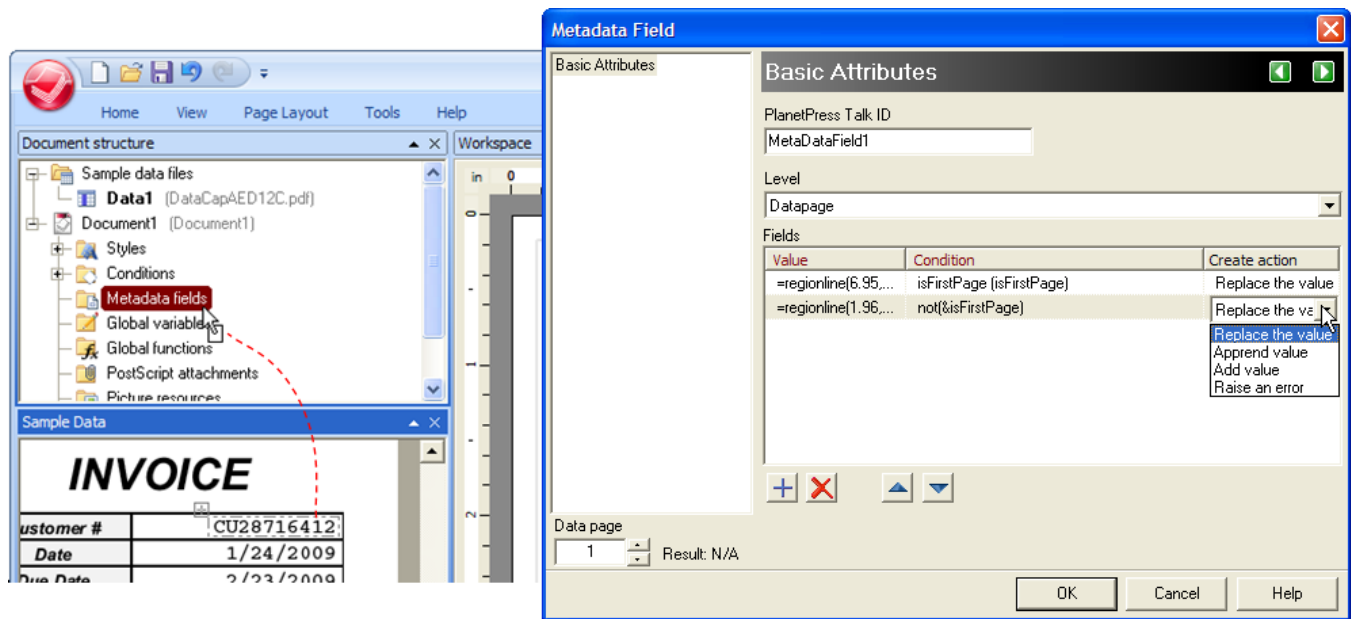
## Metadata Fields

Metadata Fields in the properties of a PlanetPress Design document allows to easily define documents or groups.

# 🖼 Setting Up a Document

Once you have a sample data file to work with, you are ready to start creating your document in PlanetPress. This chapter describes how to set up a document, the first step in document creation.

## Set Up a Document

You can view or edit the properties of a document using either the Object Inspector or the Document properties dialog box.

To view or edit properties using the Object Inspector:

1. In the Structure area, select the document icon.
2. In the Object Inspector, make any necessary modifications to the properties.

To view or edit properties using the Document properties dialog box:

1. Double-click on the **Document** node to display the Document properties dialog box.
2. Use the Document properties dialog box to edit the document properties, if necessary.
3. In the **Document** properties dialog box, click **OK**.

The properties of a document are as follows:

### Basic Attributes

- **Description**: Enter a short text description of the document.
- **PlanetPress Talk ID**: Enter a name under which the document is known in PlanetPress Talk commands. This also servers as the document's name if it is sent to a printer drive and used in Printer-Centric mode. The name you choose should be both descriptive and unique, cannot begin with a number, and can contain only the following ASCII characters: underscore (_), upper and lower case letters of the alphabet, all digits 0 through 9.If you use an underscore in the name, it should not appear as either the first or last character of the name as this may cause internal conflicts in the software. Names are case-insensitive and must be unique; no two elements in a document can have the same name. Names can be a maximum of 50 characters in length. Finally, PlanetPress Talk variable and command names are reserved words; you cannot use any of these reserved words as a name.
- **Designed For**: Select a PPD for the document. The contents of the PPD subfolder in the PlanetPress program folder determine the contents of this list. In PlanetPress Design, you can now open PPD files as well as other attachments emanating from Macintosh and UNIX workstations.The PPD that appears by default here is the one selected in the Default printer box in the User Options dialog box. The Default printer is an optimized PPD. Rather than send a request for a setting to the printer, it first queries the printer to see if that setting is already in effect. The Alternate default printer PPD does not query the printer before it sends a request for a setting.
- **Printer Password**: Enter the password for the printer on which you intend to install the document. A password value of 0 means there is no password for the printer. You can determine whether a printer requires a password by printing a PlanetPress Design printer status page.
- **Printer Working Path**: Enter the path of the folder on the printer in which you want to install the document. This is necessary only if you plan to install the document in a specific folder on the printer's hard drive or in a specific folder in its flash memory.
- **Digit Substitution**: Choose to display numbers in either Arabic, Hindi or Farsi characters.
- **Notes**: Enter any notes you want to insert at the head of the document. Notes do not print as part of the output of the document.
- **Sign**: Click to append the current date and time, the name of the owner of this copy of PlanetPress Design, and the name of the company to which this copy is licensed.

### Paper Handling

- **Default Page Size**: Select the default paper format the document uses. In most cases, you never need to set this option, unless the same job, or multi-jobs require using various paper sizes. In a small number of printers however, when you are executing a document in which several consecutive pages use the same paper format, you may need to set this option. When the document subsequently executes each page, it compares the current paper format with the one set for the page, and only issues a paper format command if the two formats are different.The formats available depend on the PPD selected in the Designed for box of the Document properties dialog box. If you select Default, no command is issued to the printer regarding the paper format and the document uses the printer setting in effect at execution time.
- **Selection Type**: Set the default input paper tray to use for this document. The trays available depend on the PPD you selected for the document. The selection you make here determines the contents of the Paper source area. Select Input tray to specify a specific input tray, and select match paper size to have the printer automatically use the tray that contains the paper format defined for this page. Select manual feed to use the printer's manual feed, and select media selection to set specific media characteristics. You can select an input tray, paper size, and paper orientation for each document page. If the printer supports it, you can also select duplex mode. Note that the document must be able to recognize the output printer during the job to perform load balancing. You can set the output printer name using a variable for either the printer output or the Windows printer output. You can add start/end document commands to support subset job handling. You can activate Run-locally+GDI with speed limits (specified as categories or absolute speed) and either block or enable the use of multi-processor systems.
    - **Input Tray**: Select the input tray you want to use in this box. This option is available only when you select Input tray in the Selection type box.
    - **Media Type**: Select the paper type for the page. This option is available only when you select media selection in the Selection type box.
    - **Media Color**: Select the paper color for the page. This option is available only when you select media selection in the Selection type box.
    - **Media Weight**: Select the paper weight for the page. This option is available only when you select media selection in the Selection type box.
- **Output Tray**: Set the output paper tray you want to use for this document. The options available depend on the PPD selected for the document in the document properties dialog box.
- **Paper Orientation**: Select the orientation you want to page to be in on the printer. The selection is made between Portrait, Landscape, and rotated versions of both (using a 180° rotation)
- **Duplexing**: Select the duplexing options for the document. The duplexing options available depend on the PPD you selected for the document. Note that duplexing options apply only to normal pages since these are the only pages that can print. If you want to print simplex and duplex in the same document, and your printer supports switching between simplex and duplex in the same job, you can set the duplexing options on a page-by-page basis using the duplexing option in the page properties dialog box. If your printer does not support switching between simplex and duplex in the same job, you can simulate the switch by setting the duplexing option here and printing a blank page on the back of each page that you want to print simplex. If you select default, no command is issued to the printer regarding duplexing for this document, and the document uses the printer setting in effect at execution time.

**Advanced Paper Handling**

- **Paper handling before the document**: Display, define, and/or edit the paper handling operations you want the printer to execute before it executes the document. The printer executes the operations sequentially, from top to bottom. Right-click in this area and use the menu to manipulate the data.
- **Paper handling after the document**: Display, define, and/or edit the paper handling operations you want the printer to execute after it executes the document.You can also delete an item by selecting it and pressing **DELETE**. You can manipulate or move data to the Paper handling before the document area by clicking it and dragging it to the new location. Double-click an item to display and, if necessary, edit the condition associated with it.

**Compilation Options**

PLANETPRESS. SUITE

For help creating documents that use Fiery® FreeForm™ or Freeform 2 features, see <u>"FreeForm Caching" (page 110)</u> or <u>"FreeForm 2 Caching" (page 111)</u>. For help creating a document in VPS or VDX format, refer to <u>"VPS Caching" (page 112)</u>.

- **Caching method**: Select the caching method you want to use.
  - Select **None** to prevent the RIP from using any caching.
  - Select **Generic** to use the standard caching provided by PostScript Level 2.
  - Select **FreeForm** and use the options that appear to either make the current document a FreeForm master document, or make the current document a document that calls a FreeForm master document.
  - Select **FreeForm 2** and use the options that appear to make the current document a FreeForm 2 master document. To make the current document call a FreeForm 2 master document, select individual pages and use the options that appear in the Page properties dialog box.
  - Select **VPS** to cache all virtual and cachable pages. Note that VPS always creates booklets.
  - Select **VDX** to cache only those pages of type virtual that are defined as cachable.
- **Use Form Versioning at printer level**: Use this option to set a version number for the document.
- **Form Version**: You use a version number to tell a document to check its version number before executing, and only execute if it is the most recent version.
- **Search Database Name**: *This legacy option is unused and disabled. It is removed in later versions of PlanetPress.*
- **PostScript Level**: Select the PostScript Level for the converted document. This should correspond to the highest level your printer supports.
- **Do not use Virtual Memory for EPS/PDF pages**: Check if your PostScript Level 2 printer prints errors when 2 or more EPS or PDF files are present on your page. This is only useful in some older PostScript Level 2 printers.
- **Caching Options group** (only appears with FreeForm and FreeForm 2 caching methods)
- **This Document is a Master**: Select to create a FreeForm or FreeForm 2 *master* document.
- **Form number**: Enter a FreeForm document reference number. If the document you are creating is a FreeForm *master* document, it will be assigned the number entered in this box. If the document you are creating is *not a master* document, it will be associated with the FreeForm master document identified by the number entered in this box.
- **Master ID**: Enter the FreeForm 2 master document name. When you create a FreeForm 2 *master* document, you must name it. This makes it possible for multiple FreeForm 2 master documents to be cached on a given printer and for any FreeForm 2 document to specifically call any cached FreeForm 2 master document.
- **Maximum Data Line Length**: Define the number of columns in a data page. Enter the value, or use the spin buttons to increment or decrement the value. The maximum value for this option is 65,535 characters. The default value is also 256 characters. You should tune this value to the longest line in your input data. Setting a maximum data line length that greatly exceeds the length of the longest line in your input data may increase execution time depending on the printer manufacturer. Also, the 65,535 value affects imaging, faxing, archiving, and caching VDX functionality. Optimized PostScript Stream is not affected by this value.
- **Behavior**
  - **Use end of job keyword**: Use this option when you want to execute several documents by sending a single job that contains all the triggers and input data for those documents. Note that this is only functional in Printer-Centric run mode and will not work in any other outputs.
  - **End of Job Keyword**: Enter the keyword for the end of job.
  - **Skip blank data pages**: Select to have the document ignore any empty data pages it encounters at runtime. Clear to have the document execute with any empty data pages it encounters.
- **PostScript Printer Form Cache**
  - **Set Form Cache**: Select to set the size of the printer form cache and/or the size of the largest item the cache can contain.
  - **Max Form Item**: Set the size (in bytes) of the largest single EPS, PDF, or bitmapped image that the form cache can contain. You use the size of the largest and most frequently used image in your document to determine an appropriate value for this option.
  - **Max Form Cache**: Set the size (in bytes) of the PostScript printer form cache. You base the setting for this option on the number of images in your documents, their sizes, and how frequently each image repeats in a document.

**Resource Options**

- **Resource location**: Select the runtime location for the resource files associated with the document when the document executes on a printer. Select In memory to copy the resource files into RAM at runtime. Select On file system to copy the resource files onto the printer's hard drive at runtime; this option is recommended for documents that occupy large amounts of space.
- **Picture Resolutions**
  - **Color (DPI)**: Enter the resolution for the color bitmapped images the document references. This resolution applies to all color bitmapped image resources in the document. In the case of dynamic images that reference color bitmapped images external to the document, the resolution in effect at the time you install and/or convert the document is the one that applies to *all* of the external color bitmapped images (the document applies this resolution at runtime, when it retrieves each of the external images).
  - **Grayscale (DPI)**:  Enter the resolution for grayscale bitmapped images the document references.
  - **Monochrome (DPI)**: Enter the resolution for monochrome bitmapped images the document references.
- **PlanetPress Capture** (see also "Capture Field Object" (page 166))
  - **Pattern Sequence**: A batch identification, useful for extending the number of possible patterns.
  - **Anoto statement position**: The position of the notice "Paper featuring Anoto functionality" which appears when adding a PlanetPress Capture field to your document. This notice is legally mandatory and will always appear on your page.
  - **Horizontal offset**: Enter a value if you wish to move the notice away from the left or right border (depending on the location of the notice)
  - **Vertical offset**: Enter a value if you wish to move the notice away from the top or bottom border (depending on the location of the notice)
  - **Include trace code**: Check to print the trace code (identification) of the pattern used on the page. This is useful for debugging and error handling.
- **Compilation Options**
  - **Convert images to monochrome**: Select when you want PlanetPress Design to automatically convert every color bitmapped image resource to grayscale when it converts the document. Select Always to have PlanetPress Design always perform the conversion. Select Never to prevent PlanetPress Design from ever performing the conversion. Select PPD to have PlanetPress Design perform the conversion only when the PostScript Printer Description (PPD) file associated with the document does not support color. This option applies only for this document. The Convert images to grayscale option in the User Options dialog box determines the option that appears here by default when you create the document.
  - **Image scanline orientation**: See "Scanline Orientation" (page n) and "Adjust the Scanline Orientation of Images" (page n).
  - **Picture quality compression level** : Specify, in percentage, the compression level of pictures in your document. A higher number means less compression, thus a larger picture size and better quality.
  - **Picture Quality**: Line Art is more useful for illustrations and text that can be converted to line art. Photo is more useful for images such as photos, landscapes, etc.

**Binding**

- **Use binding**: Select to make binding margins for the document available.
- **Odd pages**
  - **Horizontal**: Set the binding margin you want to leave along the left edge of each odd-numbered page. This value is relative to the left edge of the physical page, and units are as set in the User Options dialog box.
  - **Vertical**: Set the binding margin you want to leave along the bottom edge of each odd-numbered page. This value is relative to the bottom edge of the physical page.
- **Even pages**
  - **Horizontal**: Set the binding margin you want to leave along the right edge of each even-numbered page. This value is relative to the right edge of the physical page.

- **Vertical**: Set the binding margin you want to leave along the top edge of each even-numbered page. This value is relative to the top edge of the physical page.

**Attachments** (see also [Add PostScript Attachment Resources](#))

- **Attachments run before form**: These attachments will execute before the document.
- **Attachments run after form**: These attachments will execute at the end, after the document.

**Job Infos** ( aka jobinfo or jobinfos )

- **Info #**: The job info number.
- **Value**: Enter a static text value that you want to associate with the corresponding job info reference number. This value is typically overridden, if a job info value is passed by PlanetPress Suite Workflow Tool.
- **Ignore PlanetPress Suite Workflow Tool job info values**: Enable this option to prevent values passed by PlanetPress Suite Workflow Tool from overriding the values you have entered. This option is typically used for debugging purposes and is disabled by default.

**Scan code settings**

- **Setting file**: A scan code settings files from a Pitney Bowes system. Click Browse to locate the file on your hard drive.

**Metadata**

- **Group Marker**: Defines the boundaries of the Group level in the metadata.
    - **Marker**: Determines if there is no marker, if the condition defines the beginning (Begin) or end (End) of the Group level.
    - **Condition**: When this condition is true, a new boundary is created in the metadata.
- **Document Marker**: Defines the boundaries of the Document level in the metadata.
    - **Marker**: Determines if there is no marker, if the condition defines the beginning (Begin) or end (End) of the Group level.
    - **Condition**: When this condition is true, a new boundary is created in the metadata.
- **This document requires a metadata file**: Check to stop execution of this document if no metadata is found. This is especially useful when sending the document to a PlanetPress Workflow process.

**PlanetPress Talk after**: Enter PlanetPress Talk code that you want to execute after the document executes.

# Cacheable Execution Options

PlanetPress Suite offers compatibility with some of the major caching methods available on the market. This page explains each caching method and how it is used.

**None**

This caching method really isn't one. In essence, using this option will remove any sort of caching from the document when it is printed. This means that print jobs will most likely be much larger, but in some cases they will be compatible with older printers, especially those who do not support PostScript level 2 and 3.

**Generic**

This is the default caching method, and uses PostScript level 2 in order to cache images in your job. Any image that is repeated more than once is cached and referred to throughout your job. The more an image is used, the more the job file produced by this caching method will be smaller when compared to using the *None* option.

**VPS**

This method is used on printers that use the CREO VPS as a RIP. With this caching method, any page set as an Overlay or a Virtual page (See Page Types) can be set with the **Cacheable** option. Any static object placed on this page will be cached. For more information, see VPS Caching.

> ⚠️ If you place any variable data on an overlay or virtual page, the Cacheable option will have no use, since each individual page will be different, thus not cached.

**VDX**

This method caches virtual pages, used through the N-Up object. To the contrary of other caching methods however, VDX files need to be sent to the VDX RIP manually and are not "printed" by PlanetPress Suite. VDX output cannot be generated directly by PlanetPress Design, the document must be sent to the PlanetPress Suite Workflow Tools and used in a process. For more information, see VDX Caching.

**FreeForm 1 and FreeForm 2**

The EFI Fiery RIPs use either FreeForm 1 or FreeForm 2 caching methods, which are very similar except for the limitations of the master forms used. In FreeForm 1 you are limited to 100 Master Forms that are numbered, whereas in FreeForm 2 you can choose a name for the Master Form and the 100 limit is not present. For more information, see FreeForm Caching.

## FreeForm Caching

A feature of these RIPs is that you can create a master document, and have the content of a page of that master document appear as the background on the pages of a document when that document executes. Only one page of the master document appears on a given page of the document.

All the pages of a FreeForm document must indicate the number of the FreeForm *master* document to use when printing. If a master document contains more than one page, when a document that calls that master document executes, it cycles through the pages of the master document. For example, if the master document contains two pages, and the document that uses the master document outputs six pages, the first page of output contains the first page of the master document, the second page the second page of the master document, the third the first page of the master document.

To create a FreeForm master document:

1. In PlanetPress Design, create the document you want to use as the master document.
2. Double-click on the **Document** node to display the Document properties dialog box.
3. In the Document properties dialog box, click **Compilation options**.
4. In the **Caching method** box, select **FreeForm** and set the FreeForm options that appear.
   **This document is a master**: Select to make the current document a master document.
   **Document number**: Enter the index number for the master document. This is the number you use when you want to reference this master document. Index numbers must be in the range 1 to 100 inclusive.
5. Click **OK** to exit the Document dialog box.
6. Save the document.
7. From the **PlanetPress Design Button**, choose **Print**, select the printer or printers on which you want to install the master document, clear **Optimized PostScript Stream**, clear **Print to file**, set the **Number of copies** to 1, and click **OK**. Remember that the printers you select must be running FreeForm.

To associate a FreeForm document with a FreeForm master document:

1. In PlanetPress Design, open the document in which you want to reference the FreeForm master document.
2. Double-click on the **Document** node to display the Document properties dialog box.
3. In the Document properties dialog box, click **Compilation options**.

4. In the **Caching method** box, select **FreeForm**.
5. Set the **Document number** box to the index number of the FreeForm master document you want the document to use.
6. Click **OK** to exit the Document dialog box.
7. Save the document.
   Now when you execute the document on the printer on which the FreeForm is running, each page of the document includes the contents of the master document you specified in the document number box. Remember that in FreeForm, if a master document has more than one page, when the document that calls the master page executes, it cycles through the pages of the master document.

## FreeForm 2 Caching

The properties of all the pages of a FreeForm 2 document must include the ID of the FreeForm 2 *master* document as well as the number of the page to use when printing.

To create a FreeForm 2 master document:

1. In PlanetPress Design, create the document you want to use as the master document.
2. Double-click on the **Document** node to display the Document properties dialog box.
3. In the Document properties dialog box, click **Compilation options**.
4. In the **Caching method** box, select **FreeForm 2** and set the FreeForm 2 options that appear.
   - **This document is a master**: Select to make the current document a master document.
   - **Master ID**: Enter the name for the master document. This is the name you use when you want to reference this master document. The name can contain a maximum of 64 characters. It cannot start with either **formC** or **formU**, and cannot contain any of the following characters: char(0) through char(32), vertical bar (|), forward slash (/), backward slash (\), asterisk (*), question mark (?), double-quote ("), single quote ('), back tick (`), colon (:), less than sign (<), greater than sign (>). You should make certain the name you enter is not that of an existing master document on the printer on which you intend to install this master document. If the name is the same, this master document overwrites the existing master document when you install it.
5. Click **OK**

From the **PlanetPress Design Button**, choose **Print**, select the printer or printers on which you want to install the master document, clear **Optimized PostScript Stream**, clear **Print to file**, set the **Number of copies** to 1, and click **OK**. Remember that the printers you select must be running FreeForm 2.

To associate the pages of a FreeForm 2 document with a FreeForm 2 master document:

1. In PlanetPress Design, open the FreeForm 2 document that contains the pages that you want to associate with the Free-Form 2 master document.
2. In the Structure area, select the page you want to associate with a FreeForm 2 master document and double-click on it to display the **Page** properties dialog box.
3. In the Page properties dialog box, click **Basic attributes** and set the FreeForm 2 options.
   **Master ID**: Enter the name of the FreeForm 2 master document you want this page to reference. This option appears only if you selected FreeForm 2 in the Caching method box of the Conversion options in the Document dialog box.
   **Page number**: If the FreeForm 2 master document you entered in the Master ID box is a multi-page document, enter the page number of that document that you want this page to reference. This option appears only if you selected Free-Form 2 in the Caching method box of the Conversion options in the Document dialog box.
4. Click **OK**.
5. Repeat step 2 through step 4 for each page that you want to associate with a master document.
6. Save the document.

## VPS Caching

PlanetPress Suite gives you the ability to send documents in the VPS format that is compatible with CREO VPS RIPs on certain types of printers.

In VPS documents, overlay (fixed size) *and* virtual (variable size) pages defined as cacheable are placed in memory (cached) at the beginning of the job, and are referred to throughout the document. This means those cached elements are never repeated in the document and the job can be much smaller than a regular, non-cached job.

**To create a VPS document:**

1. In PlanetPress Design, create a new document.
2. Double-click on the **Document** node to display the Document properties dialog box.
3. In the Document properties dialog box, click **Compilation options** and in the **Caching method** box, select **VPS**.
4. Click **OK** to exit the Document properties dialog box.
5. Add a page to the document. See the section "Add a Document Page" in the chapter "Setting Up Pages."
6. In the Structure area or in the Page area, select the page and double-click on it.
7. In the Page properties dialog box, click Basic attributes.
8. Select **Cachable** and set **Page type** to Overlay or Virtual.
9. Click OK to exit the Page properties dialog box.
10. Add the content to the cacheable page.
11. Repeat step 5 through step 10 for each of the cacheable pages you want to add to the document.
12. Create the normal pages of the document, either calling a cacheable overlay page or using n-up objects as necessary to reference the content of the virtual pages. If you include an N-Up object on a page, make sure that it appears as the first object on the page in the Structure area.
13. Save the document.

The document is now ready for the printer.

## VDX Caching

You can create and print a document in VDX format using PlanetPress Design in conjunction with the PlanetPress Image action in PlanetPress Suite Workflow Tool. It is important to understand that in this case the document itself does not execute on the printer. Rather, the PlanetPress Image action merges the data with the document, and converts the result to VDX format.

To create a document in VDX format:

- In PlanetPress Design, define the document as a VDX document.
- Place any content you want the VDX format document to cache, on one or more virtual pages, and define each of those virtual pages as cacheable. In most cases you cache content that is extremely large and that is used more than once during document execution. Any content you intend to cache must be static. On the normal pages of the document, you use n-up objects to reference the cacheable content. You can scale the content of an n-up object, as well as rotate an n-up object; thus you can present the same virtual page at different sizes and angles.
- Once you complete your document, install it in PlanetPress Suite Workflow Tool.
- To execute the document and convert the result to VDX format, set up a process in PlanetPress Suite Workflow Tool and add a PlanetPress Image action to that process.
- When the process executes, the PlanetPress Image action executes the document and converts the result to VDX format.

This procedure describes how to set up a document in PlanetPress Design, so that the Create VDX action recognizes it and converts it to VDX format correctly. For help understanding and using the PlanetPress Image action, see the *PlanetPress Watch or Server 6 User Guide Addendum*.

**To create a document in VDX format:**

PLANETPRESS. SUITE

1. In PlanetPress Design, create a new document.
2. Double-click on the **Document** node to display the Document properties dialog box**.**
3. In the Document properties dialog box, click **Compilation options** and in the **Caching method** box, select **VDX**. If you do not select VDX, the Create VDX action in PlanetPress Suite Workflow Tool does not recognize the document as one it can convert to VDX format.
4. Click **OK**.
5. Add a page to the document.
6. Double-click on the **Page** node to display the Page properties dialog box.
7. In the Page properties dialog box, click **Basic attributes**.
8. Set Page type to Virtual and select **Cachable**.
9. Click **OK**.
10. Add the content to the virtual page. You can use any of the objects available in PlanetPress Design to add content to the virtual page. However, all of the content you add must be static.
11. Repeat step 5 through step 10 for each of the virtual pages you want to add to the document.
12. Create the normal pages of the document, using n-up objects as necessary to reference the content of the virtual pages.
13. Save the document.

**To display a virtual page on a normal page:**

1. On the normal page, choose **Home** | **N-UpPrinting**.
2. Move the pointer inside the Page area, click at the point at which you want to add the n-up object, and release to display the N-Up properties dialog box.
3. In the **N-Up** properties dialog box, click **Basic attributes** and enter the name, position, size, style, and condition properties for the n-up object.
4. In the **N-Up** properties dialog box, click **N-Up options** and in the **Page to execute n-up** box, select the virtual page you want this n-up object to display.
5. Set both the horizontal and vertical **Number of repeats** to **1**.
6. If necessary, use the **Scale** box to adjust the scale at which the n-up object displays the virtual page.
7. If necessary, click **Basic attributes** and adjust the **Angle** property to rotate the content of the n-up object.
8. Click **OK** to exit the N-Up properties dialog box.

## PPD Setup

PPD files, or PostScript Printer Description files are used by PostScript printer drivers to print to PostScript devices. PPDs contain printer-specific postscript commands that are used to control printer features, such as duplexing, stapling and sub-set stapling, tray calls and more. PPDs can be specific to a certain printer model, to a model line, or to a manufacturer. Generic all-purpose PPDs are provided with PlanetPress Suite and contain commands that should be understood by a large number of printers, but using the appropriate PPD is always better.

> PPDs are only used when printing through Optimized PostScript Stream or Printer-Centric mode. They are completely ignored when using Windows Driver output.

The PPD, along with the sample data file you associate with the document and the emulation you select define how your document handles its input data, and consequently determine the accuracy of the output the document produces with the data stream it receives at runtime.

Document setup also involves giving the document a name, setting default printer options for the document, associating one or more attachments with the document if necessary, setting any binding margins you want the document to use, and adjusting options that can improve the document performance at runtime.

Normally the default PPD can be used. However, if the default PPD fails, it is recommended you search for one made available by the manufacturer. You can modify any of the document setup settings at any point during the document creation process. If you change or modify the PPD associated with the document, you should verify the change does not compromise the accuracy of the output the document produces.

## Add or Remove PPDs

The default PostScript Printer Description (PPD) file PlanetPress Design provides works for the vast majority of printers. It is recommended that you use it when possible. In the event the default PPD and any of the other PPDs provided with PlanetPress Design do not work properly with your printer, you can add PPDs to your PlanetPress Design installation. Note that PlanetPress Design only accepts PPDs for PostScript Level 2 or higher printers.

To add a PPD from PlanetPress Design:

1. Double-click on the **Document** node to display the Document properties dialog box.
2. In the **Document** properties dialog box, click **Basic attributes**, and click the **Add PPD** button.
3. Use the **Select PPD File** dialog box to navigate to the PPD file you want to add, and click **Open**.
   If the file is a valid PPD file for a PostScript Level 2 or higher printer, PlanetPress Design adds it to its **PPD** folder. If you selected Invalid PPD notification in the User Options dialog box, PlanetPress Design reports any failure to add the PPD. The dialog containing the error message includes a checkbox you can use to suppress the message in future. This checkbox clears the Invalid PPD notification option in the User Options dialog box.
4. Refresh the PPD list that appears in PlanetPress Design.

To add a PPD from outside PlanetPress Design:

1. Drag the PPD you want to add to PlanetPress Design from Windows Explorer or your desktop to either the Document structure or Document page areas.
2. The current document will be configured to use that PPD automatically. If you dragged several PPDs at once you will have to select the appropriate PPD from the list available in PlanetPress Design.

To remove a PPD from PlanetPress Design:

1. Remove the PPD file from the PlanetPress Design**PPD** subfolder of the Windows Common Files folder.
2. Refresh the PPD list that appears in PlanetPress Design.

## Refresh the PPD Lists

To refresh the list of PPDs using the Refresh PPD List button:

1. Double-click on the **Document** node to display the Document properties dialog box.
2. In the **Document** properties dialog box, click **Basic attributes**, and then click the **Refresh PPD List** button.

## Specify Job Infos

To specify job info variables in a PlanetPress Design document:

1. Double-click on the **Document** node to display the Document properties dialog box.
2. Click **Job Infos**.
3. Specify any job info values. Job info values are typically passed by PlanetPress Suite Workflow Tool. Job info numbers and the information associated with them may vary depending on input types as well as PlanetPress Suite Workflow Tool configurations, with the exception of job info 0%, which is reserved for the job file name
   **Info #**: The job info number.
   **Value**: Enter a static text value that you want to associate with the corresponding job info reference number. If a job info value is passed by PlanetPress Suite Workflow Tool the static text value is overridden.

**Ignore PlanetPress Suite Workflow Tool job info values**: Enable this option to prevent values passed by PlanetPress Suite Workflow Tools from overriding the values you have entered. This option is typically used for debugging purposes and is disabled by default.

4. Click **OK**.

## Associate Attachments with a Document

When you associate an attachment with the document, you define whether you want that attachment to execute before or after the document executes. You can associate a condition with each attachment that determines whether the attachment executes.

To associate attachments with a document using drag and drop:

- In the Structure area, select the attachment resources you want to associate with the document, and drag and drop them onto the document symbol ().

To associate attachments with a document using the Document properties dialog box:

1. Double-click on the **Document** node to display the Document properties dialog box.
2. In the **Document** properties dialog box, click **Attachments**.
3. Add the attachments you want to execute before and/or after the document.
   **Attachments to execute before document**: Display, define, and/or edit the list of attachments you want to execute before the document executes. The attachments execute sequentially, from top to bottom. Right-click in this area and use the menu that appears to add or delete items, move items up or down in the list, or clear the list altogether. You can also delete an item by selecting it and pressing **DELETE**. You can also move an item up or down in the list, or move it to the Attachments run after document list, by clicking it and dragging it to the new location. Double-click an item to display it. Note that you cannot set a condition on any attachment you execute before the document executes, as these attachments execute before the document evaluates conditions.
   **Attachments to execute after document**: Display, define, and/or edit the list of attachments you want to execute after the document executes.
4. Click **OK**.

To add or edit an item in the attachments list:

1. Double-click on the **Document** node to display the Document properties dialog box.
2. In the **Document** properties dialog box, click **Attachments**.
3. To add an attachment, right-click in the appropriate list and choose **Add**.
4. Use the Attachment Selection dialog box to add a new item or edit an existing one.
   **Attachment name**: Select one of the existing attachment resources in the document. If the attachment resource does not yet exist in the document, use the Attachment button to add it.
   **Attachment button**: Click to display the Select attachments dialog box and select an attachment resource to add to the document. When you exit this dialog box, PlanetPress Design adds the attachment resource to the document, and returns you to the Attachment Selection dialog box.
   **Condition**: Specify the condition under which this attachment executes. You can select an existing condition from the drop-down list, or define a condition using PlanetPress Talk expression. The current value of the condition appears below the Condition box. Clear the contents of the box if you want the attachment to always execute.
5. Click **OK**.

# ![icon] Setting Up Pages

Each page of the document can have its own distinct characteristics.

This chapter explains what is meant by a document page, describes the difference between a page that executes and one that outputs, introduces the features that can help you lay out your pages, and provides guidelines and procedures for working with pages.

## Page Properties

To view or edit properties using the Object Inspector:

1. In the Structure area, select the page whose properties you want to view or edit.
2. In the Object Inspector, make any necessary modifications to the properties.

To view or edit properties using the Page properties dialog box:

1. In the Structure area or in the Page area, select the page and double-click on it to display the **Page** properties dialog box.
2. Use the Page properties dialog box to view and/or edit the page properties.
3. In the Page properties dialog box, click **OK**.

The properties of a page are as follows:

### Basic attributes

- **Display Name**: Enter a name for the page, which will be displayed in the Document Structure Pane.
- **Name**: Enter a name under which the page is known in PlanetPress Talk commands. The name you choose should be both descriptive and unique, cannot begin with a number, and can contain only the following ASCII characters: under-score (_), upper and lower case letters of the alphabet, all digits 0 through 9.If you use an underscore in the name, it should not appear as either the first or last character of the name as this may cause internal conflicts in the software. Names are case-insensitive and must be unique; no two elements in a document can have the same name. Names can be a maximum of 50 characters in length. Finally, PlanetPress Talk variable and command names are reserved words; you cannot use any of these reserved words as a name.
- **Execution Options**
    - **Page ejects**: Select to include this page in the output after it executes. This option applies only to pages that execute. In the Structure area, a red "X' appears in the page symbol for a page that executes but does not output.
    - **Cachable**: Select to have the printer cache this page. This option is enabled and is applied based on the document's caching method (see see ["Set Up a Document" (page 105)](#)) and on the page type as follows:

| Document caching method: | Result: |
|---|---|
| None | Option disabled. No images are cached. |
| Generic (or PostScript Level 2) | Option disabled. All images are cached. |
| FreeForm | Option enabled. Selected virtual pages are cached (overlays are not cached). |
| FreeForm 2 | Option disabled. See the options below. |
| VPS | Option enabled. Selected overlays and virtual pages are cached. |
| VDX | Option enabled for virtual pages only. Selected virtual pages are cached (overlays are not cached). |

- **Master ID**: Enter the ID of the master document this page must call. This option is only displayed if FreeForm 2 is selected as the document's caching method in the document's compilation options. See [FreeForm 2 Caching](#).

- **Page number**: Enter the number of the page within the master document identified in the **Master ID** box. This option is only displayed if FreeForm 2 is selected as the document's caching method in the document's compilation options.
- **Force front**: Select if you are printing your document in duplex mode, and want to force this page to always print on the front side of the paper. You can fine-tune this by using the When box to define the condition under which the document forces this page to print. This option is not available on virtual pages for obvious reasons. Also note that this option may not work properly when generating previews.
  - **When**: Define the condition that must resolve to True for the document to force this page to print on the front side of the paper. You can enter the condition manually, select an existing global condition from the box, or define a condition using a PlanetPress Talk expression. This option is available only when you select Force front.
- **VPS End Booklet**: Only available when VPS is set as a caching method in the document. Enter or select a condition which will trigger when to end a booklet in the Creo VPS caching method.
- **Include this page when** (See "Converted Document" (page 694))
- **Print:** Select to include this page when PlanetPress Design converts the document to produce printer output.
- **Fax:** Select to include this page when PlanetPress Design converts the document to produce output in PlanetPress Fax.
- **Archive:** Select to include this page when PlanetPress Design converts the document to produce output in PlanetPress Image.
- **Page Type and Condition**
  - **Page type**: Select the type of the page: normal, overlay, or virtual. See Page Types.
  - **Condition**: Enter the condition, if any, you want to set on the page. This determines whether or not the page executes at runtime; the page executes only if the condition is True. You can enter the condition manually, select an existing global condition from the box, or define a condition using a PlanetPress Talk expression.

**Paper handling**

- **Page size**: Select the paper format for this page. The format that appears here by default is the one set for the document in the User Options dialog box. The formats available depend on the PPD you selected in the Document properties dialog box. In addition to the formats available in the PPD, you can also select Custom and use the Page width and Page height boxes to define a custom paper size. A custom paper size is useful if the printer supports the paper size you require but the PPD does not include it in its list of supported paper formats. Before you define a custom paper size you should consult your printer documentation to verify your printer can handle the custom paper size you define. When you select a page size you should ensure you set the appropriate input and output trays for the page.

  If you are printing in 2-up mode, the Default page size box in the Document properties dialog box determines the size of the paper on which the two pages print, and the page size you select here determines the scaling required to fit the two pages on that paper size.
- **Page width**: Enter the width of the custom page size. Units are as selected in the User Options dialog box. This option is available only when you select Custom in the Page size box.
- **Page height**: Enter the height of the custom page size.
- **Duplexing**: Select the duplexing option for the page. The default duplexing option that appears here is the one you set for the document in the Document properties dialog box. The duplexing options available depend on the PPD you selected in the Document properties dialog box.
- **Paper orientation**: Select the orientation for this page.
- **Add Cut Marks:** Cut Marks are small horizontal and vertical lines that appear outside the physical boundaries of a page. They are commonly used in N-Up projects that print multiple pages on large paper. They indicate where to cut the pages in post-processing.
  - **X Offset:** Set the distance of the cut mark stems from the left and right edges of the page.
  - **Y Offset:** Set the distance to the cut mark stems from the top and bottom edges of the page.
  - **Cut Mark Length:** Set the length of the stems of the cut marks.

**Paper input/output**

- **Selection type**: Select the type of input paper tray the printer uses for this page. The Selection type that appears here by default is the one set for the document in the Document properties dialog box. The trays available depend on the PPD you selected for the document. The selection you make here determines the contents of the Paper source. Select Input tray to specify a specific input tray; and use the Input tray box that appears in the Paper source area when you select this option to select the tray. Select Match paper size to have the printer automatically use the tray that contains the paper format defined for this page. Select Manual Feed to use the printer's manual feed. Select Media selection to set specific media characteristics for the page, and use the Media type, Media color, and Media weight options that appear in the Paper source area to specify these characteristics.
- **Input tray**: Select the input tray. This option is available only when you select Input tray as the Selection type option.
- **Media type**: Select the paper type. This option is available only when you select Media selection as the Selection type option.
- **Media color**: Select the paper color. This option is available only when you select Media selection as the Selection type option.
- **Media weight**: Select the paper weight. This option is available only when you select Media selection as the Selection type option.**Output tray**: Set the output paper tray. The options available depend on the PPD selected for the document in the Document properties dialog box.

**Advanced paper handling**

- **Paper handling before the page**: Display, define, and/or edit the paper handling operations you want the printer to execute before it executes this page. The printer executes the operations sequentially, from top to bottom. Right-click in this area and use the menu that appears to add or delete items, move items up or down in the list, or clear the list altogether. You can also delete an item by selecting it and pressing **DELETE**. A warning appears in red for any operations not supported in the new PPD.
- **Paper handling after the page**: Display, define, and/or edit the paper handling operations you want the printer to execute after it executes this page.
- To add or edit Advanced Paper Handling, see Advanced Paper Handling.

**Overlays**: In order to use overlays, you need to have at least one defined. See Add or Remove Overlays.

**Attachments**: To use attachments, see Associate Attachments with a Page.

**PlanetPress Talk before**: enter PlanetPress Talk code that you want to execute before the page executes

**PlanetPress Talk after**: enter PlanetPress Talk code that you want to execute after the page executes.

# Page Types

There are three types of pages in PlanetPress Design: normal pages, overlay pages, and virtual pages.

## Normal Page

A normal page is one that can print depending on the conditions you set for that page.

## Overlay Page

An overlay page is one that you can place either underneath or over top of the contents of another page. A simple example of an overlay that goes under a page is a company logo that appears as a background graphic on all pages of the document.

An overlay page can print only if it is associated with a normal page, and only if the normal page with which it is associated prints. You can have many overlay pages associated with a single page.

In PlanetPress Design, overlay pages are displayed in the Page area in mauve with a yellow border. Mauve and yellow are both defaults, which you can modify in the User Options dialog box. In the Structure area, you can distinguish overlay pages by the horizontal lines that appear inside the page symbol.

When you rename an overlay page associated with a normal page, you break the association between both pages. To fix this, you must recreate the association.

## Virtual Page

A virtual page is a page you want to execute n-up. N-up means *n* instances of the page print on a single sheet of paper.

The only time a virtual page executes is when an n-up object executes it. If you have a virtual page in a document, and a no n-up object that executes that virtual page, the virtual page does not execute.

In PlanetPress Design, virtual pages are displayed in the Page area in gray with a yellow border. Gray and yellow are both defaults, which you can modify in the User Options dialog box. In the Structure area, you can distinguish a virtual page by its symbol.

## Execution Order of Pages

The order in which the normal pages appear in the Structure area determines the order in which the document executes them. Normal pages are always executed from top to bottom, once per datapage, unless this order has been modified by the use of a RunPage (see Creating and using Runpages), or a Repeat object (See Line Repeat and Data Overflow).

You can change the execution order of the pages by changing the order in which they appear in the Structure area.

# Advanced Paper Handling

Advanced Paper Handling (APH) commands are used to conditionally control specific options on your printer. APH is only supposed when using Printer-Centric and Optimized PostScript run modes, and will not function when using Windows Driver Output.

The APH commands available to you are determined by the PPD that is selected in your document's properties (see Set Up a Document under Basic Settings).

To add or edit an advanced paper handling operation:

1. Open the page's properties (see Set Up a Page), and click on the Advanced Paper Handling section.
2. To **add** an operation, right-click and choose **Add** in the appropriate paper handling area. To **edit** an existing operation, double-click it.
3. In the Paper Handling dialog box, enter a new paper handling operation or edit the existing one.
   - **Section**: Select the name of a paper handling property. This determines the contents of the Selection box. The PPD you selected in the Designed for box of the Document properties dialog box determines the contents of the Section box.
   - **Selection**: Select the value you want to associate with the paper handling property selected in the Section box.
   - **Condition**: Specify the condition under which this operation executes. You can select an existing condition from the drop-down list, or define a condition using a PlanetPress Talk expression. The current value of the condition appears below the Condition box. Clear the contents of the box if you want the operation to always execute.

4.  Click **OK**.

## Associate Attachments with a Page

When you associate an attachment with a page, you define whether you want that attachment to execute before or after the page executes. You can associate a condition with each attachment that determines whether the attachment executes.

To associate attachments with a page using drag and drop:

- In the Structure area, select the attachment resources you want to associate with the page, and, staying in the Structure area, drag and drop them onto the page with which you want to associate them.
  A symbol for each of the dropped attachments appears on the right of the page in the Page area.

To associate attachments with a page using the Page properties dialog box:

1.  In the Structure area, select the attachment resources you want to associate with the page, and, staying in the Structure area, drag and drop them onto the page with which you want to associate them.
2.  In the **Page** properties dialog box, click **Attachments**.
3.  Add the attachments you want to execute before and/or after the page.
    **Attachments run before page**: Display, define, and/or edit the list of attachments you want to execute before the page executes. The attachments execute sequentially, from top to bottom. Right-click in this area and use the menu that appears to add or delete items, move items up or down in the list, or clear the list altogether. You can also delete an item by selecting it and pressing **DELETE**.
    **Attachments run after page**: Display, define, and/or edit the list of attachments you want to execute after the page executes.
4.  Click **OK**.

## Add or Remove Overlays

To add or remove overlays:

1.  In the Structure area or in the Page area, select the page and double-click on it to display the Page properties dialog box for the page to which you want to add, or from which you want to remove, overlays.
2.  In the **Page** properties dialog box, click **Overlays**, and select the overlay pages you want to add to this page, and/or clear the overlay pages you want to remove from the page. You can also adjust the order in which you want to layer the overlays.
    **Underlays**: Select the overlay pages you want to place under this page, and clear any overlays you do not want to place under the page. All overlay pages in the document appear in this list. PlanetPress Design layers the selected pages under this page, in the order in which they occur in the list (topmost page in the list becomes the bottommost layer). You can drag and drop the items in the list to control the order in which PlanetPress Design layers the overlays.
    **Overlays**: Select the overlay pages you want to place on top of this page, and clear any overlays you do not want to place on top of the page.
3.  Click **OK**.
    Any overlays you add to a normal page are displayed as part of that page in the Page area, however, you cannot edit the content of an overlay from the normal page.

PLANETPRESS. SUITE

# PlanetPress Design Objects

Objects are the pieces of your document that are inserted in your page, such as text boxes, barcodes, images and data selections. This chapter explains each object and its unique properties.

The following objects are available to you in PlanetPress Design:

- "Text and Box Object" (page 130)
- "Data Selection Object" (page 140)
- "Picture Object" (page 150)
- "N-Up Object" (page 149)
- "Postal Address Object" (page 146)
- "PlanetPress Talk Object" (page 147)
- Pitney Bowes Scan Code
- "Shape Object" (page 149)
- "Barcode Object" (page 152)
- "Business Graphic Object" (page 164)
- Capture Field Object

While some properties are common to all objects, they each have their unique properties, which will be described in this chapter.

# View or Edit the Properties of an Object

To view or edit properties using the Object Inspector Pane:

1. Select the object.
2. In the Object Inspector Pane, make any necessary modifications to the properties.

To view or edit properties using the properties dialog box:

1. Double-click the object.
2. Edit the Properties for the object.
3. Click **OK**.

Common Object Properties

The following properties are common to all objects in PlanetPress Design:

## Basic Attributes

- **Display Name**: Enter a name for the object. Although PlanetPress Design supplies a default name, it is recommended you choose a name that reflects the content or purpose of the object.
- **Name**: Enter a name by which the object can be referred to through PlanetPress Talk. The name you choose should be both descriptive and unique, cannot begin with a number, and can contain only the following ASCII characters: underscore (_), upper and lower case letters of the alphabet, all digits 0 through 9.If you use an underscore in the name, it should not appear as either the first or last character of the name as this may cause internal conflicts in the software. Names are case-insensitive and must be unique; no two elements in a document can have the same name. Names can be a maximum of 50 characters in length. Finally, PlanetPress Talk variable and command names are reserved words; you cannot use any of these reserved words as a name.

- **Position and size** Note that you can also click and drag on the object after you add it to the document to adjust its position and/or size. Also note that the unit of measure used is the one set in the PlanetPress Design User Options dialog box (see Miscellaneous Preferences)
    - **Left**: Set the distance to offset the left edge of the object from the left edge of the page. If the object is within a group, this is the distance to offset the left edge of the object from the left edge of the group that contains it.
    - **Top**: Set the distance to offset the top edge of the object from the top edge of the document page.
    - **Width**: Set the width of the object. The width of the contents of an n-up object depends on the scaling of each of the repeated pages, the number of repeats of each page, and the horizontal space between each repeat; the Width box does not update to reflect these settings.
    - **Height**: Set the height of the object. The Repeat properties of an n-up object use the height as it appears in the Height box.

    > The following apply to both the Width and Height settings:
    > - Data selection, barcode and picture objects automatically adjust height and width, so you should not define them manually.
    > - If the &width or &height variable is defined in the PlanetPress Talk Before code of an object, the setting in its basic attributes will be ignored.
    > - The maximum width and height of any object is 32 inches (80 centimeters).
    > - When you adjust the width of a group, PlanetPress Design scales the width of each element in the group, such that the new width of the group accommodates all the elements and preserves their spatial relationships.

    - **Dynamic height**: This option is available only in text objects, and only when word wrap is on in the text object. You use it when you define a background box for the text object. It determines whether the background box, when word wrap is on, adjusts to fit the word-wrapped text. Select to have the height of the background box adjust to accommodate all of the word-wrapped text. Clear to leave the size of the background box fixed and independent of the word-wrapped text. You define the default setting for this option in the User Options dialog box. If you select this option, you cannot adjust the object height.
    - **Angle**: Set the angle of rotation. The pivot point for the rotation is the lower-left corner of the object. Note that if you use PlanetPress Talk code to define the angle of rotation, PlanetPress Design automatically locks the object to prevent an inadvertent change to this value due to selecting, moving, or resizing the object using the mouse or keyboard shortcuts.
- **Style and Condition**
    - **Style**: Select the style for the object. This is the style all text in the object uses by default. It is also the style any PlanetPress Talk code you enter in the PlanetPress Talk properties of the object uses by default. The default style is the style that appears by default in this box. Note that the style for a group does **not** override any styles referenced by individual objects/groups within that group.
    - **Size**: Select or type the point size for the font the style uses.
    - **Bold**: Click to turn the Bold property for the style on or off.
    - **Italic**: Click to turn the Italic property for the style on or off.
    - **Underline**: Click to turn the Underline property for the style on or off.
    - **Color button**: Click to select a color for the style using the Color Picker.
    - **Color box**: Displays the current color for the style.
    - **Ratio**: Enter the percentage by which you want to shrink or stretch the font spacing in the style.
    - **Condition**: Select the condition you want to associate with the object. You can select or enter the name of an existing global condition. You can also enter a PlanetPress Talk expression that defines the condition. The expression must evaluate to a Boolean value.

## Repeat

The options available here depend on the Repeat Mode selected. For more information, see Line Repeat and Data Overflow.

**No Repeat**: No option to configure, the object does not repeat.

**Static Repeat**:

- **Horizontally**
  - **Number of repeats**: Enter the number of times you want to repeat the object or group along the horizontal axis.
  - **Space between each repeat**: Enter the amount of space to leave between each repeat of the object or group, along the horizontal axis.
- **Vertical**
  - Number of repeats: Enter the number of times you want to repeat the object or group along the vertical axis.
  - Space between each repeat: Enter the amount of space to leave between each repeat of the object or group, along the vertical axis.
- **Expand to make a grid**:  Select this checkbox to have PlanetPress Design use the values you enter in the Number of repeat boxes to create a table rather than a single row and/or column. The number of horizontal repeats, plus one for the source object, becomes the number of rows in the table. The number of vertical repeats, plus one for the source object, becomes the number of columns in the table. The total number of cells in the table is equal to the number of rows times the number of columns.

**Line Repeat** (not available for PDF data files):

- **From line**: Enter the line on which the repeat starts.
- **To line**: Enter the line at which the repeat stops. The value "0" means it will repeat until the end of the current data page.

> You can also click the Use Data Selector button to select the line range.

- **Repeat choice**: Select the axis or axes along which PlanetPress Design repeats the source object or group.
- Select **Horizontal** to create the repeats along the horizontal axis, starting to the right of the source object or group and progressing towards the right.
- Select **Vertical** to create the repeats along the vertical axis (a column), starting below the source object or group, and progressing towards the bottom of the document page.
- Select **Both, horizontal first** to create the repeats along both the horizontal and vertical axis, starting with repeats along the horizontal axis first (a table, constructed row by row).
- Select **Both, vertical first** to create the repeats along both the horizontal and vertical axis, starting with repeats along the vertical axisfirst. Use the Iterations per row/column box to set the number of repeats in each column.
- **Horizontal displacement**: Enter the amount of space between each repeat of the source object or group, along the horizontal axis.
- **Vertical displacement**: Enter the amount of space between each repeat of the source object or group, along the vertical axis.
- **Iterations per row/column**: Enter the number of repeats in each row (if you selected Both horizontal first in Repeat choice) or in each column (if you selected Both vertical first in Repeat choice). This option is available only when you select either Both horizontal first, or Both vertical first in Repeat choice.
- Use the **Iterations per row/column** box to set the number of repeats in each row.
- **Iteration condition**: Set a condition on the current line of data. This is effective only within the line range defined in the From/to line property. It determines whether the data selections of the element that PlanetPress Design is currently processing and that reference that line or record appear in the output. If a data selection does not appear in the output (the condition resolves to False), PlanetPress Design inserts a blank space for it in the element. If the source object or group contains a data selection object, and you do not want a blank space to appear when the condition you set here resolves to False, you can set a line condition in that data selection object to remove the blank space. Any condition set on the source object or group takes precedence over this condition. In other words, if a condition set on the source

object or group resolves to False, the object or group is not displayed, regardless of whether the condition here resolves to True or False.

- **Condition to Exit and Overflow:** Use this condition to manage data overflow. Set it the to number of iterations after which this object should stop processing data from the current data page. When this condition becomes true, PlanetPress Design executes the remaining objects in the page and repeats the page until all the remaining data has been processed.

**Line Repeat (XML)**:

When using an XML Data file, the Line Repeat options change slightly. The "From line" and "To line" disappear, replaced by the following options:

- **XPath**: Displays the current path to the XML element you are repeating on. Empty until the data selector has been used to select an XML value.
- **From iteration**: Enter the iteration of the XML element on which the repeat starts.
- **To iteration**: Enter the iteration at which the repeat stops. The value "0" means it will repeat until the last XML Element.
- **Use Data Selector**: Click to open the data selector and choose which element to repeat on.
- **Do not overflow if last record reached**: Select this option to prevent an overflow from occurring if you have reached the last record.

## Snapping Points

- **Snap to previous object**: Check to snap the object to the closest snapping point set in a previous object.
  - **Snapping point selection**: Select one of the 9 positions to select where this object will snap to with the other object's snapping point.
  - **Horizontal offset**: Enter the horizontal distance you want to appear between this object's snapping point and the snapping point of the object it snaps to.
  - **Vertical offset**: Enter the vertical distance you want to appear between this object's snapping point and the snapping point of the object it snaps to.
- **Set snapping point**: Check to set this object as a snapping point for the next object(s) that have the *Snap to previous object* option set.

## Manipulation

- **Scalable orientation**: Define the axes along which you can resize the object/group when you click and drag the handles on its bounding box or select it and use keyboard shortcuts to resize it. This does not prevent changes to the Height and Width properties of the object/group using either the Object Inspector or the object's/group's properties dialog box.
  - Select **Horizontal** only to restrict resizing the object/group along the X axis.
  - Select **Vertical** only to restrict resizing the object/group along the Y axis. Select Horizontal Vertical to permit resizing along both the X and Y axes.
  - Select **None** to prevent resizing along either the X or Y axis. If you use PlanetPress Talk code to define the Angle, Height, Width, Top, or Left properties of an object/group, PlanetPress Design automatically sets Scalable orientation to None, to prevent an inadvertent change to the PlanetPress Talk code due to a resize of the object/group using the mouse or keyboard shortcuts.
- **Movable orientation**: Define the axes along which you can move the object/group when you click and drag it in the Page area or select it and use keyboard shortcuts to move it. This does not prevent changes to the Left and Top properties of the object/group using either the Object Inspector or the object's/group's properties dialog box.
  - Select **Horizontal** only to restrict moving the object/group along the X axis.
  - Select **Vertical** only to restrict moving the object/group along the Y axis.

- Select **Both** to permit moving along both the X and Y axes.
- **This object can be selected on the page**: Select to permit selection of the object/group with the mouse in the Page area. Clear to lock the object/group. When you lock an object/group, you cannot select it using the mouse in the Page area, or resize or move it using the mouse or keyboard shortcuts. This option applies only to selecting, moving and resizing the object/group in the Page area and does not affect selecting the object in the Structure area, or using the Object Inspector to resize or move the object/group. Note that when you clear this option, PlanetPress Design automatically sets both Scalable orientation and Movable orientation to None to also prevent moving and resizing the object or group using the mouse or keyboard shortcuts. You can also use the items in the Tools menu to lock and unlock objects/groups.

## PlanetPress Talk Before and PlanetPress Talk After

Both the PlanetPress Talk Before and PlanetPress Talk After options are discussed in the PlanetPress Talk Chapter. See PlanetPressTalk Before and After.

## Preview options

At the bottom of the object property window, two options are available to control the Object Preview window (see Use the Object Preview):

- **Show/Hide Object Preview**: This button controls whether or not the object preview appears.
- **Data page selector**: Controls which data page to use for the preview. Enter a data page, or use the up/down arrows to navigate through the pages.

# Line Repeat and Data Overflow

The Repeat properties of an object or group are useful when you want to create a row, column, diagonal, or table from that single object or group. PlanetPress Design repeats the object or group to create the structure you specify. The Repeat properties can speed up the document design process and in some cases make the document more efficient to execute by optimizing the PostScript code for the resulting structure as a whole.

There are two types of repeats: static and line. In a static repeat, data selections remain static; the data selections in each repeat are identical to those in the source object or group. In a line repeat, data selections change with each repeat.

A line repeat is meaningful only with an object whose data selection exists on a single line. When PlanetPress Design performs the repeat, it advances to the next line of the data page after each repeat of the object. The data selection thus changes with each repeat. In the case of a group, each of the data selections referenced by objects in the group must exist on a single line (or in the case of a database emulation or XML file, in a single record). Note that when using Line repeat with a Repeat choice of Vertical only with a Vertical displacement, an initial displacement will be performed before the first item.

Note that in both cases, the specific line number you define in the Data properties of the object or objects is not important. You determine the start and end lines for the repeats when you set up the line repeat. When you do this, PlanetPress Design automatically sets the values of the From line and To line boxes in the Data properties of each object to **&current.line**. This ensures the data selection changes with each repeat. If you use a Custom data selection, you must manually adjust the references to the line (or record) to **&current.line**; if you do not make this adjustment, the Custom data selection remains the same for all repeats.

The start and end lines you define in the Repeat properties also determines the total number of repeats. In the Repeat properties you also define the layout you want to create, the horizontal and/or vertical order in which you want the data selection

changes to occur. An Iteration condition can also be applied to determine whether the data selections that reference the current line or record that are being processed appear in the output.

**When is a Line Repeat useful?**

A line repeat is useful when all of the following are true:

- The input data contains two or more contiguous lines or records with the same structure.
- In your document, you want to reference the data on those lines or records by proceeding consecutively, one line or record at a time.
- You cannot accomplish your goal only by creating data selections that cover the line range.
- The input data contains more text than a PlanetPress Design document page can accommodate, due to layout or paper size constraints.

When PlanetPress Design is processing an object with repeat properties and this condition becomes true, it stops repeating the object, performs the remaining objects on the document page and then performs the whole page once again, only this time using the surplus data. When all the data included in the current data page has been printed, PlanetPress Design stops repeating that page and goes on to the other pages that may be included in the document.

The fact that the same page is repeated to accommodate the surplus data does not mean that the initial page and the additional pages have to have the same appearance. The status of the condition to exit and overflow may be used to turn on objects that were turned off on the first page and vice versa. Note that all these objects must belong to the same group.

**List of available variables when overflowing:**

| Variable Name | Type | Available on | Description |
|---|---|---|---|
| &current.overflowcount | integer | Object & Page | Indicates the overflow page number. Is always 0 on the first page and non-overflowing pages. Is the same before and after the object, wherever it is placed on the page or document order. |
| &current.overflowing | boolean | Object & Page | A true/false variable indicated if you are in an overflow loop. Is always false before the first overflow loop, true for all objects and pages during iteration, and false after the object after the last overflow loop. |
| &current.iteration | integer | Object only | The current iteration of the whole object, reset only when the overflow is finished on all pages. Useful for determining the total number of objects repeated in all pages. |
| &iterationcount | integer | Object only | The current iteration on this page, reset on ever new overflow page. Most common use is to determine the maximum number of objects on the page in the overflow condition, e.g. =&iterationcount >= 15 |

# Snapping Points

Snapping point are points on an object or group that you use to connect one object or group to another such that changes to the size or position of one affects all those snapped to it. As a simple example, consider the second page of a form letter in which the last paragraph (a text object) is followed by a signature (a picture object). The last paragraph can vary in length but the signature must always appear in the same position relative to the last line of the paragraph. You can accomplish this by connecting the text object to the picture object using snapping points.

> If you snap an object to the bottom of a text object whose contents vary in length, and you want that object to always remain in the same position relative to the last line of text in the text object, be sure you select Dynamic height in the Basic attributes of the text object. Recall that the Dynamic height option is available only when Word wrap is on.

An object or group has nine snapping point sites on its bounding box: one on each corner, one in the middle of each edge, and one in the center of the object.
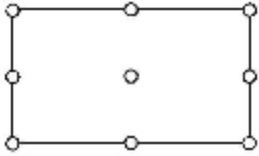
You can set a snapping point at any of the nine snapping point sites. When you snap two objects/groups together, PlanetPress Design positions the relevant snapping point of each object/group such that the two points lie one on top of the other.

You can also set a horizontal and/or vertical offset for a snapping point. You can do this to control the spacing between the two objects/groups.

> When using snapping points with objects that may dynamically change width or height, you should only use Top-Left or Bottom-Left snapping points, as other snapping points may not work properly.

Here are some key points to remember when using snapping points:

- The order in which snapping points are evaluated is top-to-bottom, as seen in the Document Structure Pane for the page. Any object that has the *Snap to previous object* option set will snap to the first object it encounter before it. If no object before it has the *Set snapping point* option set, it will snap to the top-left corner of the page it is located on.
- You may snap any number of objects to a single snapping point on an object, and you may also snap groups of objects together.
- Objects will not snap if they are not on the same page.
- Objects called using the PlanetPress Talk $element command and that have snapping points set will follow snapping behaviors as if they were originally part of the page. The order in which snapping points are evaluated is the same as if you replaced the PlanetPress Talk object with each individual object called by it.

## Object Preview Window

The Object Preview window lets you see objects before they are actually created. The object preview window has its own Message area displaying PlanetPress Talk error messages.

You use it to preview the effect of property settings, and to help determine the appropriate setting for a given property. As you change the property settings of an object or a group, the Object Preview updates to display the result of the modifications. It can also display messages from the PlanetPress Talk Converter that can help you debug the object or group.

To show or hide the Object Preview:

- In the properties dialog box of any object, click the **Show/Hide Object Preview** button. Note that by default, the Object Preview will always appear when you open an object's properties, even when you hide it. Its position is also reset. You can change this behavior in the PlanetPress Design User Preference (see Miscellaneous Preferences).

The following options are available in the Object Preview window toolbar:

- **Zoom in**: Zoom in by 25% zoom.
- **Zoom level**: Set the zoom level in percentage. The default zoom level in the Object Preview is 75%. The zoom value can be any value from 10 to 1000.
- **Zoom out**: Zoom out by 25% zoom.
- **Fit in window**: Adjusts the zoom level so the object appears in its entirety in the preview window.

- **Fit object width in window**: Adjusts the zoom level so the width of the object fits the width of the preview, ignoring the object's height.
- **Show Messages**: Click to make the message area appear or disappear.

You can reposition the contents of the Object Preview by clicking and dragging the preview around, or by using the scrollbars.

# Text and Box Object

Text and Box objects are essentially the same object. The Box object is offered as a separate menu in the ribbon for convenience - it gives you automatic starting options for borders that appear around the text object. On the other hand, the Text object will not show any borders by default.

The following properties tab apply to the Text and Box options:

## Borders

Borders are applied either through the Settings pre-defined border settings, or manually. To define the borders manually, start by choosing the color, width and style of the border you want to apply, then click on the appropriate button to add the border you want. To remove a border, click on the - *None* - line style then click on the border button you want to remove.

- **Custom Line Settings**
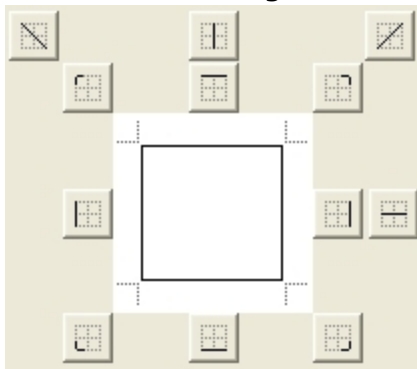
- **Top, Bottom, Left, Right sides**: Click one of these to select a side of the background box and apply the current line settings to it.
  - **Upper left, upper right, lower right, lower left corners**: Click to switch between a right-angled and rounded corner, or to apply the current line settings to a rounded corner. If you want to apply the current line settings to a rounded corner that is not currently visible, click that corner. If the rounded corner is currently visible, click the corner twice: the first click switches to the right-angled corner style, the second switches back to the rounded corner style and applies the settings to that style.
  - **Left, Right diagonal**: Click to apply the line settings to a diagonal that bisects the background box.
  - **Vertical, Horizontal line**: Click to apply the line settings to a line that bisects the background box.
- **Radius**: Set the radius of the circle to use to create round corners. The maximum value is one half of the smaller of the Width and Height values set for the object. Units are as set in the User Options dialog box. This setting applies to all rounded corners in the background box. If you resize the text object such that the value of the radius exceeds the maximum allowed, PlanetPress Design automatically adjusts the value of the radius to the maximum. In the latter this case PlanetPress Design does not update the value of the radius in the Object Inspector or the Text properties dialog box; you can update the value by entering a new value that exceeds the current maximum value.
- **Line width**: Enter the width of the line to use in the border. Units are points.
- **Line style**: Select the style for the line.
- **Line color box**: View the current color for the line.

- **Color button**: Click to select a color for the line using the Color Picker.
- **Settings**: These are pre-defined border settings you can use:
    - **None**: Select to make the border of the background box invisible.
    - **Box**: Select to use a rectangle as the border of the background box.
    - **Shadow**: Select to use a rectangle with a drop shadow as the border of the background box.
    - **Round**: Select to use a rectangle with rounded corners as the border of the background box. Use the Radius box to define the curve that occurs at each corner.

## Color

- **Fill**: Click to fill the object with a specific color or gradient. This enables the properties in the Fill group
    - **Color box**: View the current fill color for the background box.
    - **Color button**: Click to select the fill color for the background box using the Color Picker.
    - Gradient fill
    - **Gradient fill**: Select to use a gradient fill for the background box. The start color for the gradient is the fill color. Use the controls in this area to set the end color for the gradient, the number of gradations the gradient uses, and the direction of the gradient. Gradients are not available if the background box you select for the text/box object has one or more rounded corners.
        - **Color box**: View the end color for the gradient.
        - **Color button**: Click to select the end color for the gradient using the Color Picker.
        - **Steps**: Enter the number of gradations you want to use in the gradient, or use the spin buttons to set the value. The number of gradations determines how obvious the transition is between the Fill color and the Gradient fill color. The maximum number of gradations is 100.
        - **Gradient direction**: Select the direction for the gradient.
        - **Reverse**: Select to reverse the start and end colors for the gradient.
- **Shadow**: Select to display a drop shadow with the background box.
    - **Color box**: View the current color for the drop shadow.
    - **Color button**: Click Color to select a color for the shadow using the Color Picker.
    - **Width**: Enter the width for the drop shadow. Units are typographical points.
    - **Scale ratio**: Enter the percentage by which you want to scale the text object to create the drop shadow.
    - **X shift, Y shift**: Select the X and Y offsets respectively for the center of the drop shadow. The coordinates (0,0) align the center of the drop shadow with the center of the text object. Positive X values move the shadow to the right of the center of the text object, and negative X values move it to the left of the center of the text object. Positive Y values move the shadow below the center of the text object, and negative Y values move it above the center of the text object.

## Text

This is the primary settings tab for the Text and box object. It contains different sections that help you manipulate the text in the object itself.

**Menu:**

- **Format**
    - **Font...**: Opens the Font selector, which is essentially the same as the Style Toolbar as well as the Style Selector (See Apply a Style to Text in a Text Object)
    - **Margin...**: Opens the Margins dialog which lets you set the **Left** and **Top** margins for the text object.
    - **Word Wrap**: Enables or disables Word Wrap in the text object. Applies to all the contents of the text object.
    - **Skip Empty Variable Paragraphs**: Enables or disables skipping any line where variables do not have any content. Does not apply to empty lines with no variables on them.
    - **Text in box**: This option is only available when Word Wrap is turned off. Click to display a list of possible alignments available for the text in the box. 9 positions are available.

- **Tabs...**: Opens the Tabs dialog. See <u>Set Tabs</u>.
- **Paragraph...**: Opens the Alignment dialog. See <u>Adjust Alignment and Lines Per Unit settings</u>.
- **PlanetPress Talk Before Paragraph...**: Opens the PlanetPress Talk Editor to add PlanetPress Talk code before the currently selected paragraph.
- **PlanetPress Talk After Paragraph...**: Opens the PlanetPress Talk Editor to add PlanetPress Talk code after the currently selected paragraph. See <u>PlanetPressTalk Before and After</u>
- **Edit**
  - **Undo**: Undo the last command. Note that not all actions can be undone.
  - **Redo**: Redo the last command that was undone. Note that not all actions can be redone.
  - **Cut**: Cut the currently selected text. If no text is selected, has no effect.
  - **Copy**: Copies the currently selected text. If no text is selected, has no effect.
  - **Paste**: Pastes any text that is in the Windows clipboard where the cursor is located. If text is selected, the pasted text replaces the selected text.
  - **Select All**: Selects all the contents of the text box.
- **View**
  - **Control Characters**: Click to show or hide control characters such as line returns, tabs and spaces.
  - **Vertical Ruler**: Click to show or hide the vertical ruler on the left of the text box.
  - **Horizontal Ruler**: Click to show or hide the horizontal ruler on top of the text box.
  - **Background Color of Editor...**: Click to show a dialog that lets you select what color is used as a background for the text editor. Note that this does not affect the color or background of your actual textbox, but may improve readability of your text within the editor, for example if your text has a very light color.
- **Variables**
  - **Select Data**: Click to show the data selector. When you choose a data location in the data selector and click OK, one variable per line is created, and they are added in the text box.
  - **Local Variables**: Click to display a list of variables that are local to this text box. Those variables are not accessible by any other object.
  - **Global Variables**: Click to display a list of variables that are global to your form. To change the value of these variables, you have to edit the global variable from the Document Structure Area.
  - **Variables...**: Displays a dialog that shows you all local and global variables.See <u>Use Variables in a Text Object</u>.
- **Tools**
  - **Spell Check Options...**: Opens the Spell Check Options dialog. See <u>Spell Check Text in a Text Object</u>
  - **Spell Check...**: Open the spell checker. See <u>Spell Check Text in a Text Object</u>
  - **Thesaurus...**: Opens the Thesaurus window. See <u>Use the Thesaurus</u>.
  - **Word Wrap Options**:
    - **Normal (word-based)**: Click to select the word-based Word Wrap option. This option will only wrap when a space is present in the text and will not separate characters within words (or character strings, whatever they may be).
    - **Forced (character-based)**: Click to select the character-based Word Wrap option. This option will always wrap even if there is no space in the text. In this case it generates a line return to do word wrap. Note that this is not a "smart" wrap, and it will not add dashes or correctly separate words. Instead is simply wraps when the number of characters is reached on the line.

## Apply a Style to Text in a Text Object

To apply a style to text in a text object:

1. Do any of the following to define the region of text to which you want to apply the style:
   - To apply a style to existing text, highlight that text.
   - To apply the style to text you enter from this point forward, click in the Text area at the point at which you want the style to become active.
2. Choose **Format** | **Font** and use the Font dialog box, to select an existing style and, if necessary, modify its properties.

Note that the modifications you make to the style properties are local to this instance of the style and do not affect those defined for the style.

**Style box**: Select the existing style you want to use for the text. If you type a style name that does not exist, PlanetPress Design updates the selection to the style that most closely matches the name entered, or if no existing style name is a possible match, reverts to the previous selected style.

**Font size**: Enter the point size you want to use for the text.

**Font ratio**: Enter a percentage by which you want to shrink or stretch the font spacing.

**Color list**: Click and select a color for the text from the list of colors that appears.

**Color box**: View the current color for the style.

**Color button**: Click to select a color for the style using the Color Picker.

**Bold**: Click to toggle the bold property on and off.

**Italic**: Click to toggle the italic property on and off.

**Underline**: Click to toggle the underline property for the currently selected style on and off. Note that spaces may not appear underlined in PlanetPress Design, but that they will be when the document is used to generate output.

**Outline:** Click to toggle the outline property for the current selected style on and off.

To create a new style from within a text object:

1.  Use either the toolbar or the dialog box to display the **Style** properties dialog box.
    Toolbar
    In the **Style** toolbar, click the new style tool (  ).
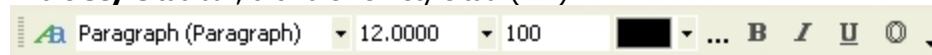
    

    A. New style tool
    Font Dialog
    Choose **Format** | **Font** and click the New Style button.
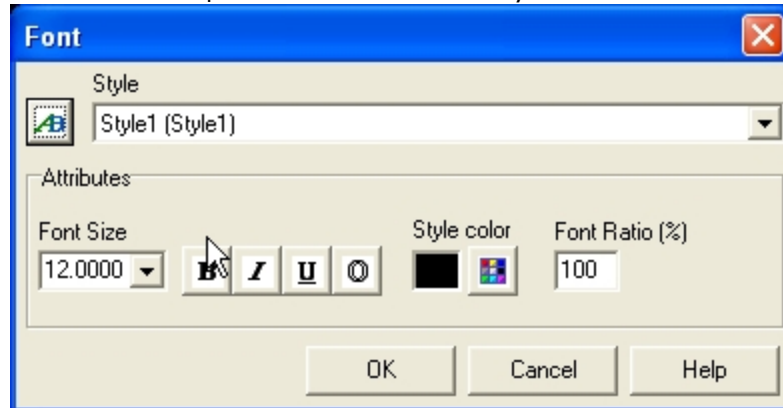
    

    A. New Style button in the Font dialog box
2.  In the Style properties dialog box, define the new style.

## Set Tabs

To set default tabs:

1.  In the Format toolbar, click the tab tool to display the Tabs dialog box.
2.  In the **Tabs** dialog box, set the default tabs using the Default tab stop box.
    **Default tab stop**: Enter the space to leave between each default tab. This is also the initial ruler position for the first default tab. You can also use the spin buttons to adjust the value. Units are as set in the User Options dialog box.
3.  Click **OK.**

To clear the default tabs:

1. Click the tab tool to display the **Tabs** dialog box.
2. In the **Tabs** dialog box, set the value in the **Default tab stop** box to zero.
3. Click **OK**.

To set tabs for individual paragraphs using the tab tool:

1. Do any of the following to define the region of text for which you want to set tabs:
   - To set tabs for a single paragraph, click anywhere in that paragraph.
   - To set tabs for one or more contiguous paragraphs, click and drag to highlight those paragraphs.
   - To set tabs for text you enter from this point forward, click at the new paragraph position.
2. Locate the tab tool to the left of the horizontal ruler.
3. Click the tab tool until it displays the alignment type of the tab you want to set.
   Left alignment tab
   Right alignment tab
   Center alignment tab
   Decimal alignment tab
4. Click in the ruler at the position at which you want to set the tab. If you want to set more than one tab of this type, click in the ruler at the appropriate position for each additional tab.
5. Repeat step 3 through step 4 for any additional tabs you want to set.

To set tabs for individual paragraphs using the Tabs dialog box:

1. Do either of the following to define the region of text for which you want to set tabs:
   - To set tabs for a paragraph, click anywhere in that paragraph. Note that you cannot set tabs for more than one paragraph at a time using the Tabs dialog box.
   - To set tabs for text you enter from this point forward, click at the new paragraph position.
2. In the **Format** toolbar, click the tab tool to display the Tabs dialog box.
3. Set the properties of the tab.
   **Tab stop**: Enter the ruler position for the tab, or use the spin buttons to adjust the value.
   **Alignment**: Select the alignment type for the tab you want to set: Left, Right, Center, or Decimal.
4. Click **Insert**.
   PlanetPress Design sets the new tab. The new tab appears in the Current tabs list, preceded by the tab marker representing its type. PlanetPress Design orders tabs in the Current tabs list in ascending order, within each alignment type.
5. Repeat step 3 through step 4 for each new tab you want to set.
6. Click **Close.**

To reposition a tab:

- In the Text area, right-click on the tab, then drag it to the new position and release.

To delete tabs from individual paragraphs:

1. Click anywhere in the paragraph containing the tab or tabs you want to delete.
2. Do either of the following:
   - In the horizontal ruler, click the tab marker for the tab you want to delete. Repeat for each tab you want to delete.
   - In the **Format** toolbar, click the tab tool to display the **Tabs** dialog box. Click **Delete All** to delete all tabs set in the defined region of text. Click **OK** to close the Tabs dialog box.

## Adjust Alignment and Lines Per Unit settings

To adjust the text justification:

1. Verify that word wrap is on. The justification tools are available only when word wrap is on.
2. Do any of the following to define the region of text to which you want to apply a new text justification setting:
   - To change the justification for a single paragraph, click anywhere in that paragraph.
   - To change the justification for one or more contiguous paragraphs, click and drag to highlight those paragraphs.
   - To change the justification for text you enter from this point forward, click at the new paragraph position.
3. Locate the **Text Justification** toolbar.
4. Click the appropriate justification tool.

   Left justify the text.

   Right justify the text.

   Center justify the text.

   Left and right justify the text.

To adjust the lines per unit:

1. Do any of the following to define the region of text to which you want to apply a new lines per unit (LPU) setting:
   - To change the LPU for a single paragraph, click anywhere in that paragraph.
   - To change the LPU for one or more contiguous paragraphs, click and drag to highlight those paragraphs.
   - To change the LPU for text you enter from this point forward, click at the new paragraph position.
2. Locate the Lines per unit box in the toolbar.
3. Do any of the following:
   - Select an LPU value from the drop-down list.
   - Enter a new integer value in the Lines per unit box.
   - Select **Automatic** to have PlanetPress Design automatically adjust the LPU to accommodate the largest font size in the paragraph. If you subsequently modify the font size of any of the styles the paragraph uses, PlanetPress Design automatically adjusts the LPU to reflect the modification. Automatic is the default value for the LPU setting.

## Spell Check Text in a Text Object

You can spell check a selection of, or all of, the text you enter in the Text area of a text object. The options you set for a spell check remain in effect across all text objects. The spell checker does not check variable

**To spell check text:**

1. If necessary, verify the dictionaries and the other options set for the spell check are correct.
2. Do either of the following:
   - If you want to spell check all text, click anywhere in the text. The spell check proceeds forward from that point to the end of the text, then starts at the beginning of the text and proceeds forward to that point.
   - If you want to spell check only a region of text, highlight that region in the Text area.
3. In the menu bar of the Text properties, choose **Tools | Spelling**.
4. When the Spelling dialog box appears, define how you want to handle the spelling error.
   - **Spelling Error**
     - **Not found**: Displays the misspelled word the spell check encountered.
     - **Replace with**: Enter the word with which you want to replace the misspelled word.
     - **Suggestions**: Displays a list of suggested replacements for the misspelled word.
   - **Actions**
     - **Ignore**: Click to have the spell check ignore this spelling error and continue with the spell check.
     - **Ignore All**: Click to have the spell check ignore all instances of this spelling error in the current spell check.
     - **Change**: Click to have the spell check replace the misspelled word with the contents of the Replace with box.

- **Change All**: Click to have the spell check replace all instances of this misspelled word in the current spell check with the contents of the Replace with box.
- **Add**: Click to have the spell check add the word to the Added Words of the custom dictionary selected for this spell check.
- **Auto-Correct**: Click to have the spell check add this misspelled word and the contents of the Replace with box to the Auto Correct Pairs of the custom dictionary selected for this spell check.
- **Undo**: Click to undo the most recent action.
- **Options**: Click to display the Spelling Options dialog box.
- **Cancel**: Click to exit the spell check.

**To set spell check options:**

1. In the menu bar of the Text properties, choose **Tools** | **Speller Options**.
2. In the **Spelling Options** dialog box, adjust the options.
3. Options
   - **Check spelling as you type**: Select to have the spell check underline misspelled words in red as you type in the Text area.
   - **Correct spelling errors as you type**: Select to have the spell check use the Auto Correct Pairs in the custom dictionary selected for the spell check, to automatically correct any misspelled words it detects as you type in the Text area.
   - **Ignore words in upper case**: Select to prevent the spell check from spell checking any words written in upper case.
   - **Ignore words containing numbers**: Select to prevent the spell check from spell checking any words that contain numbers.
   - **Ignore markup languages**: Select to prevent the spell check from spell checking any words that are part of the HTML or XML markup languages.
   - **Ignore Internet addresses**: Select to prevent the spell check from spell checking email or Web site addresses.
   - **Ignore quoted lines**: Select to have the spell check exclude from the spell check any text that appears between quotes. This applies only to double quotes ("); it does not apply to single quotes (').
   - **Ignore abbreviations**: Select to have the spell check treat any abbreviations it encounters as legal words. The abbreviation must appear in at least one of the dictionaries selected for the spell check.
   - **Prompt on repeated word**: Select to have the spell check prompt you if it encounters two instances of a word together in the text. This option is case sensitive.
   - **Automatically correct DUal capitals**: Select to have the spell check automatically correct words that begin with two upper case characters by changing the second to a lower case character.
   - Dictionary options
   - **Dictionaries list**: Select the dictionaries you want the spell check to use. The name of the dictionary appears on the left, and the name of the file that contains it on the right. Select Locate dictionaries to browse the file system and add a dictionary to this list.
   - **Custom dictionary**: Select the custom dictionary you want to use for the spell check. All of the custom dictionaries you created appear in the drop-down list in this box.
   - **Dictionaries button**: Click to create, edit, or delete a custom dictionary.

**To reset the spell check options to their default values:**

1. In the Text area, right-click and choose **Spell Check Options**.
2. In the **Spelling Options** dialog box, click **Reset Defaults**.
3. Click **OK**.

**To add a dictionary:**

1. Make sure the dictionary is accessible on the computer on which you are running PlanetPress Design. Additional dictionaries are available at **http://www.addictivesoftware.com/dicts.htm**.
2. In the Text area, right-click and choose **Spell Check Options**.
3. In the **Spelling Options** dialog box, in the **Dictionaries list**, click **Locatedictionaries**.
4. In the **Browse for Folder** dialog box, navigate to the dictionary you want to add and click **OK**.

**To create a custom dictionary:**

1. In the Text area, right-click and choose **Spell Check Options**.
2. In the lower right of the **Spelling Options** dialog box, click **Dictionaries**.
3. In the **Dictionaries** dialog box, click **New**.
4. In the **New Custom Dictionary** dialog box, enter a name for the new dictionary, and click **OK**.
5. To exit the Dictionaries dialog box and return to the Spelling Options dialog box, click **OK**.
6. Click **OK**.

**To edit a custom dictionary:**

1. In the **Spelling Options** dialog box, click **Dictionaries**.
2. In the **Dictionaries** dialog box, select the custom dictionary you want to edit and click **Edit**.
3. Click **Added Words** and enter any words you want the spell check to treat as legal words when it uses this custom dictionary.
   - **Ignore this word**: Type the word you want to add to the list of words the spell check treats as legal when it uses this custom dictionary. Click Add to add it to the Word list.
   - **Word list**: Displays the words the spell check treats as legal when it uses this custom dictionary.
   - **Add**: Click to add the word entered in the Ignore this word box, to the Word list.
   - **Delete**: Click to delete the word that is currently selected in the Word list.
4. Click **Auto Correct Pairs** and enter any automatic corrections you want the spell check to perform.
   - **Replace/With**: Use these boxes to define an automatic correction you want the spell check to perform when it uses this custom dictionary. Type the string you want the spell check to automatically replace in the Replace box, and the replacement string in the With box. Click Add to add it to the Auto-correct pairs list.
   - **Auto correct pairs list**: Displays the automatic corrections the spell check performs when it uses this custom dictionary. The misspelled string appears on the left and its correction on the right.
   - **Add**: Click to add the contents of the Replace/With boxes to the Auto-correct pairs list. This button is not available if either or both of the Replace/With boxes are empty.
   - **Delete**: Click to delete the auto-correct pair that is currently selected in the Auto-correct pairs list.
5. Click **Excluded Words** and add any words you want the spell check to always treat as misspellings.
   - **Exclude this word**: Type the word you want to add to the list of words the spell check always treats as misspellings when it uses this custom dictionary. Click Add to add it to the Excluded word list.
   - **Excluded word list**: Displays the words the spell check always treats as misspellings when it uses this custom dictionary.
   - **Add**: Click to add the word entered in the Ignore this word box, to the Excluded word list.
   - **Delete**: Click to delete the word that is currently selected in the Excluded word list.
6. Click **OK** to return to the Dictionaries dialog box.
7. In the **Dictionaries** dialog box, click **OK** to return to the **SpellingOptions** dialog box.

**To delete a custom dictionary:**

1. In the **Spelling Options** dialog box, click **Dictionaries**.
2. In the **Dictionaries** dialog box, select the custom dictionary you want to delete and click **Delete**.
3. Click **OK.**

## Use the Thesaurus

To use the Thesaurus:

1. In the Text area, highlight the word you want to search for in the thesaurus.
2. In the menu bar of the Text properties, choose **Tools** | **Thesaurus**.
   - **Looked up**: Displays the word for which the thesaurus is currently displaying choices. Use the box to view and select words you previously entered in this thesaurus session.
   - **Contexts**: Displays a list of the different contexts in which the word might occur.
   - **Choices list**: Displays a list of possible synonyms for the selected word in the selected context. Click a word in this list to have it appear in the Replace with box.
   - **Replace with**: Displays the word with which you want to replace the word that appears in the Looked up box. You can either enter the word manually or click a word in the Choices list.
3. Do any of the following:
   - To replace the word with one of the words in the Choices list, click the word in the **Choices** list, then click **Replace**.
   - To look up another word, either enter the word in the **Replace with** box and click **Lookup** or, if the word appears in the Choices list, click it and then click **Lookup**.
   - To return to a previous lookup, click **Previous** or, in the **Looked up** box, select the previous lookup.

## Change the Width of the Text Object in the Text Area

To change the width of a text object in the Text area:

1. In the Text area, locate the width marker. The width marker is initially aligned with the right edge of the right indent marker.
2. Position the pointer over the width marker to display the resize pointer, then click and drag the width marker to the new position.

## Use Variables in a Text Object

You can reference both system and global variables from a text object. The global variables you reference within a text object can be of type string, measure, integer, and currency.

If you reference a global variable in a text object and subsequently change the type of the global variable to one not supported by the text object, PlanetPress Design replaces the global variable in the text object with a blank space. The next time you open the text object, an error message appears in the Messages area of the Object Preview and in the status bar of the text object. You cannot exit the text object until you remove all references to the global variable.

You can also create local variables that are internal to the text object. The value of a local variable can be a data selection, a constant, or a PlanetPress Talk expression.

You insert a data selection in a text object by creating a local variable that contains the data selection. Note that you cannot modify the value of those variables through the PlanetPress Talk before paragraph and PlanetPress Talk after paragraph properties.

Also note that if you use a global variable to display Arabic text in a Text object, the data will not be displayed correctly in the application. To fix this, you need to initialize the variable content using the MapUTF8 PlanetPress Talk command.

To determine the variables currently available in the text object, do any of the following:

- **To determine the global variables available** Choose **Variables** | **Global Variables**.
- **To determine the local variables available** Choose **Variables** | **Data Selections Available**.

To insert an existing variable, do one of the following:

- **To insert a local variable** Choose **Variables** | **Data Selections Available.** Then choose the local variable you want to insert from the menu that appears.

- **To insert a global variable** Choose **Variables** | **Global Variables.** Then choose the global variable you want to insert from the menu that appears.
  PlanetPress Design inserts the variable in the Text area, displays its value on the current data page and highlights it in blue.

To quickly insert a data selection as a local variable:

1. Choose **Variables** | **SelectData**.
2. Use the Data Selector to select the data for the variable.
3. Close the Data Selector.
   PlanetPress Design inserts the variable, highlights it in blue, and displays its value on the current data page. PlanetPress Design also adds the variable name to the menu that appears when you right-click and choose **Data Selections Available**.
   You can use the Data page box in the toolbars to navigate through the data pages and view the value of the variable on other data pages.

To create a new local variable using the Variables dialog box:

1. Choose **Variables** | **EditDataSelection**.
2. Click **Add Variable**.
3. Click the variable name to select the variable. Then click the variable name a second time to select the name. Edit the name, then click outside the variable name.
4. Click the variable in the Variables list to select it.
5. Use either of the following to define the value of the local variable. The value of the variable on the current data page appears in the Variables list, beside its name and type.
   Define a Data Selection value
   Clear **Custom data selection** and define the new value by creating a data selection using either the line and column boxes or the Data Selector.
   If you use the Data Selector to select the data for the variable, and you select data that extends over more than a single line, PlanetPress Design uses only the data on the first line (or record) as the data selection for the variable. Note that in a database emulation, the Data Selector does not permit selections that extend over more than a single field.
   **Trim leading spaces:** Select to remove any ASCII space characters that appear before the data in the data selection for the variable.
   **Trim trailing spaces:** Select to remove any ASCII space characters that appear after the data in the data selection for the variable.
   Define a Constant value OR a PlanetPress Talk Expression
   Select **Custom data selection** and enter a constant value or a PlanetPress Talk expression in the **String to display** box.
6. Click **OK**.

To display the name of a variable in the Text area:

- Pause the pointer over the variable.

To edit the definition of a local variable:

1. Choose **Variables** | **EditDataSelection**.
2. In the Variables list, select the variable you want to edit.
3. Right-click the variable name and choose **Rename**. Edit the name, then click outside the variable name.
4. In the Variable value area, edit the value of the local variable.
5. Click **OK**.

To delete a reference to a variable:

- In the Text area, highlight the reference you want to delete and press any key.

To delete a local variable in the Text area:

1. In the Text area, select any reference to the variable you want to delete.
2. Right-click and choose **Delete data selection**.

To delete a local variable using the Variables dialog box:

1. Choose **Variables** | **EditDataSelection**.
2. In the **Variableslist**, select the local variable you want to delete.
3. Press **DELETE**.
4. Click **OK**.

To avoid an empty line in output when a line contains only variables, none of which contain data:

1. In the **Format** toolbar, locate the **Skip Empty Paragraphs** icon. If this toolbar is not visible, right-click in the toolbar area of the Text area and choose Format.
2. Set the **Skip empty paragraphs** option. Click to turn this option on or off. When the option is on, if a line in the Text area contains only variables, and none of those variables contain data, the line does not appear in the output. When this option is off, an empty line appears in the output.

# Data Selection Object

Data Selections are used to retrieve and display information from your data file with simple formatting. It is normally used either for Line Repeat, or as triggers for PlanetPress Image, PlanetPress Fax and PlanetPress Search metadata.

In addition to the common object properties, Data Selections have the following properties:

## Data

- **Data Selection**: This section changes depending on your emulation:
  - For **Line Printer**, **ASCII**, **Channel Skip** and **User-Defined** emulations, see
  - For **Database** and **CSV** emulations, see
  - For **XML** emulations, see
  - For **PDF** emulations, see
- You can also use Custom Data Selections using a PlanetPress Talk command. See Custom Data Selections.
- **Lines per inch:** Set the lines per inch for the data selection object. This makes sense only for data selection objects that contain two or more lines, or two or more records in the case of a database emulation.
- **Alignment:** Align the data selection within the data selection object, as well as the data selection object itself. The alignment of the data selection object is with respect to the Left position setting. For example, if Left=3.5000, and you select a Right alignment, PlanetPress Design aligns the data selection on the right edge of the object **and** aligns the right edge of the object to 3.5000.
- **Trim leading spaces:** Remove any ASCII space characters that appear before the data in the data selection.
- **Trim trailing spaces:** Remove any ASCII space characters that appear after the data in the data selection.

## Lines

- **Text before each line**: Enter any text you want to appear in the document before each line of the data selection. For example, if the data selection is lines 32 through 34, and you enter the string "ITEM:" in this text box, ITEM: appears three times in the document: before the data on line 32, before the data on line 33, and before the data on line 34. You can enter either text or a PlanetPress Talk expression that resolves to a text string in this box.

- **Text after each line**: Enter any text you want to appear in the document after each line of the data selection. For example, if the data selection is lines 20 through 22, and you enter the string "SHIPPED" in this text box, SHIPPED appears three times in the document: after the data on line 20, after the data on line 21, and after the data on line 22. You can enter either text, or a PlanetPress Talk expression that resolves to a text string, in this box.
- **On empty lines**: Use to control how the document treats lines that do not contain data.
  - Select **Do not skip** to have the document display a blank line when it encounters an empty line in the data selection.
  - Select **Skip completely empty line** to have the document ignore empty lines in the data selection. No blank line appears in the document in this case.
  - Select **Skip empty region** to have the document ignore the line when the portion of the line included in the data selection is empty. No blank line appears in the document in this case. In Custom data selections, Skip empty region is equivalent to Skip completely empty line.
- **Line condition**: Set the line condition, if any, that you want to apply to the data selection.
  - Select **No line condition** if you do not want to set a line condition on this data selection.
  - Select **When text is present**, **When text is absent** or **When advanced condition is true** to create conditions for each line. See Create or Remove a Line Condition.

## Archive/Email/Fax

The indexing information in this section is useful for integrating PlanetPress Design in a workflow that uses PlanetPress Image, PlanetPress Fax or PlanetPress Search. For more information on PlanetPress Fax and PlanetPress Image, see the PlanetPress Workflow Tools User Guide. For more information on PlanetPress Search, see the PlanetPress Search User Guide.

- **PlanetPress Image options group**
  - **PDF bookmark**: Select to have PlanetPress Image use this data selection as a bookmark in the PDF file it generates for this document.
  - **Index**: Select to use this data selection as an index term in PlanetPress Search. The name of the index term is the one you specify in the Name box, and its value is the value of the data selection you specify in the Data properties of the data selection object. PlanetPress Image uses this information to generate the .PDI file it creates for each PDF file it creates. The default length for the value of the index term in the PlanetPress Search database is the length of the data selection. The length of the data selection is its length after PlanetPress Design applies the settings of the Trim leading spaces and Trim trailing spaces boxes in the Data properties of the data selection object. If you have two or more data selection objects that provide values for the same index term, PlanetPress Design sets the length to that of the longest of the data selections.
  - **Index Name**: Specify the name you want to use for the PlanetPress Search index term. You can enter a name, select one from the drop-down list, or leave the box empty. If you leave the box empty, PlanetPress Design uses the name of the data selection object as the name of the index term. The drop-down list contains the names of all index terms defined to date in the document. If you select one of these, the data selection you create in this object becomes an additional value for that index term. You can create as many index terms as you require in a document. In the .PDI file the name you specify here appears as the value of an ~IndexName field. Note that the name cannot contain a colon (:) or a closing bracket (]).
  - **This is a Recollect index:** Select to use this data selection as an index term in Recollect from Rebus software. TIFFs and JPGs containing index values that match those provided by a Recollect query are displayed with a red box around the corresponding index value. There will be inconsistencies with how the red binding box containing the index is rendered in Recollect if a Data Selection that is configured as an index for Recollect is rotated or contained in an N-Up object in PlanetPress Design.
  - **Email options group**
    - **E-mail address (To)**: Select to have PlanetPress Image use this data selection as the E-mail address to which to send the PDF file.

- **E-mail address (Cc)**: Select to have PlanetPress Image use this data selection as the E-mail address to which to send the PDF file as a carbon copy (CC).
- **E-mail address (Bcc)**: Select to have PlanetPress Image use this data selection as the E-mail address to which to send the PDF file as a blind carbon copy (BCC).
- **Subject**: Select to have PlanetPress Image use this data selection as the text of the subject line of the E-mail it sends with the PDF file. Use the Trim box to the right of the Subject box to specify how you want to treat any leading or trailing spaces that appear in the data selection. Select Trim to trim both leading and trailing spaces.Select Do not trim to leave any leading or trailing spaces in the data selection. Note that the Trim leading spaces and Trim trailing spaces boxes in the Data properties of the data selection object are independent from the Trim box here and have no effect on the data selection PlanetPress Image receives.
- **Body text**: Select to have PlanetPress Image use this data selection as the body text of the E-mail it sends with the PDF file. Use the Trim box to the right of the Message text box to specify how you want to treat any leading or trailing spaces that appear in the data selection. Select Trim to trim both leading and trailing spaces. Select Trim leading to trim only leading spaces. Select Trim trailing to trim only trailing spaces. Select Do not trim to leave any leading or trailing spaces in the data selection.
- **PlanetPress Fax options group**
  - **Fax number**: Select to have PlanetPress Fax use this data selection as the fax number to which to fax the PDF file.
  - **Information**: Select to have PlanetPress Fax use this data selection as the description of the fax in both the PlanetPress Fax dialog box and the fax log file.

## Text-Based Data Selections

Text-Based data selections are used when your sample data file uses either the Line Printer Emulation, ASCII Emulation, Channel Skip Emulation or User-Defined Emulation.

The following options appear in a the Data section of any data selection object's properties when using either of these emulations:

- **From line**: Enter the line at which to start your data selection.
- **To line**: Enter the line at which to stop your data selection. If you enter 0, the data selection will extend until the end of the data page.
- **From column**: Enter the column for the character at which to start your data selection.
- **To column**: Enter the column for the character at which to stop your data selection. If you enter 0, the data selection will extend until the end of the line(s) of the data page.
- **Use Data Selector**: Click to display the Data Selector to choose which lines and columns to use instead of entering them manually.

You can also create more advanced data selections by creating Custom Data Selections.

**Using the Sample Data Pane to create a data selection**

You can create a text-based data selection by opening the Sample Data pane, selecting the data you wish to use with drag and drop, then drag the selection on the page.

You can use the following keyboard shortcuts within the Sample Data Pane to modify the selection. These shortcuts can also be used when a data selection is selected on the page and the Sample Data Pane is open, in which case modifying the selection changes the data selection object.

- **HOME, END**: Move the currently active cell to the first cell of the current line (**HOME**), or the last cell of the current line (**END**). This shortcut collapses the current data selection to only the currently active cell.

- **PAGE DOWN, PAGE UP**: Move the currently active cell forward (**PAGE DOWN**) or backward (**PAGE UP**) one screen, where the size of a screen is defined as the number of lines currently visible in the Data Pane. Thus you can adjust the size of a screen by resizing the Data Pane. This shortcut collapses the current data selection to only the currently active cell.
- **CTRL+PAGE DOWN, CTRL+PAGE UP**: Move the currently active cell right (**CTRL+PAGE DOWN**) or left (**CTRL+PAGE UP**) one screen, where the size of a screen is defined as the number of columns currently visible in the Data Pane. Thus you can adjust the size of a screen by resizing the Data Pane. This shortcut collapses the current data selection to only the currently active cell.
- **SHIFT+ARROW**: Add cells to, or remove cells from, a data selection.
- **ALT+ARROW or CTRL+SHIFT+ARROW**: Move the current data selection region within the Data Pane.
- **SHIFT+HOME, SHIFT+END**: Increase the size of the data selection to include all cells on the same lines, from the first column of the data page to the first column of the selection (**SHIFT+HOME**), or from the last column of the data selection to the last column of the data page (**SHIFT+END**). The Maximum line width set in the Data Selector determines the number of the last column.
- **ARROW**: Move the currently active cell. This shortcut collapses the current data selection to only the currently active cell.

## Database Data Selections

Database data selections are used when your sample data file uses either the Database Emulation, or the Comma Separated Value (CSV) Emulation.

The following options appear in a the Data section of any data selection object's properties when using either of these emulations:

- **Field name**: A drop-down list that displays all of the fields from your database results. Click on the list and click on the field you want to use.
- **From record**: Enter the number of the record where the data selection should start.
- **To record**: Enter the number of the record where the data selection should end. If you enter 0, the data selection will include all your records.
- **Use Data Selector**: Click to display the Data Selector to choose which field and records to use instead of entering them manually.

You can also select multiple fields by creating a Custom Data Selections.

To create a data selection in database emulation using the Data Selector:

1. In the **DataSelector**, use any of the following to create the data selection.
   FIELD NAME AND Range Coordinates
   In the **Field name** box, select the field you want to use for the data selection, and use the **From record** and **To record** boxes to specify the range of records you want to include in the data selection.
   Mouse
   Click in the field you want to use for the data selection, in the first of the records you want to include in the data selection, then drag the mouse to select the remaining records to include.
   Keyboard Shortcuts
   Click in the Data Pane and use any of the following keyboard shortcuts to navigate in the Data Pane.
   **HOME, END**: Move the currently active cell to the first field of the current record (HOME), or the last field of the current record (END). This shortcut collapses the current data selection to only the currently active cell.
   **PAGE DOWN, PAGE UP**: Move the currently active cell forward (**PAGE DOWN**) or backward (**PAGE UP**) one screen, where the size of a screen is defined as the number of lines currently visible in the Data Pane.
   **CTRL+PAGE DOWN, CTRL+PAGE UP**: See Use the Data Selector to Create a Data Selection .
   **ALT+ARROW or CTRL+SHIFT+ARROW**: See Use the Data Selector to Create a Data Selection .

**SHIFT+HOME, SHIFT+END**: See Use the Data Selector to Create a Data Selection .
**CTRL+SHIFT+ARROW**: Move the current data selection.
**ARROW**: Move the currently active cell. This shortcut collapses the current data selection to only the currently active cell.

2. Click **OK**.

## XML Data Selections

XML data selections are used when your sample data file uses the XML Emulation.

The following options appear in a the Data section of any data selection object's properties when using this emulation:

- **XPath**: Specify the path in the structure for the data selection you want to obtain.
- **From iteration and To iteration fields**: Use these fields to specify the range of the data selection. The To iteration field allows a value of 0. When To iteration is not zero, it must be higher than the value of From iteration and is the last iteration index. If it is zero, the repetition occurs from the From iteration to the maximum (or entire) number of elements.
- **Data Type:** Specify how you want to select the information: by Count, Name, or Value as follows:

  Data Type Description

  Count      Returns an integer indicating the number of nodes for the specified XPath.

  Name       Returns a string that represents the name of the current node or the first node in the specified XPath.

  Value      Returns a string that represents the content of the node in the specified XPath.
- **Element on which to iterate field**: Select the element on which to iterate. Using this in conjunction with the From iteration and To iteration fields narrows the scope of the data selection set.
- **Use Data Selector**: Click to display the Data Selector to choose which value or property to use instead of entering them manually.

To create a data selection in XML emulation using the Data Selector:

1. In the **DataSelector**, use the mouse to create the data selection.
   FIELD NAME AND Range Coordinates
   In the **Field name** box, select the field you want to use for the data selection, and use the **From record** and **To record** boxes to specify the range of records you want to include in the data selection.
   Mouse
   Click in the field you want to use for the data selection, in the first of the records you want to include in the data selection, then drag the mouse to select the remaining records to include.
   Keyboard Shortcuts
   Click in the Data Pane and use any of the following keyboard shortcuts to navigate in the Data Pane.
   **HOME, END**: Move the currently active cell to the first field of the current record (HOME), or the last field of the current record (END). This shortcut collapses the current data selection to only the currently active cell.
   **PAGE DOWN, PAGE UP**: See Use the Data Selector to Create a Data Selection .
   **CTRL+PAGE DOWN, CTRL+PAGE UP**: See Use the Data Selector to Create a Data Selection .
   **SHIFT+ARROW**: Use the Data Selector to Create a Data Selection .
   **ALT+ARROW**: Use the Data Selector to Create a Data Selection .
   **SHIFT+HOME, SHIFT+END**: Use the Data Selector to Create a Data Selection .
   **CTRL+SHIFT+ARROW**: Use the Data Selector to Create a Data Selection .
   **ARROW**: Use the Data Selector to Create a Data Selection .
2. Click **OK**.

**About PlanetPress Design XPath**

In supporting XML programming functionality, PlanetPress Design data selection makes use of a proprietary version of XPath. The proprietary version of XPath is derived from the W3C standard XPath. In the context of PlanetPress Design, XPath selects data from the XML file that is associated with a PlanetPress Design document and is used in the data emulation.

In XPath, there are various types of nodes: element, attribute, text, namespace, processing-instruction, comment, and document (root) nodes. XML documents are treated as trees of nodes. The root of the tree is called the document node (or root node).

The characteristics of PlanetPress Talk XPath are:

- Uses a 1-based system. This means XPath accesses the first element in the XML structure using the predicate [1]. W3C XPath also uses base 1.
- Returns a single value only. Each time PlanetPress Talk XPath encounters an element that is not indexed in the XML structure, an index of [1] is returned instead, thus returning the value of the first element.
- Returns the content of the specified tag—meaning the W3C XPath command /text() is inherent in PlanetPress Design XPath.
- Does not read values or attributes of the top level node if the Second Element emulation option is enabled in the Data Selector dialog box.

PlanetPress Design XPath uses the following subset of W3C XPath:

| Syntax | Description |
|---|---|
| /agency/crew | Navigates the data file content in a standard way |
| | PlanetPress Design XPath supports simple indexing embedded in square brackets. |
| /agency/crew[1] | Result |
| | Selects the first crew element that is the child of the agency element. |
| * | Matches any element node |
| /name() | Obtains the name of the element's tag |
| @ | Obtains an attribute |

Any remaining XPath expressions and syntax used by W3C XPath are not implemented by PlanetPress Talk XPath.

## PDF Data Selections

PDF data selections are used when your sample data file uses the PDF Emulation.

The following options appear in a the Data section of any data selection object's properties when using this emulation:

- **Left**: Enter the position (in inches) of the left boundary of your data selection.
- **Top**: Enter the position (in inches) of the upper boundary of your data selection.
- **Right**: Enter the position (in inches) of the right boundary of your data selection.
- **Bottom**: Enter the position (in inches) of the lower boundary of your data selection.
- **Use Data Selector**: Click to display the Data Selector to choose which area to use instead of entering the coordinates manually.

**Using the Sample Data Pane to create a data selection**

You can create a PDF data selection by opening the Sample Data pane, selecting the data you wish to use with drag and drop, then drag the selection on the page. The text contents of the selection appears on the page.

## Edit Text-Based Data Selection Size

To edit the size of a data selection using the mouse:

1. In the Page area, select the data selection object whose data selection you want to edit.
   A red rectangle appears around the selected object. It contains eight resize handles, one on each corner, and one in the middle of each edge of the rectangle.
2. Position the pointer over the appropriate resize handle to display the double-headed arrow pointer, then click and drag to adjust the size of the object. Release when the object contains the data selection you require. Note that if you defined the data selection in the object using PlanetPress Talk expressions, you cannot resize the object along any of the edges defined by those expressions.

To edit the size of a data selection or reposition it on the data page using keyboard shortcuts:

1. In the Page area, select the data selection object whose data selection you want to edit.
2. Use the following keyboard shortcuts to resize or reposition the data selection.
   PlanetPress Design updates the data selection in the selected object with each press of an **ARROW** key. More precisely, it adjusts the From line, To line, From column, and To column properties of the data selection in the selected object to reflect the changes. The Data Pane also updates to reflect the changes to the data selection.
   Resize
   **SHIFT**+**DOWN ARROW**: Increase the size of the data selection by one line along its bottom edge.
   **SHIFT**+**UP ARROW** Decrease the size of the data selection by one line along its bottom edge.
   **SHIFT**+**RIGHT ARROW**: Increase the size of the data selection by one column along its right edge.
   **SHIFT**+**LEFT ARROW**: Decrease the size of the data selection by one column along its right edge.
   Reposition
   **ALT**+**UP ARROW**: Move the selection up one line (or record in the case of a database emulation).
   **ALT**+**DOWN ARROW**: Move the selection down one line.
   **ALT**+**LEFT ARROW**: Move the selection to the left one column.
   **ALT**+**RIGHT ARROW**: Move the selection to the right one column.

# Postal Address Object

The **Postal Address Object** is used to display an address on your page and add specific address-related functionalities to your document. You can of course display an address with other objects such as a **Text Object** or a **Data Selection Object**, but the **Postal Address Object** includes features not available in other objects.

## Metadata

The Postal Address Object was originally created in order to facilitate communication with postal sorting software, which uses the metadata created by PlanetPress in order to sort the data and print in the correct order. However, this metadata can also have other uses to you, and the exact level at which the metadata level is added (and what levels are created) depends on two options: **Use Data for Postal Sorting**, and the **Group ID** input box. Please review the table below for more precise details:

| Group ID Present | Postal Sorting Checked | Metadata in level | Group ID In Metadata | Metadata divided on |
|---|---|---|---|---|
| No | No | Page | No | Datapage |
| Yes | No | Page | Yes | Datapage |
| No | Yes | Document | No | Document |
| Yes | Yes | Document | Yes | Group & Document |

In addition to the common object properties, postal address objects have the following properties in the Address data tab:

**Data Tab**

- **Custom data selections**:
  - **1 Through 8**: The 8 data selection boxes are where you put your address. Just like any other PlanetPress Talk-enabled input box, you can enter any PlanetPress Talk expression in these inputs. You can also concatenate them, for example **@(1, 1, 50) + ', ' + @(2, 1, 2) + ' ' + @(3, 1, 5)** for **City, State  ZIP.**
- **Group ID**: Enter a PlanetPress Talk expression that defines the Group ID for your data file. See Metadata (above) for more information on the behavior of this option.
- **Lines per Inch:** Determines the vertical spacing between the line by changing the lines per inch. A lower value will place more spacing (less lines per inch, thus lines take more space) and vice versa. Note that font size will also affect the spacing.
- **From bottom to top**: If you wish your address to be bottom-aligned, select this option, which will push all the lines to the bottom of the object.
- **Skip Empty Lines**: Check if you wish option is checked, empty lines in the data selections will be removed from the result on the page. If used in combination with the *From bottom to top* option, the empty lines will cause the rest of the information to move *down*, instead of *up*.
- **Use Data for Postal Sorting**: Generates additional hidden fields that are for use with Postal Sorting/Address Cleansing tasks in PlanetPress Workflow Tools. This also changes where the metadata is located (see table above).

**Advanced Options Tab**

- **Address Lines**: The display names for each of the 8 address lines and the Group ID field.
- **Metadata Field Name**: The names of the metadata fields that will be created whenever metadata is regenerated either from PlanetPress Design or PlanetPress Workflow Tools.
- **Restore Defaults**: Restore the default metadata field names.
- **Generate Names**: Generate unique and random metadata field names.

# PlanetPress Talk Object

The PlanetPress Talk object is used to add PlanetPress Talk code to your document.

In addition to the common object properties, PlanetPress Talk objects have the following properties:

PlanetPress Talk code:

- **Code area**: Enter your PlanetPress Talk code here.
- **Refresh Preview**: If the Object Preview Window is open, refreshes the preview that is generated by the PlanetPress Talk code. For performance reasons, this does not occur automatically.
- **Use PressTalk Editor**: Opens the "The PlanetPress Talk Editor" (page 199) window.

For more information on PlanetPress Talk, see the chapter on PlanetPress Talk.

# Shape Object

Shape objects are used to add a variety of lines, arrows, or geometric shapes to your document. Add shapes one at a time. You can also group one or more shapes that you add to the document and manipulate the group as a single shape.

In addition to the common object properties, shape objects have the following properties:

**Type Tab**

- **Shape type**: Select the type of style of the shape you want from the list of Box, Line, Circle, Ellipse, Arrow or Triangle.
- **Style**: Select the style for the lines in the shape.
- **Width**: Enter the width, in units of points, for lines in the shape.
- **Color box**: View the current color for the lines in the shape.
- **Color button**: Click to select a color for the lines in the shape using the Color Picker.
- Arrow (available when the *Shape type* is *Arrow*):
    - **Start Style**: Select a style for the arrowhead at the start of the arrow.
    - **End Style**: Select a style for the arrowhead at the end of the arrow.

**Color Tab**

- **Fill**: Click to fill the object with a specific color or gradient. This enables the properties in the Fill group
    - **Color box**: View the current fill color for the background box.
    - **Color button**: Click to select the fill color for the background box using the Color Picker.
    - Gradient fill
    - **Gradient fill**: Select to use a gradient fill for the background box. The start color for the gradient is the fill color. Use the controls in this area to set the end color for the gradient, the number of gradations the gradient uses, and the direction of the gradient. Gradients are not available if the background box you select for the text/box object has one or more rounded corners.
        - **Color box**: View the end color for the gradient.
        - **Color button**: Click to select the end color for the gradient using the Color Picker.
        - **Steps**: Enter the number of gradations you want to use in the gradient, or use the spin buttons to set the value. The number of gradations determines how obvious the transition is between the Fill color and the Gradient fill color. The maximum number of gradations is 100.
        - **Gradient direction**: Select the direction for the gradient.
        - **Reverse**: Select to reverse the start and end colors for the gradient.
- **Shadow**: Select to display a drop shadow with the background box.
    - **Color box**: View the current color for the drop shadow.
    - **Color button**: Click Color to select a color for the shadow using the Color Picker.
    - **Width**: Enter the width for the drop shadow. Units are typographical points.
    - **Scale ratio**: Enter the percentage by which you want to scale the text object to create the drop shadow.
    - **X shift, Y shift**: Select the X and Y offsets respectively for the center of the drop shadow. The coordinates (0,0) align the center of the drop shadow with the center of the text object. Positive X values move the shadow to the right of the center of the text object, and negative X values move it to the left of the center of the text object. Positive Y values move the shadow below the center of the text object, and negative Y values move it above the center of the text object.

# N-Up Object

The N-Up Object is used to execute a virtual page N-Up. N-Up refers to executing a page such that two or more instances of it print on one side of a sheet of paper. The page you execute N-Up must be of type virtual, however it overlays can be asso-

ciated with the virtual page. When PlanetPress Design scales the virtual page in the N-Up object, it simultaneously scales any overlays associated with that page.

>>> **Data Pages and N-Up**: It is important to note that because the N-Up object can change the data page within a single page, any other object after the N-Up object that uses data selection will always point to the last data page.

In addition to the common object properties, N-Up objects have the following properties:

- **Horizontal**
  - **Number of repeats**: Enter the number of times you want to repeat the page horizontally.
  - **Space between each repeat (Horizontally)**: Enter the amount of space you want to leave between each horizontal repeat. Units are as set in the User Options dialog box.
- **Vertical**
  - **Number of repeats**: Enter the number of times you want to repeat the page vertically.
  - **Space between each repeat**: Enter the amount of space you want to leave between each vertical repeat. Units are as set in the User Options dialog box.
- **Page options**
  - **Scale ratio**: Enter the percentage by which you want to scale each instance of the page.
  - **Page to execute n-up**: Select the virtual page you want to execute n-up.
  - **Alignment**: Select the order in which the n-up object repeats the page. You use this option when you are printing in duplex mode, have an n-up object on both of the two pages that print duplex, and need to control how the contents of the two pages align. The image and text on the front of the invitation should have the same orientation as the text on the back of the invitation, so that when the recipient turns it over, they do not have to rotate it to read the text. You use the Repeat order option in each n-up object to ensure the proper alignment of the front and back of the postcard. Each of the options here define both a starting point and direction for the repeat.
- **Data options**
  - **Change data page with each repeat**: Select to change the data page at each repeat. Clear to use the same data page for all repeats. If you select this option, be sure you understand the implications of the data page changes for any other objects in the document that use data selections. This option is not available in documents that use a user-defined emulation.
  - **Skip empty data page**: Select to skip any empty data pages. Clear to execute a repeat with a data page even when that data page is empty.
  - **Skip repeat when page condition is false**: Use this option to control whether the n-up object leaves an empty repeat when a page condition set on the virtual page resolves to False. Select to prevent the n-up object from leaving an empty repeat of the virtual page if the page condition on the virtual page resolves to False. Clear to have the n-up object leave an empty repeat of the virtual page when the condition resolves to False.
  - **Advance data page before executing**: Select to advance to the next data page before executing this n-up object. Clear to start executing this n-up object with the current data page. This option determines whether a data page change occurs after the last repeat in one n-up object, before the first repeat in the next n-up object.

# Picture Object

The picture object is basically a placeholder that can display a picture or PDF located in any of the following locations:

- As a picture resource within the PlanetPress Design Document
- As a local picture file on the hard drive of the computer or server running PlanetPress Suite
- As a picture file located in a virtual printer drive
- As a picture file located on a printer's memory (flash, hard drive or RAM)

In all cases, the image can either be static (always the same) or dynamic (depending on certain conditions or the data). Only certain image formats are supported by PlanetPress Suite. These image formats are listed in Supported Image Formats.

For more information on image formats, resolution, quality and resources, please see the chapter on Adding Resources.

In addition to the common object properties, image objects have the following properties:

**Settings**

- **Image**: Specify the image resource to associate with the picture object, using one of the following methods:
  - Click the **drop-down** to display a list of picture resources of your PlanetPress Design document. Click on the image to use it.
  - Enter an **image file name** (static or dynamic using PlanetPress Talk). PlanetPress Design will first look in its resources for the image name. If it cannot find it, it will look in the Virtual Drive (if located on a computer) or the printer's hard drive (if printed in Printer Centric or Optimized PostScript run modes).
  - Enter the **full path** of an image on the hard drive. PlanetPress will use the file on your computer. This method will not work in Printer Centric run mode.
  - Click the **Browse** button to select an image file using the Open dialog box. Once you select an image file, PlanetPress Design makes a copy of that image file and adds it to the Image resources of the document. If the image resource is a bitmapped image, it sets the image quality and scanline orientation of the resource to reflect the values set in the Document properties dialog box.
- **Page**: If the image resource is a multi-page PDF file, enter the page number of the file that you want to use as the image for this picture object. You can also use a PlanetPress Talk expression in this box to specify the page number. When PlanetPress Design adds an image resource that is in PDF format, it adds the complete file; a picture object can reference only a single page of that image resource.
- **Transparent**: If the image is a monochrome bitmapped image, select to make the background color of that image transparent. This makes the white pixels of the image transparent.
  - **Duotone**: Select to create a duotone by replacing the foreground color in the bitmapped image with a color. Click **Color** to select the color using the Color Picker. The Color box displays the current duotone color. Note that any changes to the foreground color are internal to the picture object and do not have any effect on the image resource itself.
  - **Color**: Click to select the foreground color for the duotone using the Color Picker. The color you select replaces the current foreground color in the bitmapped image.
  - **Color box**: View the current duotone color. This is the color that replaces the foreground color in the bitmapped image.
- **Clipped region**: Select to specify whether the image resource must be clipped, defining a clipped region using **Left**, **Top**, **Width** and **Height** coordinates.
  - **Invert clipped region**: Select to invert the specified clipped region, thus using the reverse of the designated region.
- **Fit setting**: Select how you want the document to size the image relative to the size of the picture object.
  - Select **Constant resolution** to display the image at the resolution specified in the relevant resolution box in the Document dialog box or at the default size. This is the least time-consuming of all the resolution options since it does not require any scaling of the image at runtime.
  - Select **Constant height** to scale the image such that its height is equal to the height of the picture object. The scaling preserves the aspect ratio of the image, and thus in some cases may result in the image extending beyond the right edge of the picture object.
  - Select **Constant width** to scale the image such that its width is equal to the width of the picture object. The scaling preserves the aspect ratio of the image, and thus in some cases may result in the image extending beyond the bottom edge of the picture object.

- Select **Best fit** to scale the image to create the best fit with the width and height of the picture object. The scaling preserves the aspect ratio of the image. This is the most time-consuming of all the Fit setting options since it requires calculating the image size that yields the best fit with the picture object. Best fit is the default Fit setting.
- **On error**: Select how you want the document to handle any image it cannot locate.
    - Select **Print square** to have the document display a square with an X in it in the image area.
    - Select **Blank** to have the document leave the image area blank.
    - Select **Stop job** to have the document abort execution and output a PostScript error.

## Barcode Object

With PlanetPress Design, you can add a variety of barcodes to your documents simply by using drag and drop, including 1d and 2d barcodes. To view a full list of barcodes supported by PlanetPress Design, see Supported Barcodes. Note however that while we support these barcode types, your own barcode reader may require specific parameters that are different than the standards provided to us and require tweaking or modifications. For detailed descriptions of each one of the options offered by specific barcodes, or for background information on barcodes, refer the documentation provided by individual barcode suppliers.

>>> None of the barcodes used in PlanetPress Design supports Double Byte or Arabic characters.

In addition to the common object properties, barcode objects have the following properties:

**Barcode Options Tab**

- **Barcode Type**: Select the type of barcode you wish to use.
- **Options group** (some options may be unavailable for certain barcode types, according to their own specifications):
- **Add human readable**: Check to have human-readable (alphanumerical) characters appear underneath the barcode for verification of its value.
- **Automatically append checksum**: Check to have the checksum automatically calculated and inserted in the barcode value.
- **Add start/stop character automatically**: Check to have start/stop characters added to the barcode value.
- **Bar width (in mils.)**: Changes the bars of the barcode. This is necessary only if the dispersion of ink on your printer's paper causes the bars to be too slim or too wide.
- **Barcode settings**: One or two groups may appear underneath the other options in order to offer more control over the barcode. These options are taken from the barcode's specifications and are explained either in Supported Barcodes or in the barcode reader's documentation.

**Data Tab**

- The options here are identical to those of the Data Selection Object.

# Supported Barcodes

This section provides basic information on the barcodes supported by PlanetPress Design, such as which characters are compatible with which barcodes, how many characters can be used in a given barcode, etc.

## Australia Post

This barcode must begin with one of the following two digit Format Control Code (FCC) number:
11, 59, or 62.

The FCC number must be followed by eight (8) digits. These are mandatory to specify the destination. Additional characters may be used for the Customer Information Field. The number characters varies according to each FCC format:

- Format 11: No extra characters.
- Format 59: Up to 8 digits or 5 letters.
- Format 62: Up to 15 digits or 10 letters.

Compatible characters for the Customer Information Field include:

- Upper and lower case letters: a to z and A to Z.
- Digits: 0 to 9.
- Symbols: space and # (pound).

A checksum is automatically calculated.

## Aztec

This barcode can be used to generate Normal, Compact or Full Aztec barcodes. This barcode supports the full ASCII character set and its maximum data capacity depends on the selected barcode type:

- Normal barcode mode: Up to 3800 digits, or up to 3000 alphanumeric characters.
- Compact barcode mode: Up to 58 digits, or up to 47 alphanumeric characters.
- Full barcode mode: Up to 310 digits, or up to 250 alphanumeric characters.

Bar width is set using millimeters and corresponds to the minimum barcode width.

Error correction levels correspond to the percentage the symbol that includes error checking data. Note that level 0 provides optimal error correction. Each barcode mode has its own value range (Normal mode values range from 0 to 99, Compact from 0 to 4; and Full from 0 to 32).

## Codabar

This barcode must start with a Start character and end with a Stop character. Characters A, B, C and D can be used as Start and Stop characters.

For the remaining barcode data, the following characters may be used:

- Digits: 0 to 9.
- Symbols: - (hyphen), $(dollar),: (colon), / (slash),. (period), + (plus).

A checksum is automatically calculated.

## CodablockF

This 2D barcode can currently only be used in documents that will be printed using either the Optimized Postscript or Windows Printing mode (printer centric mode will be added in a future release).

This barcode supports the full ASCII character set based on the 3 alphabets used in the Code128 barcode (see below). The switch between alphabets is done automatically. Special characters in the A alphabet are called with their ASCII value using the following syntax: ^nnn (such as ^010 for LineFeed).

A minimum of 3 characters is required in this barcode. The maximum string length for this barcode varies depending on the control characters used in the code. Bar width is set using millimeters and corresponds to the minimum barcode width.

## Code 11

The following characters are valid for this barcode:

- Digits: 0 to 9.
- Symbol: - (hyphen).

A checksum is optional, but should be added to ensure data reliability. When the Checksum option is selected, the required checksum digits are added automatically.

- For data with fewer than 10 characters, a single checksum digit is used.
- For data with 10 characters or more, two checksum digits are used.

## Code 16k

This 2D barcode can currently only be used in documents that will be printed using either the Optimized Postscript or Windows Printing mode (printer centric mode will be added in a future release).

This barcode supports the full ASCII character set based on the 3 alphabets used in the Code128 barcode (see below). A maximum of 5 characters per row over 16 rows is permitted. If the first 4 characters in the data are numeric, a 'SwitchC' character is automatically added before the data. Special characters within the data can be used to switch alphabets in the same way as with the Code 128 barcode.

The selected barcode mode determines the starting codeset and leading character:

- Automatic: The codeset and leading character are selected automatically.
- Mode 0: Selects codeset A.
- Mode 1: Selects codeset B.
- Mode 2: Selects codeset C.
- Mode 3: Selects codeset B and sets the leading character to'Fnc1'.
- Mode 4: Selects codeset C and sets the leading character to'Fnc1'.
- Mode 5: Selects codeset C and sets the leading character to'Shift B'.
- Mode 6: Selects codeset C and sets the leading characters to two successive'Shift B' characters.

Bar width is set using millimeters and corresponds to the minimum barcode width.

## Code 39

The following characters are valid for this barcode:

- Digits: 0 to 9.
- Upper case letters: A to Z.
- Symbols: space, - (hyphen), $(dollar), / (slash), + (plus),% (percent), * (star),. (period).

To go beyond the 44 basic characters listed above, you may use extended ASCII characters (granted that scanners are programmed for this, otherwise extended characters will simply be read as basic characters).

To use extended characters, use the encoding listed in the following table:

| Character | Encode as | Character | Encode as | Character | Encode as | Character | Encode as |
|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| NUL | %U | SP | Space | @ | %V | ` | %W |
| SOH | $A | ! | /A | A | A | a | +A |
| STX | $B | " | /B | B | B | b | +B |

| ETX | $C | # | /C | C | C | c | +C |
|-----|-----|-----|-----|-----|-----|-----|-----|
| EOT | $D | $ | /D | D | D | d | +D |
| ENQ | $E | % | /E | E | E | e | +E |
| ACK | $F | & | /F | F | F | f | +F |
| BEL | $G | ' | /G | G | G | g | +G |
| BS | $H | ( | /H | H | H | H | +H |
| HT | $I | ) | /I | I | I | i | +I |
| LF | $J | * | /J | J | J | j | +J |
| VT | $K | + | /K | K | K | k | +K |
| FF | $L | , | /L | L | L | l | +L |
| CR | $M | - | - | M | M | m | +M |
| SO | $N | . | . | N | N | n | +N |
| SI | $O | / | /O | O | O | o | +O |
| DLE | $P | 0 | 0 | P | P | p | +P |
| DC1 | $Q | 1 | 1 | Q | Q | q | +Q |
| DC2 | $R | 2 | 2 | R | R | r | +R |
| DC3 | $S | 3 | 3 | S | S | s | +S |
| DC4 | $T | 4 | 4 | T | T | t | +T |
| NAK | $U | 5 | 5 | U | U | u | +U |
| SYN | $V | 6 | 6 | V | V | v | +V |
| ETB | $W | 7 | 7 | W | W | w | +W |
| CAN | $X | 8 | 8 | X | X | x | +X |
| EM | $Y | 9 | 9 | Y | Y | y | +Y |
| SUB | $Z | : | /Z | Z | Z | z | +Z |
| ESC | %A | ; | %F | [ | %K | { | %P |
| FS | %B | << | %G | \ | %L | | | %Q |
| GS | %C | = | %H | ] | %M | } | %R |
| RS | %D | > | %I | ^ | %N | ~ | %S |
| YS | %E | ? | %J | _ | %O | DEL | %T,%X,%Y,%Z |

A checksum is optional, but should be added to ensure data reliability.

## Code 49

This 2D barcode can currently only be used in documents that will be printed using either the Optimized Postscript or Windows Printing mode (printer centric mode will be added in a future release).

The data for this barcode can include both numeric and alphanumeric characters. When only numbers are present in the data, the maximum number of characters is 82. When alphanumeric characters are present, the maximum is 50 characters.

Bar width is set using millimeters and corresponds to the minimum barcode width.

## Code 93

The following characters are valid for this barcode:

- Digits: 0 to 9.
- Upper case letters: A to Z.
- Symbols: space, - (hyphen), $(dollar), / (slash), + (plus),% (percent), * (star),. (period).

To go beyond the 44 basic characters listed above, you may use extended ASCII characters (granted that scanners are programmed for this, otherwise extended characters will simply be read as basic characters).

To use extended characters, use the encodings listed in the following table:

| Character | Encode as | Character | Encode as | Character | Encode as | Character | Encode as |
|---|---|---|---|---|---|---|---|
| NUL | (%)U | SP | Space | @ | (%)V | ` | (%)W |
| SOH | ($)A | ! | (/)A | A | A | a | (+)A |
| STX | ($)B | " | (/)B | B | B | b | (+)B |
| ETX | ($)C | # | (/)C | C | C | c | (+)C |
| EOT | ($)D | $ | (/)D | D | D | d | (+)D |
| ENQ | ($)E | % | (/)E | E | E | e | (+)E |
| ACK | ($)F | & | (/)F | F | F | f | (+)F |
| BEL | ($)G | ' | (/)G | G | G | g | (+)G |
| BS | ($)H | ( | (/)H | H | H | H | (+)H |
| HT | ($)I | ) | (/)I | I | I | i | (+)I |
| LF | ($)J | * | (/)J | J | J | j | (+)J |
| VT | ($)K | + | (/)K | K | K | k | (+)K |
| FF | ($)L | , | (/)L | L | L | l | (+)L |
| CR | ($)M | - | - | M | M | m | (+)M |
| SO | ($)N | . | . | N | N | n | (+)N |
| SI | ($)O | / | (/)O | O | O | o | (+)O |
| DLE | ($)P | 0 | 0 | P | P | p | (+)P |
| DC1 | ($)Q | 1 | 1 | Q | Q | q | (+)Q |
| DC2 | ($)R | 2 | 2 | R | R | r | (+)R |
| DC3 | ($)S | 3 | 3 | S | S | s | (+)S |
| DC4 | ($)T | 4 | 4 | T | T | t | (+)T |
| NAK | ($)U | 5 | 5 | U | U | u | (+)U |
| SYN | ($)V | 6 | 6 | V | V | v | (+)V |
| ETB | ($)W | 7 | 7 | W | W | w | (+)W |
| CAN | ($)X | 8 | 8 | X | X | x | (+)X |
| EM | ($)Y | 9 | 9 | Y | Y | y | (+)Y |
| SUB | ($)Z | : | (/)Z | Z | Z | z | (+)Z |
| ESC | (%)A | ; | (%)F | [ | (%)K | { | (%)P |
| FS | (%)B | << | (%)G | \ | (%)L | \| | (%)Q |
| GS | (%)C | = | (%)H | ] | (%)M | } | (%)R |
| RS | (%)D | > | (%)I | ^ | (%)N | ~ | (%)S |
| YS | (%)E | ? | (%)J | _ | (%)O | DEL | (%)T, (%)X, (%)Y, (%)Z |

A checksum is mandatory for this barcode (it is always added automatically).

# Code 128

Code 128 uses 3 alphabets, each containing 106 characters:

- Alphabet A contains no lower case characters but includes special characters, such as NUL, ACK and FF.
- Alphabet B contains upper and lower case characters.
- Alphabet C is used almost exclusively for double density numeric values, ranging from 00 to 99.

In the case of alphabet A, to enter the non-printable characters listed as ordinals 64 to 94 in the following table, you must use alphabet B equivalents (for the ACK character, for example, use character f).

The non-printable characters listed as ordinals 95 to 105 can be entered using their ordinal value preceded by ^. To enter FNC1, for example, you would enter ^102. To enter an actual ^ character, you would enter ^062.

Numeric values entered using alphabet C are always considered as digit pairs (0001, for example, for values 0 and 1). Only when you use alphabets A or B can you enter an odd number of digits (001, for example, for values 0, 0 and 1).

Having {} taken as a pair of characters to be converted into alphabet C will fail the job and result in a typecheck error.

The following table lists the characters that can be used in all three alphabets:

| Ordinal | Value in alphabet | | | Encoding | Ordinal | Value in alphabet | | | Encoding |
|---------|-----|-----|-----|----------|---------|-----|-----|-----|----------|
| | A | B | C | | | A | B | C | |
| 00 | SP | SP | 00 | 11011001100 | 53 | U | U | 53 | 11011101110 |
| 01 | ! | ! | 01 | 11001101100 | 54 | V | V | 54 | 11101011000 |
| 02 | " | " | 02 | 11001100110 | 55 | W | W | 55 | 11101000110 |
| 03 | # | # | 03 | 10010011000 | 56 | X | X | 56 | 11100010110 |
| 04 | $ | $ | 04 | 10010001100 | 57 | Y | Y | 57 | 11101101000 |
| 05 | % | % | 05 | 10001001100 | 58 | Z | Z | 58 | 11101100010 |
| 06 | & | & | 06 | 10011001000 | 59 | [ | [ | 59 | 11100011010 |
| 07 | ' | ' | 07 | 10011000100 | 60 | \ | \ | 60 | 11101111010 |
| 08 | ( | ( | 08 | 10001100100 | 61 | ] | ] | 61 | 11001000010 |
| 09 | ) | ) | 09 | 11001001000 | 62 | ^ | ^ | 62 | 11110001010 |
| 10 | * | * | 10 | 11001000100 | 63 | _ | _ | 63 | 10100110000 |
| 11 | + | + | 11 | 11000100100 | 64 | NUL | ` | 64 | 10100001100 |
| 12 | , | , | 12 | 10110011100 | 65 | SOH | a | 65 | 10010110000 |
| 13 | - | - | 13 | 10011011100 | 66 | STX | b | 66 | 10010000110 |
| 14 | . | . | 14 | 10011001110 | 67 | ETX | c | 67 | 10000101100 |
| 15 | / | / | 15 | 10111001100 | 68 | EOT | d | 68 | 10000100110 |
| 16 | 0 | 0 | 16 | 10011101100 | 69 | ENQ | e | 69 | 10110010000 |
| 17 | 1 | 1 | 17 | 10011100110 | 70 | ACK | f | 70 | 10110000100 |
| 18 | 2 | 2 | 18 | 11001110010 | 71 | BEL | g | 71 | 10011010000 |
| 19 | 3 | 3 | 19 | 11001011100 | 72 | BS | h | 72 | 10011000010 |
| 20 | 4 | 4 | 20 | 11001001110 | 73 | HT | I | 73 | 10000110100 |
| 21 | 5 | 5 | 21 | 11011100100 | 74 | LF | j | 74 | 10000110010 |
| 22 | 6 | 6 | 22 | 11001110100 | 75 | VT | k | 75 | 11000010010 |

| Value | Char | Char | Value | Pattern |
|---|---|---|---|---|
| 23 | 7 | 7 | 23 | 11101101110 |
| 24 | 8 | 8 | 24 | 11101001100 |
| 25 | 9 | 9 | 25 | 11100101100 |
| 26 | : | : | 26 | 11100100110 |
| 27 | ; | ; | 27 | 11101100100 |
| 28 | < | < | 28 | 11100110100 |
| 29 | = | = | 29 | 11100110010 |
| 30 | > | > | 30 | 11011011000 |
| 31 | ? | ? | 31 | 11011000110 |
| 32 | @ | @ | 32 | 11000110110 |
| 33 | A | A | 33 | 10100011000 |
| 34 | B | B | 34 | 10001011000 |
| 35 | C | C | 35 | 10001000110 |
| 36 | D | D | 36 | 10110001000 |
| 37 | E | E | 37 | 10001101000 |
| 38 | F | F | 38 | 10001100010 |
| 39 | G | G | 39 | 11010001000 |
| 40 | H | H | 40 | 11000101000 |
| 41 | I | I | 41 | 11000100010 |
| 42 | J | J | 42 | 10110111000 |
| 43 | K | K | 43 | 10110001110 |
| 44 | L | L | 44 | 10001101110 |
| 45 | M | M | 45 | 10111011000 |
| 46 | N | N | 46 | 10111000110 |
| 47 | O | O | 47 | 10001110110 |
| 48 | P | P | 48 | 11101110110 |
| 49 | Q | Q | 49 | 11010001110 |
| 50 | R | R | 50 | 11000101110 |
| 51 | S | S | 51 | 11011101000 |
| 52 | T | T | 52 | 11011100010 |

| Code A | Code B | Code C | Value | Pattern |
|---|---|---|---|---|
| FF | l | 76 | 76 | 11001010000 |
| CR | m | 77 | 77 | 11110111010 |
| SO | n | 78 | 78 | 11000010100 |
| SI | o | 79 | 79 | 10001111010 |
| DLE | p | 80 | 80 | 10100111100 |
| DC1 | q | 81 | 81 | 10010111100 |
| DC2 | r | 82 | 82 | 10010011110 |
| DC3 | s | 83 | 83 | 10111100100 |
| DC4 | t | 84 | 84 | 10011110100 |
| NAK | u | 85 | 85 | 10011110010 |
| SYN | v | 86 | 86 | 11110100100 |
| ETB | w | 87 | 87 | 11110010100 |
| CAN | x | 88 | 88 | 11110010010 |
| EM | y | 89 | 89 | 11011011110 |
| SUB | z | 90 | 90 | 11011110110 |
| ESC | { | 91 | 91 | 11110110110 |
| FS | | | 92 | 92 | 10101111000 |
| GS | } | 93 | 93 | 10100011110 |
| RS | ~ | 94 | 94 | 10001011110 |
| US | DEL | 95 | 95 | 10111101000 |
| FNC3 | FNC3 | 96 | 96 | 10111100010 |
| FNC2 | FNC2 | 97 | 97 | 11110101000 |
| SHIFT | SHIFT | 98 | 98 | 11110100010 |
| Code C | Code C | 99 | 99 | 10111011110 |
| Code B | FNC4 | Code B | 100 | 10111101110 |
| FNC4 | Code A | Code A | 101 | 11101011110 |
| FNC1 | FNC1 | FNC1 | 102 | 11110101110 |
| START A | START A | START A | 103 | 11010000100 |
| START B | START B | START B | 104 | 11010010000 |
| START C | START C | START C | 105 | 11010011100 |
| STOP | STOP | STOP | | 11000111010 |

The checksum is always calculated automatically.

## Datamatrix

This barcode supports the full ASCII character set and its maximum data capacity depends on the data type and the selected barcode mode:

- Square: Up to 2046 double-digits (00 to 99), or up to 2047 alphanumeric characters.
- Rectangular: Up to 98 double-digits (00 to 99), or up to 70 alphanumeric characters.

Bar width is set using millimeters and corresponds to the minimum barcode width.

Error correction levels depends on the selected barcode mode. Levels range from 0 to 6 in Rectangular mode, and from 0 to 24 in Square mode. Note that the number of characters available is inversily proportional to the error correction that has been set.

## Discrete 2 of 5

The following characters are valid for this barcode:

- Digits: 0 to 9.

A checksum is automatically calculated. Optional bearer bars can be added for added reliability.

## FIM

This static code can only represent one of three available FIM types: FIMA, FIMB, and FIMC.

## GS1 Databar (RSS)

This barcode supports digits only and the maximum number of characters allowed is based on the selected barcode mode.

- RSS-14, RSS-14 Truncated, RSS Limited, RSS-14 Stacked, RSS-14 Stacked Omnidirectional: Limited to 14 digits (this includes a 1 digit checksum).
- RSS Expanded and RSS Expanded Stacked: No set character limit.

Bar width is set using millimeters and corresponds to the minimum barcode width.

## IMB/OneCode

The Intelligent Mail Barcode is used by USPS to track letters and flats. The barcode has no selectable option, but the data must be formatted in a very specific way to be useable. The data must be one single string of letters, composed of the following items:

- **Barcode Identifyer**: ID String of exactly 2 characters containing only numeric digits.
- **Service Type Identifyer**: String of exactly 3 characters containing only numeric digits.
- **Mailer Identifier**: String of either 6 or 9 characters containing only numeric digits.
- **Serial Number**: String with a length of 6 or 9 digits. The Mailer ID + Serial Number must have a total length of 15.
- **Delivery Point Zipcode**: Optional string with a length of 0, 5, 9 or 11 digits.

For more information on the OneCode barcode, please see www.ribbs.usps.gov/onecodesolution.

## Interleaved 2 of 5

This code must have an even number of characters (including the checksum character), so if number of characters in the data is odd, it should be prefixed with the digit 0.

The following characters are valid for this barcode:

- Digits (0 to 9).

Bearer bars are optional, but can be added for added reliability.

The checksum is optional and can be added with the checksum option.

## Japan Post

The Japan Post barcode will accept digits and uppercase letters and the hyphen. The data consists of a 7 digit postal code plus address data. If the address data is less than 13 characters the remaining character positions are filled with control characters to make the length 20.

The postal code section may have a hyphen at the 4th character position (eg. 123-4567) although this hyphen does not appear in the encoded data. There may also be a hyphen between the postal code and the address data (eg. 154-0023-1-3-2-A-507). Again this hyphen does not appear in the encoded data. Note that the remaining hyphens are encoded.

## Maxicode

This 2D barcode can be used in documents that will be printed using the Optimized Postscript, Windows Printing or printer centric mode.

This barcode supports various data types and the maximum number of characters allowed is based on the selected barcode mode:

- Mode 2: For shipping data including a zipcode.
- Mode 3: For shipping data including a postal code.
- Mode 4: For any data (full ASCII supported) up to 84 characters.
- Mode 5: For any data (full ASCII supported) up to 68 characters.
- Mode 6: For any data (full ASCII supported) up to 84 characters.

Note that the **Barcode Options** page of the **Barcode** dialog box may also be used to enter the data. When mode 2 or 3 is selected, all the corresponding dialog box fields are accessible. When mode 4, 5 or 6 is selected, only the Message dialog box field may be used to enter data. Entering data this way supersedes any data entered using the Data page of the Barcode dialog box.

## Micro PDF417

This barcode supports various data types and the maximum number of characters allowed is based on the selected barcode mode:

- Text compaction: Alphanumeric characters up to a maximum of 250 characters.
- Numeric compaction: Numeric characters up to a maximum of 366 characters.

The number of columns in the barcode may be selected as required up to a maximum limit of 4 columns.

Bar width is set using millimeters and corresponds to the minimum barcode width.

Note that the error correction level is always set automatically.

## Micro QR

This barcode supports various data types and the maximum number of characters allowed is based on the selected barcode mode:

- Numeric: Numeric characters up to a maximum of 35 digits.
- Alphanumeric: Numeric and alphanumeric characters (limited to uppercase letters, spaces, and the following characters: dollar ($), percent (%), star (*), plus (+), minus (-), period (.), slash (/),and colon (:) only up to a maximum of 21 characters.

Bar width is set using millimeters and corresponds to the minimum barcode width.

Error correction may be set from Level 0 to Level 3.

## MSI Plessey

This code can contain a variable number of characters. The following characters are valid for this barcode:

- Digits: 0 to 9.

A checksum is added automatically.

## PDF417

This 2D barcode replaces the PDF417 barcode used in previous versions of PlanetPress Design. It can be used in documents that will be printed using the Optimized Postscript, Windows Printing or printer centric mode.

If you open document that uses the "old PDF417 barcode", existing PDF417 barcodes will remain unchanged, and any new PDF417 barcode that you may add to that document will also use the "old PDF417 barcode". If you want this older document to use the new barcode, you will have to delete all its "old PDF417 barcodes", to save the document, and then to add new PDF417 barcodes. Barcodes added to older documents that did not contain any PDF417 barcode or to new documents will be added using the "new PDF417 barcode".

This barcode supports various data types and the maximum number of characters allowed is based on the selected barcode mode:

- Text: Basic alphanumeric characters up to a maximum of 1847 characters.
- Binary/ASCII Plus: Extended ASCII character set (all 256 characters) up to a maximum of 1847 characters.
- Numeric: Numeric characters up to a maximum of 2707 digits.

The number of columns in the barcode may be selected as required up to a maximum limit of 20 columns.

Bar width is set using millimeters and corresponds to the minimum barcode width.

Error correction may be set from Level 0 to Level 8.

Place a checkmark in the **Truncated** box to make the barcode a truncated PDF417 barcode. By default, barcodes are added as standard PDF417 barcodes.

## Plessey

This code can contain a variable number of characters. The following characters are valid for this barcode:

- Digits: 0 to 9.
- Upper case letters: A to F.

A two-digit checksum is added automatically.

## USPS Postnet

Standard Postnet codes must contain exactly 5 characters. The following characters are valid for this barcode:

- Digits: 0 to 9.

Apart from the standard Postnet code, two additional codes can also be used:

- Postnet+4 (5 digits + 4 supplemental digits).
- Postnet+6 (5 digits + 6 supplemental digits).

A checksum is added automatically.

## QR Code

This barcode supports various data types and the maximum number of characters allowed is based on the selected barcode mode:

- Numeric: Numeric characters up to a maximum of 7089 digits.
- Alphanumeric: Numeric and alphanumeric characters (limited to uppercase letters, spaces, and the following characters: dollar ($), percent (%), star (*), plus (+), minus (-), period (.), slash (/), and semicolon (;) only) up to a maximum of 4296 characters. Note that lowercase characters within the data are automatically converted to uppercase. In 7.3 and higher, the semicolon (;), at sign (@) and comma (,) are accepted.

Bar width is set using millimeters and corresponds to the minimum barcode width.

Error correction may be set from Level 0 to Level 3.

## Royal Mail

The following characters are valid for this barcode:

- Digits: 0 to 9.
- Upper case letters: A to Z.

A checksum is added automatically.

## UPC

The UPC barcode in PlanetPress Design supports multiple subtypes:

### UPC-A

Standard UPC-A codes must contain exactly 12 characters (including the checksum character). The following characters are valid for this barcode:

- Digits: 0 to 9.

Apart from the standard UPC-A code, two additional codes can also be used:

- UPC-A+2 (12 digits + 2 supplemental digits).
- UPC-A+5 (12 digits + 5 supplemental digits).

A checksum is added automatically.

### UPC-E

Standard UPC-E codes must contain exactly 8 characters:

- Must start with the digit 0 (must be added if not included in data).
- Six variable digits
- One checksum digit (added automatically).

The following characters are valid for this barcode:

- Digits: 0 to 9.

Apart from the standard UPC-E code, two additional codes can also be used:

- UPC-E+2 (8 digits + 2 supplemental digits).
- UPC-E+5 (8 digits + 5 supplemental digits).

**EAN-13**

The standard code must contain exactly 13 characters (this includes one checksum character).

The following characters are valid for this barcode:

- Digits (0 to 9).

Apart from the standard EAN-13 code, two additional codes can also be used:

- EAN-13+2 (15 characters in all).
- EAN-13+5 (18 characters in all).

A checksum is automatically calculated for this barcode.

Human readable characters are always included in the code.

**EAN-8**

The standard EAN-8 code must contain exactly 8 characters (this includes one checksum character).

The following characters are valid for this barcode:

- Digits (0 to 9).

Apart from the standard EAN-8 code, two additional codes can also be used:

- EAN-8+2 (10 characters in all)
- EAN-8+5 (13 characters in all)

Human readable characters are always included in the code.

## ISBN

Standard ISBN barcodes contain 10 digits (9 digits for the data plus 1 checksum digit) or 13 digits (a 3 digit prefix, 9 digits for the data plus 1 checksum digit).

- ISBN 10 digit codes typically include 4 sections and 3 hyphens or spaces.
- ISBN 13 digit codes include 5 sections and 4 hyphens or spaces. The prefix is usually 978 or 979.

Using hyphens or spaces is not compulsory but recommended to clearly separate each part of the code.

The following characters are valid for this barcode:

- Digits: 0 to 9.
- Symbols: space, - (hyphens).

Apart from the standard ISBN code, two additional codes can also be used:

- ISBN+2 (10 or 13 characters + 2 supplemental digits).
- ISBN+5 (10 or 13 characters + 5 supplemental digits).

# Business Graphic Object

A business graphic in PlanetPress Design is a bar graph, pie chart, or line graph. PlanetPress Design builds the graphic from the static or variable data you select for the graphic. There are 4 type of business graphics available:

**Bar Graph**

In a bar graph, PlanetPress Design calculates the maximum width of each bar by dividing the width of the business graph object by the number of bars in the graph. It then calculates the actual width of the bar using the Spacing percentage property you set for the bar graph.

**Bar Graph Options**

- **Show text on graphic**: Select to display the data as text under the bar.
- **Baseline value**: Enter a baseline value for the bar graph.
- **Spacing percentage**: Enter the amount of space, expressed as a percentage, to leave to the right of each bar. This determines both the width of each bar and the amount of space to its right, as follows. PlanetPress Design divides the width of the business graphic object by the number of bars in the graph to obtain the maximum width for each bar. It then uses the Spacing percentage to determine the amount of space the bar itself occupies.
- **Automatic height adjustment**: Select to automatically make the height of the tallest bar in the bar graph, equal to the height of the business graphic object. Clear to specify the height using Units per inch/centimeter.
- **Units per inch/centimeter**: Enter the number of units PlanetPress Design displays for each unit of measure. The unit of measure is as set in the User Options dialog box.
- **Use 3D effect**: Select to use a 3D effect on each bar of the bar graph.
- **Angle**: Enter the angle, in degrees, for the 3D effect.
- **Depth**: Enter a depth to use for the 3D effect. Units are as set in the User Options dialog box.

**Line Graph**

In a line graph, PlanetPress Design calculates the position of each point by dividing the width of the business graph object by the number of points on the graph.

**Line Graph Options**

- **Show text on graphic**: Select to display the data as text above each point on the graph.
- **Baseline value**: Enter a baseline value for the line graph.
- **Automatic height adjustment**: Select to automatically make the highest point in the line graph, equal to the height of the business graphic object. Clear to specify the height using Units per inch/centimeter.
- **Units per inch/centimeter**: Enter the number of units PlanetPress Design will display for each unit of measure. The unit of measure is as set in the User Options dialog box.

**Pie Chart**

In a pie chart, all data values must be greater than or equal to zero.

**Pie Chart Options**

- **Show text on graphic**: Select to display the data as text alongside the pie chart. If this is selected, PlanetPress Design uses the style set for the business graphic in Basic attributes.
- **Use 3D effect**: Select to use a 3D effect with the pie chart.
- **Angle**: Enter the angle, in degrees, for the 3D effect.
- **Depth**: Enter a depth to use for the 3D effect. Units are as set in the User Options dialog box.

In all 3 of these PlanetPress Business Graphic options, the following tabs are also available in the object's properties:

**Data**

- **Skip lines with invalid data**: Select to remove any empty lines or lines that do not contain valid integer, measure or currency values from the data selection.
- **Thousands separator**: Enter the thousands separator the input data uses. PlanetPress Design requires this to ensure it reads the input data correctly. It does not use this or any other thousands separator in the values it displays.
- **Decimal separator**: Enter the decimal point separator the input data uses. PlanetPress Design requires this to ensure it reads the input data correctly. It does not use it as the decimal separator in the values it displays. Rather, it uses a period as the decimal separator.
- **Currency symbol**: Enter the currency symbol the input data uses. PlanetPress Design requires this to ensure it reads the input data correctly. It does not use this or any other currency symbol in the values it displays.

**Colors**

- **Color list**: Displays the colors the graph uses. Each entry you add to the list displays both a sample of the color and the numerical value of the color in the color model you are currently using. Recall that you select the default color model in the User Options dialog box. The order in which you organize the colors in the Color list determines the assignment of colors to the lines, bars, or pie sections of the graph. PlanetPress Design cycles through the colors in the Color list, starting at the top, and assigns colors to the lines, bars or pie sections of the graph.

**Excel Graphic**

See Excel Graphic.

## Excel Graphic

Microsoft Excel graphics can be added to your PlanetPress Design documents just like any other PlanetPress Design business graphic. Data selections are used to populate the graphics at runtime, so each document displays context-specific information, such as stock prices or monthly expenses. Like other images, Excel business graphics can be moved or resized. At design time, only a placeholder is displayed on the page (the actual graphics are generated at runtime).

This feature is only available for Optimized Postscript printing and for Windows Printing. Microsoft Excel 2000 or better must be installed on the server running the PlanetPress Design documents (although only the Excel template is actually required to design the PlanetPress Design document).

For each Excel business graphic you want to use in PlanetPress Design, you must create an Excel template file (see the example below for more details). At runtime, PlanetPress Design uses this file to generate the graphic and to display it on the document.

To add an Excel business graphic

- Create an Excel workbook:
    - The first sheet in your workbook must include the sample data on which your graph will be based. The data displayed in the graph must appear in the first one hundred cells of the same column. Some of the cells may remain empty, but no data should appear past cell number 100. These 100 cells must not have any special formatting (use the standard formatting). You may add a table on the same sheet for the purpose of organizing your data , but any data included in that table should be referenced to your sample data cells.
    - The graph itself must be placed on a new sheet. Any variable information appearing in the graph must be referenced to the sample data cells from the other sheet, or to cells from the table, but only if those cells are referenced to the sample data cells. If you use the **Chart Wizard** to create your Excel business graph, bear in mind that at the end of the procedure, you should select the **As new sheet** option so as not to place the graph on the same sheet as the sample data..

- Save the template file in the following location: **Documents and Settings\All Users\Application Data\Objectif Lune\PlanetPress Suite N\PlanetPress Design\** (where N is the PlanetPress Suite version number)**.**
- Add an Excel Business graph object on your PlanetPress Design document:
    - Place an Excel business graph object on your document page.
    - In the **Graphic** page of the **Excel Business Graphics properties** dialog box, select any of your Excel graph template files and set its properties.
    - In the **Data** page of the **Excel Business Graphics properties** dialog box, select the data that will be placed in the source data cells of your Excel graph template file.
    - Bear in mind that when you design a document, only a placeholder is displayed. The actual graphs are only appear when documents are generated.

Excel Business Graphics properties are as follows:

**Graphic page**

- **Excel template:** Select the template you want to use for your graphic. Click the Refresh button to redisplay all the currently available templates.
- **On error:** Select the desired behavior if an Excel graph cannot be generated.
- **Excel sheet column:** Enter the letter that corresponds to the Excel data column in which the variable data is to be entered at runtime.

**Data page**

- **Excel data graph data column:** Select the data that should be entered in the cells of the Excel data sheet at runtime.
- **Excel sheet column:** Enter the letter that corresponds to the Excel data column in which the variable data is to be entered at runtime.
- Other options are similar to the Data Selection Object properties.

---

When printing Excel Business Graphics in Optimized PostScript Stream run mode, if you notice any quality loss in the graphics you will need to adjust your document's resource options and change the Picture compression level to 100. See "Set Up a Document" (page 105).

---

# Capture Field Object

The Capture Field object is used to create input fields for use with the Anoto Digital Pen in a PlanetPress Capture workflow. A document containing capture fields is called a *capture ready document*, or simply a *capture document*. Any capture document must pass through the Capture Fields Generator action task in PlanetPress Production in order to be printable.

For information on PlanetPress Capture implementations, environment considerations, activation and more, please check the PlanetPress Workflow User Guide - PlanetPress Capture

---

PlanetPress Capture Fields cannot simply be inserted into an existing document as-is and expected to work properly, efficiently or consistently. In order to design a document with Capture Fields, you **must** review and understand the Critical PlanetPress Capture Implementation Restrictions.

---

In addition to the common object properties, Capture Field objects have the following properties:

**Capture Options tab**

---

- **Capture Field Type**: Select which type of Capture Field you want to create from the following:
  - **User Area**: An area to write or select using the Anoto Digital Pen.
  - **Pidget**: Short for "pen widget", pidgets are capture fields that are used to change options on the document or the pen by drawing a line on a special pattern with the pen. See .
- **Field Style**: Select which type of User Area or Pidget to create on the page:
  - **Checkbox**: Select to create simple checkbox.
  - **List**: Currently not implemented, acts the same as a text field.
  - **Text field**: Creates a single text field with no boundaries or divisions.
  - **Multi-Area field**: The multi-area field is actually a single field, however it presents a visual separation between rows and columns, like a table. This creates a grid of fields for data input. The multi-area field automatically sizes each field so the height and width of the field is divided equally in all its areas.
    - **Rows**: Specifies the number of horizontal separations.
    - **Columns**: Specifies the number of vertical separations.

> Field Styles are being offered as forward-compatibility with future versions of PlanetPress Suite, which should include more advanced processing of these fields as well as ICR (Intelligent Character Recognition).

  - **Stylo settings** (available with pidget fields only): Select if you want to create a pidget that defines settings in the Anoto Digital Pen's ink.
- **Stylo Type** (available when *Stylo settings* is selected): Choose which pen setting to change, such as the width and color of the ink. This changes the appearance of the ink when it gets added to the document after the pen is docked.
- **Disable rewriting**: Check to prevent a field that has already been written in to accept any further ink data. This is useful only if forms are sometimes partially processed or are processed multiple times until all mandatory fields (or a final field) are completed. The rewriting is only disabled after the pen has been docked (in a different session).
- **Required field**: Select whether the capture field is mandatory, optional or final:
  - **Optional**: The field is optional, and the capture document can be closed even if no ink is present in this field.
  - **Mandatory**: The field is mandatory and ink must be present in this field for the capture document to be closed, unless it is grouped (see *Group ID* below)
  - **Final**: Any ink present in this field will force the document to close, ignoring the presence of ink in any mandatory or optional field in the rest of the capture document.
- **Group ID**: Enter a Group ID for the Mandatory field. When multiple Mandatory fields share the same Group ID, the mandatory status becomes true for the group, instead of individual fields. The capture document can be closed if any of the fields in the group has ink, instead of each individual field. Groups only work within the same page, even if they are part of the same document.

## Considerations

There are some important considerations when using the capture field object in a PlanetPress Design document:

- Capture Fields should always be set to print on top of any other object and should not be hidden. Placing another object over a Capture Field may cause unexpected behaviors in reading pen data.
- Capture Fields are complex and ink-intensive. They consume a fair amount of ink and take a lot of time to process when present. This means it is recommended only to use fields when necessary, and not to make it any larger than you need it.
- When you add one or more Capture Fields to a page, a copyright notice automatically appears on the page indicating that the Anoto technology is being used. This notice must remain visible at all times and hiding it constitutes a violation of the PlanetPress Suite End User License Agreement (EULA). You can change the position of this notice in your document's properties (see the Resources Options section in ).
- If all of the fields in a Capture document are set to be Final, they will react the same as if they were all Optional.
- Capture Fields are not supported with the N-Up feature of PlanetPress.

- There is a 7mm (1/3 inch) wide area on each side of a field the pen may not be able to recognize depending on the direction and angle at which the pen is held while writing. In order to prevent the majority of writing errors, you can place a thin white border, 7mm from the sides of your fields. The white line will not confuse then pen and anyone writing in the field will naturally write within the box created, avoiding the 7mm area. For the same reason, the minimum size of a Capture field is 2/3 inch (14mm).
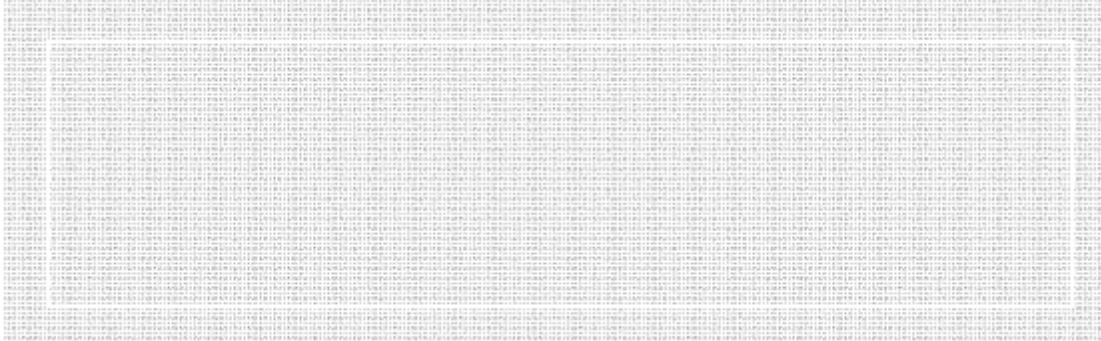
An example of a white border within a capture field.

## Pidgets

Pidgets are special static patterns you can print on a page to use with the Anoto Digital Pen. When the pen "writes" over those specific patterns, options are triggered as the pen recognizes them. In other words, pidgets are *triggers for the pen's features* and are limited to what features the pen actually has.

The pidgets themselves are small EPS files that are printed as an image. Once printed, pidgets can be re-used multiple times. This is because the pidgets are printed in true black which the pen recognizes even if there have been multiple color pen marks on top of them.

To activate a pidget, simply place the pen inside of the pidget's icon and press down. Generally, there is no need to move the pen or write anything, the pen can detect the pattern. When the pen successfully reads a pidget, it vibrates twice in quick succession.

There are two types of pidgets - those that change the digital ink settings and those that change the pen settings. At the moment, only digital ink pidgets are available but future versions of PlanetPress Suite will offer pen settings pidgets.

Available pidgets are:

- **Stroke Width**: Changes the width of the digital ink in the pen.
- **Stroke Color**: Changes the color of the digital ink in the pen.

# Capture Field Masks

When using a Field Type of Custom or Regular Expression, the mask to be used must be set with a Field Mask.

**To open the Mask List**

1. Create a new Capture Field Area
2. Set the Capture Field Type to User Area
3. Set the Field Style to Multi-Area Field
4. Change the Mask Type to either Custom or Regular Expression.
5. Next to the Format List, click the Edit Masks button.

**To add a new mask to the list**

1. Click on the ➕ button to add a new entry.
2. Double-click in the Name box and enter a name for the expression.
3. Double-click in the Value box and enter the desired expression (see below).

## Custom Masks

Custom masks are not as powerful as regular expressions but they offer two distinct advantages. First, Custom masks will automatically divide the Multi-Area Field in which they are used into the correct number of expected boxes and any static characters used in the mask will output in the Capture Field Box when printing. Second, Custom masks are simpler to use and have a one-on-one relationship between the mask and its output.

### Accepted Characters

Each character, except for \, is deemed to be a single expected character, so the number of captured characters is not variable

- 9 : Indicates any numeric character from 0 to 9.
- A: Indicates an alphabetic character from A to Z. The character will be converted to uppercase.
- X: Indicates an alphanumeric character from 0 to 9 or A to Z. Alphabetical characters will be converted to uppercase.
- \ followed by any character: Indicates a literal character. These characters will not be written by the client but rather output in the mask when printing the Capture Document.

### Examples

These examples should clarify how the accepted characters should be used:

| Mask | Description | Examples |
|------|-------------|----------|
| \(999\)999\-9999 | Canadian phone number with the area code | (514)875-5863 (800)555-1234 |
| \(999\)999\-9999\ \E\x\t\.999 | Canadian phone number with area code and extension | (514)875-5863 Ext.123 (800)555-1234 Ext.432 |
| \+9\(999\)999\-9999 | Canada/US phone number including area code and country code. | +1(800)555-1234 |
| \P\R\O\D\-XXXXX\-AAA\-99999 | A product number made of 5 alphanumerical characters, 3 letters and then 5 numbers. | PROD-5G32B-ADE-23643 PROD-IPHON-TGS-16000 |
| 999\-9999 | A Japanese postal code, which is digits only | 325-6731 812-2298 |
| 99999 | A 5-digit US zip code | 90210 32454 |
| A9A\ 9A9 | A Canadian postal code | H0H 0H0 J0S 1J0 H1V 2C8 |
| \$999\,999\.99 | Dollar amount with thousand separator, up to $999,999.99 dollars. However, note that all characters must be present, so the mask must be padded with zeroes (0) if the amount does not | $000,001.99 $321,443.54 $000,599.97 |

fully fill each box (see examples)

It is important to understand that these masks are very precise and will not accept any sort of variation. For example, the second example (phone number with extension) will only accept a 3-digit extension. A 5-digit extension could not be entered with this mask. Because the Multi-Area Field will automatically divide the field into boxes accepting one character per box, this is made clear to anyone using these boxes.

## Regular Expressions

If the data that is expected from the field can vary in contents as well as length, a Regular Expression can be used instead of a Custom mask. Regular expressions are much more powerful but they can be difficult to understand at first. Note that a Regular Expression mask will not automatically divide the Multi-Area Field into rows and columns nor will it print out static characters in the field, since the number of characters can be variable.

### Accepted Values

This documentation cannot pretend to teach anyone how to learn to use Regular Expressions nor to describe how to build regular expressions. Plenty of tutorials and documentation on Regular Expressions exist online. However, here are some quick resources that may help you in building your expressions:

> These links are provided for your education and are not supported or maintained by Objectif Lune Inc.. Use the information provided by those links at your own risk.

- Regular Expressions Cheat Sheet , available in PDF and PNG format
- Regular-Expressions.info tutorial.

### Examples

These examples are similar to the ones in the Custom Mask but may extend upon them and accept a wider variety of results.

| Regular Expression | Description | Examples |
|---|---|---|
| (\d{3})\D*(\d{3})\D*(\d{4})\D*(\d*)$ | A fully featured expression for most phone number formats used in the world. | work 1-(800)-555.1212 #1234 800-555-1212 80055512121234 |
| PROD-[0-9A-Za-z]{5}-[A-Z]{3}-\d{5} | Product Number | PROD-5G32B-ADE-23643 PROD-IPHON-TGS-16000 |
| ^\d{5}(-\d{4})?$ | A 5 or 9 digit US Zip code with optional dash. | 90210 32454-4332 |
| ^[ABCEGHJKLMNPRSTVXY]{1}\d{1}[A-Z]{1} *\d{1}[A-Z]{1}\d{1}$ | A valid Canadian postal code. | H0H 0H0 J0S 1J0 H1V 2C8 |

# Document Resources

Resources are images or PostScript attachments you add to your document. These resources can be used using Picture Objects (for images) or the Attachments properties of documents and pages (PostScript attachments).

This chapter explains resources and their location as well as how to refer to them from within your document.

> For the purpose of this chapter, PDF files are considered to be images and are used in the same way as multi-page TIFF files.

# Image Resources

An image resource is any sort of image (see ["Supported Image Formats" (page 173)](#)) that is used in your PlanetPress Design document, whether it is imported as an internal resource or used as an external resource, called when the document executes.

## Location of image resources

In order to be useful in a PlanetPress Design document, images have to be in one of the following locations:

- **Internal Images**: Internal image resources are imported within your PlanetPress Design document and follow the document wherever it is sent, including PlanetPress Workflow servers, printers and other PlanetPress Design work-stations.
- **Printer Images**: These images are located on the printer's hard drive or memory, and can only be used by documents printing in the Printer-Centric run mode.
- **Network Images**: Images located on a local path or network path. Those images can only be accessed through the Optimized PostScript run mode or the Windows Driver output. Accessing these images may require special permissions at the system or network level.

Each location has its own unique advantage and limitation, and they can be combined in hybrid implementations if necessary.

- Internal Images are best if you have a small number of images called statically or via conditions in your document, and if these images are not too large. Since they are kept inside the document, having too many images or large images may cause issues with manageability and performance.
- Printer Images are useful to maintain a smaller document size regardless of the number or size of the images being called. Images can also be shared between multiple documents when they are printed from the same printer. However, some printers may have limited space on their hard drive.
- Network Images are useful only if you are sharing images between multiple installations of PlanetPress and may slow down printing significantly as no caching is possible with this method.

## Static and Dynamic Images

A static image is an image that does not change during document execution. Static images are independent of the input data.

A dynamic image is an image that changes during document execution. Dynamic images depend on the input data and other variables to determine the image file to display.

You create both static and dynamic images using picture objects (see ["Picture Object" (page 150)](#)). In the case of a static image, the picture object references a single image resource and is selected from the image drop-down menu. In the case of a dynamic image, the reference in the picture object is a PlanetPress Talk expression that resolves to a different image name on each data page. Since dynamic images rely on PlanetPress Talk expressions, they are described in the appropriate chapter. See ["Dynamic Images" (page 207)](#).

## Supported Image Formats

What image formats does PlanetPress Design support?

The following summarizes the image formats PlanetPress Design supports for both static and dynamic images.

| Format: | Extension: | Type: | PlanetPress Design supports: |
|---|---|---|---|
| BitMap Picture | BMP | Bitmap | 1-, 4-, 8-, and 24-bit images in Windows format. Supports uncompressed and Run Length Encoded (RLE) formats. Does not support RLE for 24-bit images. |
| Encapsulated PostScript | EPS | Vector, Metafile | EPS files that conform to the Adobe **Encapsulated PostScript File Format Specification** (Adobe Technical Note #5002). |
| Joint Photography Experts Group | JPEG, JPG | Bitmap | 8-bit grayscale and 24-bit color images. Supports save quality, progressive save, and optimized Huffman codes. |
| Portable Document Format | PDF | Metafile | PDF 1.5 format (Acrobat 6) without encryption or security settings. |
| Portable Network Graphics | PNG | Bitmap | 1-, 4-, 8-, and 24-bit images. Supports gamma adjustments, progressive save, and variable compression levels. Also reads but does not consider the alpha channel. |
| Tagged Image File Format | TIF, TIFF | Bitmap | 1-, 4-, 8-, and 24-bit uncompressed images. TIFF images that use LZW compression. CCITT group 3 1D, CCITT group 3 2D, CCITT group 4, Packbits, Zlib (deflate), JPEG. Also supports multi-page images but restricts output to only the first image. PlanetPress Design now supports TIFF images that use LZW compression. |

In the case of dynamic images that reference images external to the document, in addition to the supported formats listed above, documents can also accept images in the PostScript format generated by the Image Downloader.

## Image Glossary

Defining images in PlanetPress Design requires that we first establish a few terms that will be used throughout this chapter.

### Bitmap File Formats

A bitmap file format stores an image as a set of pixels. Common bitmap formats are BMP, TIFF, PNG and JPEG. Images such as photographs typically use a bitmap format.
The disadvantages of bitmap images are loss of image quality when scaling, and large file sizes for images with large pixel dimensions and color depths.

**Vector File Formats**: A vector file format encodes an image as a set of commands or mathematical equations. Encapsulated PostScript (EPS) is a common vector format. Images such as illustrations typically use a vector format.

In a vector format, the commands precisely describe how to draw the image. The advantage of images in vector format is that they scale without loss of color, detail, or sharpness. Depending on the complexity of the image they describe, they may also have a smaller file size than their bitmapped counterpart. The disadvantage is that they require the computer or printer to interpret the commands and render the image.

### Metafile Format

A metafile image format can include both bitmapped and vector images. PlanetPress Design supports images in both Encapsulated PostScript (EPS) and Portable Document Format (PDF) metafile formats.

As for vector images, a metafile image file specifies a size for the image. In this case, if the metafile image contains bitmapped images, scaling it may affect the image quality of those bitmapped images.

### Resolution

Resolution is a broad term that can refer to different things in different contexts. A detailed discussion of the different meanings and interrelationships is beyond the scope of this guide. Instead we provide This section provides a brief overview of the different possible meanings to clarify our use of resolution in PlanetPress Design.

| Term: | Means: |
| --- | --- |
| Scan resolution | The resolution at which you scan an image on a scanner. It refers to the number of samples per inch (spi), where each sample represents a single pixel in the resulting bitmapped image. |
| Display resolution | The resolution of the monitor screen. For example 96 DPI means the screen displays 96 pixels per inch. Thus an image with pixel dimensions of 96 x 96 would measure 1 inch x 1 inch on this monitor screen. |
| Printer resolution | The number of spots of ink the printer can produce per inch, commonly measured as dots per inch (DPI). The use of DPI as a measure can be confusing as it may be interpreted as meaning one dot of ink maps to one image pixel. In fact the number of spots of ink the printer uses to represent a single pixel can vary with the halftone screen used for a print job (the lines per inch (LPI)), the color depth of the image, and the algorithm the printer uses to represent individual colors. |

**Color Depth**

Color depth refers to the number of colors each individual pixel in a bitmapped image can display. Color depth is expressed in bits per pixel where each individual bit can take on one of two values (0 or 1).

**Pixel Dimension**

Pixel dimensions refer to the number of pixels in the width and height of a bitmapped image. Thus an image with pixel dimensions of 200 x 300 has a width of 200 pixels and a height of 300 pixels. Pixel dimensions do not define a fixed size for the bitmapped image. Rather the resolution determines the final physical dimensions of the image. See "Resolution" (page n) for help understanding resolution in PlanetPress Design.

You can adjust the pixel dimensions of a bitmapped image to adjust the size it occupies at a given resolution. This is called resampling the image, and it is particularly useful for large image files that contain more pixels than are necessary to produce the highest quality output.

**Image Quality: Line Art**

In line art quality, the edges of different elements of the image are clearly visible and sensitive to any increase or decrease in pixel dimensions that occurs through resampling. As an example of line art, think of a corporate logo. A logo typically uses flat areas of color with each area clearly distinguished from the other. PlanetPress Design uses a *lossless* compression for line art quality images. A *lossless* compression preserves all image information.

**Image Quality: Photo**

In photo quality, edges of elements are not as clearly delineated and are consequently less sensitive to any increase or decrease in pixel dimensions that occurs through resampling. As an example of photo quality, think of a photograph of a natural object or landscape; there are many subtle tones and gradations, even along the edges between elements. PlanetPress Design uses a *lossy* compression (JPEG compression) for photo quality images. A *lossy* compression does not preserve all image information. In the case of photo quality, you set a compression level that determines how much image information is lost. An appropriate compression level results in no visible deterioration of quality in the document output.


# Scanline Orientation

Scanline orientation refers to the way in which the laser in a laser printer writes the page image onto the printer drum. In PlanetPress Design you can use this information to minimize the amount of runtime processing bitmapped image resources require, and thereby contribute to minimizing the overall execution time of a document.

You determine the scanline orientation for an image resource from three pieces of information: the paper orientation selected in the document for the page containing the image resource, how the paper enters the printer, and the scanline orientation of the printer. Once you have these three pieces of information, you then visualize the page of your document moving through the printer and determine whether the printer scans the images onto the page from top to bottom or side to side.

If you use the same bitmapped image resource in both a rotated and un-rotated presentation, you should add the image file as two distinct image resources, and set the scanline orientation for each image resource separately.

You set the default scanline orientation for all bitmapped image resources in the document, in the Document dialog box. You can set the scanline orientation for an individual bitmapped image resource by selecting that image resource in the Structure area, and then using the Object Inspector to edit the Scanline orientation property.

## Image Quality Settings

The image quality and compression settings in PlanetPress Design apply only to the images included in the document. They do not apply to dynamic images that are external to the document. Note however that you can adjust the image quality of dynamic images external to the document if you use the Image Downloader to download the images to their runtime location.

You can set a default image quality for all bitmapped image resources you add to the document in the Document dialog box. You can also adjust the image quality for an individual bitmapped image resource. You set a single compression level for all photo quality image resources in the document.

### Suggested Resolution Settings

Any bitmapped images embedded in PDF or EPS files that exceed the resolutions set are downsampled to this resolution. This downsampling occurs for image resources at document conversion and for external images in supported image formats, at runtime. In both cases the maximum resolution is the one in effect at document conversion. No downsampling occurs for images you download using the Image Downloader.

You set a separate resolution for color, grayscale, and monochrome images. On all but very high-end 600 DPI printers, resolutions greater than those listed below do not result in any noticeable difference in quality to the human eye.

| Color depth: | Maximum suggested image resolution: |
| --- | --- |
| Color | 150 to 200 DPI |
| Grayscale | 150 to 200 DPI |
| Monochrome | 600 DPI |

An exception to these recommendations is in the case of a line art image. Line art images should ideally be at the resolution of the printer. Thus if the printer resolution is 600 DPI, you should set the resolution of the line art image to 600 DPI.

# Caches

A cache is an amount of RAM set aside to hold items that the document references during execution. In the case of PlanetPress Design and PlanetPress Suite Workflow Tools these items are images. The cache improves performance by providing faster access to images that the document uses more than once.

There are two caches: a host-based cache, and a printer-based cache. You can adjust the sizes of both of these caches. However, the default cache sizes set for PlanetPress Design are reasonable ones that in most cases should not require adjustment.

## Host-Based Cache

The host-based cache is the cache the document uses when it executes host-based. The size of this cache is independent of the document, and its size in PlanetPress Design is independent of its size in PlanetPress Suite Workflow Tools. When you execute the document host-based from within either product, the size of the cache depends on the setting in effect in that product at execution time. In PlanetPress Design you use the Image cache option in the User Options dialog box to set the size of this cache.

When the document executes host-based in either PlanetPress Design or PlanetPress Suite Workflow Tools, it uses the host-based cache for its image resources, and for the dynamic images it retrieves from the PlanetPress Design Suite virtual disk. It never uses this cache for dynamic images it retrieves from a folder.

## Printer-Based Cache

The printer-based cache is the cache the document uses when it executes printer-based. The size of this cache depends on the document.

You use the Max form cache option in the Document dialog box to set the size of the printer cache. You can also use the Max form item option to set the maximum size of any item in the cache.

When the document executes printer-based, it uses the printer-based cache for both image resources and external images any dynamic images reference.

## Adding Image Resources

Image resources are added differently depending on their location.

**Network images** are simply placed in the correct location using your operating system or any process you may have to create, move and copy images on your computer or network. They are accessed using their full path in a PlanetPress Talk expression (see "Dynamic Images" (page 207))

**Printer images** are sent to your printer using either one of the following tools:

- "The Image Downloader" (page 639) is used to send images to one or more printer or PlanetPress Suite virtual drive.
- The Send Images to Printer and Download to Printer plugins in PlanetPress Workflow Tools can be used to send images to one or more printer or PlanetPress Suite virtual drive.

**Internal images** are imported in your PlanetPress Design document using one of the following methods:

- In the **PlanetPress Design Ribbon**, go to the **Home** tab then click **Picture Resource** in the **Document** group. Browse to the folder where the image is located, then select one or more images to import as resources.
- Copy the image file to your clipboard using **CTRL+C** in Windows Explorer. Then in PlanetPress Design, click on **Picture Resources** in the **Document Structure Pane** and use **CTRL+V** to paste the image as a picture resource.
- Select a single image in Windows Explorer then drag & drop it to the **Workspace**. This imports the picture resource and also creates a picture object and associates it to the new picture resource.
- Select multiple images in Windows Explorer then drag & drop them either in the **Workspace** or in the **picture resources** folder of the **Document Structure Pane**.

Once the picture resource is added to the document, a link is maintained with the original file on your operating system. As long as your PlanetPress Design document remains on the same computer and the original image is in the same location, if the original image is modified a dialog will be displayed in PlanetPress Design asking if you want to update the image instances in PlanetPress Design.

# Guidelines for Optimizing Images

Images increase the execution time of a document and in the case of image resources, also increase the file size of a document. Tuning images to the execution environment of the document and the image quality required in the output, keeps document file size and execution time to a minimum.

This section provides suggestions for reducing the file size of images and minimizing the amount of runtime processing they require, without compromising the required image quality. These are general guidelines. In any real-world situation, there are often trade-offs to make among image quality, image file size and document execution speed.

- **Image Formats**: Unless displaying complex images such as photos, landscapes or very complex logos, vector-based images are normally faster and offer better quality on your printer than bitmap (raster) images.
- **Image Resolution**: Always set bitmap (raster) images to Constant Resolution. If the image appears to be too large for your page, it means its resolution is too high and it should be resized to be smaller and fit within the page. A higher resolution image will not printer better on your printer - it is limited by its own resolution (normally 600 DPI) and the technology used to print colors and greyscale image.
- **Scanline Orientation**: Always use bitmap images that has been rendered in the correct scanline orientation.
- **Image Complexity**: The more complex the picture, the more processing time to print it. This applies to both vector and bitmap images and is especially true when using gradients. Gradients make vector-based images much larger and timely to process, and bitmap images remain larger since gradients cannot be easily compressed.
- **PDF and EPS files**: The more images and data PDFs and EPS files contain, the more time they take to process. This is especially true if these files contain images.
- **Multipage PDF or TIFF file**: Image resources like PDF files and TIFF files may contain multiple pages, and all their pages will be processed even if they are not all used in the document. If you are not using a large number of pages, it is better to remove them from the image resource.

## Modifying and Updating Images

Internal Images can be modified and updated in many different ways.

**To edit an internal image resource**:

1. In the **Structure area**, select the image resource you want to edit.
2. In the **Object Inspector**, locate the **Edit image** property, click the property value and click the edit button that appears on the far right of the property value.
   PlanetPress Design launches the image editor. If the image resource is a bitmapped image resource (an image resource in GIF, TIFF, PNG or BMP format), PlanetPress Design launches the image editor you defined in the User Options dialog box. If you did not define an image editor in the User Options dialog, or if the image resource is in either PDF or EPS format, PlanetPress Design launches the default editor defined for those formats in Windows.
3. Edit the image resource.
4. Save the changes and exit the image editor.

**To resample an internal image resource** (adjust the pixel dimensions), do either of the following:

- *In the Object Inspector, adjust the pixel height or pixel width of the image resource that static image references*. Note that when you adjust one pixel dimension, PlanetPress Design automatically adjusts the other dimension to preserve the aspect ratio of the image. Also note that resampling an image resource in the Object Inspector has an irreversible effect on image quality. If you want to restore both the original pixel dimensions and the original quality of the image resource, you must add the image resource to the document again.
- *Open the image resource the picture object references in an image editor and adjust its pixel dimensions*
  Any picture objects that reference that image resource update to reflect the new pixel dimensions.

**To edit the properties of an image resource:**

1. In the Structure area, select the image resource you want to view and/or edit.
2. In the Object Inspector, view and/or edit the properties.
   - **Name**: The name of the image resource in the document.
   - **Color depth**: Indicates the color depth of the image resource. Color depth is meaningful only for image resources of type bitmap. You cannot edit this property; PlanetPress Design automatically determines the color depth by examining the file specified in the File name box.
   - **Edit image**: Use to launch the image editor and edit the image resource. You define the image editor you want to use in the User Options dialog box.
   - **File name**: Specifies the pathname of the image file from which PlanetPress Design created the image resource. Editing this is equivalent to replacing the image resource.
   - **Image quality**: Select the image quality for this image resource. Select Photo if edges in the image are not sharply defined and thus not highly sensitive to any loss of image information that occurs through compression. The compression PlanetPress Design applies to line art images is lossless.
   - **Image type**: Specifies the type of the image file (bitmap, EPS, or PDF). You cannot edit this property; PlanetPress Design automatically determines the type by examining the file specified in the File name box.
   - **Height (pixels)**: Adjust the number of pixels in the height of the bitmapped image resource. If you adjust this value PlanetPress Design also automatically adjusts the value in the Width box to maintain the aspect ratio of the image. A pixel height adjustment has an irreversible effect on quality and for this reason PlanetPress Design prompts for confirmation before proceeding with the operation. The only way to restore the original pixel dimensions of an image resource at its original quality is to add the image resource to the document again.
   - **Width (pixels)**: Adjust the number of pixels in the width of the bitmapped image resource. If you adjust this value PlanetPress Design also automatically adjusts the value in the Height box to maintain the aspect ratio of the image.
   - When you import a PDF file into a PlanetPress Design document, its size may not be set accurately, so you should set it manually.
   - **Scanline orientation**: Specify the scanline orientation for the image resource. When PlanetPress Design adds an image resource to the document, it sets the scanline orientation of the image resource to the value of the Scanline orientation box in the Document properties dialog box. If you have the value properly set in the Document properties dialog box, you should only need to adjust the value here if you rotate the image resource within a picture object by a multiple of 90 degrees. If you use the same image resource in both a rotated and unrotated presentation, you should add the image file as two distinct image resources, and set the scanline orientation for each image resource separately.

When a PDF image resource is selected, the Object Inspector offers a preview of the PDF as well as a page selector. To navigate the pages of a multi-page PDF image resource, locate the page selector and either enter the page of the PDF you want to view, or use the spin buttons to navigate the pages of the PDF.

**To edit or resample a network image**:

- Open the file on the computer using your graphic editor of choice, make the modification to the image file and save the new version. PlanetPress Design will always use the latest version of your file.

## Color Management and Matching

Since PlanetPress Design displays on your screen, in PDFs and on printers, there are a few things to keep in mind if you want the colors you see to be the same as the colors your print, and if you want to match a specific, exact color.

**Monitor display VS Printing**

- The monitor produces the color you see on-screen with light while the printer produces color with pigment. The set of colors, or gamut, you can produce with light is not identical to the set you can produce with pigment. Thus there are colors you can produce on a monitor and not on a printer, and vice-versa.
- The monitor uses three primary colors of light (red, green, blue) to produce all the colors you see on-screen. It mixes different amounts of each of the primaries to produce a particular color. An on-screen color is specified as three numeric values, the first describing the amount of red, the second the amount of green, and the third the amount of blue light to use to create the color. Thus these are often referred to as RGB (Red Blue Green) colors.
- The printer uses three primary colors of ink (cyan, magenta, yellow) and black to produce all the colors it prints.
- The set of colors you can produce with light (the RGB gamut) is larger than the set of colors you can produce with pigment (the CMYK gamut). Thus monitors can produce more colors than printers.There is a significant overlap between the two gamuts however, and, in those cases, the problem becomes how to match a color that it **is** possible to create with either light or pigment, on different physical devices.

**Representation and control of color in physical devices**

The difficulty with physical devices is that none are stable enough to ensure a consistent representation of a given color. Physical devices for our purposes are monitors and printers.

- **Monitors** The same color can vary across monitors due to factors such as the phosphor specification, the calibration, and the age of the individual monitor. Even on the same monitor the color can change as the monitor ages or loses its calibration. The set of colors a monitor can display (its gamut) can also vary across monitors.
- **Printers** The same color can also vary across printers or on the same printer due to factors such as the inks a printer uses, the amount of ink in the printer at the time you print, and the physical properties of the paper on which you print.

**Color perception**

Our perception of a color can change with variations in the ambient lighting. A color that appears very rich under subdued lighting may appear washed out under bright lighting. Our perception of a color can also change due to the colors that appear alongside it. The same color on two different backgrounds can appear to be two different colors. Finally, two individuals may not see the same color.

To set up color management in PlanetPress Design:

1. Start PlanetPress Design.
2. From the **PlanetPress Design Button**, choose **Preferences** to display the Preferences dialog box.
3. In the **Preferences** dialog box, expand **Behavior**, click **Color**, and set the color management options.
   - Select **Color Management Active**. Note that you must select this option before you can select the monitor and printer color profiles.
   - Set **Monitor Profile** to the color profile for the monitor of the computer on which you are running PlanetPress Design.
   - Set **Printer Profile** to the color profile for the printer on which you intend to execute the document.
4. Click **Close**.

When you use color management profiles, PlanetPress Design attempts to display the color on your monitor as close with so it is as close as possible to the printed result.

The following limitations apply, however:

- Since external variables such as lighting, time of day and different users can impact colors, there is no guarantee of the result.
- Changing the color profile does not, in any way, change printed and PDF output colors. It is only and exclusively meant for monitor display.

- Profiles only apply to colors specified within PlanetPress Design such as font colors, backgrounds, shapes and lines. Profiles do not modify the display of images and other "external" resources.

**Color Management in PlanetPress Design**

Objectif Lune approached the issue of color management with an eye to making it as straightforward and convenient as possible for its customers. It designed the color management in PlanetPress Design to accurately display, in the Page area, the colors that print when the document executes on a printer.

When you set up color management in PlanetPress Design, you are setting up a relationship between two color profiles: a monitor profile and a printer profile. The color management workflow in PlanetPress Design consists of two basic steps:

1. Install the device color profiles.
   You install a profile for each monitor you intend to use during document design with PlanetPress Design, and for each printer on which you intend to execute your documents. PlanetPress Design uses version 2.0 of the Image Color Management (ICM) system. The ICM system supports device profiles that conform to the International Color Consortium (ICC) color profile specification.
2. In PlanetPress Design, select the appropriate monitor profile for your monitor and the appropriate printer profile for the printer on which you intend to print the document. If the printer and/or monitor you use is document-dependent, you may need to adjust the profiles you use on a per document basis.

Once you set up color management, PlanetPress Design handles color in the document as follows:

| Color source: | PlanetPress Design color management: |
|---|---|
| Document color | When you select colors using the Color Picker or define colors using numerical values, PlanetPress Design displays the specified colors by first assuming they are in the color space of the selected printer profile and then translating them into the color space of the selected monitor profile. This ensures the on-screen color represents, as closely as possible, the one the printer prints. |
| Bitmapped images | When you import static bitmapped images or reference dynamic bitmapped images, for performance reasons PlanetPress Design does not perform any color management on the images. |
| PDF and EPS image | When you import a static PDF or EPS image resource or reference a dynamic PDF or EPS image, for performance reasons PlanetPress Design displays the PDF or EPS as color or grayscale images. The images are displayed in color only when you preview or execute the document. |

# PostScript Attachments

A PostScript attachment is a file that contains standard PostScript code, which is attached in your document and printed as-is. PostScript attachments are not added in your pages and are rather executed before or after a page or document. This means that they must contain PostScript that creates and prints one or more pages in a format that your printer will understand.

You cannot edit a PostScript attachment resource from PlanetPress Design. You must edit a PostScript attachment resource in Windows using the appropriate application.

You can set PostScript attachments to execute before either one of these locations:

- The Document: In the document's properties, you can set any number of PostScript attachments to run before or after each document, which is normally equal to one data page unless you are using the N-Up object or special PlanetPressTalk command to modify its execution order.
- A specific page: In a normal page's properties, you can set any number of PostScript attachments to run before or after that page.

In both cases, you can apply a condition so the PostScript attachment will only execute at certain times depending on your data and other variables.

**To add one or more PostScript attachments as resource to your document, do one of the following:**

- In the **PlanetPress Design Ribbon**, go to the **Home** tab then click **Attachment** in the **Document** group. Browse to the folder where the PostScript file is located, then select one or more images to import as resources.
- Copy the attachment file to your clipboard using **CTRL+C** in Windows Explorer. Then in PlanetPress Design, click on **PostScript attachments** in the **Document Structure Pane** and use **CTRL+V** to paste the attachment as a resource.
- Select multiple PostScript files in Windows Explorer then drag & drop them either in the **Workspace** or in P**ostScript attachments** folder of the **Document Structure Pane**.

If you add an attachment with the same name as an attachment that currently exists in the document, PlanetPress Design appends a number to the name of the new attachment.

**To edit the properties of an attachment resource:**

1. In the Structure area, select the PostScript Attachment you want to view and/or edit.
2. In the Object Inspector, view and/or edit the properties.
    - **Name**: Specifies the name of the PostScript Attachment resource in PlanetPress Design.
    - **Original file name**: Specifies the path of the file from which PlanetPress Design created the attachment resource. Editing this is equivalent to replacing the PostScript Attachment.

## Delete a Resource

You can delete an image resource or PostScript attachment at any time by selecting it, then doing any of the following:

- Press the **Delete** key on your keyboard.
- In the **PlanetPress Design Ribbon**, go to the **Home** tab then click **Delete** in the **ClipBoard** group.
- Right-click on the object and select **Delete** from the menu.

If the resource is currently in use, deleting it without any further action would break the link to that resource from picture objects, pages and PlanetPress Talk code and cause errors when printing, previewing and converting. In order to prevent this, PlanetPress Design features the Resource Deletion window.

The Resource Deletion window appears once for each resource you are deleting. It displays the following options:

- **Deletion options**:
    - **Replace all references with**: Select to delete the resource and to replace all references to it with a reference to another image resource in the document.
        - **Resource available**: Select the image resource you want to use as the replacement reference. When you delete the resource, PlanetPress Design replaces all references to the deleted resource with a reference to the resource you select here. Select the * None * option if you want to remove the references to the resource while keeping the objects that reference the resource intact. You can use the Add Resource button to add a new image resource to this list.
        - **Add Resource button**: Click to add a new resource, which ads it to the list and imports it in your document.
    - **Delete**: Select to delete the resource and all document elements that reference it.
- **Referencing elements**: Lists all of the elements from which the resource is referenced, letting you know in a glance what objects will be modified by this dialog.

**To quickly delete any unused resource in your document:**

1. Select all the resources of the same type (images or attachments) and use one of the methods described above to delete them.
2. On each Resource Deletion dialog, simply click on Cancel.
   One dialog will appear for each used resource, but no dialog appears for unused resources and they are simply deleted.

# Conditions

This chapter explains conditions, variables, and PlanetPress Talk and how to use these features in your document.

## Conditions

What is a condition?

A condition is a PlanetPress Talk expression that performs a test on the data page and resolves to either True or False. The condition may be as simple or as complex as you require. You use conditions to make the display of a page, object, group, or line of data in a data selection object dependent on input data. The page, object, group, or line of data in a data selection object displays only if the condition resolves to True.

Conditions can be used to create OMR, by creating black lines of the size and locations specified in your OMR requirement specifications, and by applying correct conditions to these lines (such as "first page", parity checks and page numbering).

There are three kinds of conditions: global conditions, local conditions, and line conditions. You can associate at most one global or local condition at a time with a page, object, or group. A line condition is internal to a data selection object. You can associate a global or local condition with a data selection object, regardless of whether that data selection object uses a line condition. The global or local condition you set determines whether the data selection object will display. The line condition alters **what** the object displays. Note that global conditions are the only ones that appear in the Structure area, and are the only ones available for use with any of the pages, objects, or groups in the document.

It is important to understand the order in which the document processes conditions when it executes. At runtime, the document evaluates global conditions before it executes a data page. It evaluates local conditions and line conditions as it executes the pages or objects with which they are associated.

## Global Conditions

What is a global condition?

A global condition is a condition that is available for re-use throughout the document. Global conditions are the only type of conditions that appear in the Structure area, and that are thus available for re-use by other elements in the document.

You create global conditions for conditions you expect to use more than once in the document. For example, consider a form letter that includes a few pages of information relevant only for residents of Kyoto. You create a global condition to test for the presence of "Kyoto" in the client address in the data page, and associate the condition with each of the relevant pages.

## Local Conditions

What is a local condition?

A local condition is one that you associate with a particular page, object, or group. It is not available to any other page, object, or group in the document, and appears only within the element in which you define it; it does not appear in the Structure area. You cannot re-use or combine a local condition.

You can use global conditions as variables in a local condition.

## Line Conditions

What is a line condition?

A line condition is a condition that acts as a filter on a data selection. Only data selection objects can use line conditions. The line condition determines two things: what lines (or records in the case of a database emulation) to display in the data selection, and whether to display an empty line for any line (or record) that does not display. Line conditions only make sense for data selections that extend over more than one line of the data page (or record of the record set in the case of a database emulation).

A line condition performs its test on each line of the data page a data selection object references. There are three types of line conditions you can define: one that tests for the presence of a text string, one that tests for the absence of a text string, and one that you define yourself using a PlanetPress Talk expression.

**What is a Line?**

In a line condition, a line refers to an entire line of the data page, not just the portion of the line that falls inside the data selection. The document tests the entire line of the data page when it evaluates a line condition.

If you set a line condition, the document tests all lines of the data page that have a portion lying within the data selection region. If the line condition resolves to True, the portion of the line that lies in the data selection region displays. If the line condition resolves to False, the portion of the line that lies in the data selection region does not display.

## Create or Remove a Line Condition

To create a line condition using the Data Selection properties dialog box:

1. If the properties dialog box for the data selection object for which you want to create a line condition is not already open, open it doing the following: In the Structure area or in the Page area, double-click the data selection object.
2. In the **Data Selection** properties dialog box, click **Lines**, and locate the Line Condition Properties area.
3. Select the type of line condition you want to create, and set the options associated with that type. If you select When advanced condition is true, and need help understanding how to enter the PlanetPress Talk expression, see "Variable PlanetPressTalk Properties" (page 197).
   **Line condition**: Select the type of line condition you want to create. Select When text is present to create a line condition that tests for the presence of a text string. Select When text is absent to create a line condition that tests for the absence of a text string. When you select either of these, a Text to search for box and a Location box appear. Use the Text to search for box to enter the text string you want the condition to look for, and the Location box to define the line, or the line and column number, (or in the case of a database emulation, the record, or the record and field name) on which you want to perform the test Select When advanced condition is true to define your own line condition using a PlanetPress Talk expression. Use the Line displays if box that appears to enter the PlanetPress Talk expression.
   When text is present / When text is absent
   These boxes appear when you select either When text is present or When text is absent in the Line condition box.
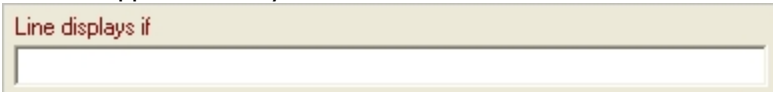
   

   **Text to search for**: Enter the string whose presence/absence on the line you want the condition to test.
   **Location**: Select the line, or the line and column number, that you want to test for the presence/absence of the string you specified in the Text to search for box. In database emulation line refers to a record. The selections available are relative to the current line. Select On line, On previous line, or On next line to test for the presence/absence of the string on the current line, the preceding line, or the succeeding line respectively. In all emulations except database, select On line at column, On previous line at column, or On next line at column to test for the presence/absence of the string at a specific column on the current line, the preceding line, or the succeeding line respectively. In this case you use the From column spin box that appears to enter the column number. You cannot specify both a record (line) and a field (column) in database emulation.

PLANETPRESS. SUITE

**From column**: Enter the column number at which to test for the string. This box appears when you select On line at column, On previous line at column, or On next line at column in the Location box. Recall that these options are not available in database emulation.

When advanced condition is true

This box appears when you select When advanced condition is true in the Line condition box.

Line displays if

**Line displays if**: Enter the PlanetPress Talk expression that defines the line condition. Remember that a condition, and thus this expression, must always resolve to either True or False.

4. Define whether or not you want the document to display a blank line when the line condition resolves to False for a given line (or record).
   **Do not display a blank line when condition is false**: Select to prevent a blank line from appearing when the line condition resolves to False. Clear to display a blank line when the line condition resolves to False.
5. This completes the line condition definition. If you have no other adjustments to make in the Data Selection properties dialog box, click **OK** to exit the dialog box.

To create a line condition using the Object Inspector:

1. Select the data selection object on which you want to set a line condition.
2. In the Object Inspector, locate the Line condition properties and adjust them to create the line condition you want to set on the object.

To remove a line condition using the Data Selection properties dialog box:

1. In the Structure area or in the Page area, double-click the data selection object.
2. In the **Data Selection** properties dialog box, click **Lines**.
3. In the **Line condition** box, select **No line condition**.
4. If you have no other adjustments to make in the Data Selection properties dialog box, click **OK** to exit the dialog box.

To remove a line condition using the Object Inspector:

1. Select the data selection object from which you want to remove a line condition.
2. In the Object Inspector, locate the Line condition property and select **No line condition**.

**Examples of Line Conditions**

You might use a line condition to accommodate an unstable data page structure. For example, consider a document that uses ASCII emulation and for which the data selection you want does not always appear in exactly the same place on every data page; the data selection shifts up or down by a few lines. The string "PREF" appears on those data lines, making it possible to uniquely identify them. You create a data selection object that spans the entire area in which the data selection can appear, and set a line condition to test for the presence of the string "PREF." The data selection applies only to those lines in the data selection region that contain the string "PREF."

## PlanetPress Supports Global Variables in the Global Function Library Manager

To view global variables in the Global Function Library Manager:

- In the PlanetPress main menu, choose **Tools** | **Application** | **Global Function Library Manager**.

In the Global functions in the current document column, global variables are sorted first followed by global functions in alphabetical order. For each global variable, the Global Function Library Manager contains variable names, data types, and default values. In the Libraries column, libraries are listed in alphabetical order.

## Create a Global Condition

To add a global condition:

1. Choose **Home** | **Document** | **Condition**.
2. In the **Condition** properties dialog box, click **Identification** and enter a name for the global condition.
   In the **Condition** properties dialog box, click **Condition** and select the type of condition from the Condition type box. Options specific to that type of condition appear below the box. Adjust these options as necessary to define the condition.
   **Invert the condition**: Invert the value of the condition. If the condition evaluates to True, its final value is False. If the condition evaluates to False, its final value is True.
3. 

   *Text-Based*

   A text-based condition tests whether a string exists in a defined region of the data page, or compares a string with one found at a specific location in the data page.

   

   **String to find**: Enter the text string you want to test for on the data page. Note that if you leave this box empty, select an operator that requires a data selection, and use the Data Selector to create the data selection for that operator, PlanetPress Design enters the first line of the data selection you create in this box.

   **Operator**: Select a location or comparison operator. Location operators look for the specified string in a specific area of the data page. Comparison operators compare the specified string with the one that starts at a specific line and column in the data page, and are intended for comparing one number to another. Comparison operators work by comparing the ASCII values of the two strings, proceeding character by character until a comparison yields an answer for the selected operator. For example, consider the case where the operator is Greater than, the string to find is 1348 and the string on the data page is 1506. The comparison operator first compares "1" to "1"; since it cannot determine from this comparison which is greater, it must proceed to a comparison of the next two characters, "3" and "5". This comparison permits it to determine which is greater thus it does not proceed with further comparisons of the string. If you select a location operator, for all except On page, you must specify the appropriate data page coordinates in the boxes that appear. If you select a comparison operator, you must specify the line and column numbers that start the string you want to compare with the specified string.

   *On value change*

---

The result of the *On value change* condition is based not on the data value itself, but on whether the data **has changed from one data page to another**.

The *On value change* condition has two parameters:

- As with all other conditions, the first parameter is mandatory and it represents the data region to monitor. Depending on the emulation, the selection can be a block of data, a single field or an advanced PlanetPress Talk statement.
    - All Text/PDF emulations: Allow the selection of a block of data.
    - XML/Database: Allow the selection of a single field.
    - Any emulation: Allow any valid PlanetPress Talk expression.
- The second parameter allows the user to specify an existing condition or a PlanetPress Talk expression that controls whether the *On value change* condition should be evaluated or not. This option is useful, for instance, to monitor the contents of an Address block, but only when the string "Page 1 of" is found on the data page as well (otherwise, the address block would be considered as "changed" on datapage 2 if it is not repeated in the data stream).

*Page based*

A page-based global condition compares the page number of the current data page with a page number you define, or compares the position of the data page in a set sequence of data pages with a page position you define.

**Type of test**: Select the type of test you want the condition to perform. There are five types of tests: On page, Less than, Greater than, Less than or equal to, Greater than or equal to. For example, if you know the first data page is always a banner page, you might want to suppress printing it by creating a condition to test for a page number equal to 1 (an On page test) and associating that condition with the first page of the document. Or, you might want to insert a separator page before the first page of every sequence of 9 data pages. In this case, you create a condition to test for the data page being the first in a sequence of 9, and then create a separator page that prints only if the condition is True.

**Page number**: Enter the page number or use the spin buttons to adjust the value. This is either the page number of the data page, or the position of the data page within a set sequence of data pages. In the latter case, you select Of, and use Sequence to define the number of pages in the sequence.

**Of**: Select to indicate the test is on a sequence of data pages.

**Sequence**: Enter the number of data pages in the sequence, or use the spin buttons to adjust the value.

*Advanced*

An advanced global condition is a condition that you define using PlanetPress Talk expressions. You can include existing global conditions in an advanced condition.

**Advanced condition**: Enter the PlanetPress Talk expression that defines the test you want the condition to perform.

4. In the **Override mode** box, select how you want PlanetPress Design to evaluate the condition during the design phase.
   **Data**: Select to have the value of the condition depend on the contents of the sample data file. This in effect removes any override, and results in the condition behaving as it will at runtime.
   **True**: Select to have the value of the condition always equal to True.
   **False**: Select to have the value of the condition always equal to False.
5. If necessary, add PlanetPress Talk code to the object.
   - In the Condition properties dialog box, click **PlanetPress Talk before** to enter PlanetPress Talk code that you want the document to execute just before it evaluates the condition, or click **PlanetPress Talk after** to enter PlanetPress Talk code that you want the document to execute just after it evaluates the condition.
6. Click **OK**.

## View or Edit the Properties of a Global Condition

To view or edit properties using the Object Inspector:

- Select the global condition in the Structure area.

To view or edit properties using the Condition properties dialog box:

1. Double-click the global condition in the Structure area.
2. Enter any edits you want to make to the global condition.
3. Click **OK**.

## Apply or Remove a Global Condition

To apply a global condition to a page, object or group:

- In the **Structure** area, drag the global condition to the page, object, or group in the Structure area to which you want to apply it, and release.
  The condition indicator appears in the upper-left corner of the page, object, or group symbol in the Structure area. The color of the indicator reflects the current value of the condition (green indicates True; red indicates False).

  Pages
      Page1  (Page1)
          border  (Box1)
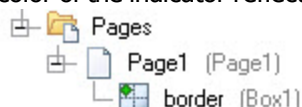
  Box object with a condition set on it

To remove an applied global condition:

- Select the page, object, or group from which you want to remove the applied global condition, and then, in the Object Inspector, locate the Condition property. Highlight the contents of the Condition property and press **BACKSPACE** to clear the property.

## Use a Global Condition as a Variable

To use a global condition as a variable:

- Place an ampersand before the first character of the condition name. For example, the following line of PlanetPress Talk code references the condition backordered as a global variable:
  show(if(&backordered,'Out of stock',''))
  When the condition evaluates to True, the line "Out of stock" prints; when it evaluates to False, nothing prints.

## Combine Global Conditions

You can use either the Condition properties dialog box or the Object Inspector to combine global conditions. When you combine conditions, it is important to understand the significance of the position in which a condition appears in the Structure area. The position of a condition in the Structure area both determines the conditions with which you can combine it, and limits how you can move it in the Structure area. More precisely, you can combine a condition only with conditions that appear above it, and you cannot move a condition above any condition it references.

You combine conditions by creating a valid Boolean expression that references conditions as variables. You construct the expression using Boolean logic, and the four Boolean operators: AND, OR, NOT, ().

| Operator | Description | Example |
|---|---|---|
| AND | Operates on two conditions. The result is True only if BOTH conditions are True. | &red AND &round |
| OR | Operates on two conditions. The result is True if EITHER condition is True or BOTH conditions are | &blue OR |

| Operator Description | | Example |
|---|---|---|
| | True. | &square |
| NOT | Operates on a single condition. The result is the inverse of the condition. | NOT (&over_50) |
| () | Assigns the highest precedence to its contents. See below for an explanation of precedence rules. | |

The expressions you create using these operators can be either simple or complex. For example:

&red OR &blue

&over_50 AND &under_300 AND NOT (&red) AND &round

(NOT (&red OR &blue)) AND (&big OR (&round AND &tin))

Precedence rules determine the order in which PlanetPress Design evaluates the expression. The final result of an evaluation can depend on the precedence rules. For example, consider that you want a line item on an invoice to print only if the part number is greater than 50, and the part is either red or round. You write:

&over_50 AND &red OR &round

Depending on the precedence rules, this could also mean print the line item only if both the part number is greater than 50 AND the part is red, or the part is round. In the latter case, line items for parts that are round print, regardless of whether their part number is greater than 50.

PlanetPress Design uses standard Boolean precedence rules:

| Operator Precedence | |
|---|---|
| () | 1 |
| AND | 2 |
| NOT | 2 |
| OR | 3 |

Thus, the contents of parentheses are always evaluated first. If two operators in an expression have the same precedence, they are evaluated left to right. Since parentheses have the highest precedence, you can override the precedence of a lower-precedence operator by enclosing it in parentheses. In the following example, although the OR operator has a lower precedence than the AND operator, the OR expression evaluates before the AND expression because parentheses enclose the OR expression.

(&before_2002 OR &under_50) AND &over_300

Thus, to return to the earlier example, to print the line item only if the part number is greater than 50, and the part is either red or round, you would rewrite the expression as:

&over_50 AND (&red OR &round)

To combine global conditions using the Condition properties dialog box:

1. In the Structure area, double-click the condition to open the **Condition** properties dialog box for the condition with which you want to combine one or more existing conditions:
2. In the **Condition** properties dialog box, click **Concatenation**.
   The name of the condition you are defining appears in the Completed condition box, and the names of the conditions you can combine with it appear in the Conditions available for concatenation box.
3. Do one or more of the following to combine conditions into a valid Boolean expression. As you work, PlanetPress Design reports any syntax errors in the Status area.
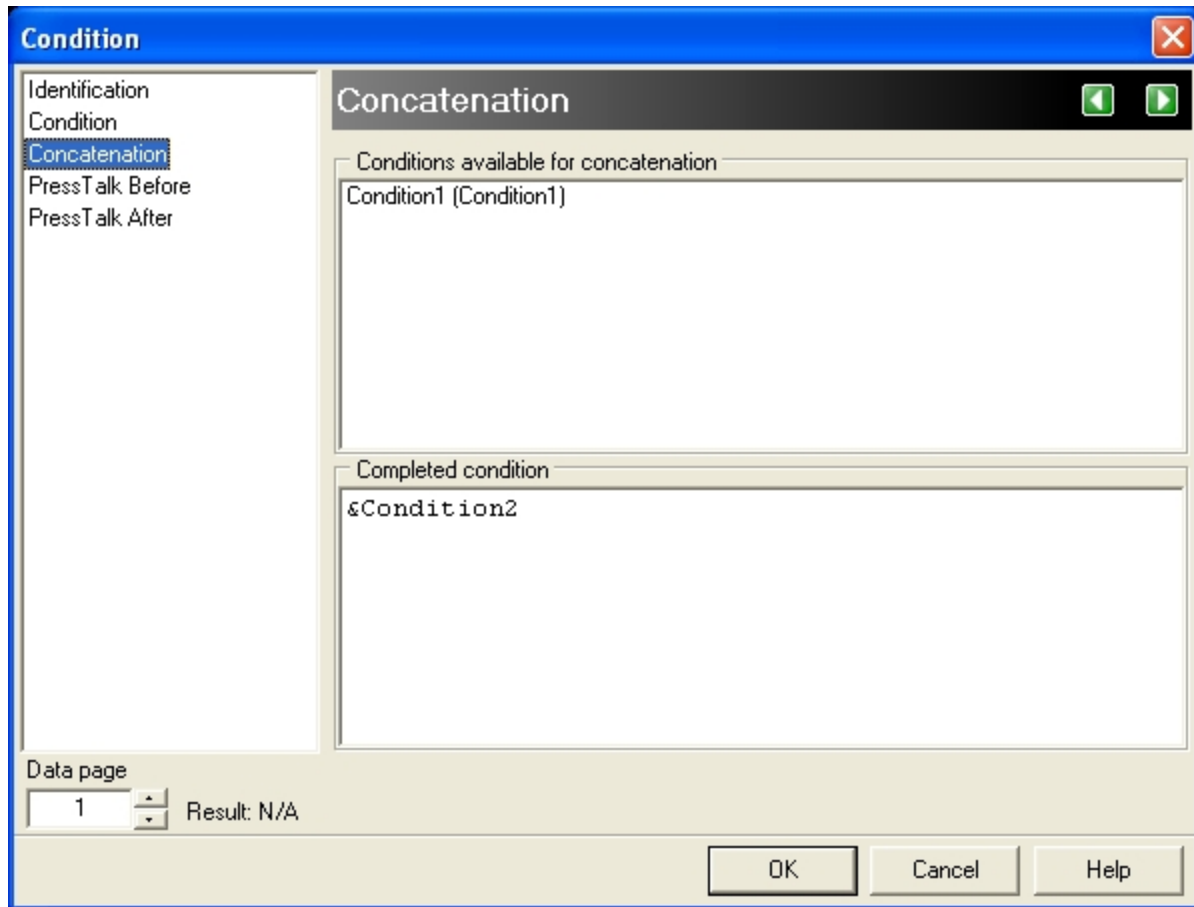
A. Status area

- Type directly in the Completed condition box. Names of conditions and Boolean operators are both case-insensitive. Note that if you want to use a NOT operator, you must enclose the condition on which you want it to operate in parenthesis. Thus you enter **NOT (&black)** rather than **NOT &black**.
- Drag and drop conditions from the Conditions available for concatenation box. Click on the name of the condition you want to add, drag it to the Completed condition box, release it at the position at which you want to insert it in the expression, and choose either **And** or **Or** from the menu PlanetPress Design displays. Legal positions for insertions are displayed in blue as you drag. Releasing over a condition name inserts the condition after that condition. Releasing over an opening parenthesis inserts the condition before that opening parenthesis. Releasing over a closing parenthesis inserts the condition after that closing parenthesis.
- Apply the NOT operator to a condition by right-clicking the condition and choosing **Insert "Not" Operator**. PlanetPress Design encloses the expression on which you clicked in parentheses, and precedes it by the NOT operator.
- Remove the NOT operator from a condition by right-clicking the condition and choosing **Remove "Not" Operator**.
- Delete a condition by right-clicking the condition and choosing **Delete Item from Condition**. Note that you cannot delete the condition you are currently defining.
- Delete any or all conditions by highlighting the section you want to delete and pressing **BACKSPACE**, or by backspacing through the section.

4. Click **OK**.

   If the Boolean expression contains a syntax error and you attempt to close the Condition properties dialog box, PlanetPress Design returns the focus to the Concatenation area of the Condition properties dialog box. You must fix the error to close the dialog box.

To combine global conditions using the Object Inspector:

1. In the Structure area, select the condition to which you want to add one or more existing conditions.
2. In the Object Inspector, locate the Combined condition property.
3. In the Combined condition property, enter a valid Boolean expression that references one or more of the conditions that appear above the currently selected condition in the Structure area, and describes how you want to combine those conditions. This expression must reference the condition whose Combined condition property you are editing.

## Override a Global Condition

To override a global condition:

1. In the Structure area, double-click the global condition to display the Condition properties dialog box for the condition whose value you want to override.
2. In the **Condition** properties dialog box, click **Condition** and select the override for the condition in the **Override mode** box.
   **Data**: Select to have the value of the condition depend on the contents of the sample data file. This in effect removes any override, and results in the condition behaving as it will at runtime.
   **True**: Select to have the value of the condition always equal to True. Note that this override is valid only during document design and has no effect at runtime.
   **False**: Select to have the value of the condition always equal to False.
3. Click **OK**.

## Delete a Global Condition

To delete one or more global conditions:

1. Select the global conditions you want to delete.
2. Choose **Home** | **Clipboard** | **Delete**.
   If no elements in the document reference any of the selected global conditions, PlanetPress Design performs the deletion.
   If any elements in the document reference any of the selected global conditions, PlanetPress Design prompts you to define how you want to handle the deletion of each referenced global condition.

To use the Condition Deletion dialog box:

1. Adjust the options to reflect how you want PlanetPress Design to handle the deletion request. The name of the global condition you selected for deletion appears in the title bar of the Condition Deletion dialog box, and the list of elements that reference it appear on the right of the dialog box.
   **Replace all references with**: Select to delete the global condition and to replace all references to it with a reference to another of the global conditions in the document.
   **Conditions available**: Select the global condition you want to use as the replacement reference. When you delete the global condition, PlanetPress Design replaces all references to the deleted global condition with a reference to the global condition you select here. You can use the Conditions button to create a new global condition to add to this box.
   **Conditions button**: Click to create a new global condition. PlanetPress Design creates the new global condition and selects it in the Conditions available box.
   **Delete all elements**: Select to delete the global condition and all document elements that reference it. All document elements that reference this global condition appear in the list on the right of the Condition Deletion dialog box.
2. Click **OK**.

## Create, Edit, or Delete Local Conditions

To create a local condition:

- Select the page, object, or group for which you want to create the condition. In the Object Inspector, locate the Condition property and enter the PlanetPress Talk expression that defines the local condition.
  The condition indicator appears on the upper-left corner of the page, object, or group symbol in the Structure area. The color of the indicator reflects the current value of the condition (green indicates True; red indicates False).

To edit a local condition:

- Select the page, object, or group whose local condition you want to edit. In the Object Inspector, locate the Condition property and edit the PlanetPress Talk expression that defines the local condition.

To delete a local condition:

- Select the page, object, or group whose local condition you want to delete. In the Object Inspector, locate the Condition property and either highlight its contents and press **BACKSPACE**, or select an existing global condition with which you want to replace the local condition.
  If the element no longer has a condition associated with it, PlanetPress Design removes the condition indicator from its symbol in the Structure area.

## Verify a Condition

To verify a global condition:

1. In the Conditions area of the Structure area, locate the global condition you want to verify.
2. Navigate through the sample data file.

To verify a local condition:

1. In the Structure area, locate the page, object, or group whose local condition you want to verify.
   The color of the condition indicator on the upper-left corner of the page, object, or group symbol in the Structure area reflects the current value of the condition. The indicator appears in green when the condition is True and in red when the condition is False.
2. Navigate through the sample data file.
   The color of the condition indicator updates to reflect the value of the local condition on the current data page.

## Add a Global Function

To add a global function:

1. Choose **Home** | **Document** | **Global Function**.
2. In the Structure area, double-click the new global function, or select it and press **ENTER**.
3. Replace the default name, **@GlobalFunction1**, with a name that reflects the purpose of the global function. Note that the initial @ character is part of the name of the function. If you omit it when you call the function from a PlanetPress Talk script, the script produces a syntax error.
4. Enter the code for the new global function.
   The function you define may or may not return a value. If it does return a value, you must assign the return value to the predefined variable **&result** on the last line of the function definition (the line that immediately precedes the **end-function()** line).
   You use the **@***name***()** command to subsequently call the function you define. Consult the **PlanetPress Talk Language Reference** for further help with this and any other command you want to use within the global function.
5. Click **OK**.

## View or Edit a Global Function

To view or edit a global function:

1. In the Structure area, double-click the global function.
2. Use the PlanetPress Talk Editor to edit the properties of the global function, if necessary.
3. Click **OK**.
   If you made a modification that may cause execution problems in the document, PlanetPress Design reports an error in the Messages area.
   If you made a modification that may cause execution problems in the document , PlanetPress Design requests confirmation before proceeding with the modification.

To change the name of a global function:

- Do any of the following:
    - *Rename the function in the Structure area*: In the Structure area, select the function, then press **F2** or click the name of the function a second time to select the name. Edit the name. You can press **ESC** at any point to abort the rename operation. When you have completed the modification, press **ENTER** or click outside the name.
    - *Rename the function in the Object Inspector*: In the Structure area, select the global function. The Object Inspector displays the properties of that function. In the Object Inspector, edit the Name property to reflect the new name, then press **ENTER** or click outside the name box.
    - *Rename the function in the PlanetPress Talk Editor*: In the Structure area, double-click the global function, or select it and press **ENTER**. The PlanetPress Talk Editor appears. Change the name of the function in the Editor then click **OK** to exit the Editor.

## Delete a Global Function

To delete one or more global functions:

1. Select the global functions you want to delete.
2. Choose **Home** | **Clipboard** | **Delete**.
   If no elements in the document reference any of the selected global functions, PlanetPress Design performs the deletion.
   If any elements in the document reference any of the selected global functions, PlanetPress Design prompts you to define how you want to handle the deletion of each of the referenced global functions.

To use the Global Function Deletion dialog box:

1. Adjust the options to reflect how you want PlanetPress Design to handle the deletion request. The name of the global function you selected for deletion appears in the title bar of the Global Function Deletion dialog box, and the list of elements that reference it appear on the right of the dialog box.
   **Replace all references with**: Select to delete the global function and to replace all references to it with a reference to another global function in the document.
   **Global functions available**: Select the global function you want to use as the replacement reference. When you delete the global function, PlanetPress Design replaces all references to the deleted global function with a reference to the global function you select here. You can use the Global Functions button to create a new global function to add to this box.
   **Global Functions button**: Click to create a new global function. PlanetPress Design creates the new global function, and selects it in the Global functions available box.
   **Delete all elements**: Select to delete the global function and all document elements that reference it. All document elements that reference this global function appear in the list on the right of the Global Function Deletion dialog box.
2. Click **OK**.

# 🔲 PlanetPress Talk

PlanetPress Talk is a complete scripting language that opens the door to more powerful and sophisticated documents.

> ⏩ Please note, as of October 20th 2011, the PlanetPress Talk Reference Guide has been merged with the PlanetPress Design User Guide. This chapter now contains the language reference and more details about our scripting language.

You can use PlanetPress Talk expressions in many places in the documents you create in PlanetPress Design. You can enter a PlanetPress Talk expression in any text box with a maroon label. You can also include complete PlanetPress Talk programs in your document. This provides the power and flexibility that make even the most complex documents possible in PlanetPress Design. You enter PlanetPress Talk programs when you create a user-defined emulation, and when you edit the PlanetPress Talk properties of an object, a page, a document, or a condition. When you edit the PlanetPress Talk properties, you can enter PlanetPress Talk code directly, or through the PlanetPress Talk Editor. Finally, you can also write your own PlanetPress Talk functions from PlanetPress Design, making it easy to extend the language to suit the requirements of your particular document or documents. When you write programs in PlanetPress Talk, you should work through the logic of the program you intend to create before you start coding. This ensures the program accomplishes what you intend it to accomplish, and makes the coding more straightforward. It can also result in insights into different ways of accomplishing the same task, or raise issues that were not immediately apparent. Note that PlanetPress Talk is case-insensitive.

> ⚠️ When a user-defined emulation is used with metadata, results and behavior are unknown and unsupported. For instance, refreshing the metadata file may cause the document to crash and/or corrupt. For this reason, it is strongly advised to create backup copies of your documents beforehand.

This chapter introduces the PlanetPress Talk scripting language, describes where and how you can enter PlanetPress Talk code in PlanetPress, explains its syntax and each of its elements, and offers some tips and tricks for writing PlanetPress Talk scripts.

## Variable PlanetPressTalk Properties

You can enter a PlanetPress Talk expression in any parse field. Parse fields can be identified by their maroon field label text. (note that maroon is a default setting that you can modify in the User Options dialog box).



A. Maroon label

The following procedure describes how to enter a PlanetPress Talk expression in such a text box. You can enter the complete PlanetPress Talk expression by hand, build it using items from the menu that appears when you right-click in the text box, or create it using a combination of the two. For help with PlanetPress Talk, consult the **PlanetPress Talk Language Reference**.

The menu that appears when you right-click in the text box contains some or all of the following items.

| Choose: | To: |
|---|---|
| Data Selection | Launch the Data Selector and create a data selection to insert in the PlanetPress Talk expression. |
| Get Data | Launch the Data Selector and retrieve characters to insert in the PlanetPress Talk expression. The string you retrieve is static. |

| Choose: | To: |
|---------|-----|
| Color | Launch the Color Picker and select a color to include in the expression. When you exit the Color Picker, the CMYK values for the color appear in the expression. |
| Style | Display a menu of the styles defined in the document and choose the name of a style to add to the expression. PlanetPress Design adds the style as a variable of type string. |
| Global Condition | Display a menu of the global conditions available in this document, and choose one of those functions to include in the expression. |
| Global Variable | Display a menu of the different types of global variables defined in the document (integer, Boolean, string,...), and choose a global variable to add to the expression. |
| System Variable | Display a menu of the different types of system variables (integer, Boolean, string,...), and choose a system variable to add to the expression. |
| Local Variable | Display a menu of the different types of local variables defined in the object (integer, Boolean, string,...), and choose a local variable to add to the expression. This menu item is available only when the object contains local variables. |
| Global Function | Display a menu of the global functions available in this document, and choose one of those functions to include in the expression. |
| Integer | Display a menu of the integer functions available in PlanetPress Talk, and choose one of those functions to include in the expression. |
| Measure | Display a menu of the measure functions available in PlanetPress Talk, and choose one of those functions to include in the expression. |
| String | Display a menu of the string functions available in PlanetPress Talk, and choose one of those functions to include in the expression. |
| Boolean | Display a menu of the Boolean functions available in PlanetPress Talk, and choose one of those functions to include in the expression. |
| Custom Data Selection CRLF | Add a new line to the Custom data selection. This menu item is available only when you are defining a Custom data selection. |

To enter a PlanetPress Talk expression in a text box:

1. Enter the equals sign (=) in the text box.
2. Construct the PlanetPress Talk expression using one or more of the following methods.
   - Type directly in the text box. Note that if you want to use an apostrophe (') or a backslash (\) in a string, you must precede it by a backslash. For example: **=if(@(1,1,4) = 'ABCD', 'Don\'t forget your alphabet' , 'Now you know your A:\\, B:\\, C:\\!')**
   - Click in the text box at the position at which you want to insert a variable, function, or data selection, then right-click and use the menu to choose the item you want to insert. PlanetPress Design inserts the chosen item in the text box. Note that if you did not enter an equals sign (=) as the first character of the text box, and this is the first item you are adding from the right-click menu, PlanetPress Design automatically inserts an equals sign in front of the item.
   - Highlight a section of the expression entered to date, and then right-click and choose a function from the menu. The function appears in the text box, with the highlighted section of the expression as its argument. In this case, you must be careful that the content of the highlighted section is in fact a valid argument for that function. If necessary, consult the **PlanetPress Talk Language Reference** for the function in question. Note that if you use a data selection to retrieve Arabic text, you must nest your data selection in a maputf8 () command. For example: maputf8( @(21,10,28) )
   - PlanetPress Design reports any errors in the expression in the Status area of the dialog box that contains it. If you attempt to close the dialog box that contains it, PlanetPress Design returns the focus to the text box. You must correct the error before you can close the dialog box.

# The PlanetPress Talk Editor

The PlanetPress Talk Editor is a feature-rich editor for entering PlanetPress Talk code. You can write, run, and debug code, as well as import and export code, all from the PlanetPress Talk Editor.

There are four main areas in the PlanetPress Talk Editor: the Commands area, the Code area, the status bar and the Spy list.
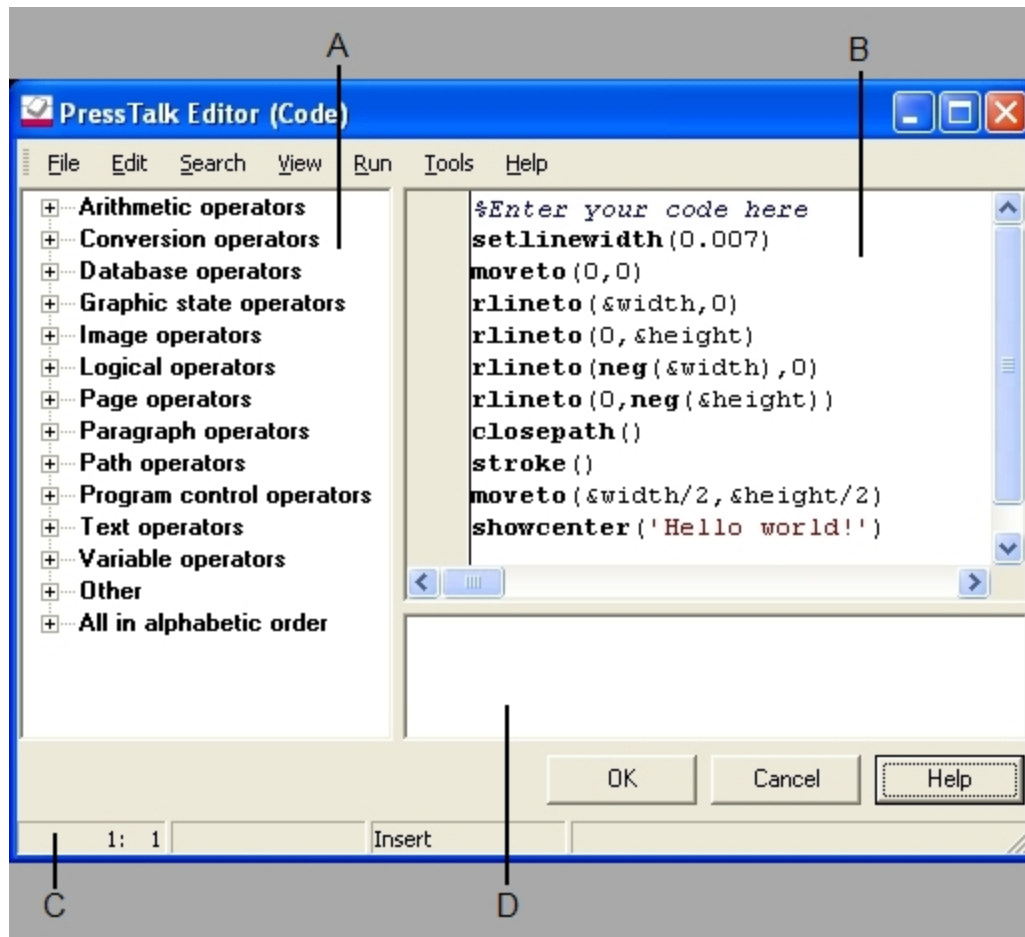


A. Commands area B. Code area C. Status bar D. Spy list

### Commands Area

The Commands area displays a list of available PlanetPress Talk commands, organized into groups. It serves both as a quick reference as well as an easy way to enter code either by dragging and dropping, or double-clicking a command.

### Code Area

The Code area displays the PlanetPress Talk code entered to date. If the Code area is currently displaying the contents of a file, the title bar of the window displays the path of that file.

### PlanetPress Talk Editor Status Bar

The status bar displays information about the line of code that is currently under the pointer, and the current execution status of the program.

The Editor also uses the status bar to display any conversion or run errors it encounters in the Code area when you attempt to exit the Editor.

---

**Spy List**

The Spy list displays any and all spies set in the object or page from within which you are using the Editor.

## PlanetPress Talk Editor Features

What are the features of the PlanetPress Talk Editor?

The PlanetPress Talk Editor provides the following features for entering, editing, and executing code..

| Feature: | Description: |
| --- | --- |
| Easily customized appearance and behavior | You can set a number of options that affect the appearance and behavior of the Editor. |
| Flexible code entry | The Editor provides several ways to enter code, and it includes a number of features to enhance the speed and efficiency of code entry. You can enter code in the PlanetPress Talk Editor by dragging and dropping commands from the Commands area, double-clicking commands in the Commands area, or by typing commands directly in the Code area as you would in a text editor. When you type commands directly in the Code area, the command name completion can enhance the speed and accuracy of command entry. The Editor also provides a number of ways to enter arguments into a command, and makes it easy to quickly display a list of the arguments a command requires. |
| Syntax highlighting | You can use syntax highlighting to make the different elements of the code in the Code area easier to distinguish visually. For example, you might choose to display all command names in a bold font with the color set to black, and all comments in a regular font with the color set to gray. |
| Convenient navigation | You can navigate in the Code area as you would in a text editor, using the arrow keys, and the scrollbars that appear on the bottom and on the right of the Code area when the code extends beyond the size of the Code area. The Editor also provides bookmarks to make it easy to jump to a specific point in the code, and a feature to make it easy to jump to a specific line number. |
| Selection and manipulation of a region of code | You can select a region of code and then cut, copy, paste, or delete that region. |
| Undo command | You can undo either a single editing command, or a group of editing commands. You can also repeat the undo to continue undoing previously entered commands or groups of commands. |
| Search, and search and replace | You can perform search as well as search and replace operations in the Editor. |
| Program execution control | You can run a program through to completion, advance execution only to the pointer position, or advance execution one line of code at a time. |
| Result preview | You can use the Object Preview to preview the result of execution. |
| Print the program | You can print the code that appears in the Code area and set a number of options that determine how the code prints. |
| Import and export programs | The Editor can import and export programs. Thus you can write code in an external text editor and import it, or save the code you write in the Editor out to a file. |
| Help ref- | The Commands area provides a quick reference for the PlanetPress Talk language, and the command name |

| Feature: | Description: |
|---|---|
| erences | completion feature and the argument list display feature provide quick references for specific commands. |

## Debugging Features

What debugging features are available in the PlanetPress Talk Editor?

The Editor provides four key features that can help you debug problem programs: execution control, breakpoints, spies, and expression evaluation. They are not available when you launch the Editor from the PlanetPress Talk properties of a document or condition, when you are editing the PlanetPress Talk before/after paragraph properties of a text object, or when you are creating or editing a global function.

1. Execution control
   The different execution methods, in particular the Run to cursor and the Step by step methods of executing a program, can be useful when you are debugging a program. You can exit program execution at any point.
2. Breakpoints
   A breakpoint stops the execution of the program at a specific line in the code. You can set the breakpoint to take effect whenever the program arrives at that line, or only when the program arrives at that line and a specified condition is True. You can set as many breakpoints as you need.
3. Spies
   A spy is a PlanetPress Talk expression that references one or more variables or expressions in your PlanetPress Talk code and evaluates to a string, integer, measure, currency or Boolean value. The expression you use for the spy may or may not be part of the existing code.
   Spies are useful for monitoring variables or expressions in your PlanetPress Talk code. The spies you create appear in the Spy list. When you execute the program, they update to reflect the changing values of the variables they reference. Spies can reference global, system, and local variables. Note, however, that the only local variables you can reference when you create a spy are those defined in the program loaded in the Editor at that time.
   There is one Spy list per individual object or page.
4. Expression evaluation
   You can evaluate any PlanetPress Talk expression that evaluates to a string, integer, measure, currency or Boolean value, whether or not that expression appears in the Code area of the Editor.
   The value of any variable the expression references is its value at the point in program execution at which you perform the evaluation. You can reference system variables, global variables, as well as any local variables that appear in the program currently loaded in the Editor.

### Other Features Useful in a Debugging Context

In addition to the four key debugging features, comments, debugging code, and the Object Preview can also be useful during debugging.

Comments can be very handy for enabling or disabling sections of code.

Debugging code is code you embed in the program for debugging purposes only; during execution, it prints information that helps you debug the program. The Messages area of the Object Preview displays the debugging code. Once you solve the problem, you remove the debugging code.

The Object Preview is useful both for viewing the result of code execution up to the current stop point in the code, and for examining error messages and debugging strings.

## Code Execution in the Editor

When are the execution and debugging features of the Editor available? When I execute a program in the Editor, what other code executes along with it?

The execution and debugging features of the Editor are available only when you launch the Editor from the PlanetPress Talk properties of an object or page, or from the Data Selector. They are not available when you launch the Editor from the PlanetPress Talk properties of a document or condition, when you are editing the PlanetPress Talk before/after paragraph properties of a text object, or when you are creating or editing a global function.

The execution feature lets you preview the result of the PlanetPress Talk code using the Preview window. When you execute a program, the Preview window displays the result of the execution up to the current stop point in the code.
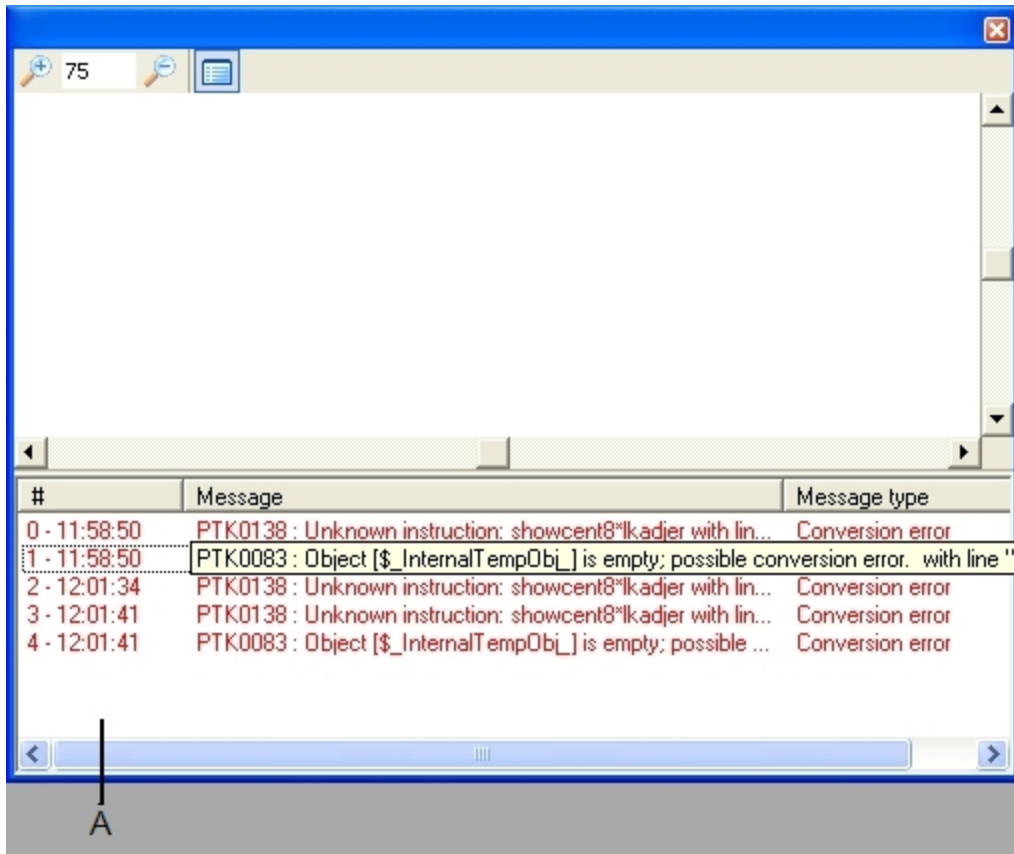


A. Messages area

It is important to understand that when you execute a program in the Editor, only in the case of a user-defined emulation is that program the only code that executes. When you execute a program from the PlanetPress Talk properties of an object or page, the entire code for that object or page executes. This ensures the execution respects any dependencies one section of code has on an earlier section of code.

Think of the entire code of an element as being composed of three parts:

1. PlanetPress Talk before code
   The code you enter in the PlanetPress Talk before property of the element.
2. Element code
   This includes the code for all the other properties of the element. For example, in an object it includes the code for the Height, Width, Left, Right, Style, Condition, and all the other properties of the object In a PlanetPress Talk object, it includes the code you enter in the PlanetPress Talk code property. In a page, it includes the code for all the properties of the page as well as the code for all the objects on the page.
3. PlanetPress Talk after code
   The code you enter in the PlanetPress Talk after property of the element.

When you execute a program in the Editor, the Editor executes all three parts of the element, in sequence.

You can stop execution, and use the debugging features only in the program you are currently editing in the Editor.

## Enter a New Program in the Editor

The way you create a new program in the PlanetPress Talk Editor depends on whether the Code area of the Editor is currently displaying the contents of a file.

To create a new program when no file is loaded in the Code area:

- Enter the new program.

To create a new program when a file is loaded in the Code area:

1. If necessary, save the program currently loaded in the Code area.
2. Press **CTRL**+**A** followed by **BACKSPACE** to clear the Code area.
3. Enter the new program.

## Import or Export a Program

The file name extension **.ptk** indicates the file contains PlanetPress Talk code.

If a file is currently loaded in the Editor, you can also use the export procedure to save the contents of the Code area to a different file.

To import an existing PlanetPress Talk program:

1. Choose **File** | **Import.**
   If you currently have an unsaved program in the Code area, the Editor prompts for confirmation to save it. Once you respond to the prompt, or if no response was necessary, the Editor displays the Open dialog box.
2. In the **Open** dialog box, navigate to the PlanetPress Talk program you want to open and click **Open**. The **.ptk** file name extension indicates the file contains PlanetPress Talk code.
   The Editor loads the contents of the file into the Code area. The Editor does not verify that the file contains valid PlanetPress Talk code when it loads it. You can verify the code by executing it in the Editor and/or examining the Messages area of the Object Preview.

To export a PlanetPress Talk program:

1. In the PlanetPress Talk Editor, choose **File** | **Export As**.
2. In the **Save As** dialog box, navigate to the folder in which you want to save the program and enter the file name for the program.
3. Click **Save**.

## Save a Program

If no file is currently loaded in the Editor, and you want to save the contents of the Code area, you use the export procedure. You also use the export procedure if a file is currently loaded in the Editor and you want to save the contents of the Code area to a different file.

To save a program:

- In the PlanetPress Talk Editor, choose **File** | **Export**.

## Print a Program

To print the program:

1. If you want to print only a region of code, select that region.
2. In the PlanetPress Talk Editor, choose **File** | **Print.**
3. Adjust the print options if necessary.
   File to print
   File name: Displays the name of the file currently loaded in the Code area.
   Print selected block: Select to print only the selected region of the program. Clear to print the complete program. This option is available only if you have code selected in the Code area.
   Options
   Line numbers: Select to print code line numbers or clear to exclude them. The line numbers appear in the Editor alongside the lines of code if the User Options are set to display them. Line numbers can provide a useful reference, particularly when debugging a program.
   Word wrap: Select to have any line of code that extends beyond the page wrap to the next line. The wrap occurs on a word break. Clear to prevent the word wrap, and thus force every line of code to occupy one line on the printed page.
   Syntax print: Select to preserve the style attribute settings (Bold, Italic, Underline) set in the User Options, in the printed output. Clear to remove these settings from the printed output. Note that Syntax print does not work if you select Line numbers.
   Left indent: Enter the left margin to use for the code on the printed page, in points. If you also select Line numbers, this is the distance between the line numbers and the code.
4. If necessary, click **Setup** to display the Printer Setup dialog box and adjust the printer options.
5. Click **OK**.

## Exit the PlanetPress Talk Editor

To exit the PlanetPress Talk Editor:

- In the PlanetPress Talk Editor, click **OK**.

The Editor verifies the code in the Code area. If it encounters any conversion or run errors, it displays the offending error in the status bar and does not exit.

If the code is error-free, the Editor exits. If a file is currently loaded in the Code area, and you made modifications to it, the Editor prompts for confirmation to save it before exiting.

## Show or Hide the Commands Area or Spy List

To show or hide the Commands area:

- Choose **View** | **Commands**.

To hide the Commands area:

- Choose **Close**.

To show or hide the Spy list:

- Choose **View** | **Spy List**.

## Adjust the Sizes of the Commands Area, Code Area and Spy List

To adjust the relative sizes of the Commands area, Code area, and Spy list:

1. Position the pointer over the right edge of the Commands area, or the top edge of the Spy list to display the double-headed arrow pointer.
2. Click and drag to adjust the relative sizes of the Commands area, Code area, and/or Spy list.

## Expand or Collapse Command Groups
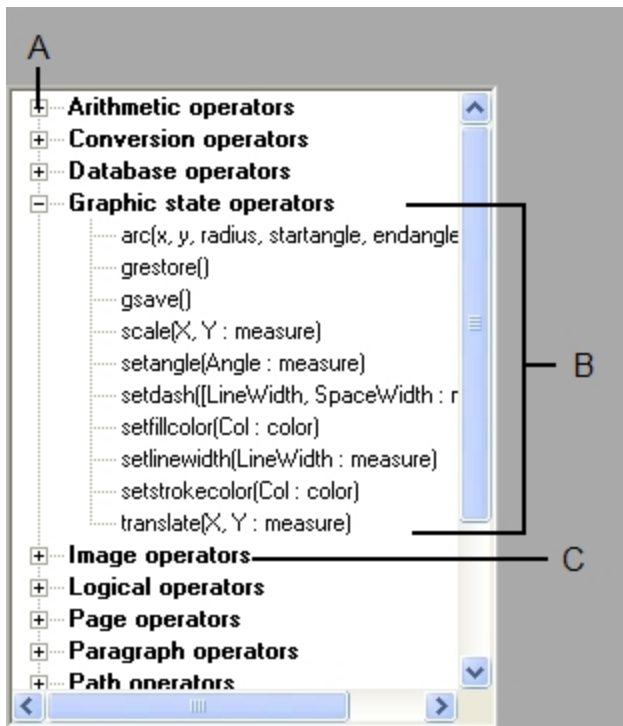


A. Expand/collapse box B. Expanded group C. Collapsed group

To expand or collapse a single command group:

- Click the expand/collapse box to the left of the group.

To expand or collapse all command groups:

- Right-click in the Commands area and choose **Expand All** or **Collapse All.**

## Enter Commands in the Editor

To make code entry faster and more adaptable to different work styles, the Editor provides a number of ways to enter code.

When you are entering commands, remember that PlanetPress Talk is case-insensitive.

To enter commands:

- In the Code area, click and type the command. When you type commands, you may find the command name completion feature useful.
  The exact insertion point depends on the type of command. If the command is a function, the Editor inserts it at the exact point indicated by the click in the Code area. If the command is a procedure, and the point indicated by the click in the Code area is anywhere **except** the first character of the line, the Editor inserts it after the line on which you clicked; if you clicked on the first character of a line, it inserts it before that line.

To enter arguments:

- Use any of the following.
    - Argument**:** You can enter any argument by typing it.
    - Data selection: To enter a data selection as an argument using the Data Selector, right-click in the Code area at
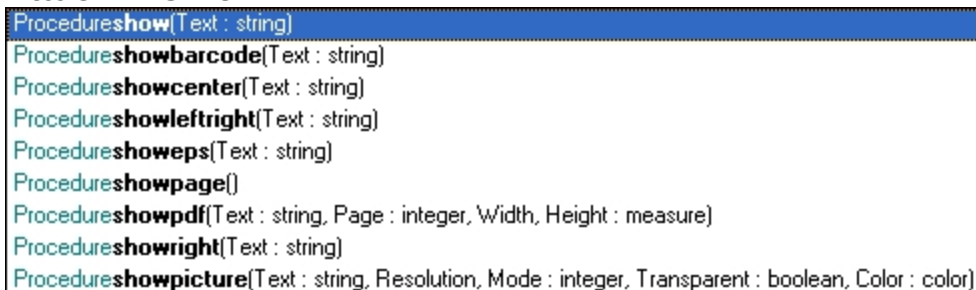
the point at which you want to enter the data selection argument, and choose **Data Selection**.

- Color specification: To enter a color specification as an argument using the Color Picker, right-click in the Code area at the point at which you want to enter the color specification, and choose **Color**.
- Style: To enter a style as an argument, right-click in the Code area at the point at which you want to enter the style, choose **Style**, and then choose the style you want to reference.
- Image resource: To enter an image resource as an argument, right-click in the Code area at the point at which you want to enter the argument, choose **Image Resource**, and then choose the image resource you want to reference.
- Condition: To enter a condition as an argument, right-click in the Code area at the point at which you want to enter the condition, choose **Global Condition**, and then choose the condition you want to reference.
- System variable (from the Commands area): To enter a system variable as an argument using the Commands area, click in the code at the point at which you want to enter the system variable. Then, locate the variable in the Commands area (either in the All in alphabetical order group or the Other group), and double-click it as you would to enter a command.
- System variable (from the Code area): To enter a system variable as an argument from the Code area, right-click in the Code area at the point at which you want to insert the system variable, choose **System Variable**, then choose the variable type, and finally choose the system variable.
- Global variable: To enter a global variable as an argument using the list of global variables, right-click in the Code area at the point at which you want to insert the global variable, choose **Global Variable**, then choose the variable type, and finally choose the global variable.
- Global functions: To enter a global function as an argument using the list of global functions, right-click in the Code area at the point at which you want to insert the global function, choose **Global Function**, then choose the global function.
- Any type of variable (global, local, or system) or a global function: To enter a variable of any type or a global function as an argument, in the Code area, click in the code at the point in the argument list at which you want to insert the variable or global function and press **CTRL**+**SPACE**. A list of variables and commands appears. In the list, double-click the variable or global function you want to insert, or if it is already highlighted, press **ENTER**, to insert it into your code. You can abort the operation at any point by pressing **ESCAPE**. Note that the position of the pointer when you press **CTRL**+**SPACE** is important. If you press **CTRL**+**SPACE** in the middle of an existing string, the Editor performs a variable or command name completion instead of an argument insertion.
- Commands: To enter a command as an argument to another command, use any of the techniques described in. You can also use the technique described above for inserting any type of variable from the Code area.

## Use Command and Variable Name Completion

To use command or variable name completion:

1. Click at the point at which you want to insert the command or variable, and type the first character or characters of its name.
2. Press **CTRL**+**SPACE**.

# Dynamic Images

Dynamic images are image resources that refer to more than one image in a dynamic fashion. These images can be located on local or remote drives, or directly on the printer, depending on the run mode that is used.

This section explains how dynamic images work.

## PlanetPress Talk Expressions for Dynamic Images that Reference Image Resources

What do I need to know to construct a PlanetPress Talk expression for a dynamic image that references image resources? What are some examples of these expressions?

You must be able to construct a PlanetPress Talk expression that resolves to the name of an image resource. In the simplest case, the name of the image resource exists in the input data, and the PlanetPress Talk expression you construct contains a single data selection command. For example, if columns 1 through 15 of the first line of each data page contain the name of the image resource, you would enter the following PlanetPress Talk expression in the Image box.

=@(1,1,15)

If the input data does not contain the complete name of the image resource, in a single location, but you can construct the name using more than one data selection, you construct an expression that contains the necessary number of data selections, concatenated to form a single string.

If the input data contains only part of the name of the image resources, and the missing part is the same for all image resources, you can construct an expression that takes the part of the name in the input data and concatenates the missing part. For example, imagine a situation in which the name of each of the image resources begins with a country code that is the same for all of the image resources, and that the country code is missing in the names in the input data. You could construct a PlanetPress Talk expression that takes the name of image from the input data and adds the country code as a prefix. So the final expression might be

='c10' + @(2,1,22)

where c10 is the country code, and the name of the image resource appears in columns 1 through 22 of the second line of each data page.

If you cannot construct an expression that resolves to the names of the image resources you want the dynamic image to reference, two alternatives are to rename the image resources in the document so they match the names in the input data, or to adjust the names of the images in the input data. In the first case, the number of image resources the dynamic image references may determine whether this is a realistic solution. In the second case the adjustment may require altering the source from which you generate the input data, and may or may not be a feasible solution.

## PlanetPress Talk Expressions for Dynamic Images that Reference External Images

What do I need to take into account when I construct a PlanetPress Talk expression for a dynamic image that references external images? What are some examples of these expressions?

The PlanetPress Talk expression you define references a selection of input data so that as the input data varies, so does the image that appears in the picture object.

When the document executes, it must retrieve the external images associated with each dynamic image in the document. These images must thus be accessible in the execution environment of the document.

When you construct the PlanetPress Talk expression that resolves to a pathname for an image, you therefore need to consider:

1. The physical location of the images you want to associate with the dynamic image.
2. Whether the document executes host-based or printer-based.
3. The image format or formats in which the images exist.

You may have to adjust one or more of these prior to document execution to make the images available to the document at runtime, in a format it can handle.

## Physical Location of Dynamic Images

The pathname you define for an external image must resolve to one of three physical locations: the printer on which the document executes, the local PlanetPress Design Suite virtual disk on the host on which the document executes, or any folder on the host on which the document executes. The local PlanetPress Design Suite virtual disk is a virtual disk PlanetPress Design creates and maintains on the hard disk of the computer on which it is running.

## Host-Based or Printer-Based Document Execution

When you preview a document in PlanetPress Design, or set up a PlanetPress Suite Workflow Tool process to execute a document, you specify whether you want the document to run host-based, or printer-based. The physical dynamic image locations available to the document at runtime depend on whether it executes host-based or printer-based.

A document that executes printer-based executes *completely* on that printer. In this case the document exists and executes on the printer, and the only dynamic image location available to it at runtime is the printer.

A document that executes host-based, executes either partially or completely on the host. If the document does not produce printer output, it executes entirely on the host. If it produces printer output, the host partially executes the document and then sends the result to a printer; the document completes execution on the printer. The partial execution that takes place on the host includes the retrieval of dynamic images. A document that executes host-based must exist on the host; you cannot run a document that is installed on a printer, host-based. If a document executes host-based, and it contains dynamic images that reference external images, those images can exist on the local PlanetPress Design Suite virtual disk on the host on which the document is executing, or in any folder on the host on which the document is executing. If the document completes execution on a printer, the external images can also exist on the printer. In this case if the document cannot find an external image on the host, it looks for it on the printer.

There are only two ways to execute a document host-based. The first is to perform a preview in PlanetPress Design, and select Run locally in the options for that preview. The second is to install the document in PlanetPress Suite Workflow Tool and set up a PlanetPress Suite Workflow Tool process that executes the document host-based. See the *PlanetPress Suite Workflow Tool User Guide* for help understanding and creating PlanetPress Suite Workflow Tool processes.

If you select host-based when you perform either a screen or hard copy preview in PlanetPress Design, the images can exist either on the local PlanetPress Design Suite virtual disk or in a folder on the host. The pathname in the picture object determines which of the two locations the document uses.

If you select host-based when you set up a PlanetPress Suite Workflow Tool process to execute a document, the document can access both the local PlanetPress Design Suite virtual disk as well as any folder on the computer on which it executes. If it completes execution on a printer, it can also access the hard disk of the printer.

## Image Format

Images must be in a format the document can accept at runtime.

Dynamic images that are resident on a printer or on the local PlanetPress Design Suite virtual disk must be in PostScript format. The only way to make images available in these locations, in a format the document can accept, is to use the Image Downloader to copy the images to those locations. The Image Downloader converts any images it copies to PostScript format.

Dynamic images that exist on a folder on the host must be in one or more of the supported image formats.

## Summary of Execution Environments

The following summarizes the different execution environments, and where the external images must reside in each case. Recall that external images on either a printer or the local PlanetPress Design Suite virtual disk must be in the PostScript format generated by the Image Downloader. External images in a folder must be in one or more of the supported formats.

| To: | Images must reside on the: |
|---|---|
| View dynamic images during document design | Host on which you are running PlanetPress Design, either in a folder or on the local PlanetPress Design Suite virtual disk. |
| Perform a screen preview | Host on which you are running PlanetPress Design.<br>If you choose a *host-based* screen preview, the images can be either in a folder or on the local PlanetPress Design Suite virtual disk.<br>If you choose a *printer-based* screen preview, the images must be on the local PlanetPress Design Suite virtual disk. |
| Perform a hard copy preview | Host on which you are running PlanetPress Design**or** printer on which you perform the preview.<br>If you choose a *host-based* hard copy preview, the images can be in a folder on the host, on the local PlanetPress Design Suite virtual disk, or on the printer.<br>If you choose a *printer-based* hard copy preview, the images must be on the printer. |
| Execute a document installed on a printer | Printer on which you execute the document. |
| Execute a document on a printer from PlanetPress Suite Workflow Tool | Host on which you are running PlanetPress Suite Workflow Tool **or** the printer on which you execute the document.<br>If you configure the PlanetPress Suite Workflow Toolprocess to execute the document *host-based*, the images can be in a folder on the host running PlanetPress Suite Workflow Tool, on the local PlanetPress Design Suite virtual disk, or on the printer.<br>If you configure the PlanetPress Suite Workflow Tool process to execute the document *printer-based*, the images must reside on the printer. |
| Execute a document in PlanetPress Image or PlanetPress Fax | Host on which you are running PlanetPress Suite Workflow Tool, or the host on which you are running a remote instance of PlanetPress Image or PlanetPress Fax.<br>If you configure the PlanetPress Image or PlanetPress Fax output to execute the document *host-based*, the images can be in a folder on the host running PlanetPress Suite Workflow Tool, on the local PlanetPress Design Suite virtual disk (if there is a PlanetPress Design installation on the host on which you are running PlanetPress Suite Workflow Tool), or on the PlanetPress Design Suite virtual disk of the remote host on which PlanetPress Image or PlanetPress Fax is running.<br>If you do not configure the PlanetPress Image or PlanetPress Fax output to execute the document *host-* |

| To: | Images must reside on the: |
|---|---|
| | *based*, the images must be either in a folder on the host running PlanetPress Suite Workflow Tool or on the local PlanetPress Design Suite virtual disk. |

## Examples of PlanetPress Talk Expressions that Resolve to Pathnames

The following are examples of PlanetPress Talk expressions that resolve to pathnames. The expression can thus include global and system variables.If the images exist on both a host and a printer, you can define an expression that specifies when to use the images on the host, and when to use those in the printer. For an example of this, see the last entry below.

| Expression: | Specifies: |
|---|---|
| **='map'** | An image resource internal to the document. If no image resource exists by that name, the document looks for an image in the root folder of the default disk of the printer or an image in the local PlanetPress Design Suite virtual disk. If the image file is on either the printer disk or the local PlanetPress Design Suite virtual disk, it is in PostScript format. |
| **=trim( @( 3,1,20 ) ) + '.png'** | An image in the document, or a host-resident or printer-resident image. The document first looks for an image internally. If it does not find it internally it looks for an image in the root folder of the default disk of the printer or an image in the local PlanetPress Design Suite virtual disk. The document constructs the specific path of the image by concatenating the data selection that specifies the name of an image file, and the file name extension **.png**. If the image file is on either the printer or the PlanetPress Design Suite virtual disk, it is in PostScript format. |
| **='S:\\parts\\wingnut.tif'** | A host-resident image file. The document expects to find the image **wingnut.tif** in the **parts** folder of the S drive of the host on which it is executing. The image is TIFF format, one of the supported image formats. |
| **='%disk1%parts\\bolt.eps'** or **='%disk1%parts/bolt.eps'** | A printer-resident image file. The document expects to find the image **bolt.eps** in the **parts** folder of the printer disk named **disk1**. The image file in this case is in PostScript format. The *.eps* file name extension reflects the Naming convention option you set in the Image Downloader. |
| **='S:\\people\\' + trim( @( 1,1,24 ) )** | A host-resident dynamic image file. The document expects to find the image in the **people** folder of the S drive of the host on which it is executing. The document constructs the specific path of the image by concatenating the path of a folder, with a data selection that specifies the name of an image file. The images in the folder on the host are in one or more of the supported image formats. |
| **='%disk2%' + trim( @( 1,1,20 ) ) + '.tif'** | A printer-resident image file. The document expects to find the image in the root of the printer disk named **disk2**. The document constructs the specific path of the image by concatenating the name of the disk, the data selection that specifies the name of an image file, and the file name extension **.tif**. The image file is in PostScript format. The *.tif* file name extension reflects the Naming convention option you set in the Image Downloader. |
| **=if ((&printermode=1), 'C:\\proofs\\' + @(1,30,12),'finals\\' + @(1,30,12))** | One set of image files for a screen preview (printermode=1) and another for when the document executes on a printer. In the case of a screen preview, it uses images found in the **proofs** folder on the C drive of the host on which PlanetPress Design is executing. The images in the folder on the host are in one or more of the supported image formats. If the document is not executing for a screen preview, it looks for an image in the **finals** folder of the default disk of the printer or an image in the **finals** folder of the local PlanetPress Design Suite virtual disk. If the image file is on either of these locations, it is in PostScript format. |

# Image Name and Pathname Resolution in Dynamic Images

How does the document resolve image names and pathnames in dynamic images?

When the document executes, it evaluates the PlanetPress Talk expression. Once it has the name or pathname, it proceeds as follows until it encounters an image file that matches that name or pathname.

1. It compares the result to the image resources in the document. Thus if you entered the PlanetPress Talk expression ='house.jpg' for the dynamic image, the document searches its image resources for one named **house.jpg**.
2. It looks for an image file at that pathname on the printer disk, or on the local PlanetPress Design Suite virtual disk.
3. It looks for an image file at that pathname on the host on which it is executing.
4. It looks for an image file at that pathname on the printer on which it is executing.
5. If it cannot find an image in any of these locations, it behaves as specified in the On error option of the picture object for that dynamic image.

The sequence the document follows to retrieve an image is important when an image by the same name exists in more than one of the locations the document examines.

## Custom Data Selections

To create a Custom data selection:

1. Open the properties dialog box for the object for which you want to create a data selection.
2. In the properties dialog box, click **Data**.
3. Select **Custom data selection**.

Create the Custom data selection using PlanetPress Talk. You can enter the selection on a single line, or divide it over several lines, depending on the selection you are trying to create and how you define it. If you create a complex data selection, breaking it up over several lines can make it more readable. When you break a selection over several lines, for each line, you determine whether to terminate it with a carriage return followed by a line feed (CRLF).

Both the *String to display* lines and the *CRLF* lines accept PlanetPress Talk expressions.

Navigating, inserting, deleting and repositioning lines:

- **To navigate within the Custom data selection**: Use either the mouse or the **ARROW** keys.
- **To Add a new line**, hit Enter on your keyboard to add a new line after the current one, or right-click on any line and select Custom Data Selection CRLF in the menu to insert the line at the mouse cursor position.
- **To delete a line**: Highlight the contents of the line and press **BACKSPACE** twice.
- **To reposition a line**: Click the line you want to reposition and use the arrow buttons on the right of the Custom data selection area to move the line up or down in the String to display lines.

Once you are done with your custom data selection, Click **OK** to close the object properties and save your changes.

If necessary, step through the sample data file to verify the data selection is accurate for the record sets in the sample data file.

## PlanetPress Talk Before and After

The properties for all documents, pages and objects contain two sections named "PlanetPress Talk Before" and "PlanetPress Talk After", which are used to execute PlanetPress Talk code before or after the object is displayed on the page.

Specific variables are available within these boxes that can be used to obtain data about the object and to modify it, for example for height, width and position. For more information, see the PlanetPress Talk Reference Guide.

> Any object that has PlanetPress Talk code entered in either the "Before" or "After" section will be identified in the Document Structure area in a maroon color instead of the normal black text. This color may be different and is configurable in the "Color Preferences" (page 75). It is the same color as PlanetPress Talk-enabled boxes.

### Create a User-Defined Emulation

> When a user-defined emulation is used with metadata, results and behavior are unknown and unsupported. For instance, refreshing the metadata file may cause the document to crash and/or corrupt. For this reason, it is strongly advised to create backup copies of your documents beforehand.

You use PlanetPress Talk to write the code that determines how the document treats each line and where it places it in the data page buffer. When you write a user-defined emulation, you must define any offset of the first data page in the data stream, the data page size, the number of data pages in the data page buffer, and any conditions that signal the end of a data page.

To set up a user-defined emulation:

1. If you have not done so already, associate a sample data file with the document.
2. Choose **Tools** | **Open Active Data**.
3. In the Data Selector, in the Emulation box, select **User-defined**.
4. In the Emulation options, click **Use PlanetPress Talk Editor**.
   The Code area of the Editor contains the code for a line printer emulation. The Spy list in the Editor contains spies for three system variables important in user-defined emulations:

   | Variable: | Type: | Contains: |
   |---|---|---|
   | **&str** | string | The current line of input data. |
   | **&current.line** | integer | The line number of the current line of data in the data page. This system variable is read-write and can be modified using the **set()** command. |
   | **&current.lpp** | integer | The number of lines per data page in the current data page. |

5. Enter the emulation code in the Code area of the PlanetPress Talk Editor. You can use the line printer emulation code as a starting point or delete it (highlight it and press **DELETE** or **BACKSPACE**) and start from scratch.
6. When you finish entering the code for the emulation, click **OK** to exit the Editor and return to the Data Selector.
7. Click **OK**.


# Creating and using Runpages

A *runpage*, also called *control page*, is a normal page in your document that is set not to eject, on which a single PlanetPress Talk object is placed. This PlanetPress Talk script is meant to control the execution of other pages and can be used for many purposes such as custom N-Up, repeating and calling overlays and underlays.

To create a runpage:

1. Start with a new, blank page.
2. Open the page's properties (see Set Up a Page).
3. Under Basic Attributes, remove the *Page ejects* option, then click **OK**.
4. Add a new PlanetPress Talk object on the page.
5. In the PlanetPress Talk editor, enter your runpage code, then click **OK**.

Commands that make up a runpage:

- The @page() and execpage() commands are used to execute any page type and place its result in memory, if you do not need to use any of those page's paper handling.
- The $element (or $page) command is used to execute a page if you want to keep the paper handling intact. It is only useful if you are using it to call a normal page, as virtual pages and overlays do not contain any paper handling.
- The showpage() command is used to display (print) the pages that are in memory.

This to keep in mind:

- If any object is placed on the runpage other than a PlanetPress Talk object, these objects will force the page to output as the last page with this element on it, whether or not the runpage has the *Page eject* option checked.
- If you have multiple PlanetPress Talk objects on the page, they will execute in order from top to bottom, one at a time.
- You can place multiple pages in memory using the @page(), execpage() and $page commands, and display them using a single showpage(). This emulates using overlays and underlays on your page. The first page to be placed in memory will be the furthest on the back of the page (underlay), while the last page to be called will be on top of all the others (overlay).
- Make sure to never call the runpage from itself, as this creates an infinite loop and a stack overflow.

## Convert an Object to PlanetPress Talk

You can convert any object on the page to its corresponding PlanetPress Talk version. This can be done to create custom code, to reproduce specific functionality or simply as an academic exercise.

It is important to understand the conversion is not reversible. Once you convert an object to PlanetPress Talk, it becomes a PlanetPress Talk object and any modifications to it must be made in PlanetPress Talk.

This procedure applies only to objects. You cannot convert groups or pages to PlanetPress Talk.

To convert an object to PlanetPress Talk:

1. Select the object.
2. Choose **Tools** | **Advanced** | **Convert to PlanetPress Talk**.

# PlanetPress Talk Basics

In this section, you learn to:

-
-
-
-
-
-
-

In this section, you find examples of how you can use PlanetPress Talk to do the following:

-
-
-
-
-

In this section you will be able to answer the following questions:

-
-
-
-
-
-
-
-
-

## Assumed Knowledge

What background does the Planet Press Talk Reference assume?

The**PlanetPress Talk Language Reference** does not provide information on the art and science of computer pro-gramming. It assumes readers already have a solid grasp of programming concepts and techniques.

## PlanetPress Talk

PlanetPress Talk is a complete scripting language that opens the door to more powerful and sophisticated documents.

This section introduces the PlanetPress Talk scripting language, describes where and how you can enter PlanetPress Talk code in PlanetPress, explains its syntax and each of its elements, and offers some tips and tricks for writing PlanetPress Talk scripts.

In this section, you learn to:

-
-
-
-

In this section, you find examples of how you can use PlanetPress Talk to do the following:

In this section you will be able to answer the following questions:

**Learning PlanetPress Talk**

One of the most efficient ways to familiarize yourself with PlanetPress Talk is to create an object, convert it to PlanetPress Talk, and examine the code, consulting the **PlanetPress Talk Language Reference** as necessary for explanations of each line of code. You can further verify your understanding by modifying the code and observing the changes that occur in the object as a result.

As an example of this technique, the following is the code for a converted data selection object. The data selection is from columns 1 through 12 on line 1 of the data page.

```
moveto(0,0)
setstyle(&Default)
margin(0,0.167)
rmoveto(stringwidth(''),0)
rmoveto(neg(stringwidth('')),0)
show(@(1,1,12))
```

Consider what each line of the code accomplishes, and notice that parameters to a function or procedure may themselves be PlanetPress Talk expressions.

| Command: | Tells the document to: |
|---|---|
| **moveto(0,0)** | Move the drawing pen to the top left of the object. |
| **setstyle(&Default)** | Set the style for the data selection to the default style |
| **margin(0,0.167)** | Set the offset for the text, relative to the top left of the object. |
| **rmoveto(stringwidth(' '),0)** | Move the pen forward the width of the data selection. Here, the purpose of this line is to set a bounding box that appears around the data selection object in the document design window. The bounding box does not print but is a visual cue that keeps the object visible even when its contents are empty. |
| **rmoveto(neg(stringwidth('** | Move the pen backward the width of the data selection. This resets the pen to its position prior to executing the previous command. It ensures the previous command does not alter the posi- |

| Command: | Tells the document to: |
|---|---|
| **')),0)** | tion of the data selection. |
| **show(@(1,1,12))** | Display the text that appears from columns 1 through 12 on line 1 of the current data page. |

At this point you might want to introduce other commands into the code, or modify some commands and verify the changes produce the result you expect. Or perhaps you want to create a new data selection that starts in the middle of the data page and extends over several lines to see how PlanetPress Talk handles multi-line data selections.

## PlanetPress Talk Terminology

What terms does the *PlanetPress Talk Language Reference* use?

The PlanetPress Talk Language Reference uses the following terms:

| Term: | Definition: |
|---|---|
| command | A generic term that encompasses operators, functions, procedures, condition structures, and loop structures. |
| expression | A piece of PlanetPress Talk code used in a PlanetPress Talk-enabled input box, which will always return a value of the type expected by that input box. For example:<br>**=strtoint(@(1,1,4))** |
| statement | A single line of PlanetPress Talk code. For example:<br>**define(&max, integer, strtoint(@(1,1,4)))** |
| script or program | A sequence of one or more statements, executed sequentially. For example the following script draws a filled shape:<br>**moveto(1,1)**<br>**lineto(1.5,2)**<br>**curveto(1,1,1,3,0.5,2)**<br>**closepath()**<br>**stroke()** |

## The Elements of PlanetPress Talk

What are the elements of the PlanetPress Talk scripting language?

PlanetPress Talk includes all of the features one expects in a scripting language. It has a defined set of data types (see "Data Types" (page 220)) and the following categories of elements that you use to build scripts:

| Category: | Description: |
|---|---|
| Variables | A variable is a value that changes over time. See "Variables" (page 222). |
| System objects | An object is a set of related system variables. For example the **physical** system object contains the system variables that describe the current x and y coordinates of the drawing pen on the physical page. You reference one of the system variables in an object by prefixing the name of the system object to the name of the variable. For example **&physical.x** references the x coordinate system variable in the **physical** system object. |
| Operators and Operator functions | An operator is a symbol that performs a common operation. For example, the **+** is an operator that adds two numbers together or concatenates two strings. An operator function is an operator that does not use a symbol to represent its operation. For example the **add** command is an operator function. Both operators and operator functions operate on one or more operands. For example the **add** command operates on two measure or integer values (the operands). See "Operators and Operator Functions" (page n) for a complete list of all operators and operator functions available in PlanetPress Talk. |
| Procedures | A procedure performs an action. For example the **curveto** procedure draws a Bezier curve, the **rectangle** procedure draws a rectangle, and the **setfillcolor** procedure sets the fill color. Most procedures require arguments. For example, the **rectangle** procedure requires six arguments: the x and y coordinates of the upper left hand corner of the rectangle, the width and height of the rectangle, and whether it is filled and/or stroked. |

| Category: | Description: |
|---|---|
| Functions | A function performs an action **and** returns a value. For example the **inttostr** function converts an integer to a string, and returns the value of the resulting string. A function thus has a data type that corresponds to that of the value it returns. You assign the result of a function to a variable or use it as an argument in another command. Most functions require arguments. For example the **inttostr** command requires a single argument: the integer you want to convert to a string; it returns a string and thus the **inttostr** function is of type string. |
| Loop structures | Repeat a sequence of one or more commands until a specified condition is met. See "Loop Structures" (page n). |
| Condition structures | Control when a sequence of one or more commands executes. See "Condition Structures" (page n). |
| Comments | Provide a means for embedding documentation in your code, or for commenting out lines of code during debugging. See "% (procedure)" (page 564). |

## PlanetPress Talk Syntax

What are the rules for creating PlanetPress Talk scripts?

The rules, or syntax, of the PlanetPress Talk language describe how you combine its elements to create scripts.

### Command Syntax

A PlanetPress Talk command has the form:

**command_name**( **arguments** )

Arguments can be constants, variables, or functions (see "Use Functions as Arguments" (page 236)).

A command can require no arguments, or one or more arguments. When a command requires more than one argument, you separate arguments by commas. The following are all examples of valid PlanetPress Talk commands:

```
fill()
moveto( 2.2, 4.5 )
mul(4.7,2.1)
rectangle(0,0,3,3,true,true)
curveto(1,1,1,3,0.5,2)
set(&current.line,&current.line + 1)
store(&current.line,trimleft(&str))
strtoint( @( 12,30,35) )
```

Spaces are not significant in PlanetPress Talk **except** between the name of a command and its opening parenthesis. An error occurs if you leave a space between the name of a command and its opening parenthesis.

```
show('this syntax is correct')
show ('this syntax produces an error')
```

### Comments

PlanetPress Talk allows the use of comments within the code. To enter an expression, simply enter a percentage sign (%) followed by the text you want to comment.

Comments may only be used within a PlanetPress Talk object, or in the "PlanetPress Talk Before" and "PlanetPress Talk After" sections of any object, and is not supported in PlanetPress Talk-enabled input boxes (for example, conditions and advanced data selections).

### Case

PlanetPress Talk is case insensitive. Thus the following commands are all equivalent:

```
showright(@(1,34,64))
ShowRight(@(1,34,64))
SHOWRIGHT(@(1,34,64))
```

It is important to understand that arguments to commands **are** case sensitive. For example, the following command returns False:

```
eq('bovine','Bovine')
```

## Unit of Measure

Unless otherwise specified, the unit of measure in PlanetPress Talk is inch. Thus, for example, the following line of code moves the current point to the X coordinate 1.5 inches, and the Y coordinate 2.5 inches.

```
moveto(1.5,2.5)
```

## Statement or Expression Evaluation

The PlanetPress Talk interpreter uses parentheses to determine the order in which to evaluate the individual parts of an expression or statement. It evaluates an expression or statement from the innermost to the outermost set of parentheses. For example, consider the statement:

```
if( (pos('BLACK', strip(' ',@(36,25,58))) > 0 ), 'old', 'new')
```

The interpreter first evaluates the data selection function (**@**), then the strip function (**strip**), then the start position function (**pos**), and finally the **if** function.

```
@(36,25,58)
strip(' ',@(36,25,58))
(pos('BLACK', strip(' ',@(36,25,58)))
if( (pos('BLACK', strip(' ',@(36,25,58))) > 0 ), 'old', 'new')
```

## Operator Precedence

The rule that says that multiplications and divisions should be performed before additions and substractions, known as operator precedence, is not supported in PlanetPress Talk, so you should use parentheses to simulate operator precedence. For example, if you use the equation 2 + 3 x 4, PlanetPress Talk will calculate a total of 20, but if you use the equation 2 + (3 x 4), PlanetPress Talk will calculate a total of 14.

## Apostrophes and Backslashes within Strings

If you want to use an apostrophe (') or a backslash (\) within a string, you must precede it by a backslash. For example:

```
show( 'Don\'t miss this offer!' )
```

## Names

The names of any variables, global functions, objects, or styles you create with PlanetPress Talk must meet the following requirements:

- Names must be unique (no two elements in a document can have the same name).
- Names cannot begin with a number, and can contain only the following ASCII characters: underscore, upper and lower case letters of the alphabet, all digits 0 through 9. If you use an underscore in the name, it should not appear as either the first or last character of the name as this may cause internal conflicts in the software.
- Names can be a maximum of 50 characters in length.
- PlanetPress Talk variable and command names are reserved words; you cannot use any of these reserved words as a name.

It is always recommended that you choose a name that reflects the content or purpose of the element you are creating. A meaningful name makes it easier to distinguish one element from another, and thus makes a document easier to maintain.

## Data Types

What data types are available in PlanetPress Talk?

PlanetPress Talk provides the following data types:

| Data Type | Example |
|---|---|
| Integer | 3 |
| Measure or Floating Point | 3,2300987 |
| Currency | 5.25 |
| String | Michel or $4,567.09 |
| Boolean | Yes or No |
| Color Array | [100, 100, 0, 0] **(for the CMYK color model)** |
| Array | [Monday Tuesday Wednesday Thursday Friday] **(a seven-element array of type string)** [12 18 103 57] **(a four-element array of type integer)** [True True False False True False False] **(a seven-element array of type Boolean)** |
| Directory | [c:\images\house.jpg c:\images\apt.jpg] **(a directory that contains two pathnames)** |

PlanetPress Talk strictly enforces its data types. For example, the **show** command accepts a string as its argument; an error occurs if you attempt to send it an integer as an argument. You can use any of the conversion operators PlanetPress Talk provides to convert from one type to another. Thus the following show command is correct:

show(inttostr(subreccount()))

See for a complete list of the operators you can use to convert from one type to another.

### Integers

You use integers for counters and simple calculations. Integers have no decimal precision.

| Integer | Limit |
|---|---|
| Smallest on screen | -2147483648 |
| Largest on screen | +2147483647 |
| Smallest PostScript | -2147483648 |
| Largest PostScript | +2147483647 |

### Measures

Measures are real numbers and are often called floating-point numbers. Measures have a precision of approximately 8 decimal places, with exact precision implementation dependent. If your document requires complex mathematical computations in which a greater decimal precision is critical, it is recommended you perform those calculations outside PlanetPress and include the result in the input data.

| Measure | Limit |
|---|---|
| Smallest on screen | 1.5 * 10-45 |
| Largest on screen | 3.4 * 1038 |

| Measure | Limit |
|---|---|
| Smallest PostScript | 10-38 |
| Largest PostScript | 1038 |

## Currency

Currency is a specific type of measure intended for use with numbers that represent monetary values. The currency type has a precision of 4 decimal places, and can take on a minimum value of -99999999999999.9999 and a maximum value of +99999999999999.9999. The only operations you can perform on a currency type are addition, subtraction, multiplication and division, and negation.

## Strings

A string is a sequence of one or more alphanumeric characters. You often use strings to display information on a page. You must enclose strings in single quotes, for example 'mystring'.

| String | Length, in characters |
|---|---|
| Maximum on screen | 2147483647 |
| Maximum PostScript | 65535 |

## Booleans

Booleans have a value of either True or False. Every condition you define in PlanetPress resolves to a Boolean value.

## Color Array

To define the color or shading of object displayed on screen or printed on paper, color values are used, which are typically based on color models. Models may include a single color (such as for the grayscale model) or multiple colors. The RGB color model uses three colors (Red, Green and Blue), while the CMYK model uses four (Cyan, Magenta, Yellow and Black). For each color, a value ranging from 0, for 0% color intensity, to 255, for 100% color intensity, is used. If the Grayscale model is used, only a single value is required. For the RGB model, three values are used, while four are needed for the CMYK model. These values are stored in an array, which can store as many as four values.

Color arrays are primarily used internally by the PlanetPress Talk interpreter. However there are a handful of commands that expect a color array as an argument (**setfillcolor** for example).

## Arrays

An array is a table of values of the same data type. You reference each element of the array (each cell of the table) independently. Arrays can be of type Boolean, integer, measure, currency, or string. The type of the array defines the type of values each of its elements can contain. Thus in a Boolean array, the value of each element is either True or False, and in a string array each element is a string value.

Arrays in PlanetPress Talk are one-dimensional. If you are not familiar with arrays from other programming or scripting languages, you can think of a one-dimensional array as a single row of a table. You specify the number of elements you want the array to contain, and initialize each of the elements, when you define the array. Note that you cannot subsequently increase or decrease the number of elements in an array. If you are uncertain of the exact number of elements the array may need to contain at runtime, you may want to create an array that can handle the most extreme case you expect the document to encounter at runtime. Note however that increasing the size of the array increases the memory required for the document, and that it is good programming practice not to create arrays that are larger than what the document requires to execute properly. An array in PlanetPress can have a maximum of 65,535 elements.

You define an array using the **define()** command, and assign values to individual elements using the assignment operator (:=), the **set()** command or the **put()** command. You retrieve the value of an individual element using either the **get()** command, or the name of the array followed by the position of that element in the array. The position is an integer value starting at

0 for the first element of the array, and incrementing by one for each additional element. See <u>"Define (procedure)" (page 559)</u>, <u>"Set (procedure)" (page 561)</u>, <u>"Put (procedure)" (page 560)</u>, and <u>"Get (function)" (page 541)</u>.

As an example, consider **&tax_rates**, an integer array containing five elements (10 20 30 40 50). You define the array as follows:

```
define( &tax_rates,arrayinteger,[10,20,30,40,50] )
```

You can use any of the following to change the value of the third element from 30 to 32.

```
&tax_rates[2] := 32
&tax_rates, 2, 32
put( &tax_rates, 2, 32 )
```

You can use any of the following to reference the third element in the array:

```
&tax_rates[2]
get(&tax_rates, 2)
```

For example:

```
&due := mul( &gross,&tax_rates[2] )
&due := mul( &gross,get(&tax_rates,2) )
```

### Directory

The directory type is a type of string array in which each element in the array contains a pathname. All pathnames in the array reference a file in the same folder. The number of elements in the array, and the initial value of each element are both determined dynamically at the time the array is created. You specify the folder, and the filter you want to apply to the list of files in that folder. When PlanetPress Talk creates the array, it locates the folder you specified, applies the filter to the list of files that folder contains, and then for each file that meets the filter criteria, adds its pathname to the directory array. Pathnames may be either printer pathnames or host pathnames.

You can subsequently set the value of any of the elements in the array, as you would for any element in an array of type string.

## Variables

What are variables?

A variable is a value that changes over time, that has a specific type, and a defined scope. Variables have names to make it possible to reference them. All variable names in PlanetPress begin with an ampersand. For example, **&printermode**.

**Type** The type of a variable defines what kind of data it contains and consequently what kinds of operations you can perform on it. The following are the data types available in PlanetPress Talk: integer, measure, currency, string, boolean, array, color array, and directory. See <u>"Data Types" (page 220)</u>.

**Scope** The scope of a variable is the context in which it is available. There are two possible scopes: local and global. A local scope means the variable is available only in the specific area of an object or group in which it is defined, and a global scope means the variable is available anywhere you can enter PlanetPress Talk code. The way you define a variable determines its scope. The variables you define using the **define** command have a local scope limited to the area of the object or group in which you define them. The variables you define as global variables in PlanetPress have a global scope. Global variables in PlanetPress appear in the Global Variables area of TreeView.

There are a number of ways to use variables in documents. See the **PlanetPress User Guide** for examples of some of the most common uses people make of variables.

**Common Uses of Variables**

The following are several common uses for variables:

- **Store values that you want to reference at various places in the document**.
- **Concatenate strings** For example, you want to create a footer for each page of a patient record document that contains the patient's name, physician's name, file number, and date the record was last updated. All the pieces of information are available in the input data.
- **Perform calculations** For example, the input data for your financial statement document does not include a figure for year-to-date earnings, but does include earnings per month.
- **Create counters** For example, the input data for your invoice document contains a variable number of line items for each invoice.

## Create a Global Variable

To create a global variable:

1. Choose **Home** | **Document** | **Global Variable**.
2. Enter the name and type of the global variable you want to create.
   **Name**: Enter a name for the global variable.
   **Variable type**: Select the type of the global variable (string, integer, measure, currency, Boolean, integer array, measure array, string array, Boolean array, or directory array).

   > 》》》 String Variables are limited to 65534 characters.

3. Define the properties specific to the type of variable you selected. Consult the *PlanetPress Talk Language Reference* for help understanding the various types.
   - If you selected a variable type of string, integer, measure, currency or Boolean, enter an initial value for the variable in the **Default value** box. Note that you cannot use PlanetPress Talk code to define an initial value.
   - If you selected a variable type of color array, click the **Color** button to use the Color Picker to select an initial color for the color array. The Color box updates to reflect the selected color.
   - If you selected a variable type of integer array, measure array, string array, or Boolean array, enter the number of elements you want the array to contain in the **Array size** box, or use the spin buttons to adjust the value.
   - If you selected a variable type of directory array, enter the path to the folder containing the files you want to reference, and any filter you want to apply to names of the files in that folder, in the **Path and filter** box.
4. Click **OK**.
   The new global variable appears in the Global variables folder in the Structure area.

   Global variables
     subtotal (subtotal)
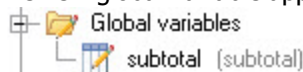
   It also appears on the Global variables submenu available from the Code area of the PlanetPress Talk Editor, from the PlanetPress Talk properties of an object, and from any text box that accepts PlanetPress Talk expressions

To reference a global variable:

- Enter the name of the global variable by hand, preceding it by an ampersand. For example, to reference the global variable **subtotal**, enter **&subtotal**. In the case of an array element, you must include the position of the element in the array.

To set a global variable:

- Use either the PlanetPress Talk assignment operator (**:=**) or **set()** command. For example:
  &code := &code + 1

---

```
set( &code, (&code + 1) )
set( &imagefiles[0], 'c\\images\\sushi.png' )
```
See the **PlanetPress Talk Language Reference** for complete information on the assignment operator and the **set()** command.

## View or Edit a Global Variable

To view or edit the properties of a global variable using the Object Inspector:

1. In the Structure area, select the global variable.
2. In the Object Inspector, make any necessary modifications to the properties.
   If you made a modification that may cause execution problems in the document, PlanetPress Design reports an error in the Messages area.
   If you make a modification that may cause execution problems in the document , PlanetPress Design requests confirmation before proceeding with the modification.

To view or edit the properties of a global variable using the Global Variable dialog box:

1. In the Structure area, double-click the global variable. Use the Global Variable dialog box to edit the properties of the global variable, if necessary.
2. Click **OK**.
   If you edited the properties of the global variable and made a modification that may cause execution problems in the document, PlanetPress Design requests confirmation before proceeding with the modification.

## Delete a Global Variable

To delete one or more global variables:

1. Select the global variables you want to delete.
2. Choose **Home** | **Clipboard** | **Delete**.
   If no elements in the document reference any of the selected global variables, PlanetPress Design performs the deletion.
   If any elements in the document reference any of the selected global variables, PlanetPress Design prompts you to define how you want to handle the deletion of each of the referenced global variables. More precisely, for each referenced global variable, it displays the Global Variable Deletion dialog box.

To use the Global Variable Deletion dialog box:

1. Adjust the options to reflect how you want PlanetPress Design to handle the deletion request. The name of the global variable you selected for deletion appears in the title bar of the Global Variable Deletion dialog box, and the list of elements that reference it appear on the right of the dialog box.

**Global Variable Deletion GlobalVariable3 (GlobalVariable3)**

PlanetPress is about to delete **GlobalVariable3** (GlobalVariable3).
How do you want to handle references to this element?

Referencing elements
Ttest1 (Ttest1)
Test (Test)

Deletion options
○ Replace all references with
Global variables available
GlobalVariable1 (GlobalVaria ▼)
○ Delete all elements

OK    Cancel    Help
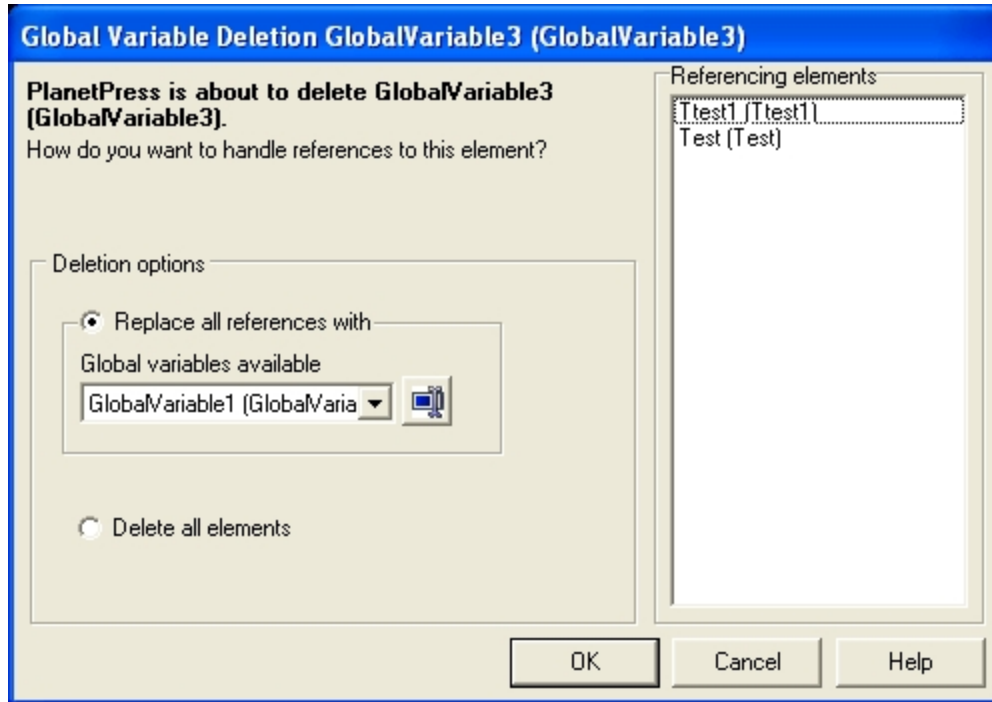
A. Global Variables button

**Replace all references with**: Select to delete the global variable and to replace all references to it with a reference to another global variable in the document.

**Global variables available**: Select the global variable you want to use as the replacement reference. When you delete the global variable, PlanetPress Design replaces all references to the deleted global variable with a reference to the global variable you select here. You can use the Global Variables button to create a new global variable to add to this box.

**Global Variables button**: Click to create a new global variable. PlanetPress Design creates the new global variable, and selects it in the Global variables available box.

**Delete all elements**: Select to delete the global variable, and all document elements that reference it. All document elements that reference this global variable appear in the list on the right of the Global Variable Deletion dialog box.

2.  Click **OK**.

## Local, Global, and System Variables

There are three categories of variables in PlanetPress Talk: system, global, and local.

1.  System Variables
    System variables are variables that the system defines. There are two types of system variables: system variables with a global scope and system variables with a local scope.
    **System variables with a global scope** System variables with a global scope are available anywhere you can insert a PlanetPress Talk expression in PlanetPress. You can use the value of a system variable with a global scope but you cannot, except in one or two cases, modify it.
    **System variables with a local scope** System variables with a local scope are available only within a well-defined context. There are currently two system variables with a local scope: **&height** and **&width**. Both are local to a specific object, group, or page. The system initializes **&height** and **&width** to, respectively, the values of the Height and Width properties of the object, group, or page, as they appear in the Basic Attributes of that object, group, or page. See "System Variables with Global Scope" (page 226) for a complete list of the system variables available in PlanetPress Talk.
2.  Global Variables

Global variables are variables that you define using the Global Variable Properties dialog box in PlanetPress. Global variables appear in TreeView in PlanetPress. Global variables have a global scope; you can reference global variables anywhere you can insert a PlanetPress Talk expression in PlanetPress.

You cannot define a global variable with the same name as a system variable.

3. Local Variables

Local variables are variables that you define using the PlanetPress Talk**define** command. Local variables have a local scope; you can reference them only within the area of the object in which you create them. You can create local variables within the PlanetPress Talk properties of an object or group, within the Text property of a text/box object, and within the PlanetPress Talk Code property of a PlanetPress Talk object.

You should never define a local variable with the same name as a system variable.

## System Variables with Local Scope

| Name: | Description: |
| --- | --- |
| **&Height** | System variable with a scope local to a specific object, group, or page. The system initializes this variable to the value of the Height property of the object's, group's, or page's Basic Attributes. You can reference and set this variable in any PlanetPress Talk code you enter in the object, group, or page. |
| **&Width** | System variable with a scope local to a specific object or group. The system initializes this variable to the value of the Width property of the object's or group's Basic Attributes. You can reference and set this variable in any PlanetPress Talk code you enter in the object, group, or page. |

## System Variables with Global Scope

| Name: | Description: |
| --- | --- |
| **&EOJob** | Read-only variable that returns True if the document has processed the last line of data or if the line that follows the last form feed character (ASCII 12) is empty. |
| **&FirstSide** | Read-only variable that returns True when a page is printing on the first side of a page in duplex mode (double-sided printing). In simplex mode (single-sided) &firstside always returns True. |
| **&PrinterMode** | Returns the current output mode for the document. This allows you to test for a specific value and react accordingly. |
| **&Str** | Contains a line of input data. |

# Current System Object

| Name: | Description: |
| --- | --- |
| **&Current.DataPage** | Integer value containing the page number of the current data page, or the number of the current record set in the case of a database emulation. |
| **&Current.Line** | Integer value containing the line number of the current line of data within the data page, or the number of the current record within the record set. This value is read-write and can be modified using the set command. |
| **&Current.LPP** | Integer value containing the number of lines per page in the current data page. |
| &Current.MediaColor | String value containing the color set in the Media color box of the Page dialog box for the current document page, or, if no value is set in that box, the color set in the Media color box in the Document dialog box for the document. The Media color box appears when you select Media selection in the Selection type box. Recall that the Media color box specifies the color of paper. "Current (system object)" (page 376) |
| &Current.MediaType | String value containing the type set in the Media type box of the Page dialog box for the current document page, or, if no value is set in that box, the type set in the Media type box in the Document dialog box for the document. The Media type box appears when you select Media selection in the Selection type box. Recall that the Media type box specifies the type of paper (plain, coated, etc.)."Current (system object)" (page 376) |
| &Current.MediaWeight | Integer value containing the weight set in the Media weight box of the Page dialog box for the current document page, or, if no value is set in that box, the weight set in the Media weight box in the Document dialog box for the document. The Media weight box appears when you select Media selection in the Selection type box. Recall that the Media weight box specifies the weight of paper, in grams per square meter. "Current (system object)" (page 376) |
| **&Current.MinFeature** | Measure value containing the width of a single pixel at the current zoom level. Precise to 1/65536 inch. This is the minimum size you can use in PlanetPress Talk code. If you use a smaller value (for example as an argument in a command), PlanetPress Talk automatically replaces that value with the value of minfeature."Current (system object)" (page 376) |
| **&Current.Orientation** | Integer value containing the orientation of the current document page (0=portrait, 1=landscape, 2=Rotated portrait, 3=Rotated landscape). "Current (system object)" (page 376) |
| **&Current.PageHeight** | Measure value containing the physical page height, in inches, of the current page in the document."Current (system object)" (page 376) |
| **&Current.PageWidth** | Measure value containing the physical page width, in inches, of the current page in the document."Current (system object)" (page 376) |
| **&Current.PrintPage** | Integer value containing the current page number in the document."Current (system object)" (page 376) |
| **&Current.x**<br>**&Current.y** | Measure value containing the current horizontal (x) or vertical (y) position of the drawing pen on the current page. This value is relative to the top left corner of the current page - whose coordinates are (0,0) - and may not necessarily be equal to the current physical position on the page (for instance, if the page is called from a PlanetPress Talk object located on the middle of the physical page)."Current (system object)" (page 376) |

## Current (system object)

This object includes a set of system variables reflecting the current state of the document. All of these variables are read-only and therefore cannot be changed using the set command, except where noted. You can access each property individually using the same syntax.

### Syntax

&current.property ⇒ integer, measure value

### Properties

datapage

Integer value containing the page number of the current data page, or the number of the current record set in the case of a database emulation.

line

Integer value containing the line number of the current line of data within the data page, or the number of the current record within the record set. This value is read-write and can be modified using the set command.

lpp

Integer value containing the number of lines per page in the current data page. In database emulation, this value can also be used in place of the subreccount() function to determine how many records a record set contains.

mediacolor

String value containing the color set in the Media color box of the Page dialog box for the current document page, or, if no value is set in that box, the color set in the Media color box in the Document dialog box for the document. The Media color box appears when you select Media selection in the Selection type box. Recall that the Media color box specifies the color of paper.

mediatype

String value containing the type set in the Media type box of the Page dialog box for the current document page, or, if no value is set in that box, the type set in the Media type box in the Document dialog box for the document. The Media type box appears when you select Media selection in the Selection type box. Recall that the Media type box specifies the type of paper (plain, coated, etc.).

mediaweight

Integer value containing the weight set in the Media weight box of the Page dialog box for the current document page, or, if no value is set in that box, the weight set in the Media weight box in the Document dialog box for the document. The Media weight box appears when you select Media selection in the Selection type box. Recall that the Media weight box specifies the weight of paper, in grams per square meter.

minfeature

Measure value containing the width of a single pixel at the current zoom level. Precise to 1/65536 inch. This is the minimum size you can use in PlanetPress Talk code. If you use a smaller value (for example as an argument in a command), PlanetPress Talk automatically replaces that value with the value of **minfeature**.

orientation

Integer value containing the orientation of the current document page (0=portrait, 1=landscape, 2=Rotated portrait, 3=Rotated landscape).

overflowcount

Integer variable storing the number of times that a data page overflowed and that a new document page was added to accommodate the overflowing data. Initially set to 0, is incremented by 1 every time the page overflows.

overflowing

Boolean variable set to False by default. When the current *data page* contains more lines than can be printed by the current object, or technically speaking when the object's condition to exit and overflow has been met, this variable is set to True. It is set back to False with every new overflowing page, the logic being that the current overflowing page will be the last, unless if the condition to exit and overflow is met yet again.

lastoverflowrepetition

Integer variable storing the number of the last iteration value when the condition to exit and overflow has been met.

pageheight

Measure value containing the physical page height, in inches, of the current page in the document.

pagename

String value containing the name of the page that is actually printing. This allows conditional processing to take place within a PPTalk object according to the name of the page being executed. Note that the value always corresponds to the name of the actual document page being printed, even if the call comes from a virtual or overlay page (in other words, the name of the page on which the overlay is layed out).

pagewidth

Measure value containing the physical page width, in inches, of the current page in the document.

printpage

Integer value containing the current page number in the document.

x,y

Measure value containing the current horizontal (x) or vertical (y) position of the drawing pen on the current page. This value is relative to the top left corner of the current page - whose coordinates are (0,0) - and may not necessarily be equal to the current physical position on the page.

### Physical (system object)

The **physical** object contains system variables that reflect the current state of the document. These system variables are read-only and therefore cannot be changed using the set command. You can access each system variable using the same syntax.

### Syntax

&physical.x ⇒ measure value

&physical.y ⇒ measure value

### Variables

x,y

Measure value containing the current horizontal (x) or vertical (y) position of the drawing pen, relative to the upper left corner of the physical page, whose coordinates are (0,0). This system object can be used to assess the amount of space left on a page.

**System (system object)**

The system system object contains system variables related to the Raster Image Processor (RIP) the document is using to execute, and document versioning. These system variables are read-only.

**Syntax**

&system.product ⇒ string value

&system.version ⇒ string value

&system.formversion ⇒ integer value

**Properties**

product

String value containing the name of the RIP the document is using to execute. For example,

PlanetPress Alambic is the name of the RIP built in to PlanetPress (the RIP PlanetPress uses when you perform a preview and select Internal interpreter as the PostScript interpreter).

version

String value containing the version number of the RIP the document is using to execute. For example, 4.0.

formversion

Integer value containing the version number of the document requested by the trigger. Recall that you can assign a version number to a document and increment the number each time you update and reinstall the document. When you execute a document that uses a version number, the document verifies its version number against the one you specify in the trigger, and it proceeds with execution only if the two version numbers match. See the **PlanetPress User Guide** for more complete information on versioning.

## Runpage

What is a runpage?

A runpage is a normal page containing a PlanetPress Talk object that controls the execution of the pages in the document. Although nothing prevents a runpage from containing printable elements, in most cases it serves only to control the execution of the other pages in the document and does not itself print.

## Using Foreign Language Text with PPtalk

Can PlanetPress Talk let me use foreign language text in PlanetPress?

You can use Text objects to display both static text and variable content, such as data selections, in the same object. Text objects also let you mix foreign language text, such as Arabic, and European language styles within the same object, although not in the same paragraph.

**Adding Foreign Text**

The Text objects dialog box lets you enter foreign language text, such as Arabic, directly in the Text area (granted that the corresponding foreign language text feature is enabled in Windows).

**Using Global Variables**

If you use a Global variable containing a foreign language text string within a Text object, you must map the characters included in that non-Unicode text string to UTF8. To do this, you must use the **mapUTF8** function. You can do this by placing a PlanetPress Talk object above the Text object in the Document Structure or in the Text object itself using the PlanetPress Talk Before properties.

**Using Dynamic Data within Local Text Variables**

You can add dynamic data to a Text object by using custom data selections. If you do so , you must use the **mapUTF8** function within the definition of each custom data selection.

**Text Direction**

A new setting was added to the paragraph options to enable right-to-left text display for paragraphs associated with a foreign language text style.
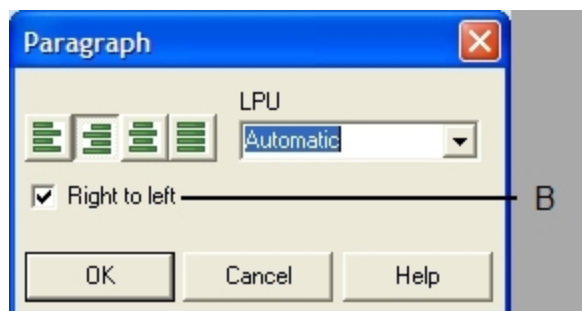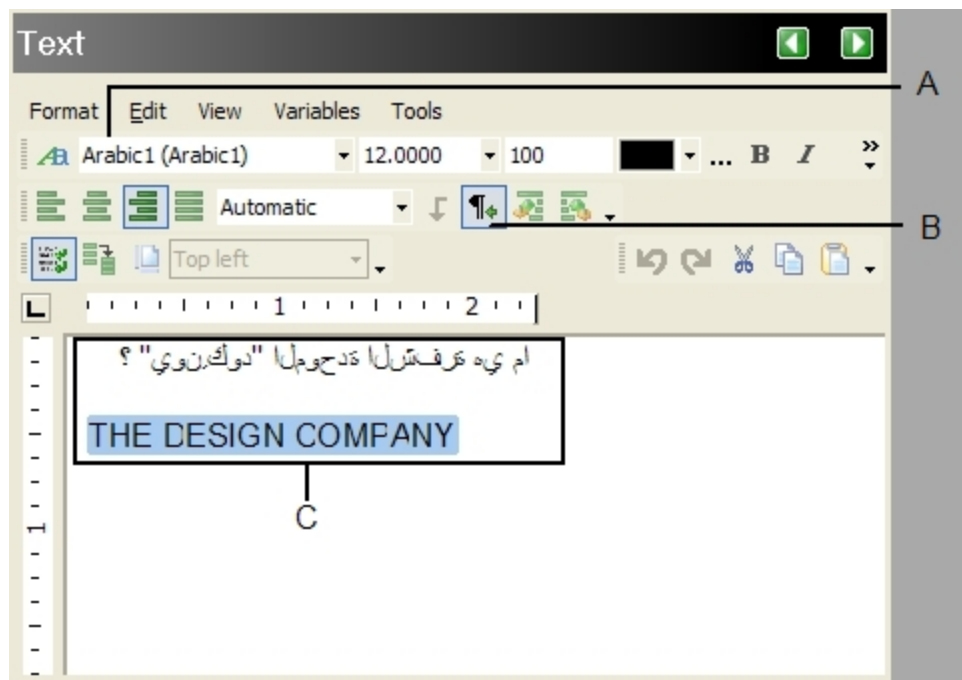
A. Arabic style ; B. Right to left text option available when an Arabic style is selected ; C. Static text and data selection as edited within the Text objects dialog box.

PLANETPRESS. SUITE

## Foreign Language Text within PlanetPress Objects

You can use PlanetPress Talk objects to display either static text or variable content. You would typically add a paragraph for static text or a simple field for variable information, such as a client's name or address.
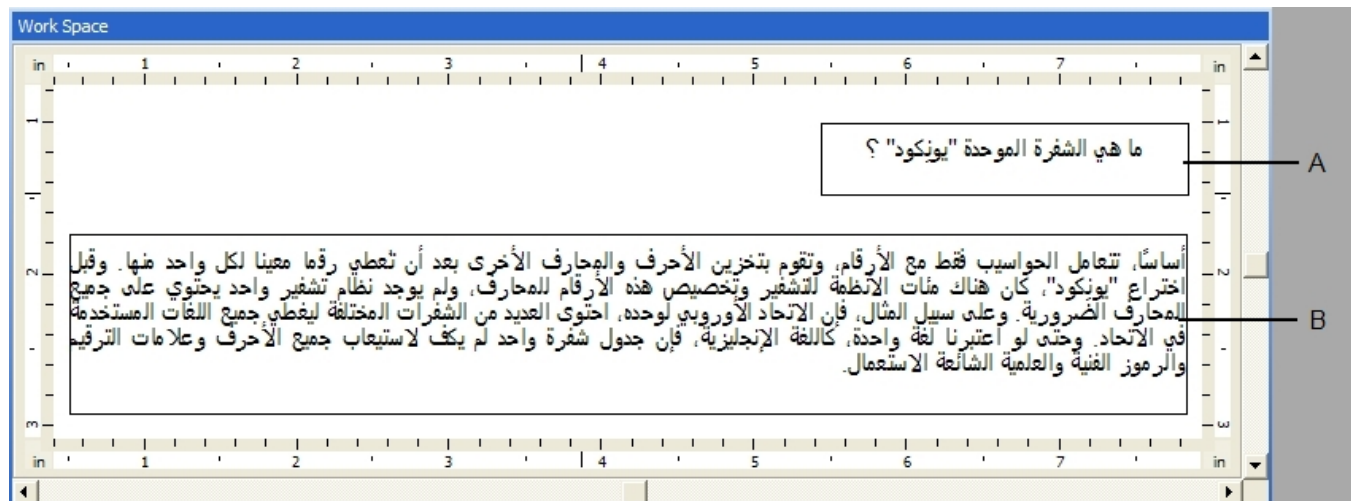


A. A PlanetPress Talk object created without defining a UTF8 paragraph ; B. Another PlanetPress Talk object, created this time with UTF8 paragraph properties (the paragraph is justified on both sides)

If you need a string, that may contain variable content, you can use the **ShowUTF8Left**, **ShowUTF8Center** and **Sho-wUTF8Right** commands. *Note that if you choose to display a box around objects that include any of these commands, the box may not be perfectly drawn. Pages that include many objects with these commands will not display as fast as other pages.*

If you need a paragraph, that may only contain static text, you must use the **BeginUTF8Paragraph** and **EndUTF8Pa-ragraph** procedures along with the **ShowUTF8** command.

The following illustration shows the properties of a PlanetPress Talk object that creates a paragraph containing Arabic text. The code associated with this object is also displayed in the PlanetPress Talk Editor.
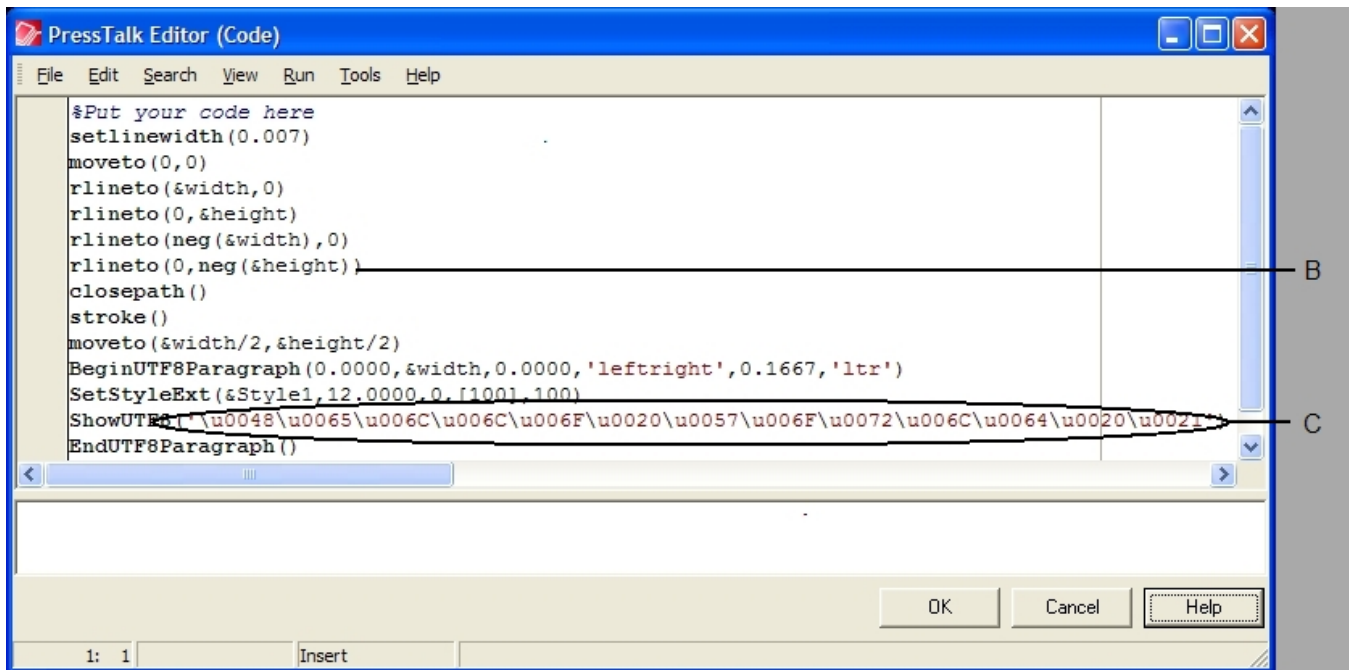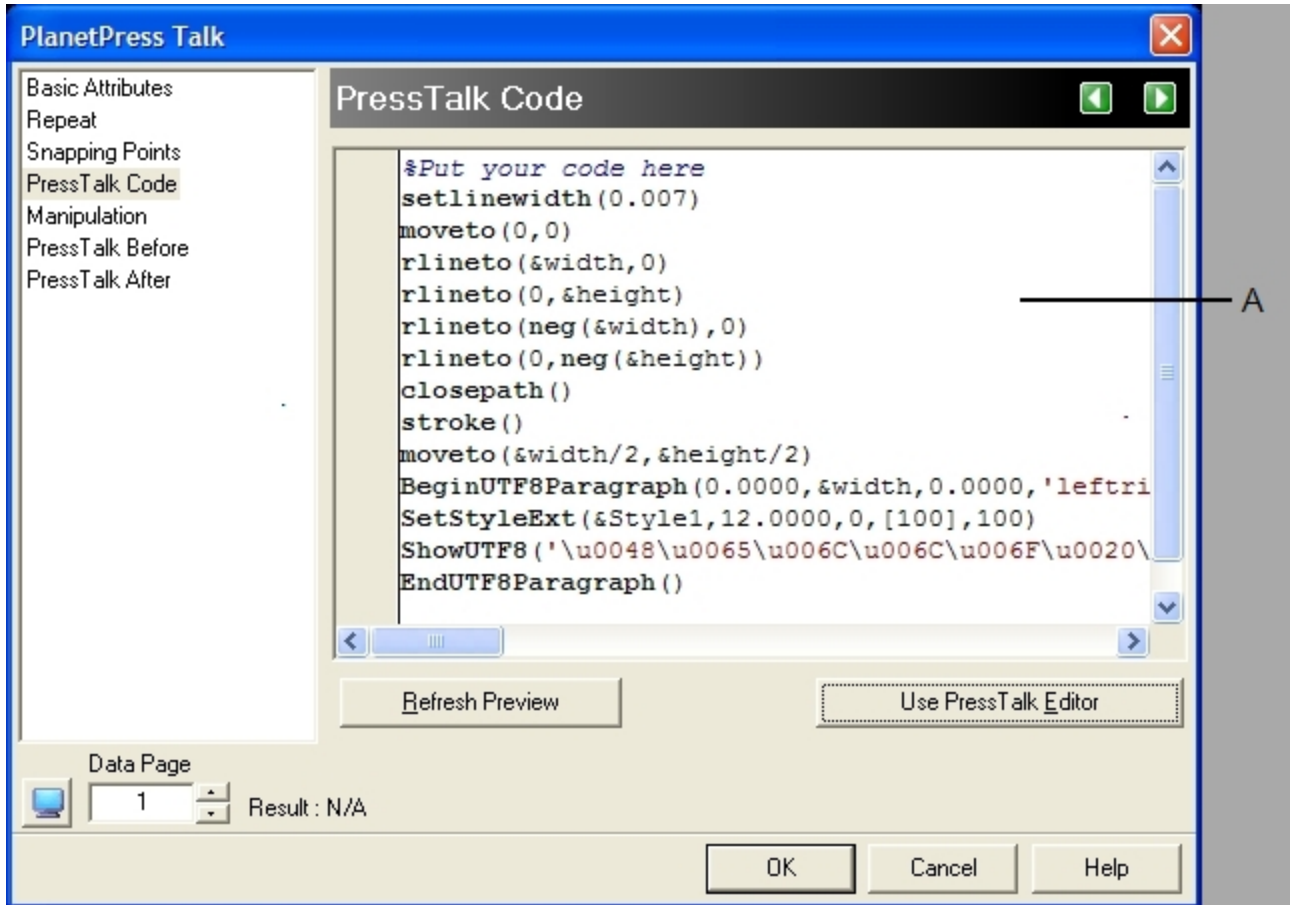
A. The properties of a PlanetPress Talk object created to display Arabic text; B. The complete PlanetPress Talk code, as displayed in the PlanetPress Talk Editor; C. All the characters included in the text string, with the exception of the basic ASCII characters, are converted to their Unicode reference by the Ansi/UTF8 Converter dialog box

The illustration below shows the properties of a PlanetPress Talk object that creates an Arabic text string.
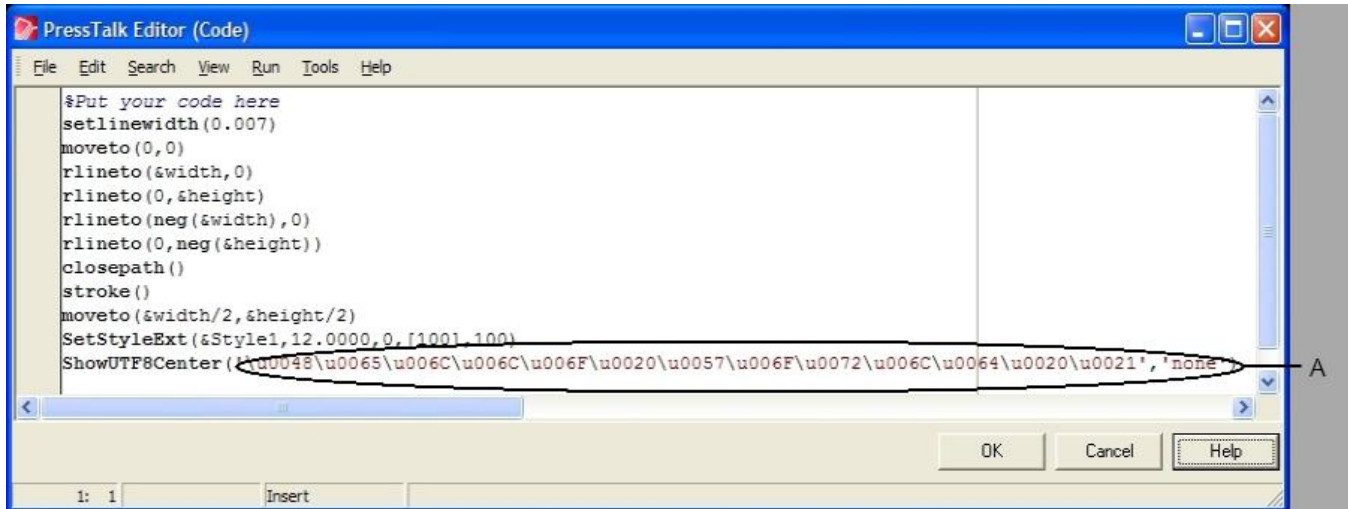
The code of a PlanetPress Talk object that displays a UTF8 static text string

A dialog box that lets you enter arabic text has also been added. If you place the cursor within a text string and press CTRL+N, the Arabic Converter dialog box will be displayed. When you are finished, click OK. The dialog box will be closed and the Arabic characters will be converted to their Unicode equivalent.
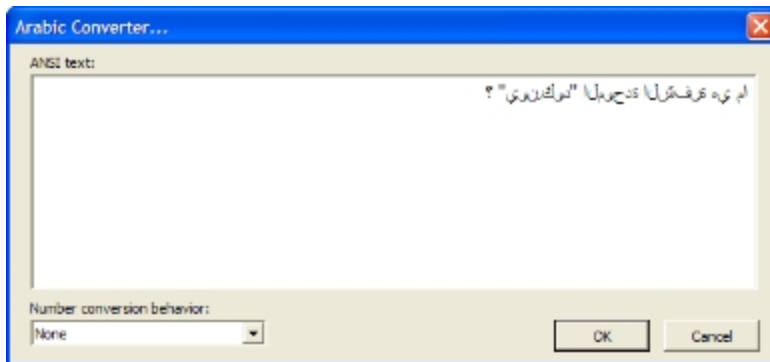


The Arabic Converter dialog box also includes number conversion options that can be used to convert numbers to either Arabic or Farsi.

To enter variable content, you can use a data selection. Use the right-click menu to display the Data Selector and select either a location within the data or a given field in the case of a database.

Once the data has been selected, the data reference appears as a parameter.

## Learning PlanetPress Talk

An efficient way to learn PlanetPress Talk is as follows. Consult the **PlanetPress User Guide** for an example of this approach using a data selection object.

1. Get a feel for the different system variables, operators, and commands PlanetPress Talk provides (read through the "PlanetPress Talk Summary" (page n)).
2. Create an object in PlanetPress, convert it to PlanetPress Talk and examine the code, consulting the **PlanetPress Talk Language Reference** as necessary for explanations of each line of code.
3. Once you understand code for that object, verify your understanding by modifying it and observing the changes that occur in the object as a result.

Note that PlanetPress Talk has many similarities to the PostScript language, and programmers familiar with PostScript should find PlanetPress Talk easy to learn and use. This should also hold for programmers familiar with the Windows Application Programming Interface (API).

## Integrate PlanetPress Talk into Documents

You can enter PlanetPress Talk code in any of the following places in PlanetPress. Note that in all places except a text box that accepts PlanetPress Talk code, you can use the PlanetPress Talk Editor to enter your scripts. The PlanetPress Talk Editor provides a rich set of features that make entering, executing, and debugging scripts fast and efficient. For complete information on the PlanetPress Talk Editor, consult the **PlanetPress User Guide**.

| Location: | Accepts: |
| --- | --- |
| PlanetPress Talk-enabled input box (any box for which the title or label is colored in maroon/dark red) | A single line of PlanetPress Talk code preceded by an equal (=) sign. |
| PlanetPress Talk properties of an object, group, page, document, or condition. | A PlanetPress Talk script. You can enter the script directly in the appropriate Properties dialog box, or using the PlanetPress Talk Editor. |
| PlanetPress Talk code property of a PlanetPress Talk object. | A PlanetPress Talk scripts. You can enter the script directly in the PlanetPress Talk Object Properties dialog box, or using the PlanetPress Talk Editor. |
| PlanetPress Talk properties of a paragraph in a text/box object | A PlanetPress Talk script. You enter the script using the PlanetPress Talk Editor. |
| User-defined emulation selection in the Data Selector. | A PlanetPress Talk script. You can enter the script using the PlanetPress Talk Editor. **Important note**: When a user-defined emulation is used with metadata, results and behavior are unknown and unsupported. For instance, refreshing the metadata file may cause the document to crash and/or corrupt. For this reason, it is strongly advised to create backup copies of your documents beforehand. |
| Global Function Properties dialog box | A PlanetPress Talk script that defines a new PlanetPress Talk function. |
| **Import** command in the PlanetPress Talk Editor | A file containing a PlanetPress Talk script. For example you can write a PlanetPress Talk script in an external text editor, save it as a **.ptk** file. You use the PlanetPress Talk Editor to import and, if necessary, edit the script. You can also export a script from within the PlanetPress Talk Editor, using the **Export** command. This makes it easy to re-use scripts. Consult the information on the PlanetPress Talk Editor in the **PlanetPress User Guide** for help importing and exporting scripts. |

## Define and Assign Values to Variables

You use the following PlanetPress Talk following commands to define and set variables:

| Use: | To: |
| --- | --- |
| **define** | Create a local variable. |
| **set** | Assign a value to a local or global variable. |
| **:=** | Assign a value to a local or global variable. |

## Select Data

You use the **@** function to select data in PlanetPress Talk. This command selects data on a single line of the data page. For example, the following returns the string of data found on line 3, from column 12 through 30 of the current data page:

@(3,12,30)

Remember that arguments can be variables. The following returns the string of data found on line **&line**, from column **&col1** to column **&col2** of the current data page:

@(&line, &col1, &col2)

You usually assign the result of a function to a variable or use it as an argument to another command. For example:

&customer_name := @(&line, &col1, &col2)

You can also use the Data Selector to insert a PlanetPress Talk data selection command. This is often quicker and more convenient than manually typing the data selection command. Consult the **PlanetPress User Guide**, and in particular the area of the PlanetPress user interface you are using to enter PlanetPress Talk code, for help.

## Use Functions as Arguments

You can nest functions so that the value returned by one function becomes the argument for another function. For example, in the following example, the string returned by the **@** function becomes one of two arguments for the **strip** function:

```
strip('*',@(36,25,58))
```

As a more complex example, consider the following:

```
if((pos('BLACK', strip( ' ', @(36,25,58))) > 0 ), 'old', 'new')
```

It returns the string 'old' if the data selection contains the string 'BLACK' and 'new' if it does not. When the PlanetPress Talk interpreter evaluates the statement, it first selects the data on line 36 of the current data page, from columns 25 through 58. It then sends that data to the **strip** command which removes all spaces in the data. It then sends the data (which now contains no spaces) to the **pos** command, which determines whether it contains the string 'BLACK'. Finally, it compares the value returned by the **pos** command to 0, and returns the value of that comparison to the **if** command.

## Debug Scripts

The PlanetPress Talk Editor has a solid set of features for debugging PlanetPress Talk scripts, including execution control, breakpoints, spies, expression evaluation, and changing the value of a variable during execution. Consult the PlanetPress Talk Editor section of the **PlanetPress User Guide** for a description of these features, and suggestions on other debugging techniques.

PlanetPress Talk includes a breakpoint command that you can enter anywhere you can enter a PlanetPress Talk script. See .

PlanetPress reports errors in PlanetPress Talk code in two places: the status bar of a dialog box when you are entering a PlanetPress Talk expression in a text box within that dialog box, and the Messages area of the PlanetPress Program Window (consult the **PlanetPress User Guide** for a description of the Messages area). You can also use these error reports to determine and fix problems in your code.

## Tips and Tricks

The following are useful to remember as you write your PlanetPress Talk scripts:

- **Defining program logic** As for any coding, when you write programs in PlanetPress Talk you should work through the logic of the program you intend to create before you start coding. This ensures the program accomplishes what you intend it to accomplish, and makes the coding more straightforward. It can also result in insights into different ways of accomplishing the same task, or raise issues that were not immediately apparent.
- **Test** If you are writing a complicated script, it is recommended you use the PlanetPress Talk Editor, and test your code regularly as you develop it. This helps you discover and solve any problems early on in the development process.
- **Simplicity** As in any programming language, aim for both simplicity and clarity in your scripts.
- **Using mathematical operators** Enclose mathematical operators in parentheses ( ) to make your code easier to read and therefore easier to maintain.
- **Performing complex calculations** PlanetPress Talk limits its precision to 1/1000 to optimize speed and thus is not designed to handle complex mathematical computations. If you require greater precision you should perform the calculation outside PlanetPress Talk and include the result in the input data for your document.
- **Case** PlanetPress Talk is case-insensitive. It is good programming practice to establish case conventions as it enhances the readability of the resulting code. It is recommended that you use lower case for all PlanetPress Talk reserved words (names of system variables, operators, and commands) as this is the default case convention of the PlanetPress Talk Editor.
- **Indent** Use indents to visually represent the structure of the script. For example, indent the commands a loop structure encloses to make it apparent where the loop begins and ends.
- **Comments** Include meaningful comments in your code to make it easier to read and maintain.

# Code Samples

This section presents short samples of PlanetPress Talk code that may be useful, and that may also stimulate your imagination for the ways in which you can use PlanetPress Talk in your documents. Consult the **PlanetPress Talk Language Reference** for help understanding specific commands.

### Print a variable number of copies of a page based on a value in the datastream

This sample prints a variable number of copies of a page based on an integer value in the data stream. It uses a global integer variable **&nbPage** that is initialized to 0, and subsequently set to the value of the data that appears on line 1, columns 1 through 2 of the data page.

The document for which it was written uses two pages: DATA and RUNPAGE. DATA is an overlay page that contains all of the data selections for the document. RUNPAGE is a runpage and the code that appears here is in a PlanetPress Talk object on RUNPAGE.

```
% ==================================================
% PRINT A VARIABLE NUMBER OF COPIES OF A PAGE BASED ON
% A VALUE IN THE DATA STREAM
% ==================================================

if(ne(&printermode,0))
 set(&nbPage,strtoint(@(1,1,2)))
 define(&it,integer,0)
```

```
for(&it,1,1,strtoint(@(1,1,2)))
  execpage('DATA')
  showpage()
 endfor()
endif()
```

**Notes:**

- The **if(ne(&printermode,0))** statement prevents the program from executing during design.
- The local variable **&it** is a counter.

## Store two lines of input data on one line of the data page

This sample is a user-defined emulation that reads two lines of input data, concatenates them, and stores them as a single line in the data page buffer. It terminates a data page when it encounters a form feed character. The emulation relies on two global variables:

| Variable: | Type: | Initialized to: | Description: |
|---|---|---|---|
| **&second** | Boolean | False | The emulation uses this to determine when the current line is a second line of data. It is True when the current line is a second line of data. |
| **&line** | string | "" (the empty string) | Holds the contents of a first line of data. |

**Code:**

```
% ====================================================
% STORE TWO LINES OF DATA ON ONE LINE OF THE DATA PAGE
% ====================================================
search(&str,'\014')
 set(&current.line,&current.line + 1)
 store(&current.line,&str)
 doform()
 set(&second,false)
 clearpage()
endsearch()

if(&second)
 store(&current.line,&line + &str)
elseif()
 set(&line,&str)
 set(&current.line,&current.line + 1)
endif()

set(&second,not(&second))
if(ge(&current.line,&current.lpp))
 doform()
 clearpage()
 set(&second,false)
endif()
```

**Notes:**

- The **search** loop executes when the emulation encounters a form feed character. The form feed signals the end of a data page.
- The first **if** statement handles the current line of data. If the current line of data is a second line, the first block of the statement executes; the emulation concatenates the line with the previous one and stores the result. If the current line of data is not a second line, it sets **&line** to the value of this(first) line of data.
- The second **if** statement executes the data page if the value of **&current.line** exceeds the maximum number of lines in a data page.
- The code toggles the value of **&second**. after it reads each line of data so it can determine whether the current line of data is a first or second line.

## Print a line of text on odd-numbered pages

This sample prints a line of text only on odd-numbered pages. It uses the system integer variable **&current.printpage**, and assumes you add a PlanetPress Talk object containing this code to every page in your document.

**Code:**

```
% =========================================================
% PRINT A LINE OF TEXT ON ODD-NUMBERED PAGES ONLY
% =========================================================
if(eq(mod(&current.printpage,2),1))
 margin(1.5,9.75)
 show('See reverse side of this page')
endif()
```

**Notes:**

- If you use a runpage, you would integrate this code into the runpage code.

## Determine the proper page to print based on the width of the data in the data page

In this sample, the document contains three different pages and a runpage that manages execution of the three pages.

Each document page has a different page width and accommodates data of a specific line length. The code determines which document page to use with a given data page based on the length of line 5 of the data page. The three page widths are 198 columns, 132 columns, and 80 columns.

**Code:**

```
% =========================================================
% SELECT THE PROPER PAGE FOR THE CURRENT DATA
% =========================================================
if(trimleft(@(5,198,1))<>'')
 execpage('198ColPage')
elseif()
 if(trimleft(@(5,132,1))<>'')
  execpage('132ColPage')
 elseif()
  execpage('80ColPage')
 endif()
endif()
```

# Language Reference

This section provides complete descriptions of all elements of the PlanetPress Talk scripting language.

Elements appear in alphabetical order, and each is illustrated with code samples you can try out in the PlanetPress Talk Editor of the PlanetPress application.

# Language Reference (Alphabetical)

This section contains a comprehensive reference to all PlanetPress Talk functions, variable, system objects and system variables. Recall that the documentation convention for representing an optional argument to a command is to enclose the argument in square brackets.

## % (procedure)

Comments out the line. This is useful for embedding documentation in your code, making it easier to read and maintain. It is also useful during debugging, for commenting out lines of code you do not want to execute.

Comments must not appear in the first line of the script you enter in the PlanetPress Talk Editor.

**Syntax**

% ... comment ....

**Argument**

None

**Code Sample Example**

This example illustrates comments in PlanetPress Talk code.

```
define(&x,integer,1)
% Creates a new local variable named x that is an
% integer and has a value of 1. The variable is used
% in a loop
for(&x,1,1,10)
        show(inttostr(&x))
endfor()
```

## Current (system object)

This object includes a set of system variables reflecting the current state of the document. All of these variables are read-only and therefore cannot be changed using the set command, except where noted. You can access each property individually using the same syntax.

**Syntax**

&current.property => value

**Properties**

*datafilename*

String value containing the full path and the internal filename, not the original values, of the currently used data file. Note: This variable contains an empty string in printer centric mode.

*datapage*

Integer value containing the page number of the current data page, or the number of the current record set in the case of a database emulation. Note that if you use Metadata and you modify it in any way (sorting, filtering, sequencing) you should use &current.metadatapage instead.

*line*

Integer value containing the line number of the current line of data within the data page, or the number of the current record within the record set. This value is read-write and can be modified using the set command.

*lpp*

Integer value containing the number of lines per page in the current data page. In database emulation, this value can also be used in place of the subreccount() function to determine how many records a record set contains.

*mediacolor*

String value containing the color set in the Media color box of the Page dialog box for the current document page, or, if no value is set in that box, the color set in the Media color box in the Document dialog box for the document. The Media color box appears when you select Media selection in the Selection type box. Recall that the Media color box specifies the color of paper.

*mediatype*

String value containing the type set in the Media type box of the Page dialog box for the current document page, or, if no value is set in that box, the type set in the Media type box in the Document dialog box for the document. The Media type box appears when you select Media selection in the Selection type box. Recall that the Media type box specifies the type of paper (plain, coated, etc.).

*mediaweight*

Integer value containing the weight set in the Media weight box of the Page dialog box for the current document page, or, if no value is set in that box, the weight set in the Media weight box in the Document dialog box for the document. The Media weight box appears when you select Media selection in the Selection type box. Recall that the Media weight box specifies the weight of paper, in grams per square meter.

*metadatapage*

Integer value containing the page number of the current metadata page. This corresponds to the filtered, sorted and sequenced metadata.

*minfeature*

Measure value containing the width of a single pixel at the current zoom level. Precise to 1/65536 inch. This is the minimum size you can use in PlanetPress Talk code. If you use a smaller value (for example as an argument in a command), PlanetPress Talk automatically replaces that value with the value of **minfeature**.

*orientation*

Integer value containing the orientation of the current document page (0=portrait, 1=landscape, 2=Rotated portrait, 3=Rotated landscape).

*overflowcount*

Integer variable storing the number of times that a data page overflowed and that a new document page was added to accommodate the overflowing data. Initially set to 0, is incremented by 1 every time the page overflows.

*overflowing*

Boolean variable set to False by default. When the current *data page* contains more lines than can be printed by the current object, or technically speaking when the object's condition to exit and overflow has been met, this variable is set to True. It is set back to False with every new overflowing page, the logic being that the current overflowing page will be the last, unless if the condition to exit and overflow is met yet again.

*lastoverflowrepetition*

Integer variable storing the number of the last iteration value when the condition to exit and overflow has been met.

*pageheight*

Measure value containing the physical page height, in inches, of the current page in the document.

*printpagename*

String value containing the name of the page that is actually printing. This allows conditional processing to take place within a PPTalk object according to the name of the page being executed. Note that the value always corresponds to the name of the actual page page being printed, even if the call comes from a virtual or overlay page (in other words, the name of the page on which the overlay is layed out).

*pagewidth*

Measure value containing the physical page width, in inches, of the current page in the document.

*printpage*

Integer value containing the current page number in the document.

*x,y*

Measure value containing the current horizontal (x) or vertical (y) position of the drawing pen on the current page. This value is relative to the top left corner of the current page - whose coordinates are (0,0) - and may not necessarily be equal to the current physical position on the page.

## &currentiteration (variable)

Variable storing the number of times that a line was repeated within an object or group. Initially set to 0 in every object, it is incremented by 1 with every line repeat. It can also be used to track repetitions within a group of objects.

### Syntax

&currentiteration (when used as a local variable)

&[GroupName].currentiteration (when used to keep track of repetitions within a group)

## &Document (system object)

This object includes a set of system variables reflecting the current state of the document. All of these variables are read-only and therefore cannot be changed using the set command. You can access each property individually using the same syntax.

**Syntax**

&document.property => value

**Properties**

*name*

String value containing the name of the PlanetPress Document.

*picturecolores*

Integer value containing the default resolution of color pictures, in DPI.

*picturegreyres*

Integer value containing the default resolution of greyscale pictures, in DPI.

*picturemonores*

Integer value containing the default resolution of monochrome pictures, in DPI.

## &EOJob (system variable)

Read-only variable that returns *True* if the document has processed the last line of data or if the line that follows the last form feed character (ASCII 12) is empty.

> This variable always returns *True* at design-time when using a user-defined emulation.

Thus if a file ends with the sequence:

```
FF CR LF
```

**&eojob** becomes true before the last line (CR LF) since the last line is empty.

**Syntax**

&eojob ➡ Boolean value

## &FirstSide (system variable)

Read-only variable that returns True when a page is printing on the first side of a page in duplex mode (double-sided printing). In simplex mode (single-sided) **&firstside** always returns True.

This variable is based on PostScript's **firstside** command as implemented in your printer. If your printer does not support **firstside**, the document attempts to detect which side of the page is currently printing. Passthrough commands can hamper the document's ability to return the correct value for **&firstside**.

**Syntax**

&firstside ➡ Boolean value

## &Height (system variable)

System variable with a scope local to a specific object, group, or page. The system initializes this variable to the value of the Height property of the object's, group's, or page's Basic Attributes. You can reference and set this variable in any PlanetPress Talk code you enter in the object, group, or page.

**Syntax**

&height ⇒ measure value

## &iterationcount (variable)

Integer variable local to the current object that is incremented each time the iteration is executed (when the iteration condition is True).

This variable is generally used to determine when an overflow will be triggered within a simple repeat & overflow.

**Syntax**

&iterationcount ⇒ integer value

Example

This example would be put in the Exit and Overflow box found in the Repeat section of an object or group's properties. In this example, an overflow is triggered whenever the number of repeating lines reaches 15.

```
=&iterationcount >= 15
```

## lastoverflowcount (variable)

Integer variable containing the number of repeats in the previous page of an overflow. For example if you are on the second page of an overflow that contains 6 items each, &lastoverflowcount will be 6 at the beginning of the page, and 12 at the end of the page (after the repeating object).

Note that this variable is not reset when the overflow finishes, meaning on a datafile that has no overflow, its value will still be the one from the last overflow to occur.

Example:

```
if(&current.overflowing)
        show('Showing '+inttostr(&lastoverflowcount) + ' to ' + &totalitemsonpage)
endif()
```

## &Metamode (variable)

⟩⟩⟩ This variable is only available in PlanetPress Suite 7.3 and higher.

The &metamode boolean variable will indicate whether the script is being run while metadata is being created (for example, with the Create Metadata task in PlanetPress Worfklow) or simply being read (with metadata conditions, etc).

**Syntax**

&metamode ⇒ integer value

**Return Values:**

- 0: No metadata, for example a Design document with no data file.
- 1: Metadata is being generated (Create Metadata / Refresh Metadata).
- 2: Metadata is in read mode, such as in PlanetPress Design's workspace.

**Use Cases:**

- Prevent circular logic when adding a page based on the metadata page count. For example, when manually adding a blank page on duplex jobs when the page number is odd, you could add a condition so that this page only "prints" when the `&metamode <> 1`.
- Prevent pages from being added to the metadata. Other than blank pages, you may want to exclude certain pages from the metadata, for example a "Terms and Conditions" page, a header or footer page, separators, etc. By setting its condition to `=&metamode <> 1`, these pages will still print but will not affect metadata page counts.

# Physical (system object)

The **physical** object contains system variables that reflect the current state of the document. These system variables are read-only and therefore cannot be changed using the set command. You can access each system variable using the same syntax.

## Syntax

&physical.x ➡ measure value

&physical.y ➡ measure value

## Variables

*x,y*

Measure value containing the current horizontal (x) or vertical (y) position of the drawing pen, relative to the upper left corner of the physical page, whose coordinates are (0,0). This system object can be used to assess the amount of space left on a page.

# &PrinterMode (system variable)

Returns the current output mode for the document. This allows you to test for a specific value and react accordingly.

## Syntax

&printermode ➡ integer value

**Possible Values**

| Value: | Description: |
|--------|--------------|
| 0 | PlanetPress during document design |
| 1 | Preview or Softproof as Printer Queue |
| 2 | Preview or Softproof as PlanetPress Image |
| 3 | Preview or Softproof as PlanetPress Fax |
| 4 | Printer Preview |
| 5 | Printer Hard Disk |
| 6 | Printer Memory |
| 7 | Host Computer |
| 8 | Flash Memory |
| 9 | PlanetPress Image |
| 10 | PlanetPress Fax |
| 11 | PlanetPress Suite Workflow Tools |

**Code Sample Example**

This example changes some colors to shades of gray for greater legibility on a fax.

**Example**

```
setlinewidth(0.1) %Select a thick pen
if(&printermode=10) %If Destination is Fax
        setfillcolor([5,5,5,5]) %Set fill to light gray
        setstrokecolor([0,0,0,100]) %Set pen to black
elseif()
        %Otherwise,set fill colour to yellow
        setfillcolor([0,0,50,0])
        setstrokecolor([100,100,0,0]) %Set pen colour to blue
endif()
rectstroke(0,0,3,3) %Display rectangle
rectfill(0,0,3,3)
```

# Script (system object)

The script system object is available in both PlanetPress Design and PlanetPress Suite Workflow Tools scripting environments. It contains properties describing the current external script.

> &script is not used within PlanetPress Design and is not a PlanetPress Talk variable. It is a variable used within scripts in JScript, VBScript, Perl or Python called from PlanetPress Talk in a PlanetPress Design Document, using the "Exec-ScriptFile (function)" (page 484) command.

**Syntax**

- &script.filename ⇒ string
- &script.paramstring ⇒ string
- &script.returnvalue ⇒ string

**Properties**

*filename*

Read-only string value containing the fully qualified path and filename of the current script.

*paramstring*

Read-only string containing additional information passed by the document to the script.

*returnvalue*

Read-write integer containing a coded value for the result of the ["ExecScriptFile (function)" (page 484)](#) command.

> Accessing custom PlanetPress Suite Workflow Tools variables from a currently running script within PlanetPress Design will yield unknown results, simply because PlanetPress Design does not have any access to the PlanetPress Suite Workflow Tools custom variables. Users should therefore be careful to use conditional statements within the script if needing to access custom variables.

# &Str (system variable)

Contains a line of input data.

**Syntax**

&str ⇒ string

**Code Sample Example**

This example illustrates a few lines of code from a user-defined emulation. In these lines, if the line of input data (minus any leading spaces)begins with the code '5', the emulation advances to the next line of the data page and stores the line of input data on that line. Notice the &current.line system variable—one of the variables in the current system object.

**Example**

```
if(eq(mid(trimleft(&str),1,1),'5'))
  set(&current.line,&current.line + 1)
  store(&current.line,trimleft(&str))
endif()
```

# &system (system object)

The &system system object contains system variables related to the Raster Image Processor (RIP) the document is using to execute, and document version. These system variables are read-only.

**Syntax**

- &system.product ⇒ string value
- &system.version ⇒ string value
- &system.formversion ⇒ integer value

**Properties**

*product*

String value containing the name of the RIP the document is using to execute. For example,

PlanetPress Alambic is the name of the RIP built in to PlanetPress (the RIP PlanetPress uses when you perform a preview and select Internal interpreter as the PostScript interpreter).

*version*

String value containing the version number of the RIP the document is using to execute. For example, 4.0.

*formversion*

Integer value containing the version number of the document requested by the trigger. Recall that you can assign a version number to a document and increment the number each time you update and reinstall the document. When you execute a document that uses a version number, the document verifies its version number against the one you specify in the trigger, and it proceeds with execution only if the two version numbers match.

## &watch (system object)

The &watch object currently contains a single property, the JobInfos array. This array contains the 9 job information variables that are sent to the document from PlanetPress Workflow, when the document is executed in a PlanetPress Workflow process.

### Syntax

- &watch.JobInfos[n] ⇒ string value

*Note: n is a digit from 1 to 9, corresponding to the PlanetPress Workflow Job Info.*

### Properties

*JobInfos[n]*

String value equal to the value of job info *n*, as set in PlanetPress Workflow.

## &Width (system variable)

System variable with a scope local to a specific object or group. The system initializes this variable to the value of the Width property of the object's or group's Basic Attributes. You can reference and set this variable in any PlanetPress Talk code you enter in the object, group, or page.

### Syntax

&width ⇒ measure value

## @ (function)

Returns a selection within the current data page for text emulations.

> The @ character can be the beginning of a [data selection](#), a [page call](#) or a [procedure or function call](#).

### Syntax

@( line, startcolumn, endcolumn ) ⇒ string value

### Arguments

*line*

Integer value specifying the line on which to read the data in the current data page.

*startcolumn, endcolumn*

Integer values specifying the start/end columns of the chunk of data to read from the current data page.

**Code Sample Example**

The first line of code assigns data from the current data page to a variable. The second line of code checks for the presence of a value inside the data page and sets a Boolean variable according to the results.

```
define(&invnum,string,@(7,50,59))
define(&isfirstpage,boolean,(strtoint(@(1,60,70))=1))
```

# @name (function/procedure)

Executes a global function or procedure created using the function() command.

> The @ character can be the beginning of a data selection, a page call or a procedure or function call.

**Syntax**

@functionname( parameters ) ⇒ integer, measure, currency, string, Boolean or no return value

**Arguments**

*functionname*

String value specifying the name of the function or procedure.

*parameters*

Comma separated list of parameters required by the specified function or procedure.

# @page (procedure)

Executes the content of the specified document page, as if it were part of the calling page. The content of the page in this case excludes the paper handling properties of the page. The height and width of the page content thus becomes the height and width of the group that would result if you created a single group of all of the page elements. If you want to include the paper handling properties in the page execution, use the *$element* syntax instead.

> The @ character can be the beginning of a data selection, a page call or a procedure or function call.

> The @*page()* and *execpage()* commands are functionally equivalent. However, the *execpage()* command has the added ability of calling variable page names.

**Syntax**

@pagename()

**Arguments**

*pagename*

String value specifying the name of the page you want to execute.

**Example**

The following two lines of code are functionally equivalent and display the content of a page named *MyFirstPage*:

```
@MyFirstPage
execpage('MyFirstPage')
```

## $element (procedure)

Executes the specified document element (object, page, resource, etc.). Note that in the case of a page, this procedure executes both the content of the page and the paper handling properties of the page. The height and width of the *calling* page are therefore set to the same values as those set by the *called* page.

Recall that every element in a document has a unique name. You can therefore call any element in the document at any time. For example, the first page of a medical record document may contain a header listing all of the patient information, while other pages in the document scale that same header to fit in the lower left corner of each page. Any modification you subsequently make to the header is automatically reflected throughout the document.

**Syntax**

*$element*

**Arguments**

*element*

String value specifying the name of the element you want to execute.

**Examples**

Example 1 illustrates basic usage of the syntax.

```
$my_square
$page1
$box5
```

Example 2 scales the *$header* element to one quarter of its original size, and rotates it 90 degrees.

```
scale( 0.25, 0.25 )
setangle( 90 )
$header
```

Example 3 creates a thumbnail of each of the first two pages and positions the thumbnails side-by-side.

```
scale( 0.10, 0.10 )
$page1
translate( 8.5, 0 )
$page2
```

## + (operator & function)

Either concatenates two or more strings or adds two numerical expressions, depending on its context.

This description details how to use it to concatenate two strings. See <u>"Add (function)" (page 549)</u> for help using it to add two numerical expressions.

**Syntax**

string1 + string2 + ...  ⇒  string

**Arguments**

*string1, string2,…*

String values.

**Code Sample Example**

The following code sample concatenates five strings into a single string that contains the first and last name of a customer.

**Code Sample**

```
show('First Name: ' + @(2,12,36) + ' ' + 'Last Name: ' + @(3,12,36))
```

# – (operator)

See <u>"Sub (function)" (page 537)</u> and <u>"Neg (function)" (page 533)</u>.

# Asterisk (*) operator

See <u>"Mul (function)" (page 533)</u>.

# / (operator)

See <u>"Div (function)" (page 530)</u>.

# Greater than (>) operator

See <u>"GT (function)" (page 552)</u>.

# Greater or equal to (>=) operator

See <u>"GE (function)" (page 551)</u>.

# Less than (<) operator

See <u>"LT (function)" (page 554)</u>.

# Less or equal to (<=) operator

See <u>"LT (function)" (page 554)</u>.

# = (operator)

Assign a value to a variable. This provides a more convenient alternative to the **set()** command. Note that spaces between the operator and its operands are optional.

**Code Samples**

```
&price := 18.53
&day[4] := 'Friday'
&last:=True
&mustard := [0,20,90,16]
```

# <> (operator)

See

# Add (function)

Adds two numerical expressions. This is the functional equivalent to the + operator, when you use the + operator with numerical expressions.

## Syntax

add( expression, expression2 ) ⇒ integer, measure, currency value

## Arguments

*expression1, expression2*

Integer, measure, or currency values. Both expressions must be the same type. The returned value is set according to the type of the parameters.

## Code Sample Examples

Examples 1 and 2 show both ways of adding numbers.

## Example 1

```
show(inttostr(2+2)) %Displays 4
```

## Example 2

```
show(floattostr(add(4.7,2.1))) %Displays 6.8
```

# And (Boolean operator function)

Returns true if both specified expressions are true, otherwise it is false.

## Syntax

and( expression1, expression2 ) ⇒ Boolean value

expression1 and expression2 ⇒ Boolean value

## Arguments

*expression1, expression2*

Boolean values.

**Code Sample Examples**

These examples illustrate the possible values for AND.

```
and(true, false) %Returns false
and(false, true) %Returns false
and(false, false) %Returns false
and(true, true) %Returns true
```

# Arc and ArcN (procedures)

Arc() draws an arc in a counter-clockwise direction, while ArcN() draws an arc in a clockwise direction. If there is a current point set, the command draws a straight line from the current point to the start point of the arc. Whether or not a current point is set when the command executes, after execution the current point is the end point of the arc.

You define the arc you want to draw by specifying the x and y coordinates of the center of the circle, the radius of the circle, and the start and end points for the arc. You specify each of the start and end points of the arc as an angle; the command uses the angle to position the point. For example, to position the start point, the command draws an invisible line at the specified start angle from the center of the circle, the length of the radius; it positions the start point at the end of that line.

| | |
|---|---|
| Example using Arc() | Example using ArcN(). |

Example of an arc drawn from start point D to end point F: A. Center of circle (x,y) B. Radius C. Start angle D. Start point of arc E. End angle F.

**Syntax**

arc( x, y, arc_length, start_angle, end_angle )

**Arguments**

*x, y*

Measure values representing the x and y coordinates respectively of the center of the circle.

*arc_length*

Measure value representing the length of the arc.

*start_angle*

Measure value representing the start angle, in degrees, for the arc. Degrees are relative to a standard X, Y axis, with a value of 0 degrees lying flush with the X axis. The value of the start angle can be either positive or negative.

*end_angle*

Measure value representing the end angle, in degrees, for the arc. Degrees are relative to a standard X, Y axis, with a value of 0 degrees lying flush with the X axis. The value of the end angle can be either positive or negative.

**Code Sample Examples**

**Example 1**

Example 1 draws a 180 degree arc 3 inches in length. The center of the circle is at (0,0). The **rmoveto()** command that precedes the **arc()** command ensures the current point is set at the center of the circle and prevents a line from being drawn from the current point to the start point of the arc.

```
rmoveto(3,0)
arc(0,0,3,0,180)
stroke()
```

**Example 2**

Example 2 draws an arc 2.5 inches in length. The center for the arc is four inches from the left edge of the page and three inches from the top edge. The arc starts at 30 degrees and ends at 60 degrees. The **rmoveto()** command that precedes the **arc()** command moves the current point to the center of the circle, resulting in a line that is drawn from the center of the circle to the start point of the arc.

```
rmoveto(4,3)
arc(4,3,2.5,30,60)
stroke()
```

# BeginDocument/EndDocument (procedure)

Delimits the boundaries of a metadata document within a PlanetPress Design document.

**Syntax**

begindocument()

or

enddocument()

**Arguments**

None

**Code Sample Examples**

```
if (&is_FirstPage)
      begindocument()
EndIf()
if(&is_LastDocPage)
```

```
        enddocument()
EndIf()
```

## BeginGroup/EndGroup (procedure)

Delimits the boundaries of a metadata group within a PlanetPress Design document.

**Syntax**

begingroup()

or

endgroup()

**Arguments**

None

**Code Sample Examples**

```
if(&is_FirstInvPage)
        begingroup()
EndIf()
if(&is_LastInvPage)
        endgroup()
EndIf()
```

## BeginParagraph ... EndParagraph (procedure)

Delimits a paragraph of formatted text.

The **beginparagraph** ... **endparagraph** structure can contain only the following commands: **setstyle()**, **setstyleext()**, **show()**, and **%** (indicating a commented line). Any other command included within this structure will yield unpredictable results. A **margin()** command should precede the **beginparagraph...endparagraph** structure, and each variable you reference within the structure must have its own procedure., as demonstrated in the second of the examples below. Also note that the first command within a **beginparagraph...endparagraph** structure must be the **setstyle()** command.

No **crlf** procedures are permitted inside a paragraph structure, since **crlfs** are paragraph delimiters.

If you do not need to change fonts or use variables within a paragraph, it is strongly recommended you use the text object in PlanetPress, as PlanetPress optimizes text objects to improve performance.

When a long string containing no spaces between its characters appears in the context of a **show()** command and the **beginparagraph** ... **endparagraph()** command, the line of text will not wrap. The same is true for a text object; no wrapping occurs when a long string has no spaces between its characters.

Note that when a Text object is converted to PlanetPress Talk, an extra argument is is added to the **beginparagraph** ... **endparagraph** syntax. Ignore this argument since it is strictly for internal use.

**Syntax**

beginparagraph( lmargin, rmargin, firstindent, align, leading,[opt, forcedwordwrap])

...

endparagraph()

## Arguments

*lmargin*

Measure value specifying the left margin, in inches, of the text relative to the left border of the object.

*rmargin*

Measure value specifying the right margin, in inches, of the text relative to the left border of the object.

*firstindent*

Measure value specifying the indent, in inches, of the first line of the paragraph, relative to the lmargin parameter.

*align*

String value specifying the text alignment within the paragraph. Possible values are 'left', 'right', 'center' and 'leftright' (for both left and right justification of text).

*leading*

Measure value specifying the amount of vertical space, in inches, between each line of the paragraph.

*opt*

Optional boolean value used internally. Default value is false.

*forcedwordwrap*

Optional boolean value that forces wordwrap at the defined paragraph size. Default value is false.

## Code Sample Examples

Example that displays a paragraph aligned to the right.

## Example 1

```
beginparagraph(1,3,0,'right',0.16)
       setstyle(&Default)
       show('This long line of text should wrap around')
       show('This long line of text should wrap around')
endparagraph()
```

Example that shows how to include a variable within a formatted paragraph.

## Example 2

```
define(&var,string,'very')
beginparagraph(1,3,0,'left',0.16)
       setstyle(&Default)
       show('This ')
       show(&var)
       show(' long line of text should wrap around')
endparagraph()
```

# BeginUTF8Paragraph ... EndUTF8Paragraph (procedure)

Delimits a paragraph of formatted UTF8 text.

This procedure is based on BeginParagraph ... EndParagraph (procedure), with the following differences:

- ShowUTF8 (procedure) must be used instead of show()
- SetStyleExt (procedure) must use a style that has been associated with the True Type Font "TT Host UTF8Arabic".
- Only UTF8 text can be displayed in a PlanetPress Talk object defined using the **beginUTF8paragraph** ... **endUTF8paragraph** structure. Non-UTF8 static text should therefore be converted to its UTF8 reference using the Ansi/UTF8 Converter (place the cursor within the string and press CTRL+N). Variable text should be converted using MapUTF8 (function).

## Syntax

beginUTF8paragraph( lmargin, rmargin, firstindent, align, leading, majormode, [forcedwordwrap] )

% Your Paragraph data here

endUTF8paragraph()

## Arguments

Same as BeginParagraph ... EndParagraph (procedure).

## Code Sample Example

This example displays a justified paragraph with the UTF8 text running from right to left.

## Example

```
setlinewidth(0.007)moveto(0,0)
rlineto(&width,0)
rlineto(0,&height)
rlineto(neg(&width),0)
rlineto(0,neg(&height))
closepath()
stroke()
moveto(&width/2,&height/18)
BeginUTF8Paragraph(0.0000,&width,0.0000,'leftright',0.1667,'rtl')
       SetStyleExt(&Style1,12.0000,0,[100],100)
       ShowUTF8('\u0623\u0633\u0627\u0633\u064B\u0627\u060C')
EndUTF8Paragraph()
```

# BitmapWidth/BitmapHeight (function)

Returns the width or height, in inches, of a bitmap image resource, at the specified resolution.

## Syntax

Bitmapwidth( name, resolution ) ⇒ measure value

Bitmapheight( name, resolution ) ⇒ measure value

## Argument

*name*

String value that specifies the name of the bitmap image resource.

*resolution*

Integer value that specifies the resolution at which you want to measure the width or height of the bitmap.

**Code Sample Example**

This example sets the variable maxwidth to the width of the bitmap 'fingerprint' at a resolution of 150 DPI, and the variable maxheight to the height of the same bitmap at a resolution of 200 DPI.

**Example**

```
&maxwidth := bitmapwidth( 'fingerprint', 150 )
&maxheight := bitmapheight( 'fingerprint', 200 )
```

# Breakpoint (procedure)

Stops the execution of a PlanetPress Talk object when the specified expression is True. This breakpoint is effective only when running in debug mode in the PlanetPress Talk editor.

**Syntax**

breakpoint( expression1 )

**Argument**

*expression1*

Boolean value.

**Code Sample Example**

This example sets up a loop, stops the loop on its 5th iteration, and then displays the value.

**Example**

```
define(&x,integer,1)for(&x,1,1,10)
      breakpoint(&x = 5)
      show(inttostr(&x))
endfor()
```

# C128 (function)

>>> This function is deprecated and should be replaced by ShowBarcodeCode128 (procedure) whenever encountered. It is kept for backwards compability.

Converts a two-character string to Code 128 bar code, character set C data.

**Syntax**

c128( string ) ➡ string value

**Argument**

*string*

String value composed of two characters.

**Code Sample Example**

This example converts the data on line 5, columns 12 to 13, to Code 128 bar code C character set data.

**Example**

```
c128(@(5,12,13))
```

# CallPPD (procedure)

Calls a section of the PostScript Printer Description (PPD) file and executes it on the printer. Use this to set printer options such as the input tray or duplexing options. The tokens and subtokens you use as arguments with this function must be defined in the PPD file associated with your document or page in the document's or page's Basic Attributes. PlanetPress searches the PPD file for the token-subtoken combination and then prompts the printer to execute the corresponding code.

**Syntax**

callppd( token, subtoken )

**Arguments**

*token*

This is the string to search for in the PPD file (for example, Duplex).

*subtoken*

This is string to search for that is a setting for the token (for example, None).

**Example**

```
if((&Condition1) or (&Condition2))
      callppd("Tray", "2")
elseif()
      if((&Condition3) or (&Condition4))
             callppd("Tray", "3")
      elseif()
             callppd("Tray", "4")
      endif()
endif()
```

# Ceil (function)

Returns the smallest integer greater than or equal to the specified measure value.

**Syntax**

ceil( value ) ➡ integer value

**Argument**

*value*

Measure value. The absolute value of the measure value must be less than the maximum value of an integer in PostScript (2,147,483,647).

**Code Sample Examples**

These examples illustrate various applications of **ceil()**.

```
ceil( -2.8 ) %Returns -2
ceil( 2.8 ) %Returns 3
ceil( -5.0 ) %Returns -5
```

# Char (function)

Returns the character whose ASCII value is specified.

**Syntax**

char( value )  ⇒  string value

**Argument**

*value*

Integer value between 0 and 255, specifying the ASCII index of the character.

**Example**

```
show(char(84)+char(97)+char(108)+char(107))
% This displays the word "Talk"
```

# CharPath (procedure)

Creates a path in the shape and size of a text string, in the currently active style.

**Syntax**

charpath(string)

**Argument**

None

**Example**

This example creates a path of the word "Text" and applies it as a clipping path on the coffee_time bitmap. This creates the word Text where the filling texture is the cofee_time image.

```
moveto(&width / 2, &height / 2)
charpath('TEST')
clippath()
moveto(0,0)
showbitmap('coffee_time',&document.picturecolorres,&width,&height)
```

**Result:**

# TEST

## ClearPage (procedure)

Clears the current data page and loads the next one. This command should only be used before reading the next page of data from the input stream. It is strictly for use within a User-Define Emulation and generally follows DoForm (procedure).

**Syntax**

clearpage()

**Argument**

None

Example:

This example the default User-Defined Emulation that corresponds to a Line Printer data file.

```
search(&str,'\014')
        set(&current.line,&current.line + 1)
        store(&current.line,&str)
        doform()
        clearpage()
endsearch()
set(&current.line,&current.line + 1)
store(&current.line,&str)
if(ge(&current.line,&current.lpp))
        doform()
        clearpage()
endif()
```

## ClipPath (procedure)

The Clippath() command takes the last drawn path and selects it as the clipping region. Any subsequent drawing will only be visible inside that clipping region. The clipping will stop at the next call to GRestore(). All PlanetPress objects are natively wrapped in a GSave/GRestore(), so clipping between objects will not work.

**Syntax**

clippath()

**Argument**

None

**Example**

This example displays the coffee_time image partially, where only a square defined by the rlineto() is visible.

```
moveto((2.0000/(6.6529)) * &width, (0.2000/(4.9897)) * &height)
rlineto(0, (3.0000/(4.9897)) * &height)
rlineto((3.0000/(6.6529)) * &width, 0)
rlineto(0, neg((3.0000/(4.9897)) * &height))
closepath()
clippath()
moveto(0, 0)
showbitmap('coffee_time',&document.picturecolorres,&width,&height)
```

# ClosePath (procedure)

Closes any open path and completes the current shape, if need be, by drawing a line from the last point in the shape to the starting point. This ensures the shape is closed, thus enabling filling to take place if specified. Only closed shapes can be filled.

**Syntax**

closepath()

**Argument**

None

**Code Sample Example**

The first line of code sets the starting point of a triangle, the second and third lines of code draw the first and second lines (or sides) of the triangle, the fourth line of code closes the triangle shape by implicitly issuing a **moveto** command.

**Example**

```
moveto(1.0,1.0)lineto(1.5,2.0)
lineto(0.5,2.0)
closepath()
fill()
```

# Cos (function)

Returns the cosine value of the specified angle.

**Syntax**

cos( value ) ⇒ measure value

---

PLANETPRESS. SUITE

**Argument**

*value*

Measure value specifying the angle whose cosine is returned.

**Code Sample Example**

This example displays a sinusoidal graph.

**Example**

```
moveto(0,0)
define(&i,integer,0)
for(&i,1,10,360)
        lineto(cos(inttofloat((&i)/ 4)),sin(inttofloat(&i)))
endfor()
closepath()
```

# CRLF (procedure)

Moves the current point by simulating a carriage return/line feed combination. The horizontal position is reset to the value of the current left margin, while the vertical position is offset by the leading value specified.

**Syntax**

crlf( [leading] )

**Argument**

*leading*

Measure value setting the vertical distance, in inches, between two lines. If you do not provide this argument, the value by default is 0.167 inches, which is roughly equivalent to a 6 LPI (lines per inch) setting.

**Code Sample Example**

This example demonstrates how to change the vertical spacing of lines within an object.

**Example**

```
margin(1,1)
show('These lines are displayed')
crlf(0.125)
show('at 8 LPI.')
crlf(0.5)
show('While these lines are displayed')
crlf(0.5)
show('at 2 LPI')
```

# CurToStr (function)

Converts a currency expression into a string value.

**Syntax**

curtostr( expression ) ⇒ string value

**Argument**

*expression*

Currency value to convert.

## CurToFloat (function)

Converts a currency expression into a measure value.

**Syntax**

curtofloat( expression ) ⇒ measure value

**Argument**

*expression*

Currency value to convert.

## CurveTo/RCurveTo (procedure)

Creates a Bezier curve. Bezier curves are shapes defined using 4 points: the starting and ending points are physical positioning points while the second and third points are control points on the curve. In PlanetPress Talk, the starting point is implicit and automatically set to the current cursor position. The first control point defines the direction the curve takes when moving from the starting point, and before shifting to the second control point, which defines a second curve before reaching the ending point.

A complete explanation of Bezier curves is beyond the scope of this document. Feel free, however, to try out this command using the PlanetPress Talk Editor in PlanetPress in order to study its possibilities.

To draw a simple curve, use the same (x,y) coordinates for both control points.

**Syntax**

curveto( x1, y1, x2, y2, x3, y3 )

**Arguments**

*x1, y1, x2, y2, x3, y3*

Pairs of measure values representing the horizontal/vertical position, in inches, of each point defining the Bezier curve. When using rcurveto, these values are relative to the current point of origin. When using curveto, they are absolute values.

**Code Sample Example**

This example draws a triangle with a twisted bottom side. The triangle has a blue outline and is filled with yellow.

**Example**

```
setstrokecolor([100,100,0,0])
setfillcolor([0,0,100,0])
moveto(1,1)
lineto(1.5,2)
```

PLANETPRESS. SUITE

```
curveto(1,1,1,3,0.5,2)
closepath()
strokeandfill()
```

**Result:**



## Date (function)

Returns the print date. Note that this function does not work in Printer-Centric mode, since it is impossible for the document to get the current date from a printer. So if you send your document to a printer and then simply send data with the appropriate trigger to that printer, the document will run on the printer and the function will return an empty string. Use the Run locally option, available in the PlanetPress Suite Workflow Tools, to ensure that the document runs on a computer rather than on a printer.

**Syntax**

date(longdate)

**Argument**

*longdate*

Boolean specifying whether to use short or long date. True returns the date in long format, False, in short format. The precise format of the short and long date value strings are set in the Windows Regional options.

**Code Sample Example**

This example shows how to add the current date in long form on a document page.

**Example**

```
show(date(true))
```

**Result**

On a system set to French (Canada), returns .

# Define (procedure)

This command creates a new local variable, or redefines an existing one. The define procedure must appear before the variable is actually used. PlanetPress Talk scripts require all variables to be defined either as a global variable in your document, or a local scope variable within your PlanetPress Talk script.

## Syntax

define( name, type, value )

## Arguments

*name*

The name of the variable, prefixed with the ampersand (&) character. The name must conform to the rules for names described in ["Names" (page 219)](#).

*type*

Constant value specifying the variable type, either integer, measure, string, Boolean, or an array type. The array types are color, arrayinteger, arraymeasure, arraycurrency, arrayBoolean, arraystring or directory. Once a type is assigned to a variable, it cannot be changed, unless the variable is redefined.

*value*

An initial value for the variable, of the same type as the variable (for example if the variable is of type integer, this value must be an integer). This value can also be a valid PlanetPress Talk expression. In the case of any array variable other than a color array or an array of type directory, the initial value can be either the number of elements you want the array to contain, or the set of values you want the array to contain. In the former case you assign values to the array elements in subsequent commands. In the latter case, you separate each value by a comma, and enclose the complete set of elements in square brackets. In the case of a string array, you must also quote each of the array elements. In an array of type directory, you specify the pathname to the folder and the file name filter you want to apply to each of the files in that folder; this in turn determines the number of elements in the array.

## Code Sample Example

```
define( &customer_name, string, 'Smith' )
define( &invoice_number, integer, strtoint(@(1,1,10)) )
define( &left_margin, measure, 1.5 )
define( &is_bottom, boolean, (&current.line > 50) )
define( &my_array,arrayinteger,[12,3,76,109,4] )
define( &my_array,arrayinteger, 5 )
define( &prices,arraycurrency, 6)
define( &greetings,arraystring,['hello','bonjour','ola'] )
define( &mustard,color,[0,12,84,16] )
define( &image_paths, directory, 'c:\\images\\*.JPG' )
define( &cost, currency, 0.00 )
```

# DefineData (procedure)

Change the active data file to the one specified in the *path* parameter.

Changing the data file in the middle of document processing can have unexpected consequences and should only be

used if you understand the implications of doing so.

**Syntax**

definedata( path )

**Argument**

*path*

String value specifying the path of the data file.

## DefineImageIndex (procedure)

Defines a PlanetPress Search index term. PlanetPress Image uses this information to generate the .PDI file it creates for each PDF file it creates. You use SetImageIndex (procedure) to associate a data value with the index term.

**Syntax**

defineimageindex( name, [length] )

**Arguments**

*name*

String value specifying a name for the index term.

*length*

String value specifying a length for the index term.

**Code Sample Examples**

**Example 1**

This example defines the index term 'PONumber' and assigns it a length of 8 characters. The PDI file generated by PlanetPress Image contains the line: ~IndexName=PONumber,8.

```
defineimageindex( 'PONumber', '8')
```

**Example 2**

This example defines the same index term, this time without defining a length for the term. In this case the line in the PDI file becomes: ~IndexName=PONumber

```
defineimageindex( 'PONumber' )
```

## Definemeta (function)

Defines a field when running a PlanetPress Design Document with metadata file creation. In every cases, the name of the field is case-insensitive and must begin with a letter (A-Z, a-z), followed by any number of letters, numbers (0-9), underscore ('_') or dash ('-') characters.

**Syntax**

definemeta(name: string, data: string, [flags: integer, path: string])

**Argument**

*name*

The name of the metadata field to define. Unless the **path** argument specifies otherwise, the new field is defined at *page level*.

*data*

An initial string value of the defined metadata field. This value can also be a valid PlanetPress Talk expression.

*flags*

Optional integer value to define the behavior of the command when a field with the same name already exists in the current level. The default behavior uses 0 and overwrites the old value with the new one.

| Name | Value | Behavior |
|------|-------|----------|
| DefineOverwrite | 0 | The content of the **value** parameter overwrites the current value of the field if it exists, or creates it if it does not. |
| DefineAppend | 1 | The content of the **value** parameter is appended to the existing field. |
| DefineDuplicate | 2 | A new field with the same name is created, appending an index at the end of its name. |
| DefineError | 3 | The command fails and the job crashes, yielding an error dialog. |

*path*

Optional string value to specify the level where to create the metadata field. Path components cannot be indexed, and if no path is provided, the field is created at the current page level.

> The node names of the path argument can have an optional zero-based index, indicated between square brackets ('[ ]'), to access a specific node. If no index is provided, the first node is used.

**Code Sample Example**

This example illustrates how to assign data from the current line printer data file to a document-based metadata field, specifying the level where the field will be created.

```
% Create simple 'InvoiceNumber' holding the value found on line 10,
% between columns 35 and 42 of a line printer emulation data file.
definemeta('InvoiceNumber', @(10,35,42))
% Create the same 'InvoiceNumber' field but duplicating it if it already exists.
definemeta('InvoiceNumber', @(10,35,42), 2)
% Create the same 'InvoiceNumber' field but at the Document level this time,
% and raising an error if it already exists.
definemeta('InvoiceNumber', @(10,35,42), 3, 'Job.Group.Document')
```

# Div (function)

Divides an expression with another. This is the functional equivalent to the / operator.

**Syntax**

div( expression, expression2 ) ⇒ integer, measure, currency value

**Arguments**

*expression1, expression2*

Integer, measure or currency values. Both expressions must be the same type. The returned value is set according to the type of the parameters.

**Code Sample Examples**

Examples 1 and 2 show both ways of dividing numbers.

**Example 1**

```
show(inttostr(2/2)) %Displays 1
```

**Example 2**

```
show(floattostr(div(4.7,2.1))) %Displays 2.2381
```

# DoForm (procedure)

Runs the current document. This command is usually issued once a complete data page has been received and committed to buffer. This command should only be used before reading the next page of data from the input stream. It is strictly for use within a User-Define Emulation.

**Syntax**

doform()

**Argument**

None

**Code Sample Example**

This example is an extract from a user-defined emulation. The search() command looks for a formfeed and adds its line number. The line number is stored as a string and the document is run. The data page is cleared and the search is over.

**Example**

```
search(&str,'\014')
      set(&current.line,&current.line + 1)
      store(&current.line,&str)
      doform()
      clearpage()
endsearch()
```

# endpageset (procedure)

Ends any page set and subset when printing using a Windows driver. When creating a VDX PlanetPress Design document, this command signals the end of a booklet rather than a simple page subset.

**Syntax**

endpageset()

**Argument**

None

**Code Sample Examples**

In both modes, the command is called the same way.

```
% Print using a Windows driver mode
if(&is_CA)
  endpageset()
  selectprinter('MyCanadianPrinter')
else
  endpageset()
  selectprinter('MyOtherPrinter')
endif()

% Work with VDX
if(&is_LastBookletPage)
  endpageset()
endif()
```

## EPSWidth/EPSHeight (function)

Returns the width or height, in inches, of an Encapsulated PostScript (EPS) image resource.

**Syntax**

epswidth( name ) ⇒ measure value

epsheight( name ) ⇒ measure value

**Argument**

*name*

String value that specifies the name of the EPS image resource.

**Code Sample Example**

This example sets the variable maxwidth to the width, and the variable maxheight to the height of the EPS image resource wing_nut.

**Example**

```
&maxwidth := EPSwidth('wing_nut')
&maxheight := EPSheight('wing_nut')
```

## Eq (function)

Compares two expressions of any type and returns true if both are equal, false otherwise. This is the functional equivalent of the = operator.

**Syntax**

eq( expression1, expression2 ) ⇒ boolean value

**Arguments**

*expression1, expression2*

Values of any type. Both expressions must be of the same type.

**Code Sample Examples**

Examples 1 and 2 shows both ways of comparing two expressions.

**Example 1**

```
if( eq(&current.line, 33) )
  show('Middle of page')
endif()
```

**Example 2**

```
if(&current.line = 33)
  show('Middle of page')
endif()
```

## ExecPage (procedure)

Executes the content of the specified document page, as if it were part of the calling page. The content of the page in this case excludes the paper handling properties of the page. If you want to include the paper handling properties in the page execution, see **"$element (procedure)" (page 611)**.

The **execpage()** and **@page()** commands are equivalent.

**Syntax**

execpage( pagename )

**Argument**

*pagename*

String value specifying the name of the page to run. The page must exist, and its name is case insensitive.

**Code Sample Example**

In this example, the current page executes pages 2, 3 or 4 according to a data selection. This allows you to create pages of 'subroutines' that can be called from any other page.

**Example**

```
define(&x,integer,strtoint(@(7,4,8)))
if(&x < 1000)
  execpage('PAGE1')
elseif()
  if(&x > 1000)
      %This is equivalent to execpage('PAGE3')
    @PAGE3
  elseif()
    execpage('PAGE2')
  endif()
endif()
```

## ExecScriptFile (function)

Allows PlanetPress Design documents to execute the content of an external script file. The document waits until the script has completed before resuming its own execution.

**Note**: This function does not support printer centric mode; results are undefined.

### Syntax

ExecScriptFile(filename: string, parameters: string, scriptlanguage: integer) ➡ integer

### Arguments

*filename*

Fully qualified path and name of the script to call and execute.

*parameters*

User-defined string value allowing a document to pass additional information to the script, such as data selections, PlanetPress Talk variables, etc..

*scriptlanguage*

The programming language in which the script is written. The valid values are:

0 Auto-detect based on file extension: VBScript (.vbs) | JavaScript (.js) | Perl (.pl) | Python (.py)
1 VBScript
2 JavaScript
3 Perl
4 Python

### Return value

Successful execution of the external script file will return 0 by default, although this value may be set by users through the "Script (system object)" (page 247) system variable. Note that some negative values are internally used to indicate different types of failure, always using negative values, which means conflicts may happen when users set the *Script.ReturnValue* variable with an existing negative value. The proper practice is thus to return only positive values.

Here are the existing negative values:

-1 Script file does not exist or is not accessible.

-2 Invalid script language.

-3 Error during script execution.

## Use of PlanetPress APIs within called scripts

When a script is called from within PlanetPress Design using ExecScriptFile(), all of the PlanetPress environment is available to the script, including the Watch, Metadata, Capture and Alambic objects.

> Using these objects from a script within a Design document is not supported and can lead to unexpected results. Furthermore, setting a Watch variable or job info from within the script will not return this value to the document, as these values are always passed once when the document is called, never during execution.

Accessing Parameters from the script

The parameters value is a string which is sent to the executed script. It is available using the Script.Paramstring system variable from within the code. If you want to send multiple parameters, you will have to separate them with a special character and use this character as a separator to split the string or create an array with it, such as (example in JavaScript):

```
params = Script.Paramstring.split(";")
```

## Code Example

```
ExecScriptFile('c:\myScript.vbs', 'name;'+@(1,1,10), 1)
```

## Sample Scripts

2 sample scripts are available with any PlanetPress Suite installation.
They are are located in *C:\Documents and Settings\All Users\Application Data\Objectif Lune\PlanetPress Suite N\Scripts*, where N is the PlanetPress Suite version number. These 2 sample scripts are used by PlanetPress Design's Excel Graph functionality, allowing to insert a business graphic from a Microsoft Excel file.

# Exit (procedure)

Exits from a for...endfor or repeat...until loop.

## Syntax

exit()

## Argument

None

## Code Sample Example

This example shows how to break from a loop when a specific word, in this case 'Invoice,' is found somewhere on the data page.

```
define(&x,integer,1)
for(&x,1,1,&current.lpp)
  if(pos('Invoice',@(&x,1,80))>0)
    exit()
```

```
    endif
endfor()
```

## ExpandString (function)

Returns a string that contains the data from a Workflow tool variable. Any variable available to the Workflow tool can be used here.

### Syntax

ExpandString(stringtoexpand: string) ⇒ string

### Arguments

*stringtoexpand*

The Workflow tool variable string to expand. This can be local and global variables, job infos, as well as any other variable such as %o, %j, %y, etc.

### Return Value

Returns the resulting expanded string.

### Code Sample Example

```
define(&result, string, '')
&result := ExpandString('%{global.GlobalVar}')
show(&result)
% Also works with system variables
&result := ExpandString('%o')
show(&result)
```

## Field (function)

Returns the contents of the specified field for the current record. This function is only useful in database emulation mode.

### Syntax

field( fieldname[, recordnumber] ) ⇒ string value

### Argument

*fieldname*

String value specifying the name of the field whose value you want to retrieve.

*recordnumber*

Integer value specifying the number of the record, in the current record set, whose value you want to retrieve. If you omit this argument, the document uses the value of **&current.line**.

This argument is case sensitive. Thus the field names OrderNumber and ORDERNUMBER are not equivalent.

### Code Sample Example

This example shows how to print the name and content of the current record in a generic fashion.

**Example**

```
define(&x,integer,1) //Define loop variable
define(&fn,string,'') //Define name variable
margin(0,0)
for(&x,1,1,fieldcount()) //Loop through all fields
  set(&fn, fieldname(&x)) //Store field name
  show(&fn) //Display field name
  moveto(2,&current.y) //Move right
  show(field(&fn)) //Display field contents
  crlf() //Move to next line
endfor()
```

## FieldCount (function)

Returns a count of fields in the current database record. Since all records always hold the same number of fields, this value will not change unless a new sample data file is used. This function is only useful in database emulation mode.

### Syntax

fieldcount() ➡ integer value

### Code Sample Example

"Field (function)" (page 540)

## FieldName (function)

Returns the name of the specified field for the current record. This function is only useful in database emulation mode.

### Syntax

fieldname( index ) ➡ string value

### Argument

*index*

Integer value specifying the index of the field whose name should be retrieved. Correct values range from 1 to fieldcount().

### Code Sample Example

"Field (function)" (page 540)

## Fill (procedure)

Fills the current closed shape, using the colour specified with the **setfillcolor** command. Only the inside of the shape is filled, not its outline. To fill and outline the shape, use **strokeandfill** command instead. If you only want to outline the shape, use **stroke**. Only closed shapes can be filled.

### Syntax

fill()

### Argument

None

PLANETPRESS. SUITE

**Code Sample Example**

This example draws a yellow filled triangle.

**Example**

```
setfillcolor([0,0,100,0])
moveto(1,1)
lineto(1.5,2)
lineto(0,2)
closepath()
fill()
```

## Find (function)

Checks for the presence of a string within a rectangular region of the current data page. If the specified string is found, the function returns true. This allows you to look for strings within a more general area, rather than at some precise location.

**Syntax**

find( string1, column1, line1, column2, line2 ) ⇒ Boolean value

**Argument**

*string1*

String value to look for.

*column1, line1*

Integer values for the top left corner of the rectangular region to search in the current data page.

*column2, line2*

Integer values for the bottom right corner of the rectangular region to search in the current data page.

**Code Sample Example**

This example looks for the word 'Invoice' within a region of the data page.

**Example**

```
if(find('Invoice',60,1,80,5))
  show('Invoice')
elseif()
  show('Sales Order')
endif()
```

## FloatToCur (function)

Converts a measure expression into a currency value.

**Syntax**

floattocur( expression ) ⇒ currency value

**Argument**

*expression*

Measure value to be converted. The decimal part is rounded down to 2 decimal points.

**Code Sample Example**

This example converts a floating point value into a currency value.

```
floattocur(3.263457) %Returns 3.26
```

# FloatToInt (function)

Converts a measure expression into an integer value.

**Syntax**

floattoint( expression ) ⇒ integer value

**Argument**

*expression*

Measure value to be converted. The decimal part is rounded down to a whole number.

**Code Sample Examples**

These examples illustrate **floattoint()**.

**Example 1**

```
floattoint(3.2) %Returns 3
```

**Example 2**

```
floattoint(11.6) %Returns 12
```

# FloatToStr (function)

Converts a measure expression into a string value.

**Syntax**

floattostr( expression[, precision]) ⇒ string value

**Argument**

*expression*

Measure value to be converted.

*precision*

The number of decimal places to include in the result string.

**Code Sample Examples**

These examples illustrate **floattostr()**.

**Example 1**

```
floattostr(3.14) %Returns 3.14
```

**Example 2**

```
floattostr(11) %Returns 11.0
```

**Example 3**

```
floattostr(34.6453455, 4) %Returns 34.6453
```

# For... EndFor (procedure)

This command structure allows a series of nested commands to be repeated a specified number of times. Each **for** statement must have a corresponding **endfor** statement in order for PlanetPress Talk to know which commands to include in the loop. Note that you cannot increment the counter for a **for()** loop, inside the loop.

If you open a document created a with a version of PlanetPress earlier than 5.2.0, and if that document includes a **for()** loop with an increment greater than 1, you will notice that the loop is executed one less time than with your older version software both in PlanetPress itself and when you use the Optimized PostScript Stream option. The behavior on printers (using the Printer Centric option), on the other hand, remains unchanged.

**Syntax**

for( varname, startvalue, increment, stopvalue )

...

endfor()

**Arguments**

*varname*

Name of the variable to use for iterations. The variable must already exist.

*startvalue*

Integer value to initialize varname.

*increment*

Integer value used to increment varname after each iteration.

*stopvalue*

Integer value after which iterations stop.

**Code Sample Example**

This example simply prints 5 lines of text.

To cycle backwards through values, make **startvalue** larger than **stopvalue**, and specify a negative increment.

**Example**

```
define(&x,integer,1)
for(&x,1,1,5)
  show('Line n° '+inttostr(&x))
  crlf()
endfor()
```

## Function @name (procedure)

Defines a new PlanetPress Talk function or procedure. If you define a function that you want to return a value, you assign that value to the predefined variable **&result** on the last line of the function definition. You call the function or procedure you define using the **@name()** command. See "@name (function/procedure)" (page 561).

You can reference a global function from within another global function only if the global function you are referencing already exists.

You cannot reference a function from within itself, so recursive functions are not possible within PlanetPress Talk.

**Syntax**

**Syntax for a function that returns a value:**

function @name( arglist ): type

...

&result := return-_value

endfunction()

**Syntax for a function that does not return a value (procedure):**

function @name( arglist )

...

endfunction()

**Arguments**

*name*

Name to use for the function, prefixed with the @ character. The name must conform to the rules for names described in "Names" (page 219). It is good practice to be careful naming functions that you define in different documents, to ensure that if at some point you want to copy code between documents, the names of functions do not conflict.

*arglist*

List of arguments the function requires, where each element in the list has the syntax varname : type. For example, &date : string. An ampersand always precedes the first letter of varname. A comma separates each element in the list. Note that all arguments are pass by value (as opposed to pass by reference). Pass by value means that the function you define cannot modify any of the arguments it receives. Any modifications you make to an argument within the body of the function are made to a local copy, and do not affect the original.

*type*

Type of value (integer, measure, currency, string, Boolean) the function returns.

## Code Sample Example

This example displays a string.

```
function @showtext(&mystring: string)
  moveto(1,1)
  show(&mystring)
endfunction()
```

## Code Sample Example

This example illustrates a procedure that creates a 3D pie chart.

## Example

```
function @_3DPie( &W:measure, &H:measure, &HB:measure, &S:measure, &E:measure,
&L:string, &c:color )
  %==================================================
  % &W is the total width of the pie chart
  % &H is the total height of the pie chart
  % &HB is the height of the lower band
  % &S is the starting angle (0..360)
  % &E is the ending angle (0..360 and bigger than &S)
  % &L is the label text
  %==================================================
  %Local variables
  %==================================================
  define(&SB,measure,0)
  define(&EB,measure,0)
  define(&Scaling,measure,0)
  define(&R,measure,&W/2)
  %==================================================
  &Scaling := (&H-&HB)/&W
  gsave()
  if(&Scaling<0.01)
    &Scaling := 0.01
  endif()
  if(&Scaling>1)
    &Scaling := 1
  endif()
  scale(1,&Scaling)
  setstyle(&Style1)
  &HB := &HB / &Scaling
  translate(&R,&R)
  if(&E > 180)
    if(&E > 360)
      &EB := 360
    elseif()
```

```
      &EB := &E
    endif()
    %=================================================
    if(&S < 180)
      &SB := 180
    elseif()
      &SB := &S
    endif()
    %=================================================
    set-
fill-
color([getcyan(&c),getmagenta(&c),getyellow(&c),if(getblack(&c)>50,100,getblack(&c)+20)])
    moveto(cos(&SB) * &R,neg(sin(&SB) * &R))
    rlineto(0,&HB)
    arc(0,&HB,&R,&SB,&EB)
    rlineto(0,neg(&HB))
    closepath()
    strokeandfill()
  endif()
  %=================================================
  setfillcolor(&c)
  pie(0,0,&R,&S,&E)
  strokeandfill()
  &SB := (&S+&E)/2
  %=================================================
  if(gt(&SB,180))
    margin(cos(&SB) * &R,neg(sin(&SB) * &R)+&HB+(0.2/&Scaling))
  elseif()
    margin(cos(&SB) * &R,neg(sin(&SB) * &R)-(0.05/&Scaling))
  endif()
  %=================================================
  scale(1,1/&Scaling)
  if(and(gt(&SB,90),lt(&SB,270)))
    showright(&L+' ')
  elseif()
    show(' '+&L)
  endif()
  grestore()
```

## GE (function)

Compares two expressions of any type and returns true if the first is greater than or equal to the second, false otherwise. This is the functional equivalent to the >= operator.

**Syntax**

ge( expression1, expression2 ) ⇒ Boolean value

**Arguments**

*expression1, expression2*

Values of any type. Both expressions must be of the same type.

**Code Sample Examples**

Example 1 and Example 2 show both ways of comparing two expressions. Example 2 illustrates the >= operator.

**Example 1**

```
if( ge(&current.line, 50) )
  show('Bottom of page')
endif()
```

**Example 2**

```
if(&current.line >= 50)
  show('Bottom of page')
endif()
```

## Get (function)

Returns an element of an array.

**Syntax**

get( array, position ) array element

**Argument**

*array*

Name of the array.

*position*

Integer specifying the position in the array, of the element you want to retrieve. Recall that the first position in an array is 0, the second is 1, the third 2, etc.

**Code Sample Examples**

Examples 1 and 2 are equivalent representations of using the **get()** command.

**Example 1**

```
&month := get(&MyArray,12)
```

**Example 2**

```
&month := &MyArray[12]
```

## GetBlack (function)

Returns the value of the Black component of a color array.

**Syntax**

getblack( color ) ⇒ integer

**Arguments**

*color*

Color array for which you want to determine the Black value.

**Code Sample Example**

```
set-
fill-
color([getcyan(&c),getmagenta(&c),getyellow(&c),if(getblack(&c)>50,100,getblack(&c)+20)])
```

## GetCyan (function)

Returns the value of the Cyan component of a color array.

**Syntax**

getcyan( color ) ⇒ integer

**Arguments**

*color*

Color array for which you want to determine the Cyan value.

**Code Sample Example**

```
set-
fill-
color([getcyan(&c),getmagenta(&c),getyellow(&c),if(getblack(&c)>50,100,getblack(&c)+20)])
```

## GetMagenta (function)

Returns the value of the Magenta component of a color array.

**Syntax**

getmagenta( color ) ⇒ integer

**Arguments**

*color*

Color array for which you want to determine the Magenta value.

**Code Sample Example**

```
set-
fill-
color([getcyan(&c),getmagenta(&c),getyellow(&c),if(getblack(&c)>50,100,getblack(&c)+20)])
```

# Getmeta (function)

Returns the value of a metadata attribute or field.

**Syntax**

getmeta(name: string; [flags : integer; path : string]) ⇒ string

**Arguments**

*name*

The name of the metadata value to retrieve. Unless the **GetAttribute** flag argument is specified, the function looks for the name in the *field* collection.

*flags*

Optional integer value to define the behavior of the command when a field with the same name already exists in the current level.  The flag value to enter should be the sum of all desired flags. (Eg: GetAttribute and FailIfNotFound would give a flag value of '5').

The default behavior uses 0.

| Name | Value | Behavior |
|---|---|---|
| GetAttribute | 1 | Search for the **name** argument in the attribute collection instead of the default field collection. |
| NoCascade | 2 | Search only the level specified by the **path** argument (defaults to *Page* level when **path** argument is empty), instead of default behavior, going from the *Page* level to the *Job* level. |
| FailIfNotFound | 4 | Raise an error and crash the job is the specified **name** is not found instead of returning an empty string. |
| SelectedNodesOnly | 8 | Returns values from the selected nodes only. |

*path*

Optional string value to specify the level where to start looking for the field or attribute. Path components cannot be indexed, and if no path

is provided, the search is done from the the *Page* level.

**Note**: The node names of the path argument can have an optional zero-based index, indicated between square brackets ('[ ]'), to access a specific node. If no index is provided, the first node is used.

**Important Note**: If used from PlanetPress Suite Workflow Tools, every argument is **mandatory**.

**Code Sample Example**

These examples show the three different syntaxes for the GetMeta() function.

```
% Retrieve value for field "mydata". If that field exists in both the current group and
the current datapage collections, for instance,

% this call to the GetMeta() function will return the value of the field from the datap-
age.

GetMeta('mydata')
```

```
% Retrieve the second value of the current page, containing two fields called "address",
while the group collection also has four "address" fields.

GetMeta('address[1]')
```

```
% Note that using GetMeta('address[2]') would return the third value of the group level.
To prevent the function from cascading up to the group level,

% the NoCascade flag must be provided. And to retrieve the first two "address" fields of
the group collection, the third argument must be used, set to

% the proper path, thusly:

GetMeta('address[1]', 2, 'Job.Group')
```

## Getmetacount (function)

Returns the number of times an attribute or a field is encountered in the current metadata hierarchy. It can also be used to verify whether a field or attribute exists by comparing the returned value to zero. Note that attributes can only appear once, and thus can only yield two possible values, either zero or one.

**Syntax**

getmetacount(name: string[, flags: integer, path: string]) ⇒ integer

**Arguments**

*name*

The name of the metadata value to count.

*flags*

Optional integer value to define the behavior of the command.The default behavior uses 0. The flag value to enter should be the sum of all desired flags. (Eg: GetAttribute and FailIfNotFound would give a flag value of '5').

| Name | Value | Behavior |
|---|---|---|
| GetAttribute | 1 | Search for the **name** argument in the attribute instead of the default field . |
| NoCascade | 2 | Search only the level specified by the **path** argument (defaults to *Page* level when **path** argument is empty), instead of default behavior, going from the *Page* levelto the *Job* level. |
| FailIfNotFound | 4 | Raise an error and crash the job is the specified **name**is not found instead of returning an empty string. |

*path*

Optional string value to specify the level where to start counting the field or attribute. Path components cannot be indexed, and if no path

is provided, the search is done from the the *Page* level.

**Note**: The node names of the path argument can have an optional zero-based index, indicated between square brackets ('[ ]'), to access a specific node. If no index is provided, the first node is used.

Also note that , if no attribute nor field is found, the command returns zero, unless the **FailIfNotFound** flag is specified.

### Code Sample Example

This example shows the syntax for the GetMetacount() function.

```
% Retrieve and count the number of iterations of "address" for the current datapage
Getmetacount('address')
```

## GetNextDataPage(procedure)

Advance to the next data page of the sample data file. Note that this command may have unpredicted behavior if not used correctly in documents that use a user-defined emulation.

This command should only be executed when printermode is not in design mode. The result will be incorrect on screen, but there will be no
unexpected behaviors. Therefore, the only solution is for end-users to make sure the command doesn't get executed while in Design mode by bracketing it within a if(&printermode<>0) ... endif()
structure.

### Syntax

getnextdatapage()

## GetYellow (function)

Returns the value of the Yellow component of a color array.

**Syntax**

getyellow( color ) ⇒ integer

**Arguments**

*color*

Color array for which you want to determine the Yellow value.

**Code Sample Example**

```
set-
fill-
color([getcyan(&c),getmagenta(&c),getyellow(&c),if(getblack(&c)>50,100,getblack(&c)+20)])
```

## GRestore (procedure)

This command restores all system parameters previously saved with the **gsave** command. If no **gsave** command was issued previously, the results are unpredictable. For more information, refer to the **gsave** command.

**Syntax**

grestore()

**Argument**

None

**Code Sample Example**

See "GSave (procedure)" (page 565).

## GSave (procedure)

This command saves all current system parameters, which can later be restored using the **grestore** command. By bracketing parts of your PlanetPress Talk programs with **gsave**/**grestore** commands, you can make sure the current system state is preserved and remains unaffected by whatever operations your program executes. For instance, each PlanetPress object, once converted to its PlanetPress Talk scripting language equivalent, begins with a **gsave** command and ends with a **grestore**, thus ensuring objects do not interfere with each other, or with the system.

The system parameters gsave saves include current line width, current stroke color, and current fill color.

**Syntax**

gsave()

**Argument**

None

## Code Sample Example

In this example, the current position is set first and saved.

**gsave** and **grestore** are required to be issued in pairs, meaning that you need the equal number of **gsave** and **grestore** commands to run in any given object. However, an uneven number of commands can be located in the code, if some of them are within IF statements, so that when the code is actually run, the pairing occurs correctly in all instances.

### Example

```
moveto( 1, 1 )
%Position is now 1,1
gsave()
%Save current state
lineto( 2, 1 )
%Position is now 2,1
lineto( 2, 2 )
%Position is now 2,2
grestore()
%Restore previous state: position is now 1,1
```

# GT (function)

Compares two expressions of any type and returns true if the first is greater than the second, false otherwise. This is the functional equivalent to the > operator.

### Syntax

gt( expression1, expression2 ) ➡ Boolean value

### Arguments

*expression1, expression2*

Values of any type. Both expressions must be of the same type.

### Code Sample Examples

Examples 1 and 2 illustrate both ways of comparing two expressions.

### Example 1

```
if( gt(&current.line, 50) )

   show('Bottom of page')

endif()
```

**Example 2**

```
if(&current.line > 50)

  show('Bottom of page)

endif()
```

## If (function)

Evaluates an expression and returns one of two specified values according to the results of the evaluation.

**Syntax**

if( expression, trueresult, falseresult ) ⇒ any type

**Arguments**

*expression*

Boolean value. If it evaluates to true, the function returns the contents of trueresult, otherwise it returns the contents of falseresult.

*trueresult, falseresult*

Values of any type, as long as both are of the same type. The return value of the function is therefore of the same type as trueresult and falseresult.

**Code Sample Example**

This example changes the text to display according to the current value of a variable.

**Example**

```
show(if(&pagenumber=1,'Customer Copy','Store Copy'))
```

## If ... ElseIf... EndIf (procedure)

This command structure can define up to two conditional blocks. Statements nested within the first block are executed only if the condition specified in the if statement returns true. If it returns false, the program branches to the first statement in the second block, if one has been defined using the elseif statement. Otherwise, control flows to the first statement following the endif command.

With PlanetPress Suite version 7.0, the elseif statement has been optimized to evaluate an argument, exactly like an if statement, and elseif statements from other programming languages. In order to complete the if...elseif...endif command structure, however, an else statement was also added, acting exactly like the previous, empty elseif. Note that the use of the else

statement is strongly recommended, as it is optimized by comparison to the elseif statement. Also note that documents created with previous versions and using the if...elseif...endif structure will not be updated, for full backward compatibility.

**Syntaxes**

Prior to version 7.0

> if( condition )
>
> ...
>
> elseif()
>
> ...
>
> endif()

Version 7.0 and above

> if( condition1 )
>
> ...
>
> elseif( condition2 )
>
> ...
>
> else()
>
> ...
>
> endif()

**Argument**

*condition, condition1, condition2*

Value of type Boolean to evaluate.

**Code Sample Example**

This example simply prints 5 lines of text, selecting fonts according to line numbers.

**Example prior to version 7.0**

```
define(&x,integer,1)

%Define loop variable and then set up loop

for(&x,1,1,5)

  if((&x mod 2)=0)

    %If &x is an even number

    setstyle(&bluefont)

    %use the blue font

  elseif() %otherwise

    setstyle(&Default)
```

```
   %use the default font

 endif()

 show('Some text to display')

 %Display the text

 crlf()

 %Skip to the next line
endfor()
```

## Example with version 7.0

```
define(&x, integer, inttostr(@(1, 1, 10)))

if(&x = 1)

  setstyle(&bluefont)

elseif(&x = 2)

  setstyle(&redfont)

else()

  setstyle(&default)
endif()
```

**Related topics:**

- "If (function)" (page 552)

## InStream... EndInStream (procedure)

Ceate an alias for a resource file (image, data file, attachment, etc.). You can then reference the resource file using the alias instead of the path. This eliminates the need to repeat a path in more than one place in your code, and thus makes your code easier to maintain. For example, if you change the resource, you need only modify the path in the **instream()** command, rather than everywhere you reference the original resource.

You cannot use the **instream()** command inside any function you define using **function @name()**.

**Syntax**

instream external resname

path

endinstream()

If you want to create a reference to a text document (for example a sample data file), replace "external" by "cleartext."

**Argument**

*resname*

String value specifying the alias to use to refer to this resource file.

*path*

String value specifying the path of the resource file.

**Code Sample Example**

This example creates the alias 'photo' that refers to the resource file *c:\images\employees\JAdler.bmp*.

**Example**

```
instream external photo

  c:\images\employees\JAdler.bmp

endinstream()

%You can then reference the resource using the alias

showbitmap('photo',72,1,1)
```

## IntToCur (function)

Converts an integer expression into a measure value.

**Syntax**

inttocur( expression ) ➡ currency value

**Argument**

*expression*

Integer value to be converted.

## IntToFloat (function)

Converts an integer expression into a measure value.

**Syntax**

inttofloat( expression ) ➡ measure value

---

**Argument**

*expression*

Integer value to be converted.

**Code Sample Example**

This example uses a loop to draw a series of blue rectangles.

**Example**

```
define(&x,integer,1)

%Define loop variable

setfillcolor([100,100,0,0])

%Fill blue

for(&x,1,1,10)

  %Set up loop

  %We must convert &x because rectfill

  %expects measure values

  rectfill(inttofloat(&x)/2,1,0.1,1)

  %Draw rectangle

endfor()
```

# IntToStr (function)

Converts an integer expression into a string value.

**Syntax**

inttostr( expression ) ⇒ string value

**Argument**

*expression*

Integer value to be converted.

**Code Sample Example**

This example prints the iterations of a loop.

**Example**

```
define(&x,integer,1)

%Define loop variable

for(&x,1,1,10)
```

```
    %Set up loop

    show('Iteration ' + inttostr(&x) )

    %Show text

    crlf()

    %Skip line
endfor()
```

## IsNumber (function)

Tests a string and returns true if the string is a measure or integer value.

### Syntax

isnumber( string ) ⇒ Boolean value

### Argument

*string*

String value.

### Code Sample Example

This example displays the word Yes if the string is either a measure or an integer, and No if it is not.

### Example

```
show(if(isnumber(@(1,1,10)),'Yes','No'))
```

## IsPageEmpty (function)

Returns False if the current data page contains data, or True if it does not.

### Syntax

ispageempty() ⇒ Boolean

## LE (function)

Compares two expressions of any type and returns true if the first is less than or equal to the second, false otherwise. This is the functional equivalent to the <= operator.

### Syntax

le( expression1, expression2 ) ⇒ Boolean value

### Arguments

*expression1, expression2*

Values of any type. Both expressions must be of the same type.

**Code Sample Example**

This example shows both ways of comparing two expressions.

**Example**

```
if(le(&current.line, 15) )

  show('Top of page')

endif()

%Same statement, using the <= operator

if(&current.line <= 50)

  show('Top of page')

endif()
```

# Left (function)

Extracts the specified leftmost characters of a string.

**Syntax**

left( string, number ) ⇒ string value

**Arguments**

*string*

String value from which to extract characters.

*number*

Integer value specifying how many characters to extract from the beginning of the string.

**Code Sample Example**

This example prints a string up to a certain position.

**Example**

```
define(&data,string,'This is a~sample string')

%Define string

define(&x,integer, pos('~', &data))

%Find tilde

%Print the string up to the tilde character, if found

if(&x>1)

  show(left(&data,&x-1))
```

```
endif()
```

## Length (function)

Returns the length of a string, or the number of elements in an array.

### Syntax

length( element ) ⇒ integer value

### Argument

*element*

Either a string value, or an array variable.

### Code Sample Example

### Example 1

This example creates an a string array with the same number of elements as the &parts array.

```
define( &partnames, arraystring, length( &parts ) )
```

### Example 2

This example prints part of a string, starting after a specific delimiter.

```
define(&data,string,'This is a~sample string')

define(&x,integer, pos('~', &data))

%Find tilde

if(&x>1)

   show(right(&data,length(&data)-&x))

   %Show remainder of string

endif()
```

## LineTo/RLineTo (procedure)

Draws a line starting from the current point up to the specified coordinates.

### Syntax

lineto( x, y )

**Arguments**

*x, y*

Measure values representing the horizontal/vertical position, in inches, of the ending point of the line. When using **rlineto**, these values are relative to the current point of origin. When using **lineto**, they are absolute values.

**Code Sample Example**

This example draws an empty triangle with a blue outline.

**Example**

```
setstrokecolor([100,100,0,0])    %Set pen colour to blue

moveto(1,1) %Set starting point

lineto(1.5,2) %Draw first line

lineto(0.5,2) %Draw second line

closepath() %Close shape

stroke() %Draw shape
```

# LowerCase (function)

Convert a string to all lower case characters.

**Syntax**

lowercase( string ) ⇒ string

**Argument**

*string*

String value you want to convert to lower case. The string may be a mix of upper and lower case characters.

**Code Sample Example**

```
&partname := lowercase( @(30,16,39) )
```

# LT (function)

Compares two expressions of any type and returns true if the first is less than the second, false otherwise. This is the functional equivalent to the < operator.

**Syntax**

lt( expression1, expression2 ) ⇒ Boolean value

## Arguments

*expression1, expression2*

Values of any type. Both expressions must be of the same type.

## Code Sample Example

This example shows both ways of comparing two expressions.

## Example

```
if(lt(&current.line, 15))

  show('Top of page')

endif()

%Same statement, using the < operator

if(&current.line < 50)

  show('Top of page')

endif()
```

## MapUTF8 (function)

Converts a text string from its original encoding to UTF8. This function can be used within a ShowUTF8, ShowUTF8Left, a ShowUTF8Right, and a ShowUTF8Center procedure.

## Syntax

maputf8( text, encoding, reversedmapping )

## Arguments

*text*

Static text string or data selection to be converted.

*encoding*

String identifying the original encoding or reference to a style defined in the document that identifies the original encoding.

*reversedmapping*

Boolean specifying whether the text to be converted should be mapped in regular or reversed order. Possible values are true (reversed mapping) and false (no reversed mapping). In the case of preformatted Arabic data, this argument should be set to false.

## Supported Encodings

The following Arabic original encodings are supported for the conversion process:

| | | | | |
|---|---|---|---|---|
| AL-ARABI | ASMO-708 | ASMO-708-UNIX | IBM-864 | ISO-8859-6 (Arabic) |
| ASMO-449+ | ASMO-708+ | HP-ARABIC8 | IRAN SYSTEM | MS-CP-1256 |

**Code Sample Examples**

Example 1 illustrates a variable text string that is taken from the data, converted from the MS-CP-1256 encoding to UTF8 and then displayed from the insertion point to the right. Example 2 illustrates the same variable text string, but this time it is converted from the encoding specified in the style named "MyArabEncoding" to UTF8.

**Example 1**

```
setlinewidth(0.007)
moveto(0,0)
rlineto(&width,0)
rlineto(0,&height)
rlineto(neg(&width),0)
rlineto(0,neg(&height))
closepath()
stroke()
moveto(&width/2,&height/2)
SetStyleExt(&Style1,12.0000,0,[100],100)
showUTF8right(maputf8(@(2,15,35), 'MS-CP-1256', False),'normal')
```

**Example 2**

```
setlinewidth(0.007)
moveto(0,0)
rlineto(&width,0)
rlineto(0,&height)
rlineto(neg(&width),0)
rlineto(0,neg(&height))
closepath()
stroke()
moveto(&width/2,&height/2)
SetStyleExt(&Style1,12.0000,0,[100],100)
showUTF8right(maputf8(@(2,15,35), &MyArabEncoding, False),'normal')
```

## Margin (procedure)

Offsets text within a series of commands, relative to the original top left position of the object. Margin also sets the horizontal position to be used when **crlf** commands are encountered. All text-rendering functions that make use of margins use this setting.

**Syntax**

margin( x, y )

**Arguments**

*x, y*

Measure values representing the horizontal/vertical position, in inches, of the starting point for the text within an object.

**Code Sample Example**

This example displays shadowed text by repeating the same text commands twice, using styles of different colours and slightly offsetting the margin.

**Example**

```
margin(1,1) %Set margins to ( 1,1 )

setstyle(&blackfont) %Blackfont must already exist

show('This is shadowed text')

margin(.98,.98) %Offset margins slightly

setstyle(&bluefont) %Bluefont must already exist

show('This is shadowed text')
```

## Mid (function)

Extracts a specified number of characters from a string, starting at a specific position.

**Syntax**

mid( string, start, length ) ⇒ string value

**Arguments**

*string*

String value from which to extract characters.

*start*

Integer value specifying where the first character to extract is located.

*length*

Integer value specifying how many characters to extract, starting from number onwards.

**Code Sample Example**

This example extracts a bracket-delimited word from a sentence.

**Example**

```
define(&data,string,'This is a [bracketed] word')

define(&i,integer, pos('[', &data))

%Find the '[' character

define(&j,integer, pos(']', &data))

%Find the ']' character

if((&i>0) and (&j>&i))

  %If both were found

  show(mid(&data,&i+1,(&j-&i)-1))

  %Print the word

endif()
```

## Mod (function & procedure)

Returns the remainder resulting from the division of two numbers.

**Syntax**

mod( number1, number2 ) $\Rightarrow$ integer value

number1 mod number2 $\Rightarrow$ integer value

**Arguments**

*number1*

Integer value to be divided.

*number2*

Integer value with which to divide number1.

**Code Sample Example**

This example uses MOD to distinguish between odd and even numbers.

**Example**

```
define(&i,integer, 1)

%Define loop variable

for(&i,1,1,10) %Set loop
```

```
    show(inttostr(&i)+' is an') %Display number

    if(mod(&i,2)=1)

    %Determine if number is odd or even

        show(' odd ')

    elseif()

        show(' even ')

    endif()

    show('number.')

    crlf()

endfor()
```

## MoveTo/RMoveTo (procedure)

Moves the current point to the specified coordinates.

**Syntax**

moveto( x, y )

**Arguments**

*x, y*

Measure values representing the new horizontal/vertical position, in inches, of the current point. In **rmoveto**, these values are relative to the current point of origin. In **moveto**, they are absolute values.

**Code Sample Example**

This example displays a triangle within a rectangle.

**Example**

```
rectstroke(.5,.5,2,2) %Draw rectangle

moveto(1,1) %Reset current point

lineto(1.5,2) %Draw first line

lineto(0.5,2) %Draw second line

closepath() %Close shape

stroke() %Draw shape
```

## Mul (function)

Multiplies two expressions. This is the functional equivalent of the * operator.

**Syntax**

mul( expression, expression2 ) ➾ integer, measure, currency value

**Arguments**

*expression1, expression2*

Integer, measure or currency values. Both expressions must be of the same type. The returned value is set accordingly to the type of the parameters.

**Code Sample Examples**

Examples 1 and 2 show both ways of multiplying numbers.

**Example 1**

```
show(inttostr(2*2)) %Displays 4
```

**Example 2**

```
show(floattostr(mul(4.7,2.1))) %Displays 9.87
```

# NE (function)

Compares two expressions of any type and returns true if they are not equal, false otherwise. This is the functional equivalent to the <> operator.

**Syntax**

ne( expression1, expression2 ) ➾ Boolean value

**Arguments**

*expression1, expression2*

Values of any type. Both expressions must be of the same type.

**Code Sample Examples**

Examples 1 and 2 show both ways of comparing two expressions.

**Example 1**

```
if( ne(&current.line, 1) )

  show('Not a new page')

endif()
```

**Example 2**

```
if(&current.line <> 1) %Same statement, using the <> operator

  show('Not a new page')

endif()
```

## Neg (function)

Returns the negative value of the specified expression. This is the functional equivalent to the - sign.

**Syntax**

neg( expression1 ) ⇒ integer, measure, currency value

**Argument**

*expression1*

Integer, measure or currency value. The returned value is the same type as expression1.

**Code Sample Examples**

Examples 1and 2 show both ways of negating a number. Negating negative numbers yields a positive result.

**Example 1**

```
-12 %Returns -12

-&max %Returns negative of the value of the variable &max

neg(-12.45) %Returns 12.45
```

## Not (Boolean operator function)

Negates the specified expression. Not to be confused with the **neg** function: not is a Boolean operator function, not a mathematical one.

**Syntax**

not( expression1 ) ⇒ Boolean value

**Argument**

*expression1*

Boolean value.

**Code Sample Example**

This example shows how NOT can be used in place of operators.

**Example**

```
define(&x,integer,1) %Define loop variable
for(&x,1,1,10) %Set loop
  if(not(&x=5)) %Same as if(&x<>5)
    show('This is NOT the 5th line') %Show some text
  elseif()
    show('This IS the 5th line') %Show alternate
  endif()
endfor()
```

## Object $name()... EndObject (procedure)

Creates an object. You can reference the object you create using the $ operator. See "$element (procedure)" (page 611).

> ⚠ This command can only be used from within the PressTalk Before and PressTalk After of a PlanetPress Design document, and cannot exist within a page's properties or the PressTalk of any object, including

**Syntax**

```
object $name( top, left, width, height, condition, angle[, setsnappingpoint[, snap-
toprevious]] )
 %Object Contents
endobject()
```

**Argument**

*$name* (PressTalk Name)

Name to use for the object, preceded by the dollar sign character ($). The name must conform to the rules for names described in "Names" (page 219).

*top* (Measure)

Measure value specifying the distance, in inches, to offset the top edge of the picture object, from the top edge of the document page. If you set the snaptoprevious argument to a value other than 0, the value you set for the top argument becomes the vertical offset for the Snap to previous snapping point.

*left* (Measure)

Measure value specifying the distance, in inches, to offset the left edge of the picture object, from the left edge of the document page. If you set the snaptoprevious argument to a value other than 0, the value you set for the left argument becomes the horizontal offset for the Snap to previous snapping point.

*width, height* (Measure)

Measure values specifying the width and height respectively, in inches, of the object.

*condition* (Boolean)

Boolean value, or PlanetPress Talk expression that resolves to a Boolean value, specifying a condition on the object.

*angle* (Measure)

Measure value specifying the angle of rotation, in degrees. The pivot point for the rotation is the bottom left corner of the object.

*setsnappingpoint* (optional, Boolean)

Boolean value (True or False) specifying whether the object has its Set snapping point property set. If you set this argument to True, the last line of the code for the object must be a **moveto()** command that moves the current point to the position at which you want to set the object's snapping point.

*snaptoprevious* (optional, Integer)

Integer value specifying whether the object has its Snap to previous property set, and if so, the position of that snapping point. If you set the Snap to previous property, you can use the top and left arguments to specify, respectively, a vertical and horizontal offset for the snapping point.

- 0 = do not set the Snap to previous snapping point
- 1 = top left
- 2 = top middle
- 3= top right
- 4= middle left
- 5 = middle middle
- 6= middle right
- 7 = bottom left
- 8 = bottom middle
- 9 = bottom right

### Code Sample Example

The following creates a rectangle object (2 inches wide by 3 inches high), rotates it 45 degrees, and sets its Snap to previous snapping point at the top left of the object with a vertical offset of 1 inch and a horizontal offset of 1.5 inches.

```
object $Box2(1.0,1.5,2.0,3.0,true,45.0,false,1)
  setstyle(&Style1)
  MoveTo(0,0)
  SetLineWidth(0.0070)
  SetStrokeColor([0,0,0,100])
  SetFillColor([0,0,0,0])
  LineTo(&width,0.0)
  LineTo(&width,&height)
  LineTo(0.0,&height)
  LineTo(0.0,0.0)
  ClosePath()
  Stroke()
endobject()
```

## Or (Boolean operator function)

Returns true if either or both specified expressions are true, false otherwise.

**Syntax**

or( expression1, expression2 ) ⇒ Boolean value

expression1 or expression2 ⇒ Boolean value

**Arguments**

*expression1, expression2*

Boolean values.

**Code Sample Examples**

These examples illustrate the possible values for OR.

**Example 1**

```
or(true, false) %Returns true
```

**Example 2**

```
or(false, true) %Returns true
```

**Example 3**

```
or(false, false) %Returns false
```

**Example 4**

```
or(true, true) %Returns true
```

## Ord (function)

Returns the ASCII value of a character.

**Syntax**

ord( string ) ⇒ integer value

**Argument**

*string*

String value whose ASCII value is returned. Only the first character is considered.

**Code Sample Example**

This example shows how to determine if a character is lowercase or uppercase.

**Example**

```
define(&data,string,'a') %Define sample string
if((ord(&data)>=97) and (ord(&data)<=122))
  %Test lowercase
  show(&data+' is lowercase')
elseif()
  if((ord(&data)>=65) and (ord(&data)<=90))
    %Test uppercase
    show(&data+' is uppercase')
  elseif()
    show(&data+' is not between A and Z or a to z')
  endif()
endif()
```

## OutputDebugString (procedure)

Outputs a string to the PlanetPress Talk Messages window in PlanetPress.

**Syntax**

outputdebugstring(message)

**Argument**

*message*

String value to display in the PlanetPress Talk Messages window.

## PassThrough (procedure)

Sends a literal PostScript command to the printer.

**Syntax**

passthrough( string, [mode])

**Argument**

*string*

String value to be sent to the printer. This string is sent as is, with no further modification.

*mode*

Integer used as a bitfield to specify when to send the passthrough. Use a value of 1 to send the passthrough in PostScript-printer centric mode only, a value of 2 to send the passthrough in PostScript-Optimized PostScript Stream only, a value of 3 to send the passthrough in both PostScript-printer centric AND PostScript-Optimized PostScript Stream modes (default behavior if the parameter is not present), and a value of 4 to send the passthrough in Windows printing mode only.

**Code Sample Example**

Refer to your PostScript manual for valid commands.

## PDFPageCount (function)

Returns the number of pages in the specified PDF file.

**Syntax**

pdfpagecount( filename ) integer

**Argument**

*filename*

String value specifying the path of the PDF file.

## PDFWidth/PDFHeight (function)

Returns the width or height, in inches, of a page of a Portable Document Format (PDF) image resource.

**Syntax**

pdfwidth( name, page ) ⇒ measure value

pdfheight( name, page ) ⇒ measure value

**Argument**

*name*

String value that specifies the name of the PDF image resource.

*page*

Integer value that specifies the page of the PDF

**Code Sample Example**

This example sets the variable maxwidth to the width of page 3, and the variable maxheight to the height of page 4, of the PDF named parts_manual.

**Example**

```
&maxwidth := pdfwidth( 'parts_manual',3 )
&maxheight := pdfheight( 'parts_manual',4 )
```

## Pie (procedure)

Creates a pie slice shape, which can then be rendered using **stroke**, **fill**, or **strokeandfill**.

**Syntax**

pie( x, y, radius, startangle, endangle )

**Arguments**

*x, y*

Measure values representing the horizontal/vertical position, in inches, of a circle's center representing the whole pie from which to draw a slice.

*radius*

Measure value representing the length, in inches, of the segments of the pie, i.e. its radius.

*startangle, endangle*

Measure values representing the angles, in degrees, of both segments of the pie slice. Values increase counter clock wise, with 0 extending right from the center of the circle.

**Code Sample Example**

This example draws a whole pie chart with a shadow effect.

**Example**

```
setfillcolor([0,0,0,50]) %Set shadow colour to gray
pie(1.5,1.5,1,60,290) %Create shadow for first slice
translate(0.2,0.2) %Separate shadows
pie(1.5,1.5,1,290,420) %Create shadow for second slice
fill() %Display shadows
translate(-0.3, 0.3) %Offset chart from its shadow
setstrokecolor([0,0,100,0]) %Set pen colour to yellow
setfillcolor([0,100,100,0]) %Set fill colour to red
pie(1.5,1.5,1,60,290) %Create first slice
strokeandfill() %Draw first slice
```

```
translate(0.2,0.2) %Separate slices

setfillcolor([100,100,0,0]) %Set fill colour to blue

pie(1.5,1.5,1,290,420) %Create second slice

strokeandfill() %Draw second slice
```

---

## PixelHeight (function)

Returns the height, in pixels, of the specified image.

**Syntax**

pixelheight( image ) ⇒ integer

**Argument**

*image*

String value specifying the path of the file containing the image.

## PixelWidth (function)

Returns the width, in pixels, of the specified image.

**Syntax**

pixelwidth( image ) ⇒ integer

**Argument**

*image*

String value specifying the path of the file containing the image.

## Pos (function)

Returns the starting position of a specified string within another string, or 0 if the specified string cannot be found.

**Syntax**

pos( string1, string2 ) ⇒ integer value

**Arguments**

*string1*

String value to search for.

*string2*

String value in which to search.

**Code Sample Example**

This example uses POS to look for a word in a data selection.

**Example**

```
define(&s,string,@(1,10,40)) %Initialize var. with data
if(pos('INVOICE',&s)>0) %Look for the word INVOICE
  show('Invoice') %if found, display some text
elseif()
  %Otherwise,display something else
  show('Sales order')
endif()
```

## Put (procedure)

Assigns a value to an element of an array.

### Syntax

put( &array, index, value )

### Arguments

*&array*

Array variable containing the element whose value you want to change.

*index*

Integer value representing the position of the element in the array.

*value*

The value you want to assign to the element. The value must be of the same the type as the array. Thus if the array is of type string, the value must be of type string.

## Random (function)

Returns a measure value between 0 and 1, non-inclusive. Since this function uses the current system time when run inside the printer, it returns a true random value. On the computer, however, the function uses the current data page as its seed to ensure the returned value is constant when you navigate from page to page on your document.

### Syntax

random() ⇒ measure value

### Argument

None

### Code Sample Example

This example displays 10 random numbers between 1 and 20.

**Example**

```
define(&x,integer,1)
for(&x,1,1,10)
  show(inttostr(floattoint(random()*20)+1))
  crlf()
endfor()
```

## Rectangle (procedure)

Creates and draws a rectangle shape. The rectangle can optionally be filled using the colour specified with the **setfillcolor** command. The border can optionally be drawn using the colour specified with the **setstrokecolor** command.

**Syntax**

rectangle( x, y, x1, y1, filled, stroked )

**Arguments**

*x, y*

Measure values representing the horizontal/vertical position, in inches, of the starting point.

*x1, y1*

Measure values representing the horizontal/vertical position, in inches, of the bottom right corner of the rectangle.

*filled*

Boolean value specifying whether the shape should be filled using the colour specified with the setfillcolor command.

*stroked*

Boolean value specifying whether the shape should be outlined using the colour specified with the setstrokecolor command.

**Code Sample Example**

This example draws a yellow rectangle with a black border.

**Example**

```
setstrokecolor([0,0,0,100])
setfillcolor([0,0,50,0])
rectangle(0,0,3,3,true,true)
```

# RectFill (procedure)

Creates and draws a filled rectangle shape. The colour used to fill the shape can be specified using the **setfillcolor** command.

**Syntax**

rectfill( x, y, width, height )

**Arguments**

*x, y*

Measure values representing the horizontal/vertical position, in inches, of the starting point.

*width, height*

Measure values representing the width/height, in inches, of the rectangle shape.

**Code Sample Example**

This example draws two identical rectangles.

To draw a rectangle that is both stroked and filled, use **rectstroke()** and **rectfill()** in succession, or **rectfillstroke()**.

**Example**

```
setfillcolor([0,100,100,0])
%Draw first rectangle using rectfill with red
rectfill(1,1,2,2)
%Offset starting position slightly to distinguish
%between each rectangle and fill second one with blue.
translate(.2,.2)
setfillcolor([100,100,0,0])
%Draw second rectangle using conventional commands
moveto(1,1) %Reset current point
lineto(3,1) %Draw first line
lineto(3,3) %Draw second line
lineto(1,3) %Draw third line
closepath() %Close shape
fill() %Draw shape
```

## RectFillStroke (procedure)

Creates a rectangle that has both an outline and a fill color. You set the color for the outline using setstrokecolor and the fill color using setfillcolor.

**Syntax**

rectfillstroke( x, y, width, height )

**Arguments**

*x,y*

Measure values representing the x and y coordinates respectively of the top left corner of the rectangle.

*width, height*

Measure values specifying the width and height respectively of the rectangle.

## RectStroke (procedure)

Creates and draws an empty rectangle shape. The colour of the pen used to draw the rectangle can be set using **set-strokecolor**.

**Syntax**

rectstroke( x, y, width, height )

**Arguments**

*x, y*

Measure values representing the horizontal/vertical position, in inches, of the starting point.

*width, height*

Measure values representing the width/height, in inches, of the rectangle shape.

**Code Sample Example**

This example draws two identical rectangles.

To draw a rectangle that is both stroked and filled, use **rectstroke** and **rectfill** in succession.

**Example**

```
rectstroke(1,1,2,2) %Draw first rectangle

%Offset starting position slightly to distinguish

%between each rectangle; the second rectangle is blue.

translate(.2,.2)

setstrokecolor([100,100,0,0])

%Draw second rectangle using conventional commands

moveto(1,1) %Reset current point

lineto(3,1) %Draw first line
```

```
lineto(3,3) %Draw second line

lineto(1,3) %Draw third line

closepath() %Close shape

stroke() %Draw shape
```

## Region (function)

Returns an array of strings, containing the text found inside left-top, right-bottom coordinates.
Note: This function only works in Windows Printing or Optimized PostScript Stream. If used in printer centric mode, it will return an empty string.

### Syntax

region(left, top, right, bottom) ⇒ array of strings

### Argument

*left,top,right,bottom*

Measure values specifying the coordinates, in inches or centimeters, for the area on which to read the data in the current data page.
Note that since this function returns an array of string, using region to display data on a page does not make much sense; for better results, consider using regionline().

### Code Sample Example

This example illustrates how to assign data from the current data page to a array of strings variable by using the *region()* command. Then, a *for()* loop is used to show every item in the array of strings **&detailarray**, followed by a carriage return.

### Example

```
define(&detailarray, arraystring, region(1.92,3.53,4.65,4.39))
define(&i, integer, 0)

for(&i, 0, 1, Length(&detailarray)-1)
  show(&detailarray[&i])
  crlf()
endfor()
```

**Related topics**:

- "Regionline (function)" (page 501)

## Regionline (function)

Returns an array of strings, containing the text found inside left-top, right-bottom coordinates.
Note: This function only works in Windows Printing or Optimized PostScript Stream. If used in printer centric mode, it will return an empty string.

### Syntax

regionline(x1, y1, x2, y2: measure; [index: integer = 0] ⇒ string

### Argument

*x1, y1, x2, y2*

Measure values specifying the coordinates, in inches or centimeters, for the area on which to read the data in the current data page.

*index*

Integer value specifying the line number to return. If no <>index<> paramter is specified, the default value is 0.

### Code Sample Example

This example illustrates how to assign data from the current data page to a string variable by using the regionline() command.

### Example

```
define(&invnum, string, regionline(2.1807,0.3018,3.8843,0.9054, 0))
```

**Related topics**:

- "Region (function)" (page 500)

## Repeat... Until (procedure)

Repeat a sequence of commands until a condition is true. It is similar to a for... endfor loop in that you can nest the loops. It is different from a **for... endfor** loop in that the loop always executes at least once, and that you tie the number of times the loop repeats to a Boolean condition. For example you might tie the number of times the loop repeats to the occurrence of a piece of data in the data stream.

### Syntax

repeat

...

until( expression )

### Argument

*expression*

Boolean value. If it evaluates to true, the loop ends. If it evaluates to false, the loop repeats.

---

**Code Sample Example**

This example prints the string 'Cheers' ten times, one underneath the other, with a dashed line above and below the ten instances.

**Example**

```
show('--------------')

crlf(0.16)

&count := 1

repeat

  &count := &count + 1

  show('Cheers')

  crlf(0.16)

until(&count = 10)

show('--------------')

crlf(0.16)
```

## ResourceType (function)

Returns the type of a resource. Recall that resources are either images or attachments.

**Syntax**

resourcetype( name ) ⇒ integer value

**Argument**

*name*

String value that specifies the name of the resource.

**Return Values**

| Return value: | Indicates: |
|---|---|
| 0 | The resource does not exist or could not be found. |
| 1 | A color bitmap. |
| 2 | A monochrome bitmap. |
| 3 | A grayscale bitmap. |
| 4 | An Encapsulated PostScript (EPS) image. |
| 5 | A PostScript file. |
| 6 | A Portable Document Format (PDF) file. |

**Code Sample Example**

This example sets the variable &resolution to 200 DPI if the resource is a color bitmap, and to 75 DPI if it is not.

**Example**

```
if( eq(resourcetype( 'employee' ), 1))

  &resolution := 200

elseif

  &resolution := 75

endif
```

## Right (function)

Extracts the specified rightmost characters of a string.

**Syntax**

right( string, number ) ⇒ string value

**Arguments**

*string*

String value from which to extract characters.

*number*

Integer value specifying how many characters to extract from the end of the string.

**Code Sample Example**

This example prints part of a string, starting after a specific delimiter.

**Example**

```
define(&data,string,'This is a~sample string')

define(&x,integer, pos('~', &data)) %Find tilde

if(&x>1)

  show(right(&data,length(&data)-&x))

  %Show remainder of string

endif()
```

## RunPS (procedure)

This command is typically used to call resources, usually images, downloaded either with the Image Downloader in PlanetPress or Send Images to Printer action tasks in the PlanetPress Suite Workflow Tools. *Note that this command does not work with any other PostScript files.*

**Syntax**

runps( filename )

**Argument**

*filename*

String value. Name of the resource file.

## Scale (procedure)

Scales the result of all commands that follow the scale() command.

**Syntax**

scale(width, height)

**Argument**

*width*

Measure value specifying the factor by which to scale the width.

*height*

Measure value specifying the factor by which to scale the height.

**Code Sample Example**

```
scale( 2,0.5 )
```

## Search ... EndSearch (procedure)

This command structure allows a series of nested commands to be repeated as long as a specific string of characters (the search string) is found within another string (the target string). After each iteration, the target string contains all the data between occurrences of the search string, excluding the search string. Once the loop is over, target contains the remaining characters of the original data.

**Syntax**

search( target, search )

...

endsearch()

**Arguments**

*target*

Name of a variable of type string in which to perform the search.

*search*

String value to search for.

**Code Sample Example**

This example splits a string delimited with semi-colons.

In the **OnReadDataPage** event of the user-defined emulation, the **&str** system variable contains each new line of data being fed to the document. Also, do not assign anything to the variable while inside the **search-endsearch** loop; it will get overwritten.

**Example**

```
define(&v1,string,'This;is;a;test') %Create the variable

search(&v1,';') %Start loop

  show(&v1) %Show token

  crlf() %Skip line

endsearch()

show(&v1) %Show remaining text
```

## SelectMedia (procedure)

On screen, displays the physical page. On a PostScript printer, calls the specified paper.

Variables used as arguments in this command are not parsed when printing using the Optimized PostScript Stream mode.

**Syntax**

selectmedia( width, height, [media], [color], [weight], orientation, mode )

**Arguments**

*width, height*

Measure values specifying the width and height of the physical page in inches.

*media*

A string value specifying the type of paper to use.

*color*

String value specifying the color of the paper.

*weight*

Measure value specifying the weight of the paper (in grams per square meter). You can convert weight in pounds to weight in grams by multiplying by the weight in pounds by 3.76.

*orientation*

Integer representing the orientation of the page (0=Portrait, 1=Landscape, 2=Reverse portrait, and 3=Reverse landscape).

*mode*

Reserved for future use. Specify 0 for now.

### Code Sample Example

```
define(&mcolor,string,'')

&mcolor := 'White'

selectmedia(8.5,11,'Danny',&mcolor,20,0,0)

The code above will generate the following PostScript code (notice how the variable was
not replaced by its value).

612 792 0 ^SM (Danny) &mcolor 20 ^SPM
```

## SelectPrinter (procedure)

This command is used to select the printer on which the next page set will be sent. It can be used to select the same physical printer with different options, or a different printer altogether. It is meant to be used with the **EndPageSet()** command. This command can currently be used only for Windows Printing.

### Syntax

selectprinter(printername)

### Arguments

*printername*

String identifying a printer queue.

### Example

Refer to the code example given in the section describing the **EndPageSet** command on page 1.

## Set (procedure)

This command assigns a new value to a variable. Only user-defined variables can be set. System variables are read-only. Note that you can also use the assignment operator to assign a value to a variable. See "= (operator)" (page 391).

### Syntax

set( name, value )

**Arguments**

*name*

Variable name, prefixed with the ampersand (&) character. The variable name must already exist.

*value*

New value to store in the variable. This value can be any valid PlanetPress Talk expression, as long as its type is the same as the variable's original type.

**Code Sample Example**

```
set( &invoice_number, (&invoice_number + 1) )

set( &is_bottom, (not(&is_top) and not(&is_middle)) )

&tax_rates[2] :=32

put( &tax_rates[2], @(1,3,5) )

set( &months, ['JAN','FEB','MAR','APR','MAY'] )

set( &mustard_color,[0,20,90,16] )

set( &prices, [12.5632,18.9932,23.6651,27.0300] )

put( &imagefiles[0], 'c\\images\\sushi.png' )
```

# SetAngle (procedure)

This command rotates all subsequent commands by the angle specified. Most commands and objects that are displayed/printed on the document are affected by the **setangle** command.

**Syntax**

setangle( angle )

**Argument**

*angle*

Measure value specifying the angle of rotation in degrees for all subsequent commands. Use a positive value for counter-clockwise motion and a negative value for clockwise motion. This value is relative to the current angle, not absolute.

**Code Sample Example**

This example illustrates **setangle()**.

**setangle()** sets a rotation for all subsequent commands. Consequently, **setangle** commands are cumulative. Therefore, to restore the previous angle of rotation, you must issue a **setangle** command that negates whatever rotation applied.

**Example**

```
setangle(45.00) %Rotate subsequent commands by 45 degrees

show('Some text') %Display at a 45 degree angle
```

```
setangle(45.00) %Rotate subsequent commands by 45 degrees

show('Some text') %Display at a 90 degree angle

setangle(-90) %Cancel all rotations
```

## SetBodyText (procedure)

Defines the text that appears as the body of an email message sent by PlanetPress Image.

**Syntax**

setbodytext( bodytext )

**Arguments**

*bodytext*

String value specifying the text to use as the body of the email message.

## SetDash(procedure)

Set the properties of a dashed line. Any drawing done after **setdash()** draws dashed instead of solid lines. You can revert to solid lines by calling **setdash()** with an empty array as a parameter.

**Syntax**

setdash( [dashlength, spacelength] )

**Argument**

*dash*

Measure value specifying the length, in inches, of each of the dashes in the dashed line.

*space*

Measure value specifying the length, in inches, of the space between each of the dashes in the dashed line.

**Code Sample Examples**

These examples illustrate **setdash()**.

**Example 1**

```
setdash( [0.5,0.3] )
```

**Example 2**

```
setdash( [] )
```

     PLANETPRESS. SUITE

## SetDataPage(procedure)

Specify the data page of the sample data file.

**Syntax**

setdatapage( pagenum )

**Argument**

*pagenum*

Integer value representing the page number of the data page.

## SetEmailAddress (procedure)

Defines an email address for PlanetPress Image.

**Syntax**

setemailaddress( address )

**Arguments**

*address*

String value specifying the email address.

## SetEmailSubject (procedure)

Defines the subject line of an email message sent by PlanetPress Image.

**Syntax**

setemailsubject( subjectline )

**Arguments**

*subjectline*

String value specifying the subject line.

## SetEmulation(procedure)

Sets the emulation to use.

**Syntax**

setemulation( emulation )

**Argument**

*emulation*

Integer value indicating the type of emulation.

| Value: | Emulation: |
|--------|-----------|
| 0 | Line printer |
| 1 | ASCII |
| 2 | Comma Separated Value (CSV) |
| 3 | Channel skip |
| 4 | Database |
| 5 | XML |
| 6 | PDF |

## SetUserCRLF (procedure)

Lets you define a custom global function that will be executed whenever a line return appears in a Text object where this procedure is added. The SetUserCRLF() procedure is normally added in the PressTalk Before parameter of your Text object.

Syntax

setusercrlf( procedurename:procedure )

Arguments

*Procedurename*

The name of an existing global function in your PlanetPress Design document, preceded as usual with @.

Example

This example will ensure that multi-line text will never overlap an image on the page (where the image's position and size has been saved as global variables).

In the PressTalk Before of an object, the following line is added:

```
setusercrlf(@MyCRLF)
```

This global function is created:

```
function @MyCRLF(&Leading:measure)
 if(((&physical.y+(&leading*2))>&LogoPos[0]) and ((&physical.y+&leading)<(&LogoPos[0])))
  translate(0,(&LogoPos[1]-&LogoPos[0])+&leading*2)
 endif()
 moveto(0,&current.Y+&Leading)
endfunction()
```

## SetFaxInformation (procedure)

Defines the fax description that appears in both the PlanetPress Fax dialog box and the PlanetPress Fax log file.

**Syntax**

setfaxinformation( info )

**Arguments**

*info*

String value specifying a description of the fax.

**Code Sample Example**

```
setfaxinformation( 'purchase confirmation' )
```

# SetFaxNumber (procedure)

Defines a fax number for PlanetPress Fax.

**Syntax**

setfaxnumber( faxnumber )

**Arguments**

*faxnumber*

String value specifying the fax number.

**Code Sample Example**

```
setfaxnumber( '(514)276-7633' )
```

# SetFillColor (procedure)

Sets the colour used to paint the insides of any closed shapes when issuing a **fill** or a **strokeandfill** command.

**Syntax**

setfillcolor( colour )

**Argument**

*colour*

Colour array that can be expressed expressed using one, three or four values ranging from 0 to 255. For Grayscale, a single value is used in the form [K]. For the RGB model, three values are used in the form [R, G, B]. For the CMYK model, four values are used in the form [C, M, Y, K].

**Code Sample Example**

This example draws three filled rectangles.

**Example**

```
setfillcolor([255]) %Sets fill colour to black using the Grayscale model

rectfill(1, 1, 1, 1) %Draws a black square

setfillcolor([125,0,0]) %Sets fill colour to medium red using the RGB model

rectfill(1, 1, 1, 1) %Draws a medium red square
```

```
setfillcolor([0,0,25,0]) %Sets fill colour to light yellow using the CMYK model
rectfill(0, 0, 1, 1) %Draws a light yellow square
```

## SetImageIndex (procedure)

Associates a data value with a PlanetPress Search index term defined using the **DefineImageIndex()** command. PlanetPress Image uses this information when it generates the .PDI file it creates for each PDF file it creates.

This function has no effect when the output of the document is not a PDF file generated through PlanetPress Image.

**Syntax**

setimageindex( name, data )

**Arguments**

*name*

String value specifying the name of the index term with which you want to associate a data value.

*data*

String value specifying the data value you want to associate with the index term.

**Code Sample Examples**

These examples illustrate **setimageindex()**.

**Example 1**

```
setimageindex( 'PONumber', '2005')
```

**Example 2**

```
setimageindex( 'PONumber', @(2,24,32) )
```

## SetLineWidth (procedure)

Sets the width of the pen used for drawing lines, boxes and other objects. When setting the line width, remember that the exact location of the pen is on the middle of the line, with the stroke spilling over equally on both sides.

**Syntax**

setlinewidth( width )

**Argument**

*width*[SetLineWidth (procedure)](#)

Measure value setting the width, in inches, of the drawing pen.

**Code Sample Examples**

**Example 1**

This example sets the pen width to 1/4 inch then draws a 2"x2" rectangle using that pen.

```
setlinewidth(0.25)
rectstroke(0,0,2,2)
```

## SetLPP(procedure)

Sets the number of lines per data page. This function is only useful when working with User-Defined Emulation, if you have a method of determining the number of LPP from within your data.

**Syntax**

setlpp( lines )

**Argument**

*lines*

Integer value specifying the number of lines per page.

## SetPDFBookmark (procedure)

Creates a PDF bookmark at the current page, for the PDF files PlanetPress Image generates.

>>> This function has no effect when the output of the document is not a PDF file generated through PlanetPress Image.

**Syntax**

setpdfbookmark( name )

**Arguments**

*name*

String value specifying the name of the bookmark that appears in the PDF.

**Code Sample Examples**

These examples illustrate **setpdfbookmark()**.

**Example 1**

```
setpdfbookmark('Table of Contents')
```

**Example 2**

```
setpdfbookmark( @(10,23,45) )
```

**Example 3**

```
setpdfbookmark( @(10,23,45) + '|' + @(11,23,45) )
```

# SetStrokeColor (procedure)

Sets the colour of the pen used for drawing lines, boxes and other objects.

## Syntax

setstrokecolor( colour )

## Argument

*colour*

Colour array that can be expressed expressed using one, three or four values ranging from 0 to 255. For Grayscale, a single value is used in the form [K]. For the RGB model, three values are used in the form [R, G, B]. For the CMYK model, four values are used in the form [C, M, Y, K].

## Code Sample Example

This example draws two empty rectangles of different colors.

## Example

```
setstrokecolor([0,0,100]) %Sets stroke color to medium blue using the RGB model
rectstroke(0,0,1,1) %Draws a medium blue rectangle
setstrokecolor([0,0,255,0]) %Sets stroke color to bright yellow using the CMYK model
rectstroke(1,1,1,1) %Draws a bright yellow rectangle
```

# SetStyle (procedure)

Sets the style to be used for all subsequent commands. The specified style must already exist in the document.

**SetStyle()** supercedes the **SetFont()** command. Although PlanetPress Talk still recognizes **SetFont()**, this behavior is unlikely to continue in future versions, and it is therefore highly recommended that you use the **SetStyle()** command in your scripts.

## Syntax

setstyle( stylename )

**Argument**

*stylename*

Name of the style to use for all subsequent commands. The name of the style must be preceded by an ampersand. You can reference the bold, italic, or underlined versions of a font by appending the appropriate letter or letters to the name of the style. Append *.b* for the bold version, *.i* for the italic version, *.bi* for the bold italic version, and *.u* for the underlined version.

**Code Sample Example**

This example displays a line of text using two different existing styles called Blackfont and Bluefont.

**Example**

```
margin(1,1)
setstyle(&blackfont)
show('The word ')
setstyle(&bluefont.i)
show('blue')
setstyle(&blackfont)
show(' is now in blue italic')
```

## SetStyleExt (procedure)

Select an existing style and set the size, color, style property, or font ratio for its font. This eliminates the need to create several styles that use the same font. **setstyleext**() does not support bold, italic, or underline styles with double-byte character sets.

**Syntax**

setstyleext(stylename, fontsize, styleproperty, fontcolor, fontratio)

**Arguments**

*stylename*

The name of the style. The name of the style must be preceded by an ampersand. You can set the bold, italic, or bold italic property of the style by appending .b, .i, or .bi respectively to the name of the style.

> The ability to append .b, .i, or .bi to style names that have double-byte character sets is not supported.

*fontsize*

Measure value representing the point size for the font. A value of -1 use the default font size of the style specified by stylename.

*styleproperty*

Integer value representing the style property for the font:

- 0: Normal
- 1: Underline
- 2: Outline

*fontcolor*

Colour array that can be expressed using one, three or four values ranging from 0 to 255. For Grayscale, a single value between 0 and 100 is used in the form [K], representing percentage of gray (100 is black, 0 is white). For the RGB model, three values are used in the form [R, G, B]. For the CMYK model, four values are used in the form [C, M, Y, K].

*fontratio*

Integer value representing the percentage by which to shrink or stretch the font spacing. This value adjusts both the width of each glyph and the spacing between glyphs. This is in contrast to kerning, which modifies the spacing between characters without modifying the width of characters.

Example

```
setstyleext(&Style2.bi, 25, [100], 100)
```

# Show / ShowCenter / ShowRight (procedure)

These three commands are used to display simple text on the document. They are identical in all aspects, except in the way they behave relative to the current margin settings: **show** uses the margin as a starting point to extend the string to the right; **showright** uses the margin as the ending point of the string; and **showcenter** extends the string equally on both sides of the margin. These commands are equivalent to *print*, *echo* or *write* commands in other scripting languages.

## Syntax

show( text )

showcenter( text )

showright( text )

## Argument

*text*

String value to be displayed.

## Code Sample Example

This example illustrates the difference between all three variations of using the SHOW command.

## Example

```
margin(3,1)
showright('This line is to the left of the margin')
crlf()
showcenter('This line is centered on the margin')
crlf()
show('This line is to the right of the margin')
```

# ShowBarCode (procedure)

**Note: This PressTalk command is deprecated.  Use instead the barcode-specific commands.**

Displays a specified string as a barcode.

**Syntax**

showbarcode( string )

**Arguments**

*string*

Value of type string to display as a barcode.

showbarcode needs 3 additional parameters before it can display the barcode. These parameters must be set before calling **showbarcode**, using the following variables:

*BarWidth*

Measure value setting the width of each bar, in inches.

*BarHeight*

Measure value setting the height of each bar, in inches.

*BarType*

Integer value specifying the type of barcode to print. Current possible values include:

| BarType: | Code: |
|----------|-------|
| 0 | Code 39 |
| 1 | Code 128 |
| 2 | UPC_A |
| 3 | UPC_E |
| 4 | UPC/EAN 8 |
| 5 | UPC/EAN 13 |
| 6 | PostNet |
| 7 | PDF-417 |

**Code Sample Example**

This example prints a Code 128 barcode.

The **BarWidth**, **BarHeight** and **BarType** variables must be created exactly as shown here, with the same case changes in the variable names.

**Example**

```
define(&BarWidth,measure,0.012)

define(&BarHeight,measure,1.000)

define(&BarType,integer,1)

%Set bar color to black and display barcode

setfillcolor([100,100,100,100])

showbarcode('Bar 128')
```

## ShowBarcode2of5(procedure)

Displays a specified string as a Standard 2 of 5 barcode.

**Syntax**

showbarcode2of5( str, barwidth, readable, bearers)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

*readable*

Boolean that enables display of the human readable portion of the barcode

*bearers*

Boolean that enables bearer bars surrounding the barcode

**Code Sample Example**

```
showbarcode2of5('123456789',12,true,true)
```

## ShowBarcodeAustPost (procedure)

Displays a specified string as an Australia Post barcode.

**Syntax**

showbarcodeaustpost( str)

**Arguments**

*str*

Value of type string to display as a barcode.

**Code Sample Example**

```
showbarcodeaustpost('56439111ABA 9')
```

## ShowBarcodeAztec (procedure)

Displays a specified string as an Aztec barcode.

**Syntax**

showbarcodeaztec( str, mode, errorcorrection, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*mode*

Mode to use for the barcode.

| Value | Mode |
|-------|------|
| 0 | Normal |
| 1 | Compact |
| 2 | Full Range |

*errorcorrection*

Error correction level to use in the barcode.

| Mode | Error Correction Range |
|------|------------------------|
| 0 | 0-99 |
| 1 | 0-4 |
| 2 | 0-32 |

*barwidth*

Measure value of the width a a single barcode bar, in mils (0.001 inch).

**Code Sample Example**

```
showbarcodeaztec('A 12345 B 434343',5,35,24)
```

# ShowBarcodeCodabar (procedure)

Displays a specified string as a Codabar barcode.

**Syntax**

showbarcodecodabar( str, start, stop, barwidth, readable)

**Arguments**

*str*

Value of type string to display as a barcode.

*Start*

The Start character of the barcode

---

A

B

C

D

*Stop*

The Stop character of the barcode

 A

 B

 C

 D

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

*readable*

Boolean that enables display of the human readable portion of the barcode

**Code Sample Example**

```
showbarcodecodabar('12345','A','A',12,true)
```

## ShowBarcodeCodablockF (procedure)

Displays a specified string as a Codablock F barcode.

### Syntax

showbarcodecodablockf( str, barwidth)

### Arguments

*str*

Value of type string to display as a barcode.

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

**Code Sample Example**

```
showbarcodecodablockf('12345',12)
```

## ShowBarcodeCode11 (procedure)

Displays a specified string as a Code 11 barcode.

**Syntax**

showbarcodecode11( str, barwidth, checksum, readable, readchecksum)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

*Checksum*

Boolean allows the automatic calculation of the barcode checksum

*readable*

Boolean that enables display of the human readable portion of the barcode

*readchecksum*

Boolean that enables display of the checksum in the human readable portion of the barcode

**Code Sample Example**

```
showbarcodecode11('123456789',12,true,true,false)
```

## ShowBarcodeCode128 (procedure)

Displays a specified string as a Code 128 barcode.

**Syntax**

showbarcodecode128( str, subset, barwidth, readable)

**Arguments**

*str*

Value of type string to display as a barcode.

*subset*

Integer specifying the subset of the barcode to use.

Value Subset

0    A

1    B

2    C

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

*readable*

Boolean that enables display of the human readable portion of the barcode

**Code Sample Example**

```
showbarcodecode128('123456789',0,12,true)
```

# ShowBarcodeCode16k (procedure)

Displays a specified string as a Code 16k barcode.

**Syntax**

showbarcodecode16k( str, mode, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*Mode*

Mode (0-7) of the Code 16k barcode

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

**Code Sample Example**

```
showbarcodecode16k('12345',4,12)
```

# ShowBarcodeCode39 (procedure)

Displays a specified string as a Code 3 of 9 barcode.

**Syntax**

showbarcodecode39( str, barwidth, checksum, readable, readchecksum)

**Arguments**

*str*

Value of type string to display as a barcode.

*BarWidth*

Measure value setting the width of each bar, in mils (0.001 inch).

*Checksum*

Boolean value that indicates if the checksum should automatically be calculated.

*Readable*

Boolean value that indicates if the human readable should be shown

*ReadChecksum*

Boolean value that indicates if the checksum value should be included in the human readable

**Code Sample Example**

```
showbarcodecode39('123456',12,true,true,false)
```

# ShowBarcodeCode49 (procedure)

Displays a specified string as a Code 49 barcode.

**Syntax**

showbarcodecode49( str, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

**Code Sample Example**

```
showbarcodecode49('12345', 12)
```

# ShowBarcodeCode93 (procedure)

Displays a specified string as a Code 93 barcode.

**Syntax**

showbarcodecode93( str, barwidth, readable)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

*readable*

Boolean that enables display of the human readable portion of the barcode

**Code Sample Example**

```
showbarcodecode93('123456789',12,true)
```

# ShowBarcodeDatamatrix (procedure)

Displays a specified string as a Datamatrix barcode.

**Syntax**

showbarcodedatamatrix( str, mode, errorcorrection, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*Mode*

Integer value of the Datamatrix Mode to use.

| Value | Mode |
|-------|------|
| 0 | Square |
| 1 | Rectangular |

*errorCorrection*

Error correction level to use with the barcode.

| Mode | Allowed Range |
|------|---------------|
| Square | 1-24 |
| Rectangular | 1-6 |

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

**Code Sample Example**

```
showbarcodedatamatrix('12345', 0, 5, 12)
```

# ShowBarcodeEAN8 (procedure)

Displays a specified string as a EAN-8 barcode.

**Syntax**

showbarcodeean8( str, subset,[readable, barwidth] )

**Arguments**

*str*

Value of type string to display as a barcode.

*Subset*

The subset of EAN-8 to use

0 EAN-8
1 EAN-8 ext.2
2 EAN-8 ext.5

*readable*

Optional boolean that enables display of the human readable portion of the barcode. Default = True.

*barwidth*

Optional width of one barcode bar, in mils (0.001 inch). Default = 12mils.

**Code Sample Example**

```
showbarcodeean8('0133558',0)
```

# ShowBarcodeEAN13 (procedure)

Displays a specified string as a EAN-13 barcode.

**Syntax**

showbarcodeean13( str, subset,[readable, barwidth] )

**Arguments**

*str*

Value of type string to display as a barcode.

*Subset*

The subset of EAN-13 to use

 0 EAN-13
 1 EAN-13 ext.2
 2 EAN-13 ext.5

*readable*

Optional boolean that enables display of the human readable portion of the barcode. Default = True.

*barwidth*

Optional width of one barcode bar, in mils (0.001 inch). Default = 12mils.

**Code Sample Example**

```
showbarcodeean13('977147396801',0)
```

## ShowBarcodeFIM (procedure)

Displays a specified string as a FIM barcode.

**Syntax**

showbarcodefim( str)

**Arguments**

*str*

Value of type string to display as a barcode.

**Code Sample Example**

```
showbarcodefim('A')
```

## ShowBarcodeI2of5 (procedure)

Displays a specified string as an Interleaved 2 of 5 barcode.

**Syntax**

showbarcodei2of5( str, barwidth, checksum, readable, readchecksum, bearers)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

*Checksum*

Boolean to add checksum automatically to barcode

*readable*

Boolean that enables display of the human readable portion of the barcode

*readchecksum*

Boolean that enables display of the checksum in the human readable portion of the barcode

*bearers*

Boolean that enables bearer bars surrounding the barcode

## Code Sample Example

```
showbarcodei2of5('123456789',12,true,true,false,true)
```

# ShowBarcodeISBN (procedure)

Displays a specified string as an ISBN barcode.

## Syntax

showbarcodeisbn( str, subset,[readable, barwidth] )

## Arguments

*str*

Value of type string to display as a barcode.

*Subset*

The subset of ISBN to use

 0 ISBN
 1 ISBN ext.2
 2 ISBN ext.5

*readable*

Optional boolean that enables display of the human readable portion of the barcode. Default = True.

*barwidth*

Optional width of one barcode bar, in mils (0.001 inch). Default = 12mils.

## Code Sample Example

```
showbarcodeisbn('978-1-86074-271-2-54495',2)
```

## ShowBarcodeJapanpost (procedure)

Displays a specified string as a Japan Post barcode.

**Syntax**

showbarcodejapanpost( str, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*BarWidth*

Measure value setting the width of each bar, in mils (0.001 inch).

**Code Sample Example**

```
showbarcodejapanpost('6540123789-A-K-Z',12)
```

## ShowBarcodeMaxicode (procedure)

Displays a specified string as a Maxicode barcode.

**Syntax**

showbarcodemaxicode( str, mode)

**Arguments**

*str*

Value of type string to display as a barcode.

*Mode*

Integer value (2-6) of the Maxicode Mode to use.

**Code Sample Example**

```
show-
bar-
codemaxicode('^059^042^041^059^040<RS>01^02996152382802^029840^029001^0291Z00004951^029UPSN^029(
 ALPHA DR^029PITTSBURGH^029PA^030<ET>',2)
```

## ShowBarcodeMicroPDF (procedure)

Displays a specified string as a Micro PDF-417 barcode.

**Syntax**

showbarcodemicropdf( str, mode, columns, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*Mode*

Integer value of the Micro PDF mode to use.

| Value | Mode |
|-------|------|
| 0 | Text |
| 1 | Numeric |

*columns*

Integer value (1-4) of the number of columns to use.

*barwidth*

Integer value of the barwidth of a single bar in the Micro PDF

**Code Sample Example**

```
showbarcodemicropdf('123456',0,4,12)
```

## ShowBarcodeMicroQR (procedure)

Displays a specified string as a Micro QR Code barcode.

**Syntax**

showbarcodemicroqr( str, mode, errorcorrection, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*Mode*

Integer value of the Micro QR mode to use.

| Value | Mode |
|-------|------|

| 0 | Alpha |
|---|---|
| 1 | Numeric |

*errorcorrection*

Integer value of the error correction level to use.

| Value | Correction Level |
|---|---|
| 0 | L |
| 1 | S |
| 2 | Q |

*barwidth*

Integer value of the barwidth of a single bar in the Micro QR

**Code Sample Example**

```
showbarcodemicroqr('123456',0,2,12)
```

# ShowBarcodeMSI (procedure)

Displays a specified string as a MSI barcode.

**Syntax**

showbarcodemsi( str, barwidth, readable)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Measure of the width of a single barcode bar, in mils (0.001 inch).

*readable*

Boolean value determining if the human readable portion of the barcode is visible.

**Code Sample Example**

```
showbarcodemsi('1234567')
```

# ShowBarcodeOnecode (procedure)

Displays a specified string as a Onecode/IMB barcode.

**Syntax**

showbarcodeonecode( str)

**Arguments**

*str*

Value of type string to display as a barcode.

**Code Sample Example**

```
showbarcodeonecode('01234567094987654321101234567891')
```

# ShowBarcodePDF417 (procedure)

Displays a specified string as a PDF 417 Code barcode.

**Syntax**

showbarcodepdf417( str, mode, columns, truncated, errorcorrection, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*Mode*

Integer value of the PDF 417 mode to use.

Value Mode

0    Text
1    ASCII Plus
2    Numeric

*columns*

Integer value (1-20) of the number of columns to use.

*truncated*

Boolean value to decide to use the truncated version of the barcode

*errorcorrection*

Integer value (0-8) of the error correction level to use.

*barwidth*

Integer value of the barwidth of a single bar in the PDF 417

**Code Sample Example**

```
showbarcodepdf417('123456',0,2,false,5,12)
```

## ShowBarcodePlessey (procedure)

Displays a specified string as a plessey barcode.

**Syntax**

showbarcodeplessey( str, barwidth, readable)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Measure of the width of a single barcode bar, in mils (0.001 inch).

*readable*

Boolean value determining if the human readable portion of the barcode is visible.

**Code Sample Example**

```
showbarcodeplessey('1234567')
```

## ShowBarcodePostnet (procedure)

Displays a specified string as a Postnet barcode.

**Syntax**

showbarcodepostnet( str, subset)

**Arguments**

*str*

Value of type string to display as a barcode.

*subset*

Subset of the Postnet barcode to use

| Subset | Zipcode Length |
|--------|----------------|
| 0      | 5 digits       |
| 1      | 9 digits       |
| 2      | 11 digits      |

**Code Sample Example**

```
showbarcodepostnet('12345',0)
```

## ShowBarcodeQRCode (procedure)

Displays a specified string as a QR Code barcode.

**Syntax**

showbarcodeqrcode( str, mode, errorcorrection, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*Mode*

Integer value of the QR Code mode to use.

| Value | Mode |
|-------|---------|
| 0 | Numeric |
| 1 | Alpha |

*errorcorrection*

Integer value of the error correction level to use.

| Value | Mode |
|-------|------|
| 0 | L |
| 1 | S |
| 2 | Q |
| 3 | H |

*barwidth*

Integer value of the barwidth of a single bar in the QR Code

**Code Sample Example**

```
showbarcodeqrcode('123456',0,2,12)
```

## ShowBarcodeRoyalMail (procedure)

Displays a specified string as a Royal Mail barcode.

**Syntax**

showbarcoderoyalmail( str)

**Arguments**

*str*

Value of type string to display as a barcode.

**Code Sample Example**

```
showbarcoderoyalmail('LE28HS9Z')
```

## ShowBarcodeRSS (procedure)

Displays a specified string as a RSS barcode.

**Syntax**

showbarcoderss( str, mode, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*Mode*

Integer value of the RSS mode to use.

| Value | Mode |
|-------|------|
| 0 | RSS-14 |
| 1 | RSS-Truncated |
| 2 | RSS-Limited |
| 3 | RSS-Stacked |
| 4 | RSS-Stacked Omni |
| 5 | RSS-Expanded |
| 6 | RSS-Expanded Stacked |

*barwidth*

Integer value of the barwidth of a single bar in the RSS Code

**Code Sample Example**

```
showbarcoderss('123456',0,12)
```

## ShowBarcodeUPCA (procedure)

Displays a specified string as a UPC-A barcode.

**Syntax**

showbarcodeupca( str, subset,[readable, barwidth] )

**Arguments**

*str*

Value of type string to display as a barcode.

*Subset*

The subset of UPC-A to use

0 UPC-A
1 UPC-A ext.2
2 UPC-A ext.5

*readable*

Optional boolean that enables display of the human readable portion of the barcode. Default = True.

*barwidth*

Optional width of one barcode bar, in mils (0.001 inch). Default = 12mils.

**Code Sample Example**

```
showbarcodeupca('123456789012',0)
```

# ShowBarcodeUPCE (procedure)

Displays a specified string as a UPC-E barcode.

**Syntax**

showbarcodeupce( str, subset,[readable, barwidth] )

**Arguments**

*str*

Value of type string to display as a barcode.

*Subset*

The subset of UPC-E to use

0 UPC-E
1 UPC-E ext.2
2 UPC-E ext.5

*readable*

Optional boolean that enables display of the human readable portion of the barcode. Default = True.

*barwidth*

Optional width of one barcode bar, in mils (0.001 inch). Default = 12mils.

## Code Sample Example

```
showbarcodeupce('123456',0)
```

## ShowBitmap (procedure)

Displays a bitmap resource.

### Syntax

showbitmap( resname, resolution, width, height[, transparent[, duotone]] )

### Arguments

*resname*

String value containing either the name of the bitmap resource within the document, or the path to a bitmap file.

*resolution*

Integer value specifying the resolution, in pixels per inch, at which the bitmap displays. A larger number yields a smaller, clearer image, since more pixels are squeezed into a smaller space. But the bitmap then requires more memory. Except in some very specific applications, it is rarely desirable to use resolutions exceeding 100 ppi.

*width, height*

Measure values specifying the width/height, in inches, the bitmap occupies when it displays. Specifying a value of zero for one but not both of the two dimensions scales the bitmap to fit the non-zero dimension. Specifying both a width and a height scales the bitmap to the best fit possible with that width and height, while maintaining its aspect ratio. Specifying both a width and a height of zero displays the bitmap at its default resolution.

*transparent*

Boolean value specifying whether part of the bitmap is transparent. When set to true, the bitmap image is combined with the background; otherwise it is pasted on top.

*duotone*

Colour array specifying the colour to paint all non-white pixels in the bitmap.

### Code Sample Examples

### Example 1

This example displays the bitmap image street_photo at a resolution of 72 pixels per inch, and a width of 1 inch, as a transparent bitmap image with non-white pixels set to blue.

```
showbitmap('street_photo',72,1,0,true,[100,100,0,0])
```

## Example 2

This example displays the bitmap image sunset at a resolution of 300 pixels per inch.

```
showbitmap('sunset',300,0,0)
```

## Example 3

This example prints either an image, or, if the image cannot be found, the pathname to the image.

```
define( &image_paths, directory, 'c:\\images\\*.JPG' )

define( &i, integer, 1 )

if( resourcetype( &image_paths[&i] ) <> 0 )

  showbitmap( &image_paths[&i], 300, 0, 0)

  crlf( bitmapheight( &image_paths[&i] ) )

elseif()

  show( &image_paths[&i] )

  crlf( 0.16 )

endif()
```

## ShowCaptureUserArea (procedure)

Displays a Capture Field object.

### Syntax

showcaptureuserarea( name: string, areatype: integer, left: measure,top: measure, width: measure, height: measure, requiredfield: integer, disablerewrite: boolean, groupid: integer )

### Arguments

*name*

String that defines the name of the Capture Field object (aka "PressTalk ID"). Must be the same as defined in the metadata and can only be letters or numbers (no spaces or special characters).

*areatype*

Integer that defines what kind of area type to use, from the following list:

- 0 - Checkbox
- 1 - List Fields

- 2 - Text Area
- 3 - Multi-Area Field

*left, top*

Measure variables that define the position of the Capture Field, in inches, from the left and top of the page.

*width, height*

Measure variables that define the size, in inches, of the Capture Field.

*requiredfield*

Integer that defines what kind of requirement this field has, from the following list:

- 0 - Optional Field
- 1 - Mandatory Field
- 2 - Final Field

*disablerewrite*

Boolean that defines whether the field will accept new ink if some ink is already present in previous processing of the document. *True* prevents rewriting.

*groupid*

Integer value that defines a group for mandatory fields.

**Example**

```
definemeta('CapPatternSequence','', 0, 'job.group.document')
definemeta('CaptureField', 'Capture1;' + IntToStr(0) + ';2;' + FloatToStr((&physical.x *
72),4)  + ';' + FloatToStr((&physical.y*72),4) + ';' + FloatToStr(3.7861*72) + ';' +
FloatToStr(1.4223*72) + ';false;0', 2, 'job.group.document.datapage.page')
showcaptureuserarea('Capture1',2,&physical.x,&physical.y,&width,&height,0,false,0)
```

# ShowEPS (procedure)

Displays an EPS image resource.

**Syntax**

showeps( resname, width, height )

**Argument**

*resname*

String value containing either the name of the EPS resource within the document, or the path to an EPS file .

*width, height*

Measure values specifying the width/height, in inches, the EPS occupies when it displays. Specifying a value of zero for one but not both of the two dimensions scales the EPS to fit the non-zero dimension. Specifying both a width and a height scales the EPS to the best fit possible with that width and height, while maintaining its aspect ratio. Specifying both a width and a height of zero displays the EPS at its default resolution.

**Code Sample Example**

This example displays the EPS image 'my_eps', scaling it as necessary to fit a width of 4.5 inches.

**Example**

```
showeps('my_eps', 4.5, 0)
```

## ShowLeftRight (procedure)

Displays a string of characters, in inches, using a specified width.

**Syntax**

showleftright( text, width )

**Arguments**

*text*

String value to be displayed.

*width*

Measure value representing the amount of horizontal space to use for displaying the string. showleftright adjusts the amount of spaces between characters to insure the string fits exactly within the specified width.

**Code Sample Example**

This example shows how to fit strings of different sizes within the same box that has a width of 3 inches.

**Example**

```
margin(0,0)
rectstroke(0,0,3,3)
moveto(0,0.5)
showleftright('This fits well inside the box',3)
crlf(0.1599)
showleftright('But this one is a little more tight',3)
```

## ShowPage (procedure)

Instructs the printer to output the page and move on to the next instruction.

This command has no effect on screen, because it is intended to be executed only inside the printer.

**Syntax**

showpage()

**Argument**

None

**Code Sample Example**

This example specifies the number of pages to print at line 1, between columns 1 and 5.

**Example**

```
Define(&NumberOfPages,integer,strtoint(trim(@(1,1,5))))

Define(&x,integer,1)

for(&x,1,1,&NumberOfPages)

  execpage('Page1')

  showpage()

endfor()
```

## ShowPDF (procedure)

Displays a page of a PDF image resource.

**Syntax**

showpdf( resname, pagenum, width, height )

**Argument**

*resname*

String value containing either the name of the PDF resource within the document, or the path to a PDF file .

*pagenum*

Integer value specifying the page number within the PDF image resource.

*width, height*

Measure values specifying the width/height, in inches, the PDF page occupies when it displays. Specifying a value of zero for one but not both of the two dimensions scales the PDF page to fit the non-zero dimension. Specifying both a width and a height scales the PDF page to the best fit possible with that width and height, while maintaining its aspect ratio. Specifying both a width and a height of zero displays the PDF page at its default resolution.

**Code Sample Example**

This example displays page 5 of the PDF image resource 'new_models', at its default resolution.

**Example**

```
showpdf('new_models', 5, 0, 0)
```

## ShowUTF8 (procedure)

This command is used to display UTF8 text on the document. It is designed to be used within a **begi-nUTF8paragraph...endUTF8paragraph** structure. Its uses the paragraph properties defined in the **begi-nUTF8paragraph** command.

**Syntax**

showUTF8( text )

**Argument**

*text*

Static text string or data selection to be displayed. Bear in mind that the information, wether static or variable, must be in UTF8. The **maputf8** function can be used to convert the information to UTF8.

**Code Sample Example**

This example displays a UTF8 string within a justified paragraph with the text running from right to left.

**Example**

```
setlinewidth(0.007)

moveto(0,0)

rlineto(&width,0)

rlineto(0,&height)

rlineto(neg(&width),0)

rlineto(0,neg(&height))

closepath()

stroke()

moveto(&width/2,&height/18)


BeginUTF8Paragraph(0.0000,&width,0.0000,'leftright',0.1667,'rtl')

  SetStyleExt(&Style1,12.0000,0,[100],100)

  ShowUTF8('\u0623\u0633\u0627\u0633\u064B\u0627\u060C')

EndUTF8Paragraph()
```

# ShowUTF8Left / ShowUTF8Right / ShowUTF8Center (procedure)

These three commands are used to display simple text strings on the document. They are identical in all aspects, except in the way they behave relative to the text insertion point: **showUTF8Left** displays the text from the insertion point to the left; **showUTF8Right** displays the text from the insertion point to the right; and **showUTF8Center** displays the text equally on both sides of the insertion point.

These commands cannot be used with a **beginUTF8paragraph...endUTF8paragraph** structure.

**Syntax**

showUTF8left( text, mode)

showUTF8right( text, mode )

showUTF8center( text, mode )

**Argument**

*text*

Static text string or data selection to be displayed. Bear in mind that the information, wether static or variable, must be in UTF8. The **maputf8** function can be used to convert the information to UTF8.

*mode*

String value specifying the contextual analysis is to be performed. Possible values are 'normal', 'reverse', and 'none' (for no contextual analysis).

**Code Sample Examples**

**Example 1**

This example illustrates a static text string (the name Saudi Arabia) escaped to its UTF8 reference form that will be displayed from the insertion point to the left.

```
setlinewidth(0.007)

moveto(0,0)

rlineto(&width,0)

rlineto(0,&height)

rlineto(neg(&width),0)

rlineto(0,neg(&height))

closepath()

stroke()

moveto(&width/2,&height/2)

SetStyleExt(&Style1,12.0000,0,[100],100)

showUTF8left('\u0627\u0644\u0633\u0639\u0648\u062F\u064A\u0629',        'normal')
```

**Example 2**

This example shows a variable text string that is taken from the data, converted from the MS-CP-1256 encoding to UTF8 and then displayed from the insertion point to the right.

```
setlinewidth(0.007)

moveto(0,0)

rlineto(&width,0)

rlineto(0,&height)

rlineto(neg(&width),0)

rlineto(0,neg(&height))

closepath()

stroke()

moveto(&width/2,&height/2)

SetStyleExt(&Style1,12.0000,0,[100],100)

showUTF8right(maputf8(@(2,15,35), 'MS-CP-1256', False),'normal')
```

## Sin (function)

Returns the sine value of the specified angle.

**Syntax**

sin( value ) ⇒ measure value

**Argument**

*value*

Measure value specifying the angle whose sine is returned.

**Code Sample Example**

This example draws a somewhat sinusoidal graph.

**Example**

```
moveto(0,0)

define(&i,integer,0)

for(&i,1,10,360)

  lineto(cos(inttofloat((&i)/ 4)),sin(inttofloat(&i)))

endfor()

closepath()
```

## StopJob (procedure)

Terminates execution of the document and returns control to the PostScript interpreter.

**Syntax**

stopjob()

## Store (procedure)

Stores a string of characters on a specific line in the current data page.

**Syntax**

store( line, string )

**Arguments**

*line*

Integer value specifying the line on which to store the string.

*string*

String value to store in the data page.

**Code Sample Example**

This example is taken for an OnReadDataPage event for a user defined emulation. It shows how to store lines of text in the current data page. It searches for a formfeed, increases the current line, stores the string, executes the document, resets the page, and then ends the search.

**Example**

```
search(&str,'\014')

  set(&current.line,&current.line+1)

  store(&current.line,&str)

  doform()

  clearpage()

endsearch()
```

## StringReplace (function)

Replaces all occurrences of a pattern within a string, with another pattern.

**Syntax**

stringreplace( string, old, new) ⇒ string

**Arguments**

*string*

String in which you want to replace a pattern.

*old*

String value that defines the pattern you want to replace.

*new*

String value that represents the replacement pattern.

**Code Sample Example**

This example replaces the dollar currency symbol with the pound currency symbol.

**Example**

```
stringreplace('1,000.00$','$','£')
```

# StringWidth (function)

Returns the physical display width, in inches, of a string.

**Syntax**

stringwidth( string ) ⇒ measure value

**Argument**

*string*

String value whose width is returned.

**Code Sample Example**

This example draws a tight box around a variable length data selection.

**Example**

```
margin(0,0)

%Define the data. This could be a data selection.

define(&data,string,'This is a sample string')

set(&data,trimright(&data))

%Store the length
```

PLANETPRESS. SUITE

```
define(&dw,measure,stringwidth(&data))

moveto(1,1)

%Display the string, then draw a box around it.

show(&data)

rectstroke(1,0.86,&dw,0.2)
```

## StringWidthUTF8 (function)

Returns the length, in inches, of the string as it would be displayed on a page, using the current font. Bear in mind that the combination of many bytes sometimes results in a single glyph and that text orientation can also affect the number of glyphs displayed.

### Syntax

stringwidthutf8(text, mode)

### Arguments

*text*

UTF8 text string to be measured.

*mode*

String specifying how the contextual analysis is to be performed. Possible values are 'normal', 'reverse', and 'none' (for no contextual analysis).

## Strip (function)

Removes all occurrences of a string within another string.

### Syntax

strip( string1, string2 ) ⇒ string value

### Arguments

*string1*

String value to be removed from string2.

*string2*

String value to change.

### Code Sample Example

This example strips all dollar signs from a variable.

### Example

```
define(&data,string,'$11.95')
```

```
show(strip('$',&data))
```

## Stroke (procedure)

Outlines the current shape using the pen colour and width specified with the **setstrokecolor** and **setlinewidth** command. The shape is not filled. To fill and outline the shape, use **strokeandfill** instead. If you only want to fill the shape, use **fill**.

**Syntax**

stroke()

**Argument**

None

**Code Sample Example**

This example draws a blue triangle with a blue border.

**Example**

```
setstrokecolor([100,100,0,0]) %Set pen colour to blue

moveto(1,1)

lineto(1.5,2)

lineto(0,2)

closepath()

stroke()
```

## StrokeAndFill (procedure)

Outlines the current shape using the pen colour and width specified with the **setstrokecolor** and **setlinewidth** command. The shape is then filled using the colour specified with the **setfillcolor** command. To simply outline the shape, use **stroke** instead. If you only want to fill the shape, use **fill**.

**Syntax**

strokeandfill()

**Argument**

None

**Code Sample Example**

This example draws a yellow triangle with a blue outline.

**Example**

```
setstrokecolor([100,100,0,0])

setfillcolor([0,0,100,0])

moveto(1,1)

lineto(1.5,2)

lineto(0,2)

closepath()

strokeandfill()
```

## StrToCur (function)

Converts a string value into a currency value.

**Syntax**

strtocur( string )  ⇒  currency value

**Argument**

*string*

String value to convert.

## StrToFloat (function)

Converts a string into a measure value.

**Syntax**

strtofloat( string )  ⇒  measure value

**Argument**

*string*

String value to convert.

**Code Sample Example**

This example reads a column of 6 values then converts the values to floating point values and then adds the ir values to the &total variable. The total value is subsequently converted back to a string and the result is printed.

**Example**

```
define(&total,measure,0.0)

define(&x,integer,0)
```

```
for(&x,1,1,6)

   set(&total,&total+strtofloat(@(&x,10,18)))

endfor()

show(floattostr(&total))
```

## StrToInt (function)

Converts a string into an integer value.

### Syntax

strtoint( string ) ➡ integer value

### Argument

*string*

String value to convert.

### Code Sample Example

This example reads a column of 6 values then converts the values to integers and then adds the integer to the &total variable. The total value is subsequently converted back to a string and the result is printed.

### Example

```
define(&total,integer,0)

define(&x,integer,0)

for(&x,1,1,6)

   set(&total,&total+strtoint(@(&x,10,18)))

endfor()

show(inttostr(&total))
```

## Sub (function)

Subtracts one expression from another. This is the functional equivalent of the - operator.

### Syntax

sub( expression, expression2 ) ➡ integer, measure, currency value

### Arguments

*expression1, expression2*

Integer, measure or currency values. Both expressions must be the same type. The returned value is set according to the type of the parameters.

**Code Sample Example**

Example 1 and Example 2 illustrates both ways of subtracting numbers.

**Example 1**

```
show(inttostr(2-2))
```

**Example 2**

```
show(floattostr(sub(4.7,2.1)))
```

# SubRecCount (function)

Returns the number of records per data page. Since data pages can hold a variable number of records, this value may change whenever you skip through data pages. Every data page always has at least one child, otherwise it wouldn't be stored in the database. Subreccount can return 0 only if the emulation is set to something other than database mode, or if the converted database is invalid. This function is only useful in database emulation mode.

**Syntax**

subreccount ⇒ integer value

**Code Sample Example**

This example prints the line items from an invoice. It defines the loop and total variables, loops through all records, shows and positions invoice information, and finally shows the invoice total. The fields listed here are for illustration purposes only.

**Example**

```
define(&x,integer,1)
define(&tot,measure,0.0)
for(&x,1,1,subreccount())
  show(field('PartNum'))
  moveto(1.5,&current.y)
  show(field('PartDesc'))
  moveto(4.5,&current.y)
  show(field('Qty'))
  moveto(5.5,&current.y)
```

```
    show(field('UnitPrice'))

    moveto(6.5,&current.y)

    show(field('Total'))

    %Add line total to tot variable

    set(&tot,&tot+ strtofloat(field('Total')))

    moveto(0,&current.y+.25)

endfor()

moveto(6.5,8)

show(floattostr(&tot))
```

## Time (function)

Returns the print time. Note that this function *will only work if the document runs on a computer*, since it is impossible for the document to get the current time from a printer. So if you send your document to a printer and then simply send data with the appropriate trigger to that printer, the document will run on the printer and the function will return an empty string. Use the Run locally option, available in the PlanetPress Suite Workflow Tools, to ensure that the document runs on a computer rather than on a printer.

### Syntax

time(displayseconds)

### Argument

*displayseconds*

Boolean specifying whether to display the time using seconds. True returns the time with seconds, False, without. The precise format of the of the time value strings are set in the Windows Regional options.

### Code Sample Example

This example shows how to add the current time with seconds (11:14:46 AM, as opposed to 11:14 AM, for example) on a document page.

### Example

```
show(time(true))
```

# Translate (procedure)

Offsets an object's position from its original spot on the page. The point of origin of any object is (0,0) by default. All commands within an object are relative to that point of origin. Moving, or translating the point of origin allows you to reset the point of origin within the boundaries of the object.

## Syntax

translate( x, y )

## Arguments

*x, y*

Measure values representing the horizontal/vertical offset, in inches, of the new point of origin within the object. These values are relative to the current point of origin, not absolute.

## Code Sample Examples

### Example 1

This example moves the point of origin 1 inch right and down from the original point.

```
translate(1,1)

show('some text')
```

### Example 2

This example moves the point of origin 1 inch right and down from the current point.

```
translate(1,1)

show('some more text')
```

### Example 3

This example moves the point of origin back to its original position. **Translate** sets the new point of origin for all subsequent commands. Consequently, **translate** commands are cumulative. Therefore, to restore the previous point of origin, you must issue a **translate** command that negates whatever offset was previously applied.

```
translate(-2,-2)
```

# Trim (function)

Removes both leading and trailing spaces from a string.

**Syntax**

trim( string ) ➡ string value

**Argument**

*string*

String value from which you want to trim leading and trailing spaces.

**Code Sample Example**

This example trims leading and trailing spaces from the data selection on line 3, columns 8 to 24.

**Example**

```
trim(@(3,8,24))
```

# TrimLeft (function)

Removes leading spaces from a string.

**Syntax**

trimleft( string ) ➡ string value

**Argument**

*string*

String value to trim.

**Code Sample Example**

This example makes sure * delimiters are added immediately before and after a data selection.

**Example**

```
show('*'+trimleft(trimright(@(12,10,35)))+'*')
```

# TrimRight (function)

Removes trailing spaces from a string.

**Syntax**

trimright( string ) ➡ string value

**Argument**

*string*

String value to trim.

**Code Sample Example**

["TrimLeft (function)" (page 547)](#)

## UpperCase (function)

Convert a string to all upper case characters.

**Syntax**

uppercase( string ) ➡ string

**Argument**

*string*

String value you want to convert to upper case. The string may be a mix of upper and lower case characters.

**Code Sample Example**

This example illustrates **uppercase()**.

**Example**

```
&month := uppercase( @(1,60,70) )
```

## xmlCount()

Counts the number of children for a specific element, according to standard XPath syntax. The return value is an integer and always returns 0 if the specified XPath is invalid or if no value is found. The XPath is always relative to the XML root path.

**Syntax**

xmlCount(path: string) ➡ integer value

**Arguments**

*path*

The XML Path of the value you want to retrieve. The path must correspond to a single entry in the XML, thus is not a complete XPath with regular expressions and variables.

**Code Sample**

The following gets the number of items in an invoice, loops through each item and displays the item's description with show().

```
define(&i, integer, 0)
define(&currentItemDescription, string, '')
```

```
define(&numberOfItems, integer, xmlcount('/MyData/CUSTOMER/INVOICES/INVOICE/ITEM'))

for(&i, 1, 1, &numberOfItems)
       &currentItemDescription := xmlget('-
/MyData/CUSTOMER/INVOICES/INVOICE/ITEM['+inttostr(&i)+']/Description')
       show(&currentItemDescription)
       crlf()
endfor()
```

## xmlGet()

Retrieves a data selection value through the structure of an XML file. This function uses a single parameter specifying the Path of the value to be retrieved. This function returns a string. The return value is an empty string if no value is found or if the specified Element is invalid or not found.

**Syntax**

xmlGet(path: string) ⇒ string value

**Arguments**

*path*

The XML Path of the value you want to retrieve. The path must correspond to a single entry in the XML, thus is not a complete XPath with regular expressions and variables.

**Code Sample**

These are examples of xmlGet() commands returning data from an XML file. Note that these will most likely not work with your own XML even if you tried, as the path is completely dependent on the exact structure of your XML Data file.

```
define(&myVar, string, '')
&myVar := xmlget('/PLANETPRESS_DATA_FILE[1]-
/CUSTOMER[1]/INVOICES[1]/INVOICE[1]/ITEM[1]/Description[1]')
&myVar := xmlget('/MyData/CUSTOMER[1]/INVOICES[1]/INVOICE[1]/ITEM[1]/Description[1]')
% This next line uses the &i variable to dynamically select an item in the invoice. Use-
ful in a repeat & overflow
% configuration, or a manual PlanetPress Talk loop through te data. Note that the var-
iable must be converted to a string.
&myVar := xmlget('-
/MyData/CUSTOMER[1]/INVOICES[1]/INVOICE[1]/ITEM['+inttostr(&i)+']/Description[1]')
```

Example 2

The following gets the number of items in an invoice, loops through each item and displays the item's description with show().

```
define(&i, integer, 0)
define(&currentItemDescription, string, '')

define(&numberOfItems, integer, xmlcount('/MyData/CUSTOMER/INVOICES/INVOICE/ITEM'))
```

```
for(&i, 1, 1, &numberOfItems)
      &currentItemDescription := xmlget('-
/MyData/CUSTOMER/INVOICES/INVOICE/ITEM['+inttostr(&i)+']/Description')
      show(&currentItemDescription)
      crlf()
endfor()
```

## XOr (Boolean operator function)

Returns true if only one of two specified expressions is true, false otherwise.

### Syntax

xor( expression1, expression2 ) ⇒ ' boolean value

expression1 xor expression2 ⇒ boolean value

### Arguments

*expression1, expression2*

Boolean values.

### Code Sample Examples

These examples illustrate the possible values for XOR.

### Example 1

```
xor(true, false) %Returns true
```

### Example 2

```
xor(false, true) %Returns true
```

### Example 3

```
xor(false, false) %Returns false
```

### Example 4

```
xor(true, true) %Returns false
```

# Language Reference (by element type)

This section contains a reference to PlanetPress Talk organized in by element type.

## System Variables

### &EOJob (system variable)

Read-only variable that returns *True* if the document has processed the last line of data or if the line that follows the last form feed character (ASCII 12) is empty.

> This variable always returns *True* at design-time when using a user-defined emulation.

Thus if a file ends with the sequence:

```
FF CR LF
```

**&eojob** becomes true before the last line (CR LF) since the last line is empty.

**Syntax**

&eojob ⇒ Boolean value

### &FirstSide (system variable)

Read-only variable that returns True when a page is printing on the first side of a page in duplex mode (double-sided printing). In simplex mode (single-sided) **&firstside** always returns True.

This variable is based on PostScript's **firstside** command as implemented in your printer. If your printer does not support **firstside**, the document attempts to detect which side of the page is currently printing. Passthrough commands can hamper the document's ability to return the correct value for **&firstside**.

**Syntax**

&firstside ⇒ Boolean value

### &Height (system variable)

System variable with a scope local to a specific object, group, or page. The system initializes this variable to the value of the Height property of the object's, group's, or page's Basic Attributes. You can reference and set this variable in any PlanetPress Talk code you enter in the object, group, or page.

**Syntax**

&height ⇒ measure value

## &PrinterMode (system variable)

Returns the current output mode for the document. This allows you to test for a specific value and react accordingly.

**Syntax**

&printermode ➡ integer value

**Possible Values**

| Value: | Description: |
|---|---|
| 0 | PlanetPress during document design |
| 1 | Preview or Softproof as Printer Queue |
| 2 | Preview or Softproof as PlanetPress Image |
| 3 | Preview or Softproof as PlanetPress Fax |
| 4 | Printer Preview |
| 5 | Printer Hard Disk |
| 6 | Printer Memory |
| 7 | Host Computer |
| 8 | Flash Memory |
| 9 | PlanetPress Image |
| 10 | PlanetPress Fax |
| 11 | PlanetPress Suite Workflow Tools |

**Code Sample Example**

This example changes some colors to shades of gray for greater legibility on a fax.

**Example**

```
setlinewidth(0.1) %Select a thick pen
if(&printermode=10) %If Destination is Fax
      setfillcolor([5,5,5,5]) %Set fill to light gray
      setstrokecolor([0,0,0,100]) %Set pen to black
elseif()
      %Otherwise,set fill colour to yellow
      setfillcolor([0,0,50,0])
      setstrokecolor([100,100,0,0]) %Set pen colour to blue
endif()
rectstroke(0,0,3,3) %Display rectangle
rectfill(0,0,3,3)
```

## &Str (system variable)

Contains a line of input data.

**Syntax**

&str ➡ string

**Code Sample Example**

This example illustrates a few lines of code from a user-defined emulation. In these lines, if the line of input data (minus any leading spaces)begins with the code '5', the emulation advances to the next line of the data page and stores the line of input data on that line. Notice the &current.line system variable—one of the variables in the current system object.

**Example**

```
if(eq(mid(trimleft(&str),1,1),'5'))
   set(&current.line,&current.line + 1)
   store(&current.line,trimleft(&str))
endif()
```

# &Width (system variable)

System variable with a scope local to a specific object or group. The system initializes this variable to the value of the Width property of the object's or group's Basic Attributes. You can reference and set this variable in any PlanetPress Talk code you enter in the object, group, or page.

**Syntax**

&width ⇒ measure value

# System Objects

# Current (system object)

This object includes a set of system variables reflecting the current state of the document. All of these variables are read-only and therefore cannot be changed using the set command, except where noted. You can access each property individually using the same syntax.

**Syntax**

&current.property => value

**Properties**

*datafilename*

String value containing the full path and the internal filename, not the original values, of the currently used data file. Note: This variable contains an empty string in printer centric mode.

*datapage*

Integer value containing the page number of the current data page, or the number of the current record set in the case of a database emulation. Note that if you use Metadata and you modify it in any way (sorting, filtering, sequencing) you should use &current.metadatapage instead.

*line*

Integer value containing the line number of the current line of data within the data page, or the number of the current record within the record set. This value is read-write and can be modified using the set command.

*lpp*

Integer value containing the number of lines per page in the current data page. In database emulation, this value can also be used in place of the subreccount() function to determine how many records a record set contains.

*mediacolor*

String value containing the color set in the Media color box of the Page dialog box for the current document page, or, if no value is set in that box, the color set in the Media color box in the Document dialog box for the document. The Media color box appears when you select Media selection in the Selection type box. Recall that the Media color box specifies the color of paper.

*mediatype*

String value containing the type set in the Media type box of the Page dialog box for the current document page, or, if no value is set in that box, the type set in the Media type box in the Document dialog box for the document. The Media type box appears when you select Media selection in the Selection type box. Recall that the Media type box specifies the type of paper (plain, coated, etc.).

*mediaweight*

Integer value containing the weight set in the Media weight box of the Page dialog box for the current document page, or, if no value is set in that box, the weight set in the Media weight box in the Document dialog box for the document. The Media weight box appears when you select Media selection in the Selection type box. Recall that the Media weight box specifies the weight of paper, in grams per square meter.

*metadatapage*

Integer value containing the page number of the current metadata page. This corresponds to the filtered, sorted and sequenced metadata.

*minfeature*

Measure value containing the width of a single pixel at the current zoom level. Precise to 1/65536 inch. This is the minimum size you can use in PlanetPress Talk code. If you use a smaller value (for example as an argument in a command), PlanetPress Talk automatically replaces that value with the value of **minfeature**.

*orientation*

Integer value containing the orientation of the current document page (0=portrait, 1=landscape, 2=Rotated portrait, 3=Rotated landscape).

*overflowcount*

Integer variable storing the number of times that a data page overflowed and that a new document page was added to accommodate the overflowing data. Initially set to 0, is incremented by 1 every time the page overflows.

*overflowing*

Boolean variable set to False by default. When the current *data page* contains more lines than can be printed by the current object, or technically speaking when the object's condition to exit and overflow has been met, this variable is set to True. It is set back to False with every new overflowing page, the logic being that the current overflowing page will be the last, unless if the condition to exit and overflow is met yet again.

*lastoverflowrepetition*

Integer variable storing the number of the last iteration value when the condition to exit and overflow has been met.

*pageheight*

Measure value containing the physical page height, in inches, of the current page in the document.

*printpagename*

String value containing the name of the page that is actually printing. This allows conditional processing to take place within a PPTalk object according to the name of the page being executed. Note that the value always corresponds to the name of the actual page page being printed, even if the call comes from a virtual or overlay page (in other words, the name of the page on which the overlay is layed out).

*pagewidth*

Measure value containing the physical page width, in inches, of the current page in the document.

*printpage*

Integer value containing the current page number in the document.

*x,y*

Measure value containing the current horizontal (x) or vertical (y) position of the drawing pen on the current page. This value is relative to the top left corner of the current page - whose coordinates are (0,0) - and may not necessarily be equal to the current physical position on the page.

## Physical (system object)

The **physical** object contains system variables that reflect the current state of the document. These system variables are read-only and therefore cannot be changed using the set command. You can access each system variable using the same syntax.

### Syntax

&physical.x ⇒ measure value

&physical.y ⇒ measure value

### Variables

*x,y*

Measure value containing the current horizontal (x) or vertical (y) position of the drawing pen, relative to the upper left corner of the physical page, whose coordinates are (0,0). This system object can be used to assess the amount of space left on a page.

## &system (system object)

The &system system object contains system variables related to the Raster Image Processor (RIP) the document is using to execute, and document version. These system variables are read-only.

### Syntax

- &system.product ⇒ string value
- &system.version ⇒ string value
- &system.formversion ⇒ integer value

### Properties

*product*

String value containing the name of the RIP the document is using to execute. For example,

PlanetPress Alambic is the name of the RIP built in to PlanetPress (the RIP PlanetPress uses when you perform a preview and select Internal interpreter as the PostScript interpreter).

*version*

String value containing the version number of the RIP the document is using to execute. For example, 4.0.

*formversion*

Integer value containing the version number of the document requested by the trigger. Recall that you can assign a version number to a document and increment the number each time you update and reinstall the document. When you execute a document that uses a version number, the document verifies its version number against the one you specify in the trigger, and it proceeds with execution only if the two version numbers match.

## Assignment Operator

## = (operator)

Assign a value to a variable. This provides a more convenient alternative to the **set()** command. Note that spaces between the operator and its operands are optional.

**Code Samples**

```
&price := 18.53
&day[4] := 'Friday'
&last:=True
&mustard := [0,20,90,16]
```

## Mathematical Operators and Operator Functions

## + (operator & function)

Either concatenates two or more strings or adds two numerical expressions, depending on its context.

This description details how to use it to concatenate two strings. See <u>"Add (function)" (page 549)</u> for help using it to add two numerical expressions.

**Syntax**

string1 + string2 + ... ⇒ string

**Arguments**

*string1, string2,...*

String values.

**Code Sample Example**

The following code sample concatenates five strings into a single string that contains the first and last name of a customer.

---

**Code Sample**

```
show('First Name: ' + @(2,12,36) + ' ' + 'Last Name: ' + @(3,12,36))
```

## Add (function)

Adds two numerical expressions. This is the functional equivalent to the + operator, when you use the + operator with numerical expressions.

### Syntax

add( expression, expression2 ) ➡ integer, measure, currency value

### Arguments

*expression1, expression2*

Integer, measure, or currency values. Both expressions must be the same type. The returned value is set according to the type of the parameters.

### Code Sample Examples

Examples 1 and 2 show both ways of adding numbers.

### Example 1

```
show(inttostr(2+2)) %Displays 4
```

### Example 2

```
show(floattostr(add(4.7,2.1))) %Displays 6.8
```

## – (operator)

See "Sub (function)" (page 537) and "Neg (function)" (page 533).

## Sub (function)

Subtracts one expression from another. This is the functional equivalent of the - operator.

### Syntax

sub( expression, expression2 ) ➡ integer, measure, currency value

### Arguments

*expression1, expression2*

Integer, measure or currency values. Both expressions must be the same type. The returned value is set according to the type of the parameters.

### Code Sample Example

Example 1 and Example 2 illustrates both ways of subtracting numbers.

**Example 1**

```
show(inttostr(2-2))
```

**Example 2**

```
show(floattostr(sub(4.7,2.1)))
```

## Asterisk (*) operator

See "Mul (function)" (page 533).

## Mul (function)

Multiplies two expressions. This is the functional equivalent of the * operator.

### Syntax

mul( expression, expression2 ) ⇒ integer, measure, currency value

### Arguments

*expression1, expression2*

Integer, measure or currency values. Both expressions must be of the same type. The returned value is set accordingly to the type of the parameters.

### Code Sample Examples

Examples 1 and 2 show both ways of multiplying numbers.

### Example 1

```
show(inttostr(2*2)) %Displays 4
```

### Example 2

```
show(floattostr(mul(4.7,2.1))) %Displays 9.87
```

## / (operator)

See "Div (function)" (page 530).

## Div (function)

Divides an expression with another. This is the functional equivalent to the / operator.

**Syntax**

div( expression, expression2 ) ⇒ integer, measure, currency value

**Arguments**

*expression1, expression2*

Integer, measure or currency values. Both expressions must be the same type. The returned value is set according to the type of the parameters.

**Code Sample Examples**

Examples 1 and 2 show both ways of dividing numbers.

**Example 1**

```
show(inttostr(2/2)) %Displays 1
```

**Example 2**

```
show(floattostr(div(4.7,2.1))) %Displays 2.2381
```

## Neg (function)

Returns the negative value of the specified expression. This is the functional equivalent to the - sign.

**Syntax**

neg( expression1 ) ⇒ integer, measure, currency value

**Argument**

*expression1*

Integer, measure or currency value. The returned value is the same type as expression1.

**Code Sample Examples**

Examples 1and 2 show both ways of negating a number. Negating negative numbers yields a positive result.

**Example 1**

```
-12 %Returns -12

-&max %Returns negative of the value of the variable &max

neg(-12.45) %Returns 12.45
```

## Cos (function)

Returns the cosine value of the specified angle.

PLANETPRESS. SUITE

**Syntax**

cos( value ) ⇒ measure value

**Argument**

*value*

Measure value specifying the angle whose cosine is returned.

**Code Sample Example**

This example displays a sinusoidal graph.

**Example**

```
moveto(0,0)
define(&i,integer,0)
for(&i,1,10,360)
        lineto(cos(inttofloat((&i)/ 4)),sin(inttofloat(&i)))
endfor()
closepath()
```

## Sin (function)

Returns the sine value of the specified angle.

**Syntax**

sin( value ) ⇒ measure value

**Argument**

*value*

Measure value specifying the angle whose sine is returned.

**Code Sample Example**

This example draws a somewhat sinusoidal graph.

**Example**

```
moveto(0,0)

define(&i,integer,0)

for(&i,1,10,360)

  lineto(cos(inttofloat((&i)/ 4)),sin(inttofloat(&i)))

endfor()

closepath()
```

# Random (function)

Returns a measure value between 0 and 1, non-inclusive. Since this function uses the current system time when run inside the printer, it returns a true random value. On the computer, however, the function uses the current data page as its seed to ensure the returned value is constant when you navigate from page to page on your document.

## Syntax

random() ⇒ measure value

## Argument

None

## Code Sample Example

This example displays 10 random numbers between 1 and 20.

## Example

```
define(&x,integer,1)
for(&x,1,1,10)
  show(inttostr(floattoint(random()*20)+1))
  crlf()
endfor()
```

# Ceil (function)

Returns the smallest integer greater than or equal to the specified measure value.

## Syntax

ceil( value ) ⇒ integer value

## Argument

*value*

Measure value. The absolute value of the measure value must be less than the maximum value of an integer in PostScript (2,147,483,647).

## Code Sample Examples

These examples illustrate various applications of **ceil()**.

```
ceil( -2.8 ) %Returns -2
ceil( 2.8 ) %Returns 3
ceil( -5.0 ) %Returns -5
```

# String Operator

## + (operator & function)

Either concatenates two or more strings or adds two numerical expressions, depending on its context.

This description details how to use it to concatenate two strings. See for help using it to add two numerical expressions.

**Syntax**

string1 + string2 + ... ⇒ string

**Arguments**

*string1, string2,...*

String values.

**Code Sample Example**

The following code sample concatenates five strings into a single string that contains the first and last name of a customer.

**Code Sample**

```
show('First Name: ' + @(2,12,36) + ' ' + 'Last Name: ' + @(3,12,36))
```

# Boolean Operator Functions

## And (Boolean operator function)

Returns true if both specified expressions are true, otherwise it is false.

**Syntax**

and( expression1, expression2 ) ⇒ Boolean value

expression1 and expression2 ⇒ Boolean value

**Arguments**

*expression1, expression2*

Boolean values.

**Code Sample Examples**

These examples illustrate the possible values for AND.

```
and(true, false) %Returns false
and(false, true) %Returns false
```

```
and(false, false) %Returns false
and(true, true) %Returns true
```

## Not (Boolean operator function)

Negates the specified expression. Not to be confused with the **neg** function: not is a Boolean operator function, not a mathematical one.

### Syntax

not( expression1 ) ⇒ Boolean value

### Argument

*expression1*

Boolean value.

### Code Sample Example

This example shows how NOT can be used in place of operators.

### Example

```
define(&x,integer,1) %Define loop variable

for(&x,1,1,10) %Set loop

  if(not(&x=5)) %Same as if(&x<>5)

    show('This is NOT the 5th line') %Show some text

  elseif()

    show('This IS the 5th line') %Show alternate

  endif()

endfor()
```

## Or (Boolean operator function)

Returns true if either or both specified expressions are true, false otherwise.

### Syntax

or( expression1, expression2 ) ⇒ Boolean value

expression1 or expression2 ⇒ Boolean value

### Arguments

*expression1, expression2*

Boolean values.

**Code Sample Examples**

These examples illustrate the possible values for OR.

**Example 1**

---

```
or(true, false) %Returns true
```

---

**Example 2**

---

```
or(false, true) %Returns true
```

---

**Example 3**

---

```
or(false, false) %Returns false
```

---

**Example 4**

---

```
or(true, true) %Returns true
```

---

# XOr (Boolean operator function)

Returns true if only one of two specified expressions is true, false otherwise.

**Syntax**

xor( expression1, expression2 )⇒ ' boolean value

expression1 xor expression2 ⇒ boolean value

**Arguments**

*expression1, expression2*

Boolean values.

**Code Sample Examples**

These examples illustrate the possible values for XOR.

### Example 1

```
xor(true, false) %Returns true
```

### Example 2

```
xor(false, true) %Returns true
```

### Example 3

```
xor(false, false) %Returns false
```

### Example 4

```
xor(true, true) %Returns false
```

## Comparison Operators and Operator Functions

## GT (function)

Compares two expressions of any type and returns true if the first is greater than the second, false otherwise. This is the functional equivalent to the > operator.

### Syntax

gt( expression1, expression2 ) ⇒ Boolean value

### Arguments

*expression1, expression2*

Values of any type. Both expressions must be of the same type.

### Code Sample Examples

Examples 1 and 2 illustrate both ways of comparing two expressions.

### Example 1

```
if( gt(&current.line, 50) )

  show('Bottom of page')

endif()
```

### Example 2

```
if(&current.line > 50)

  show('Bottom of page)

endif()
```

## Greater than (>) operator

See .

## GE (function)

Compares two expressions of any type and returns true if the first is greater than or equal to the second, false otherwise. This is the functional equivalent to the >= operator.

### Syntax

ge( expression1, expression2 ) ⇒ Boolean value

### Arguments

*expression1, expression2*

Values of any type. Both expressions must be of the same type.

### Code Sample Examples

Example 1 and Example 2 show both ways of comparing two expressions. Example 2 illustrates the >= operator.

### Example 1

```
if( ge(&current.line, 50) )
  show('Bottom of page')
endif()
```

### Example 2

```
if(&current.line >= 50)
  show('Bottom of page')
endif()
```

## Greater or equal to (>=) operator

See .

## LT (function)

Compares two expressions of any type and returns true if the first is less than the second, false otherwise. This is the functional equivalent to the < operator.

**Syntax**

lt( expression1, expression2 ) ⇒ Boolean value

**Arguments**

*expression1, expression2*

Values of any type. Both expressions must be of the same type.

**Code Sample Example**

This example shows both ways of comparing two expressions.

**Example**

```
if(lt(&current.line, 15))

   show('Top of page')

endif()

%Same statement, using the < operator

if(&current.line < 50)

   show('Top of page')

endif()
```

## Less than (<) operator

See .

## LE (function)

Compares two expressions of any type and returns true if the first is less than or equal to the second, false otherwise. This is the functional equivalent to the <= operator.

**Syntax**

le( expression1, expression2 ) ⇒ Boolean value

**Arguments**

*expression1, expression2*

Values of any type. Both expressions must be of the same type.

**Code Sample Example**

This example shows both ways of comparing two expressions.

**Example**

```
if(le(&current.line, 15) )

   show('Top of page')
```

```
endif()

%Same statement, using the <= operator

if(&current.line <= 50)

  show('Top of page')

endif()
```

## Less or equal to (<=) operator

See "LT (function)" (page 554).

## Eq (function)

Compares two expressions of any type and returns true if both are equal, false otherwise. This is the functional equivalent of the = operator.

### Syntax

eq( expression1, expression2 ) ⇒ boolean value

### Arguments

*expression1, expression2*

Values of any type. Both expressions must be of the same type.

### Code Sample Examples

Examples 1 and 2 shows both ways of comparing two expressions.

### Example 1

```
if( eq(&current.line, 33) )
  show('Middle of page')
endif()
```

### Example 2

```
if(&current.line = 33)
  show('Middle of page')
endif()
```

## = (operator)

Assign a value to a variable. This provides a more convenient alternative to the **set()** command. Note that spaces between the operator and its operands are optional.

### Code Samples

```
&price := 18.53
&day[4] := 'Friday'
&last:=True
&mustard := [0,20,90,16]
```

## NE (function)

Compares two expressions of any type and returns true if they are not equal, false otherwise. This is the functional equivalent to the <> operator.

### Syntax

ne( expression1, expression2 ) ⇒ Boolean value

### Arguments

*expression1, expression2*

Values of any type. Both expressions must be of the same type.

### Code Sample Examples

Examples 1 and 2 show both ways of comparing two expressions.

### Example 1

```
if( ne(&current.line, 1) )

  show('Not a new page')

endif()
```

### Example 2

```
if(&current.line <> 1) %Same statement, using the <> operator

  show('Not a new page')

endif()
```

## <> (operator)

See

## Conversion Operator Functions

# FloatToCur (function)

Converts a measure expression into a currency value.

**Syntax**

floattocur( expression ) ⇒ currency value

**Argument**

*expression*

Measure value to be converted. The decimal part is rounded down to 2 decimal points.

**Code Sample Example**

This example converts a floating point value into a currency value.

```
floattocur(3.263457) %Returns 3.26
```

# FloatToInt (function)

Converts a measure expression into an integer value.

**Syntax**

floattoint( expression ) ⇒ integer value

**Argument**

*expression*

Measure value to be converted. The decimal part is rounded down to a whole number.

**Code Sample Examples**

These examples illustrate **floattoint()**.

**Example 1**

```
floattoint(3.2) %Returns 3
```

**Example 2**

```
floattoint(11.6) %Returns 12
```

# FloatToStr (function)

Converts a measure expression into a string value.

**Syntax**

floattostr( expression[, precision]) ⇒ string value

**Argument**

*expression*

Measure value to be converted.

*precision*

The number of decimal places to include in the result string.

**Code Sample Examples**

These examples illustrate **floattostr()**.

**Example 1**

```
floattostr(3.14) %Returns 3.14
```

**Example 2**

```
floattostr(11) %Returns 11.0
```

**Example 3**

```
floattostr(34.6453455, 4) %Returns 34.6453
```

# IntToCur (function)

Converts an integer expression into a measure value.

**Syntax**

inttocur( expression ) ➡ currency value

**Argument**

*expression*

Integer value to be converted.

# IntToFloat (function)

Converts an integer expression into a measure value.

**Syntax**

inttofloat( expression ) ➡ measure value

**Argument**

*expression*

Integer value to be converted.

**Code Sample Example**

This example uses a loop to draw a series of blue rectangles.

**Example**

```
define(&x,integer,1)

%Define loop variable

setfillcolor([100,100,0,0])

%Fill blue

for(&x,1,1,10)

   %Set up loop

   %We must convert &x because rectfill

   %expects measure values

   rectfill(inttofloat(&x)/2,1,0.1,1)

   %Draw rectangle

endfor()
```

## IntToStr (function)

Converts an integer expression into a string value.

**Syntax**

inttostr( expression ) ⇒ string value

**Argument**

*expression*

Integer value to be converted.

**Code Sample Example**

This example prints the iterations of a loop.

**Example**

```
define(&x,integer,1)

%Define loop variable

for(&x,1,1,10)

   %Set up loop

   show('Iteration ' + inttostr(&x) )

   %Show text

   crlf()

   %Skip line
```

```
endfor()
```

## StrToCur (function)

Converts a string value into a currency value.

**Syntax**

strtocur( string ) ⇨ currency value

**Argument**

*string*

String value to convert.

## StrToFloat (function)

Converts a string into a measure value.

**Syntax**

strtofloat( string ) ⇨ measure value

**Argument**

*string*

String value to convert.

**Code Sample Example**

This example reads a column of 6 values then converts the values to floating point values and then adds the ir values to the &total variable. The total value is subsequently converted back to a string and the result is printed.

**Example**

```
define(&total,measure,0.0)

define(&x,integer,0)

for(&x,1,1,6)

  set(&total,&total+strtofloat(@(&x,10,18)))

endfor()

show(floattostr(&total))
```

## StrToInt (function)

Converts a string into an integer value.

**Syntax**

strtoint( string ) ⇒ integer value

**Argument**

*string*

String value to convert.

**Code Sample Example**

This example reads a column of 6 values then converts the values to integers and then adds the integer to the &total variable. The total value is subsequently converted back to a string and the result is printed.

**Example**

```
define(&total,integer,0)

define(&x,integer,0)

for(&x,1,1,6)

   set(&total,&total+strtoint(@(&x,10,18)))

endfor()

show(inttostr(&total))
```

## Loop Structures

## For... EndFor (procedure)

This command structure allows a series of nested commands to be repeated a specified number of times. Each **for** statement must have a corresponding **endfor** statement in order for PlanetPress Talk to know which commands to include in the loop. Note that you cannot increment the counter for a **for()** loop, inside the loop.

If you open a document created a with a version of PlanetPress earlier than 5.2.0, and if that document includes a **for()** loop with an increment greater than 1, you will notice that the loop is executed one less time than with your older version software both in PlanetPress itself and when you use the Optimized PostScript Stream option. The behavior on printers (using the Printer Centric option), on the other hand, remains unchanged.

**Syntax**

for( varname, startvalue, increment, stopvalue )

...

endfor()

**Arguments**

*varname*

Name of the variable to use for iterations. The variable must already exist.

*startvalue*

Integer value to initialize varname.

*increment*

Integer value used to increment varname after each iteration.

*stopvalue*

Integer value after which iterations stop.

**Code Sample Example**

This example simply prints 5 lines of text.

To cycle backwards through values, make **startvalue** larger than **stopvalue**, and specify a negative increment.

**Example**

```
define(&x,integer,1)
for(&x,1,1,5)
  show('Line n° '+inttostr(&x))
  crlf()
endfor()
```

## Repeat... Until (procedure)

Repeat a sequence of commands until a condition is true. It is similar to a for... endfor loop in that you can nest the loops. It is different from a **for... endfor** loop in that the loop always executes at least once, and that you tie the number of times the loop repeats to a Boolean condition. For example you might tie the number of times the loop repeats to the occurrence of a piece of data in the data stream.

**Syntax**

repeat

…

until( expression )

**Argument**

*expression*

Boolean value. If it evaluates to true, the loop ends. If it evaluates to false, the loop repeats.

**Code Sample Example**

This example prints the string 'Cheers' ten times, one underneath the other, with a dashed line above and below the ten instances.

**Example**

```
show('--------------')

crlf(0.16)
```

```
&count := 1

repeat

   &count := &count + 1

   show('Cheers')

   crlf(0.16)

until(&count = 10)

show('--------------')

crlf(0.16)
```

## Search ... EndSearch (procedure)

This command structure allows a series of nested commands to be repeated as long as a specific string of characters (the search string) is found within another string (the target string). After each iteration, the target string contains all the data between occurrences of the search string, excluding the search string. Once the loop is over, target contains the remaining characters of the original data.

### Syntax

search( target, search )

...

endsearch()

### Arguments

*target*

Name of a variable of type string in which to perform the search.

*search*

String value to search for.

### Code Sample Example

This example splits a string delimited with semi-colons.

In the **OnReadDataPage** event of the user-defined emulation, the **&str** system variable contains each new line of data being fed to the document. Also, do not assign anything to the variable while inside the **search-endsearch** loop; it will get overwritten.

### Example

```
define(&v1,string,'This;is;a;test') %Create the variable

search(&v1,';') %Start loop

   show(&v1) %Show token

   crlf() %Skip line
```

```
endsearch()

show(&v1) %Show remaining text
```

## Exit (procedure)

Exits from a for...endfor or repeat...until loop.

**Syntax**

exit()

**Argument**

None

**Code Sample Example**

This example shows how to break from a loop when a specific word, in this case 'Invoice,' is found somewhere on the data page.

```
define(&x,integer,1)
for(&x,1,1,&current.lpp)
  if(pos('Invoice',@(&x,1,80))>0)
    exit()
  endif
endfor()
```

## Condition Structures

## If ... ElseIf... EndIf (procedure)

This command structure can define up to two conditional blocks. Statements nested within the first block are executed only if the condition specified in the if statement returns true. If it returns false, the program branches to the first statement in the second block, if one has been defined using the elseif statement. Otherwise, control flows to the first statement following the endif command.

With PlanetPress Suite version 7.0, the elseif statement has been optimized to evaluate an argument, exactly like an if statement, and elseif statements from other programming languages. In order to complete the if...elseif...endif command structure, however, an else statement was also added, acting exactly like the previous, empty elseif. Note that the use of the else statement is strongly recommended, as it is optimized by comparison to the elseif statement. Also note that documents created with previous versions and using the if...elseif...endif structure will not be updated, for full backward compatibility.

**Syntaxes**

Prior to version 7.0

    if( condition )

      ...

elseif()

    ...

    endif()

Version 7.0 and above

    if( condition1 )

    ...

    elseif( condition2 )

    ...

    else()

    ...

    endif()

**Argument**

*condition, condition1, condition2*

Value of type Boolean to evaluate.

**Code Sample Example**

This example simply prints 5 lines of text, selecting fonts according to line numbers.

**Example prior to version 7.0**

```
define(&x,integer,1)

%Define loop variable and then set up loop

for(&x,1,1,5)

  if((&x mod 2)=0)

    %If &x is an even number

    setstyle(&bluefont)

    %use the blue font

  elseif() %otherwise

    setstyle(&Default)

    %use the default font

  endif()

  show('Some text to display')

  %Display the text

  crlf()

  %Skip to the next line
```

```
endfor()
```

**Example with version 7.0**

```
define(&x, integer, inttostr(@(1, 1, 10)))

if(&x = 1)

   setstyle(&bluefont)

elseif(&x = 2)

   setstyle(&redfont)

else()

   setstyle(&default)
endif()
```

**Related topics:**

-

## If (function)

Evaluates an expression and returns one of two specified values according to the results of the evaluation.

**Syntax**

if( expression, trueresult, falseresult ) ⇒ any type

**Arguments**

*expression*

Boolean value. If it evaluates to true, the function returns the contents of trueresult, otherwise it returns the contents of falseresult.

*trueresult, falseresult*

Values of any type, as long as both are of the same type. The return value of the function is therefore of the same type as trueresult and falseresult.

**Code Sample Example**

This example changes the text to display according to the current value of a variable.

**Example**

```
show(if(&pagenumber=1,'Customer Copy','Store Copy'))
```

# Procedures

## % (procedure)

Comments out the line. This is useful for embedding documentation in your code, making it easier to read and maintain. It is also useful during debugging, for commenting out lines of code you do not want to execute.

Comments must not appear in the first line of the script you enter in the PlanetPress Talk Editor.

### Syntax

% ... comment ....

### Argument

None

### Code Sample Example

This example illustrates comments in PlanetPress Talk code.

```
define(&x,integer,1)
% Creates a new local variable named x that is an
% integer and has a value of 1. The variable is used
% in a loop
for(&x,1,1,10)
      show(inttostr(&x))
endfor()
```

## @name (function/procedure)

Executes a global function or procedure created using the function() command.

> The @ character can be the beginning of a [data selection](#), a [page call](#) or a [procedure or function call](#).

### Syntax

@functionname( parameters ) ➡ integer, measure, currency, string, Boolean or no return value

### Arguments

*functionname*

String value specifying the name of the function or procedure.

*parameters*

Comma separated list of parameters required by the specified function or procedure.

## @page (procedure)

Executes the content of the specified document page, as if it were part of the calling page. The content of the page in this case excludes the paper handling properties of the page. The height and width of the page content thus becomes the height and width of the group that would result if you created a single group of all of the page elements. If you want to include the paper handling properties in the page execution, use the *$element* syntax instead.

> The @ character can be the beginning of a data selection, a page call or a procedure or function call.

> The *@page()* and *execpage()* commands are functionally equivalent. However, the *execpage()* command has the added ability of calling variable page names.

**Syntax**

@pagename()

**Arguments**

*pagename*

String value specifying the name of the page you want to execute.

**Example**

The following two lines of code are functionally equivalent and display the content of a page named *MyFirstPage*:

```
@MyFirstPage
execpage('MyFirstPage')
```

## $element (procedure)

Executes the specified document element (object, page, resource, etc.). Note that in the case of a page, this procedure executes both the content of the page and the paper handling properties of the page. The height and width of the *calling* page are therefore set to the same values as those set by the *called* page.

Recall that every element in a document has a unique name. You can therefore call any element in the document at any time. For example, the first page of a medical record document may contain a header listing all of the patient information, while other pages in the document scale that same header to fit in the lower left corner of each page. Any modification you subsequently make to the header is automatically reflected throughout the document.

**Syntax**

$*element*

**Arguments**

*element*

String value specifying the name of the element you want to execute.

**Examples**

Example 1 illustrates basic usage of the syntax.

```
$my_square
$page1
$box5
```

Example 2 scales the *$header* element to one quarter of its original size, and rotates it 90 degrees.

```
scale( 0.25, 0.25 )
setangle( 90 )
$header
```

Example 3 creates a thumbnail of each of the first two pages and positions the thumbnails side-by-side.

```
scale( 0.10, 0.10 )
$page1
translate( 8.5, 0 )
$page2
```

## Arc and ArcN (procedures)

Arc() draws an arc in a counter-clockwise direction, while ArcN() draws an arc in a clockwise direction. If there is a current point set, the command draws a straight line from the current point to the start point of the arc. Whether or not a current point is set when the command executes, after execution the current point is the end point of the arc.

You define the arc you want to draw by specifying the x and y coordinates of the center of the circle, the radius of the circle, and the start and end points for the arc. You specify each of the start and end points of the arc as an angle; the command uses the angle to position the point. For example, to position the start point, the command draws an invisible line at the specified start angle from the center of the circle, the length of the radius; it positions the start point at the end of that line.



Example using Arc()



Example using ArcN().

Example of an arc drawn from start point D to end point F: A. Center of circle (x,y) B. Radius C. Start angle D. Start point of arc E. End angle F.

**Syntax**

arc( x, y, arc_length, start_angle, end_angle )

**Arguments**

*x, y*

Measure values representing the x and y coordinates respectively of the center of the circle.

*arc_length*

Measure value representing the length of the arc.

*start_angle*

Measure value representing the start angle, in degrees, for the arc. Degrees are relative to a standard X, Y axis, with a value of 0 degrees lying flush with the X axis. The value of the start angle can be either positive or negative.

*end_angle*

Measure value representing the end angle, in degrees, for the arc. Degrees are relative to a standard X, Y axis, with a value of 0 degrees lying flush with the X axis. The value of the end angle can be either positive or negative.

**Code Sample Examples**

**Example 1**

Example 1 draws a 180 degree arc 3 inches in length. The center of the circle is at (0,0). The **rmoveto()** command that precedes the **arc()** command ensures the current point is set at the center of the circle and prevents a line from being drawn from the current point to the start point of the arc.

```
rmoveto(3,0)
arc(0,0,3,0,180)
stroke()
```

**Example 2**

Example 2 draws an arc 2.5 inches in length. The center for the arc is four inches from the left edge of the page and three inches from the top edge. The arc starts at 30 degrees and ends at 60 degrees. The **rmoveto()** command that precedes the **arc()** command moves the current point to the center of the circle, resulting in a line that is drawn from the center of the circle to the start point of the arc.

```
rmoveto(4,3)
arc(4,3,2.5,30,60)
stroke()
```

# BeginDocument/EndDocument (procedure)

Delimits the boundaries of a metadata document within a PlanetPress Design document.

**Syntax**

begindocument()

or

enddocument()

**Arguments**

None

**Code Sample Examples**

```
if (&is_FirstPage)
      begindocument()
EndIf()
if(&is_LastDocPage)
      enddocument()
EndIf()
```

## BeginGroup/EndGroup (procedure)

Delimits the boundaries of a metadata group within a PlanetPress Design document.

**Syntax**

begingroup()

or

endgroup()

**Arguments**

None

**Code Sample Examples**

```
if(&is_FirstInvPage)
      begingroup()
EndIf()
if(&is_LastInvPage)
      endgroup()
EndIf()
```

## BeginParagraph … EndParagraph (procedure)

Delimits a paragraph of formatted text.

The **beginparagraph** … **endparagraph** structure can contain only the following commands: **setstyle()**, **setstyleext()**, **show()**, and **%** (indicating a commented line). Any other command included within this structure will yield unpredictable results. A **margin()** command should precede the **beginparagraph…endparagraph** structure, and each variable you reference within the structure must have its own procedure., as demonstrated in the second of the examples below. Also note that the first command within a **beginparagraph…endparagraph** structure must be the **setstyle()** command.

No **crlf** procedures are permitted inside a paragraph structure, since **crlfs** are paragraph delimiters.

If you do not need to change fonts or use variables within a paragraph, it is strongly recommended you use the text object in PlanetPress, as PlanetPress optimizes text objects to improve performance.

When a long string containing no spaces between its characters appears in the context of a **show()** command and the **begin-paragraph** ... **endparagraph()** command, the line of text will not wrap. The same is true for a text object; no wrapping occurs when a long string has no spaces between its characters.

Note that when a Text object is converted to PlanetPress Talk, an extra argument is is added to the **beginparagraph** ... **endparagraph** syntax. Ignore this argument since it is strictly for internal use.

**Syntax**

beginparagraph( lmargin, rmargin, firstindent, align, leading,[opt, forcedwordwrap])

...

endparagraph()

**Arguments**

*lmargin*

Measure value specifying the left margin, in inches, of the text relative to the left border of the object.

*rmargin*

Measure value specifying the right margin, in inches, of the text relative to the left border of the object.

*firstindent*

Measure value specifying the indent, in inches, of the first line of the paragraph, relative to the lmargin parameter.

*align*

String value specifying the text alignment within the paragraph. Possible values are 'left', 'right', 'center' and 'leftright' (for both left and right justification of text).

*leading*

Measure value specifying the amount of vertical space, in inches, between each line of the paragraph.

*opt*

Optional boolean value used internally.  Default value is false.

*forcedwordwrap*

Optional boolean value that forces wordwrap at the defined paragraph size.  Default value is false.

**Code Sample Examples**

Example that displays a paragraph aligned to the right.

**Example 1**

```
beginparagraph(1,3,0,'right',0.16)
       setstyle(&Default)
       show('This long line of text should wrap around')
       show('This long line of text should wrap around')
endparagraph()
```

Example that shows how to include a variable within a formatted paragraph.

---

**Example 2**

```
define(&var,string,'very')
beginparagraph(1,3,0,'left',0.16)
      setstyle(&Default)
      show('This ')
      show(&var)
      show(' long line of text should wrap around')
endparagraph()
```

## BeginUTF8Paragraph ... EndUTF8Paragraph (procedure)

Delimits a paragraph of formatted UTF8 text.

This procedure is based on BeginParagraph ... EndParagraph (procedure), with the following differences:

- ShowUTF8 (procedure) must be used instead of show()
- SetStyleExt (procedure) must use a style that has been associated with the True Type Font "TT Host UTF8Arabic".
- Only UTF8 text can be displayed in a PlanetPress Talk object defined using the **beginUTF8paragraph** ... **endUTF8paragraph** structure. Non-UTF8 static text should therefore be converted to its UTF8 reference using the Ansi/UTF8 Converter (place the cursor within the string and press CTRL+N). Variable text should be converted using MapUTF8 (function).

### Syntax

beginUTF8paragraph( lmargin, rmargin, firstindent, align, leading, majormode, [forcedwordwrap] )

% Your Paragraph data here

endUTF8paragraph()

### Arguments

Same as BeginParagraph ... EndParagraph (procedure).

### Code Sample Example

This example displays a justified paragraph with the UTF8 text running from right to left.

### Example

```
setlinewidth(0.007)moveto(0,0)
rlineto(&width,0)
rlineto(0,&height)
rlineto(neg(&width),0)
rlineto(0,neg(&height))
closepath()
stroke()
moveto(&width/2,&height/18)
BeginUTF8Paragraph(0.0000,&width,0.0000,'leftright',0.1667,'rtl')
      SetStyleExt(&Style1,12.0000,0,[100],100)
      ShowUTF8('\u0623\u0633\u0627\u0633\u064B\u0627\u060C')
EndUTF8Paragraph()
```

## Breakpoint (procedure)

Stops the execution of a PlanetPress Talk object when the specified expression is True. This breakpoint is effective only when running in debug mode in the PlanetPress Talk editor.

**Syntax**

breakpoint( expression1 )

**Argument**

*expression1*

Boolean value.

**Code Sample Example**

This example sets up a loop, stops the loop on its 5th iteration, and then displays the value.

**Example**

```
define(&x,integer,1)for(&x,1,1,10)
        breakpoint(&x = 5)
        show(inttostr(&x))
endfor()
```

## CallPPD (procedure)

Calls a section of the PostScript Printer Description (PPD) file and executes it on the printer. Use this to set printer options such as the input tray or duplexing options. The tokens and subtokens you use as arguments with this function must be defined in the PPD file associated with your document or page in the document's or page's Basic Attributes. PlanetPress searches the PPD file for the token-subtoken combination and then prompts the printer to execute the corresponding code.

**Syntax**

callppd( token, subtoken )

**Arguments**

*token*

This is the string to search for in the PPD file (for example, Duplex).

*subtoken*

This is string to search for that is a setting for the token (for example, None).

**Example**

```
if((&Condition1) or (&Condition2))
        callppd("Tray", "2")
elseif()
        if((&Condition3) or (&Condition4))
                callppd("Tray", "3")
        elseif()
```

```
            callppd("Tray", "4")
        endif()
endif()
```

## CharPath (procedure)

Creates a path in the shape and size of a text string, in the currently active style.

**Syntax**

charpath(string)

**Argument**

None

**Example**

This example creates a path of the word "Text" and applies it as a clipping path on the coffee_time bitmap. This creates the word Text where the filling texture is the cofee_time image.

```
moveto(&width / 2, &height / 2)
charpath('TEST')
clippath()
moveto(0,0)
showbitmap('coffee_time',&document.picturecolorres,&width,&height)
```

**Result:**



## ClearPage (procedure)

Clears the current data page and loads the next one. This command should only be used before reading the next page of data from the input stream. It is strictly for use within a User-Define Emulation and generally follows DoForm (procedure).

**Syntax**

clearpage()

**Argument**

None

Example:

This example the default User-Defined Emulation that corresponds to a Line Printer data file.

```
search(&str,'\014')
        set(&current.line,&current.line + 1)
        store(&current.line,&str)
        doform()
        clearpage()
endsearch()
set(&current.line,&current.line + 1)
store(&current.line,&str)
if(ge(&current.line,&current.lpp))
        doform()
        clearpage()
endif()
```

## ClipPath (procedure)

The Clippath() command takes the last drawn path and selects it as the clipping region. Any subsequent drawing will only be visible inside that clipping region. The clipping will stop at the next call to GRestore(). All PlanetPress objects are natively wrapped in a GSave/GRestore(), so clipping between objects will not work.

**Syntax**

clippath()

**Argument**

None

**Example**

This example displays the coffee_time image partially, where only a square defined by the rlineto() is visible.

```
moveto((2.0000/(6.6529)) * &width, (0.2000/(4.9897)) * &height)
rlineto(0, (3.0000/(4.9897)) * &height)
rlineto((3.0000/(6.6529)) * &width, 0)
rlineto(0, neg((3.0000/(4.9897)) * &height))
closepath()
clippath()
moveto(0, 0)
showbitmap('coffee_time',&document.picturecolorres,&width,&height)
```

## ClosePath (procedure)

Closes any open path and completes the current shape, if need be, by drawing a line from the last point in the shape to the starting point. This ensures the shape is closed, thus enabling filling to take place if specified. Only closed shapes can be filled.

**Syntax**

closepath()

**Argument**

None

**Code Sample Example**

The first line of code sets the starting point of a triangle, the second and third lines of code draw the first and second lines (or sides) of the triangle, the fourth line of code closes the triangle shape by implicitly issuing a **moveto** command.

**Example**

```
moveto(1.0,1.0)lineto(1.5,2.0)
lineto(0.5,2.0)
closepath()
fill()
```

# CRLF (procedure)

Moves the current point by simulating a carriage return/line feed combination. The horizontal position is reset to the value of the current left margin, while the vertical position is offset by the leading value specified.

**Syntax**

crlf( [leading] )

**Argument**

*leading*

Measure value setting the vertical distance, in inches, between two lines. If you do not provide this argument, the value by default is 0.167 inches, which is roughly equivalent to a 6 LPI (lines per inch) setting.

**Code Sample Example**

This example demonstrates how to change the vertical spacing of lines within an object.

**Example**

```
margin(1,1)
show('These lines are displayed')
crlf(0.125)
show('at 8 LPI.')
crlf(0.5)
show('While these lines are displayed')
crlf(0.5)
show('at 2 LPI')
```

# CurveTo/RCurveTo (procedure)

Creates a Bezier curve. Bezier curves are shapes defined using 4 points: the starting and ending points are physical positioning points while the second and third points are control points on the curve. In PlanetPress Talk, the starting point is implicit and automatically set to the current cursor position. The first control point defines the direction the curve takes when moving

from the starting point, and before shifting to the second control point, which defines a second curve before reaching the ending point.

A complete explanation of Bezier curves is beyond the scope of this document. Feel free, however, to try out this command using the PlanetPress Talk Editor in PlanetPress in order to study its possibilities.

To draw a simple curve, use the same (x,y) coordinates for both control points.

**Syntax**

curveto( x1, y1, x2, y2, x3, y3 )

**Arguments**

*x1, y1, x2, y2, x3, y3*

Pairs of measure values representing the horizontal/vertical position, in inches, of each point defining the Bezier curve. When using rcurveto, these values are relative to the current point of origin. When using curveto, they are absolute values.

**Code Sample Example**

This example draws a triangle with a twisted bottom side. The triangle has a blue outline and is filled with yellow.

**Example**

```
setstrokecolor([100,100,0,0])
setfillcolor([0,0,100,0])
moveto(1,1)
lineto(1.5,2)
curveto(1,1,1,3,0.5,2)
closepath()
strokeandfill()
```

**Result:**



# Define (procedure)

This command creates a new local variable, or redefines an existing one. The define procedure must appear before the variable is actually used. PlanetPress Talk scripts require all variables to be defined either as a global variable in your document, or a local scope variable within your PlanetPress Talk script.

**Syntax**

define( name, type, value )

**Arguments**

*name*

The name of the variable, prefixed with the ampersand (&) character. The name must conform to the rules for names described in "Names" (page 219).

*type*

Constant value specifying the variable type, either integer, measure, string, Boolean, or an array type. The array types are color, arrayinteger, arraymeasure, arraycurrency, arrayBoolean, arraystring or directory. Once a type is assigned to a variable, it cannot be changed, unless the variable is redefined.

*value*

An initial value for the variable, of the same type as the variable (for example if the variable is of type integer, this value must be an integer). This value can also be a valid PlanetPress Talk expression. In the case of any array variable other than a color array or an array of type directory, the initial value can be either the number of elements you want the array to contain, or the set of values you want the array to contain. In the former case you assign values to the array elements in subsequent commands. In the latter case, you separate each value by a comma, and enclose the complete set of elements in square brackets. In the case of a string array, you must also quote each of the array elements. In an array of type directory, you specify the pathname to the folder and the file name filter you want to apply to each of the files in that folder; this in turn determines the number of elements in the array.

**Code Sample Example**

```
define( &customer_name, string, 'Smith' )
define( &invoice_number, integer, strtoint(@(1,1,10)) )
define( &left_margin, measure, 1.5 )
define( &is_bottom, boolean, (&current.line > 50) )
define( &my_array,arrayinteger,[12,3,76,109,4] )
define( &my_array,arrayinteger, 5 )
define( &prices,arraycurrency, 6)
define( &greetings,arraystring,['hello','bonjour','ola'] )
define( &mustard,color,[0,12,84,16] )
define( &image_paths, directory, 'c:\\images\\*.JPG' )
define( &cost, currency, 0.00 )
```

## DefineData (procedure)

Change the active data file to the one specified in the *path* parameter.

> Changing the data file in the middle of document processing can have unexpected consequences and should only be used if you understand the implications of doing so.

**Syntax**

definedata( path )

**Argument**

*path*

String value specifying the path of the data file.

# DefineImageIndex (procedure)

Defines a PlanetPress Search index term. PlanetPress Image uses this information to generate the .PDI file it creates for each PDF file it creates. You use SetImageIndex (procedure) to associate a data value with the index term.

**Syntax**

defineimageindex( name, [length] )

**Arguments**

*name*

String value specifying a name for the index term.

*length*

String value specifying a length for the index term.

**Code Sample Examples**

**Example 1**

This example defines the index term 'PONumber' and assigns it a length of 8 characters. The PDI file generated by PlanetPress Image contains the line: ~IndexName=PONumber,8.

```
defineimageindex( 'PONumber', '8')
```

**Example 2**

This example defines the same index term, this time without defining a length for the term. In this case the line in the PDI file becomes: ~IndexName=PONumber

```
defineimageindex( 'PONumber' )
```

# DoForm (procedure)

Runs the current document. This command is usually issued once a complete data page has been received and committed to buffer. This command should only be used before reading the next page of data from the input stream. It is strictly for use within a User-Define Emulation.

**Syntax**

doform()

**Argument**

None

**Code Sample Example**

This example is an extract from a user-defined emulation. The search() command looks for a formfeed and adds its line number. The line number is stored as a string and the document is run. The data page is cleared and the search is over.

**Example**

```
search(&str,'\014')
        set(&current.line,&current.line + 1)
        store(&current.line,&str)
        doform()
        clearpage()
endsearch()
```

# endpageset (procedure)

Ends any page set and subset when printing using a Windows driver. When creating a VDX PlanetPress Design document, this command signals the end of a booklet rather than a simple page subset.

**Syntax**

endpageset()

**Argument**

None

**Code Sample Examples**

In both modes, the command is called the same way.

```
% Print using a Windows driver mode
if(&is_CA)
  endpageset()
  selectprinter('MyCanadianPrinter')
else
  endpageset()
  selectprinter('MyOtherPrinter')
endif()

% Work with VDX
if(&is_LastBookletPage)
  endpageset()
endif()
```

# ExecPage (procedure)

Executes the content of the specified document page, as if it were part of the calling page. The content of the page in this case excludes the paper handling properties of the page. If you want to include the paper handling properties in the page execution, see **"$element (procedure)" (page 611)**.

The **execpage()** and **@page()** commands are equivalent.

**Syntax**

execpage( pagename )

**Argument**

*pagename*

String value specifying the name of the page to run. The page must exist, and its name is case insensitive.

**Code Sample Example**

In this example, the current page executes pages 2, 3 or 4 according to a data selection. This allows you to create pages of 'subroutines' that can be called from any other page.

**Example**

```
define(&x,integer,strtoint(@(7,4,8)))
if(&x < 1000)
  execpage('PAGE1')
elseif()
  if(&x > 1000)
      %This is equivalent to execpage('PAGE3')
    @PAGE3
  elseif()
    execpage('PAGE2')
  endif()
endif()
```

# Exit (procedure)

Exits from a for...endfor or repeat...until loop.

**Syntax**

exit()

**Argument**

None

**Code Sample Example**

This example shows how to break from a loop when a specific word, in this case 'Invoice,' is found somewhere on the data page.

```
define(&x,integer,1)
for(&x,1,1,&current.lpp)
  if(pos('Invoice',@(&x,1,80))>0)
    exit()
  endif
endfor()
```

## Fill (procedure)

Fills the current closed shape, using the colour specified with the **setfillcolor** command. Only the inside of the shape is filled, not its outline. To fill and outline the shape, use **strokeandfill** command instead. If you only want to outline the shape, use **stroke**. Only closed shapes can be filled.

**Syntax**

fill()

**Argument**

None

**Code Sample Example**

This example draws a yellow filled triangle.

**Example**

```
setfillcolor([0,0,100,0])
moveto(1,1)
lineto(1.5,2)
lineto(0,2)
closepath()
fill()
```

## For... EndFor (procedure)

This command structure allows a series of nested commands to be repeated a specified number of times. Each **for** statement must have a corresponding **endfor** statement in order for PlanetPress Talk to know which commands to include in the loop. Note that you cannot increment the counter for a **for()** loop, inside the loop.

If you open a document created a with a version of PlanetPress earlier than 5.2.0, and if that document includes a **for()** loop with an increment greater than 1, you will notice that the loop is executed one less time than with your older version software both in PlanetPress itself and when you use the Optimized PostScript Stream option. The behavior on printers (using the Printer Centric option), on the other hand, remains unchanged.

**Syntax**

for( varname, startvalue, increment, stopvalue )

...

endfor()

**Arguments**

*varname*

Name of the variable to use for iterations. The variable must already exist.

*startvalue*

Integer value to initialize varname.

*increment*

Integer value used to increment varname after each iteration.

*stopvalue*

Integer value after which iterations stop.

### Code Sample Example

This example simply prints 5 lines of text.

To cycle backwards through values, make **startvalue** larger than **stopvalue**, and specify a negative increment.

### Example

```
define(&x,integer,1)
for(&x,1,1,5)
  show('Line n° '+inttostr(&x))
  crlf()
endfor()
```

## Function @name (procedure)

Defines a new PlanetPress Talk function or procedure. If you define a function that you want to return a value, you assign that value to the predefined variable **&result** on the last line of the function definition. You call the function or procedure you define using the **@name()** command. See "@name (function/procedure)" (page 561).

You can reference a global function from within another global function only if the global function you are referencing already exists.

> You cannot reference a function from within itself, so recursive functions are not possible within PlanetPress Talk.

### Syntax

### Syntax for a function that returns a value:

function @name( arglist ): type

...

&result := return-_value

endfunction()

### Syntax for a function that does not return a value (procedure):

function @name( arglist )

...

endfunction()

### Arguments

*name*

Name to use for the function, prefixed with the @ character. The name must conform to the rules for names described in . It is good practice to be careful naming functions that you define in different documents, to ensure that if at some point you want to copy code between documents, the names of functions do not conflict.

*arglist*

List of arguments the function requires, where each element in the list has the syntax varname : type. For example, &date : string. An ampersand always precedes the first letter of varname. A comma separates each element in the list. Note that all arguments are pass by value (as opposed to pass by reference). Pass by value means that the function you define cannot modify any of the arguments it receives. Any modifications you make to an argument within the body of the function are made to a local copy, and do not affect the original.

*type*

Type of value (integer, measure, currency, string, Boolean) the function returns.

## Code Sample Example

This example displays a string.

```
function @showtext(&mystring: string)
  moveto(1,1)
  show(&mystring)
endfunction()
```

## Code Sample Example

This example illustrates a procedure that creates a 3D pie chart.

## Example

```
function @_3DPie( &W:measure, &H:measure, &HB:measure, &S:measure, &E:measure,
&L:string, &c:color )
  %==================================================
  % &W is the total width of the pie chart
  % &H is the total height of the pie chart
  % &HB is the height of the lower band
  % &S is the starting angle (0..360)
  % &E is the ending angle (0..360 and bigger than &S)
  % &L is the label text
  %==================================================
  %Local variables
  %==================================================
  define(&SB,measure,0)
  define(&EB,measure,0)
  define(&Scaling,measure,0)
  define(&R,measure,&W/2)
  %==================================================
  &Scaling := (&H-&HB)/&W
  gsave()
  if(&Scaling<0.01)
    &Scaling := 0.01
  endif()
```

```
if(&Scaling>1)
  &Scaling := 1
endif()
scale(1,&Scaling)
setstyle(&Style1)
&HB := &HB / &Scaling
translate(&R,&R)
if(&E > 180)
  if(&E > 360)
    &EB := 360
  elseif()
    &EB := &E
  endif()
  %=============================================
  if(&S < 180)
    &SB := 180
  elseif()
    &SB := &S
  endif()
  %=============================================
  set-
fill-
color([getcyan(&c),getmagenta(&c),getyellow(&c),if(getblack(&c)>50,100,getblack(&c)+20)])
  moveto(cos(&SB) * &R,neg(sin(&SB) * &R))
  rlineto(0,&HB)
  arc(0,&HB,&R,&SB,&EB)
  rlineto(0,neg(&HB))
  closepath()
  strokeandfill()
endif()
%=============================================
setfillcolor(&c)
pie(0,0,&R,&S,&E)
strokeandfill()
&SB := (&S+&E)/2
%=============================================
if(gt(&SB,180))
  margin(cos(&SB) * &R,neg(sin(&SB) * &R)+&HB+(0.2/&Scaling))
elseif()
  margin(cos(&SB) * &R,neg(sin(&SB) * &R)-(0.05/&Scaling))
endif()
%=============================================
scale(1,1/&Scaling)
if(and(gt(&SB,90),lt(&SB,270)))
  showright(&L+' ')
elseif()
  show(' '+&L)
endif()
grestore()
```

# GetNextDataPage(procedure)

Advance to the next data page of the sample data file. Note that this command may have unpredicted behavior if not used correctly in documents that use a user-defined emulation.

This command should only be executed when printermode is not in design mode. The result will be incorrect on screen, but there will be no
unexpected behaviors. Therefore, the only solution is for end-users to make sure the command doesn't get executed while in Design mode by bracketing it within a if(&printermode<>0) ... endif()
structure.

## Syntax

getnextdatapage()

# GRestore (procedure)

This command restores all system parameters previously saved with the **gsave** command. If no **gsave** command was issued previously, the results are unpredictable. For more information, refer to the **gsave** command.

## Syntax

grestore()

## Argument

None

## Code Sample Example

See .

# GSave (procedure)

This command saves all current system parameters, which can later be restored using the **grestore** command. By bracketing parts of your PlanetPress Talk programs with **gsave**/**grestore** commands, you can make sure the current system state is preserved and remains unaffected by whatever operations your program executes. For instance, each PlanetPress object, once converted to its PlanetPress Talk scripting language equivalent, begins with a **gsave** command and ends with a **grestore**, thus ensuring objects do not interfere with each other, or with the system.

The system parameters gsave saves include current line width, current stroke color, and current fill color.

## Syntax

gsave()

## Argument

None

## Code Sample Example

In this example, the current position is set first and saved.

**gsave** and **grestore** are required to be issued in pairs, meaning that you need the equal number of **gsave** and **grestore** commands to run in any given object. However, an uneven number of commands can be located in the code, if some of them are within IF statements, so that when the code is actually run, the pairing occurs correctly in all instances.

**Example**

```
moveto( 1, 1 )

%Position is now 1,1

gsave()

%Save current state

lineto( 2, 1 )

%Position is now 2,1

lineto( 2, 2 )

%Position is now 2,2

grestore()

%Restore previous state: position is now 1,1
```

# If ... ElseIf... EndIf (procedure)

This command structure can define up to two conditional blocks. Statements nested within the first block are executed only if the condition specified in the if statement returns true. If it returns false, the program branches to the first statement in the second block, if one has been defined using the elseif statement. Otherwise, control flows to the first statement following the endif command.

With PlanetPress Suite version 7.0, the elseif statement has been optimized to evaluate an argument, exactly like an if statement, and elseif statements from other programming languages. In order to complete the if...elseif...endif command structure, however, an else statement was also added, acting exactly like the previous, empty elseif. Note that the use of the else statement is strongly recommended, as it is optimized by comparison to the elseif statement. Also note that documents created with previous versions and using the if...elseif...endif structure will not be updated, for full backward compatibility.

**Syntaxes**

Prior to version 7.0

    if( condition )

    ...

    elseif()

    ...

    endif()

Version 7.0 and above

```
if( condition1 )

...

elseif( condition2 )

...

else()

...

endif()
```

**Argument**

*condition, condition1, condition2*

Value of type Boolean to evaluate.

**Code Sample Example**

This example simply prints 5 lines of text, selecting fonts according to line numbers.

**Example prior to version 7.0**

```
define(&x,integer,1)

%Define loop variable and then set up loop

for(&x,1,1,5)

  if((&x mod 2)=0)

    %If &x is an even number

    setstyle(&bluefont)

    %use the blue font

  elseif() %otherwise

    setstyle(&Default)

    %use the default font

  endif()

  show('Some text to display')

  %Display the text

  crlf()

  %Skip to the next line

endfor()
```

**Example with version 7.0**

```
define(&x, integer, inttostr(@(1, 1, 10)))

if(&x = 1)

  setstyle(&bluefont)

elseif(&x = 2)

  setstyle(&redfont)

else()

  setstyle(&default)
endif()
```

**Related topics:**

- "If (function)" (page 552)

## InStream... EndInStream (procedure)

Ceate an alias for a resource file (image, data file, attachment, etc.). You can then reference the resource file using the alias instead of the path. This eliminates the need to repeat a path in more than one place in your code, and thus makes your code easier to maintain. For example, if you change the resource, you need only modify the path in the **instream()** command, rather than everywhere you reference the original resource.

You cannot use the **instream()** command inside any function you define using **function @name()**.

**Syntax**

instream external resname

path

endinstream()

If you want to create a reference to a text document (for example a sample data file), replace "external" by "cleartext."

**Argument**

*resname*

String value specifying the alias to use to refer to this resource file.

*path*

String value specifying the path of the resource file.

**Code Sample Example**

This example creates the alias 'photo' that refers to the resource file *c:\images\employees\JAdler.bmp*.

**Example**

```
instream external photo

  c:\images\employees\JAdler.bmp

endinstream()

%You can then reference the resource using the alias

showbitmap('photo',72,1,1)
```

# LineTo/RLineTo (procedure)

Draws a line starting from the current point up to the specified coordinates.

**Syntax**

lineto( x, y )

**Arguments**

*x, y*

Measure values representing the horizontal/vertical position, in inches, of the ending point of the line. When using **rlineto**, these values are relative to the current point of origin. When using **lineto**, they are absolute values.

**Code Sample Example**

This example draws an empty triangle with a blue outline.

**Example**

```
setstrokecolor([100,100,0,0])   %Set pen colour to blue

moveto(1,1) %Set starting point

lineto(1.5,2) %Draw first line

lineto(0.5,2) %Draw second line

closepath() %Close shape

stroke() %Draw shape
```

## Margin (procedure)

Offsets text within a series of commands, relative to the original top left position of the object. Margin also sets the horizontal position to be used when **crlf** commands are encountered. All text-rendering functions that make use of margins use this setting.

### Syntax

margin( x, y )

### Arguments

*x, y*

Measure values representing the horizontal/vertical position, in inches, of the starting point for the text within an object.

### Code Sample Example

This example displays shadowed text by repeating the same text commands twice, using styles of different colours and slightly offsetting the margin.

### Example

```
margin(1,1) %Set margins to ( 1,1 )

setstyle(&blackfont) %Blackfont must already exist

show('This is shadowed text')

margin(.98,.98) %Offset margins slightly

setstyle(&bluefont) %Bluefont must already exist

show('This is shadowed text')
```

## MoveTo/RMoveTo (procedure)

Moves the current point to the specified coordinates.

### Syntax

moveto( x, y )

### Arguments

*x, y*

Measure values representing the new horizontal/vertical position, in inches, of the current point. In **rmoveto**, these values are relative to the current point of origin. In **moveto**, they are absolute values.

### Code Sample Example

This example displays a triangle within a rectangle.

### Example

```
rectstroke(.5,.5,2,2) %Draw rectangle
```

```
moveto(1,1) %Reset current point

lineto(1.5,2) %Draw first line

lineto(0.5,2) %Draw second line

closepath() %Close shape

stroke() %Draw shape
```

## Object $name()... EndObject (procedure)

Creates an object. You can reference the object you create using the $ operator. See "$element (procedure)" (page 611).

> ⚠ This command can only be used from within the PressTalk Before and PressTalk After of a PlanetPress Design document, and cannot exist within a page's properties or the PressTalk of any object, including

**Syntax**

```
object $name( top, left, width, height, condition, angle[, setsnappingpoint[, snap-
toprevious]] )
 %Object Contents
endobject()
```

**Argument**

*$name* (PressTalk Name)

Name to use for the object, preceded by the dollar sign character ($). The name must conform to the rules for names described in "Names" (page 219).

*top* (Measure)

Measure value specifying the distance, in inches, to offset the top edge of the picture object, from the top edge of the document page. If you set the snaptoprevious argument to a value other than 0, the value you set for the top argument becomes the vertical offset for the Snap to previous snapping point.

*left* (Measure)

Measure value specifying the distance, in inches, to offset the left edge of the picture object, from the left edge of the document page. If you set the snaptoprevious argument to a value other than 0, the value you set for the left argument becomes the horizontal offset for the Snap to previous snapping point.

*width, height* (Measure)

Measure values specifying the width and height respectively, in inches, of the object.

*condition* (Boolean)

Boolean value, or PlanetPress Talk expression that resolves to a Boolean value, specifying a condition on the object.

*angle* (Measure)

Measure value specifying the angle of rotation, in degrees. The pivot point for the rotation is the bottom left corner of the object.

*setsnappingpoint* (optional, Boolean)

Boolean value (True or False) specifying whether the object has its Set snapping point property set. If you set this argument to True, the last line of the code for the object must be a **moveto()** command that moves the current point to the position at which you want to set the object's snapping point.

*snaptoprevious* (optional, Integer)

Integer value specifying whether the object has its Snap to previous property set, and if so, the position of that snapping point. If you set the Snap to previous property, you can use the top and left arguments to specify, respectively, a vertical and horizontal offset for the snapping point.

- 0 = do not set the Snap to previous snapping point
- 1 = top left
- 2 = top middle
- 3= top right
- 4= middle left
- 5 = middle middle
- 6= middle right
- 7 = bottom left
- 8 = bottom middle
- 9 = bottom right

**Code Sample Example**

The following creates a rectangle object (2 inches wide by 3 inches high), rotates it 45 degrees, and sets its Snap to previous snapping point at the top left of the object with a vertical offset of 1 inch and a horizontal offset of 1.5 inches.

```
object $Box2(1.0,1.5,2.0,3.0,true,45.0,false,1)
  setstyle(&Style1)
  MoveTo(0,0)
  SetLineWidth(0.0070)
  SetStrokeColor([0,0,0,100])
  SetFillColor([0,0,0,0])
  LineTo(&width,0.0)
  LineTo(&width,&height)
  LineTo(0.0,&height)
  LineTo(0.0,0.0)
  ClosePath()
  Stroke()
endobject()
```

# OutputDebugString (procedure)

Outputs a string to the PlanetPress Talk Messages window in PlanetPress.

## Syntax

outputdebugstring(message)

**Argument**

*message*

String value to display in the PlanetPress Talk Messages window.

# PassThrough (procedure)

Sends a literal PostScript command to the printer.

**Syntax**

passthrough( string, [mode])

**Argument**

*string*

String value to be sent to the printer. This string is sent as is, with no further modification.

*mode*

Integer used as a bitfield to specify when to send the passthrough. Use a value of 1 to send the passthrough in PostScript-printer centric mode only, a value of 2 to send the passthrough in PostScript-Optimized PostScript Stream only, a value of 3 to send the passthrough in both PostScript-printer centric AND PostScript-Optimized PostScript Stream modes (default behavior if the parameter is not present), and a value of 4 to send the passthrough in Windows printing mode only.

**Code Sample Example**

Refer to your PostScript manual for valid commands.

# Pie (procedure)

Creates a pie slice shape, which can then be rendered using **stroke**, **fill**, or **strokeandfill**.

**Syntax**

pie( x, y, radius, startangle, endangle )

**Arguments**

*x, y*

Measure values representing the horizontal/vertical position, in inches, of a circle's center representing the whole pie from which to draw a slice.

*radius*

Measure value representing the length, in inches, of the segments of the pie, i.e. its radius.

*startangle, endangle*

Measure values representing the angles, in degrees, of both segments of the pie slice. Values increase counter clock wise, with 0 extending right from the center of the circle.

**Code Sample Example**

This example draws a whole pie chart with a shadow effect.

**Example**

```
setfillcolor([0,0,0,50]) %Set shadow colour to gray

pie(1.5,1.5,1,60,290) %Create shadow for first slice

translate(0.2,0.2) %Separate shadows

pie(1.5,1.5,1,290,420) %Create shadow for second slice

fill() %Display shadows

translate(-0.3, 0.3) %Offset chart from its shadow

setstrokecolor([0,0,100,0]) %Set pen colour to yellow

setfillcolor([0,100,100,0]) %Set fill colour to red

pie(1.5,1.5,1,60,290) %Create first slice

strokeandfill() %Draw first slice

translate(0.2,0.2) %Separate slices

setfillcolor([100,100,0,0]) %Set fill colour to blue

pie(1.5,1.5,1,290,420) %Create second slice

strokeandfill() %Draw second slice
```

## Put (procedure)

Assigns a value to an element of an array.

**Syntax**

put( &array, index, value )

**Arguments**

*&array*

Array variable containing the element whose value you want to change.

*index*

Integer value representing the position of the element in the array.

*value*

The value you want to assign to the element. The value must be of the same the type as the array. Thus if the array is of type string, the value must be of type string.

## Rectangle (procedure)

Creates and draws a rectangle shape. The rectangle can optionally be filled using the colour specified with the **setfillcolor** command. The border can optionally be drawn using the colour specified with the **setstrokecolor** command.

**Syntax**

rectangle( x, y, x1, y1, filled, stroked )

**Arguments**

*x, y*

Measure values representing the horizontal/vertical position, in inches, of the starting point.

*x1, y1*

Measure values representing the horizontal/vertical position, in inches, of the bottom right corner of the rectangle.

*filled*

Boolean value specifying whether the shape should be filled using the colour specified with the setfillcolor command.

*stroked*

Boolean value specifying whether the shape should be outlined using the colour specified with the setstrokecolor command.

**Code Sample Example**

This example draws a yellow rectangle with a black border.

**Example**

```
setstrokecolor([0,0,0,100])
setfillcolor([0,0,50,0])
rectangle(0,0,3,3,true,true)
```

## RectFill (procedure)

Creates and draws a filled rectangle shape. The colour used to fill the shape can be specified using the **setfillcolor** command.

**Syntax**

rectfill( x, y, width, height )

**Arguments**

*x, y*

Measure values representing the horizontal/vertical position, in inches, of the starting point.

*width, height*

Measure values representing the width/height, in inches, of the rectangle shape.

**Code Sample Example**

This example draws two identical rectangles.

To draw a rectangle that is both stroked and filled, use **rectstroke()** and **rectfill()** in succession, or **rectfillstroke()**.

**Example**

```
setfillcolor([0,100,100,0])

%Draw first rectangle using rectfill with red

rectfill(1,1,2,2)

%Offset starting position slightly to distinguish

%between each rectangle and fill second one with blue.

translate(.2,.2)

setfillcolor([100,100,0,0])

%Draw second rectangle using conventional commands

moveto(1,1) %Reset current point

lineto(3,1) %Draw first line

lineto(3,3) %Draw second line

lineto(1,3) %Draw third line

closepath() %Close shape

fill() %Draw shape
```

# RectFillStroke (procedure)

Creates a rectangle that has both an outline and a fill color. You set the color for the outline using setstrokecolor and the fill color using setfillcolor.

**Syntax**

rectfillstroke( x, y, width, height )

**Arguments**

*x,y*

Measure values representing the x and y coordinates respectively of the top left corner of the rectangle.

*width, height*

Measure values specifying the width and height respectively of the rectangle.

## RectStroke (procedure)

Creates and draws an empty rectangle shape. The colour of the pen used to draw the rectangle can be set using **set-strokecolor**.

### Syntax

rectstroke( x, y, width, height )

### Arguments

*x, y*

Measure values representing the horizontal/vertical position, in inches, of the starting point.

*width, height*

Measure values representing the width/height, in inches, of the rectangle shape.

### Code Sample Example

This example draws two identical rectangles.

To draw a rectangle that is both stroked and filled, use **rectstroke** and **rectfill** in succession.

### Example

```
rectstroke(1,1,2,2) %Draw first rectangle

%Offset starting position slightly to distinguish

%between each rectangle; the second rectangle is blue.

translate(.2,.2)

setstrokecolor([100,100,0,0])

%Draw second rectangle using conventional commands

moveto(1,1) %Reset current point

lineto(3,1) %Draw first line

lineto(3,3) %Draw second line

lineto(1,3) %Draw third line

closepath() %Close shape

stroke() %Draw shape
```

## Repeat... Until (procedure)

Repeat a sequence of commands until a condition is true. It is similar to a for... endfor loop in that you can nest the loops. It is different from a **for... endfor** loop in that the loop always executes at least once, and that you tie the number of times the

loop repeats to a Boolean condition. For example you might tie the number of times the loop repeats to the occurrence of a piece of data in the data stream.

**Syntax**

repeat

...

until( expression )

**Argument**

*expression*

Boolean value. If it evaluates to true, the loop ends. If it evaluates to false, the loop repeats.

**Code Sample Example**

This example prints the string 'Cheers' ten times, one underneath the other, with a dashed line above and below the ten instances.

**Example**

```
show('--------------')

crlf(0.16)

&count := 1

repeat

  &count := &count + 1

  show('Cheers')

  crlf(0.16)

until(&count = 10)

show('--------------')

crlf(0.16)
```

# RunPS (procedure)

This command is typically used to call resources, usually images, downloaded either with the Image Downloader in PlanetPress or Send Images to Printer action tasks in the PlanetPress Suite Workflow Tools. *Note that this command does not work with any other PostScript files.*

**Syntax**

runps( filename )

**Argument**

*filename*

String value. Name of the resource file.

## Scale (procedure)

Scales the result of all commands that follow the scale() command.

**Syntax**

scale(width, height)

**Argument**

*width*

Measure value specifying the factor by which to scale the width.

*height*

Measure value specifying the factor by which to scale the height.

**Code Sample Example**

```
scale( 2,0.5 )
```

## Search ... EndSearch (procedure)

This command structure allows a series of nested commands to be repeated as long as a specific string of characters (the search string) is found within another string (the target string). After each iteration, the target string contains all the data between occurrences of the search string, excluding the search string. Once the loop is over, target contains the remaining characters of the original data.

**Syntax**

search( target, search )

...

endsearch()

**Arguments**

*target*

Name of a variable of type string in which to perform the search.

*search*

String value to search for.

**Code Sample Example**

This example splits a string delimited with semi-colons.

In the **OnReadDataPage** event of the user-defined emulation, the **&str** system variable contains each new line of data being fed to the document. Also, do not assign anything to the variable while inside the **search-endsearch** loop; it will get overwritten.

**Example**

```
define(&v1,string,'This;is;a;test') %Create the variable

search(&v1,';') %Start loop

  show(&v1) %Show token

  crlf() %Skip line

endsearch()

show(&v1) %Show remaining text
```

## SelectMedia (procedure)

On screen, displays the physical page. On a PostScript printer, calls the specified paper.

Variables used as arguments in this command are not parsed when printing using the Optimized PostScript Stream mode.

**Syntax**

selectmedia( width, height, [media], [color], [weight], orientation, mode )

**Arguments**

*width, height*

Measure values specifying the width and height of the physical page in inches.

*media*

A string value specifying the type of paper to use.

*color*

String value specifying the color of the paper.

*weight*

Measure value specifying the weight of the paper (in grams per square meter). You can convert weight in pounds to weight in grams by multiplying by the weight in pounds by 3.76.

*orientation*

Integer representing the orientation of the page (0=Portrait, 1=Landscape, 2=Reverse portrait, and 3=Reverse landscape).

*mode*

Reserved for future use. Specify 0 for now.

**Code Sample Example**

```
define(&mcolor,string,'')

&mcolor := 'White'
```

```
selectmedia(8.5,11,'Danny',&mcolor,20,0,0)
```

The code above will generate the following PostScript code (notice how the variable was not replaced by its value).

```
612 792 0 ^SM (Danny) &mcolor 20 ^SPM
```

---

## SelectPrinter (procedure)

This command is used to select the printer on which the next page set will be sent. It can be used to select the same physical printer with different options, or a different printer altogether. It is meant to be used with the **EndPageSet()** command. This command can currently be used only for Windows Printing.

### Syntax

selectprinter(printername)

### Arguments

*printername*

String identifying a printer queue.

### Example

Refer to the code example given in the section describing the **EndPageSet** command on page 1.

## Set (procedure)

This command assigns a new value to a variable. Only user-defined variables can be set. System variables are read-only. Note that you can also use the assignment operator to assign a value to a variable. See <u>"= (operator)" (page 391)</u>.

### Syntax

set( name, value )

### Arguments

*name*

Variable name, prefixed with the ampersand (&) character. The variable name must already exist.

*value*

New value to store in the variable. This value can be any valid PlanetPress Talk expression, as long as its type is the same as the variable's original type.

### Code Sample Example

---

```
set( &invoice_number, (&invoice_number + 1) )

set( &is_bottom, (not(&is_top) and not(&is_middle)) )

&tax_rates[2] :=32
```

---

```
put( &tax_rates[2], @(1,3,5) )

set( &months, ['JAN','FEB','MAR','APR','MAY'] )

set( &mustard_color,[0,20,90,16] )

set( &prices, [12.5632,18.9932,23.6651,27.0300] )

put( &imagefiles[0], 'c\\images\\sushi.png' )
```

## SetAngle (procedure)

This command rotates all subsequent commands by the angle specified. Most commands and objects that are displayed/printed on the document are affected by the **setangle** command.

### Syntax

setangle( angle )

### Argument

*angle*

Measure value specifying the angle of rotation in degrees for all subsequent commands. Use a positive value for counter-clockwise motion and a negative value for clockwise motion. This value is relative to the current angle, not absolute.

### Code Sample Example

This example illustrates **setangle()**.

**setangle()** sets a rotation for all subsequent commands. Consequently, **setangle** commands are cumulative. Therefore, to restore the previous angle of rotation, you must issue a **setangle** command that negates whatever rotation applied.

### Example

```
setangle(45.00) %Rotate subsequent commands by 45 degrees

show('Some text') %Display at a 45 degree angle

setangle(45.00) %Rotate subsequent commands by 45 degrees

show('Some text') %Display at a 90 degree angle

setangle(-90) %Cancel all rotations
```

## SetBodyText (procedure)

Defines the text that appears as the body of an email message sent by PlanetPress Image.

### Syntax

setbodytext( bodytext )

## Arguments

*bodytext*

String value specifying the text to use as the body of the email message.

# SetDash(procedure)

Set the properties of a dashed line. Any drawing done after **setdash()** draws dashed instead of solid lines. You can revert to solid lines by calling **setdash()** with an empty array as a parameter.

## Syntax

setdash( [dashlength, spacelength] )

## Argument

*dash*

Measure value specifying the length, in inches, of each of the dashes in the dashed line.

*space*

Measure value specifying the length, in inches, of the space between each of the dashes in the dashed line.

## Code Sample Examples

These examples illustrate **setdash()**.

## Example 1

```
setdash( [0.5,0.3] )
```

## Example 2

```
setdash( [] )
```

# SetDataPage(procedure)

Specify the data page of the sample data file.

## Syntax

setdatapage( pagenum )

## Argument

*pagenum*

Integer value representing the page number of the data page.

## SetEmailAddress (procedure)

Defines an email address for PlanetPress Image.

**Syntax**

setemailaddress( address )

**Arguments**

*address*

String value specifying the email address.

## SetEmailSubject (procedure)

Defines the subject line of an email message sent by PlanetPress Image.

**Syntax**

setemailsubject( subjectline )

**Arguments**

*subjectline*

String value specifying the subject line.

## SetEmulation(procedure)

Sets the emulation to use.

**Syntax**

setemulation( emulation )

**Argument**

*emulation*

Integer value indicating the type of emulation.

| Value: | Emulation: |
|--------|------------|
| 0 | Line printer |
| 1 | ASCII |
| 2 | Comma Separated Value (CSV) |
| 3 | Channel skip |
| 4 | Database |
| 5 | XML |
| 6 | PDF |

## SetUserCRLF (procedure)

Lets you define a custom global function that will be executed whenever a line return appears in a Text object where this procedure is added. The SetUserCRLF() procedure is normally added in the PressTalk Before parameter of your Text object.

Syntax

setusercrlf( procedurename:procedure )

Arguments

*Procedurename*

The name of an existing global function in your PlanetPress Design document, preceded as usual with @.

Example

This example will ensure that multi-line text will never overlap an image on the page (where the image's position and size has been saved as global variables).

In the PressTalk Before of an object, the following line is added:

```
setusercrlf(@MyCRLF)
```

This global function is created:

```
function @MyCRLF(&Leading:measure)
 if(((&physical.y+(&leading*2))>&LogoPos[0]) and ((&physical.y+&leading)<(&LogoPos[0])))
  translate(0,(&LogoPos[1]-&LogoPos[0])+&leading*2)
 endif()
 moveto(0,&current.Y+&Leading)
endfunction()
```

## SetFaxInformation (procedure)

Defines the fax description that appears in both the PlanetPress Fax dialog box and the PlanetPress Fax log file.

### Syntax

setfaxinformation( info )

### Arguments

*info*

String value specifying a description of the fax.

### Code Sample Example

```
setfaxinformation( 'purchase confirmation' )
```

## SetFaxNumber (procedure)

Defines a fax number for PlanetPress Fax.

### Syntax

setfaxnumber( faxnumber )

**Arguments**

*faxnumber*

String value specifying the fax number.

**Code Sample Example**

```
setfaxnumber( '(514)276-7633' )
```

## SetFillColor (procedure)

Sets the colour used to paint the insides of any closed shapes when issuing a **fill** or a **strokeandfill** command.

**Syntax**

setfillcolor( colour )

**Argument**

*colour*

Colour array that can be expressed expressed using one, three or four values ranging from 0 to 255. For Grayscale, a single value is used in the form [K]. For the RGB model, three values are used in the form [R, G, B]. For the CMYK model, four values are used in the form [C, M, Y, K].

**Code Sample Example**

This example draws three filled rectangles.

**Example**

```
setfillcolor([255]) %Sets fill colour to black using the Grayscale model

rectfill(1, 1, 1, 1) %Draws a black square

setfillcolor([125,0,0]) %Sets fill colour to medium red using the RGB model

rectfill(1, 1, 1, 1) %Draws a medium red square

setfillcolor([0,0,25,0]) %Sets fill colour to light yellow using the CMYK model

rectfill(0, 0, 1, 1) %Draws a light yellow square
```

## SetImageIndex (procedure)

Associates a data value with a PlanetPress Search index term defined using the **DefineImageIndex()** command. PlanetPress Image uses this information when it generates the .PDI file it creates for each PDF file it creates.

> ⟫⟩ This function has no effect when the output of the document is not a PDF file generated through PlanetPress Image.

**Syntax**

setimageindex( name, data )

**Arguments**

*name*

String value specifying the name of the index term with which you want to associate a data value.

*data*

String value specifying the data value you want to associate with the index term.

**Code Sample Examples**

These examples illustrate **setimageindex()**.

**Example 1**

```
setimageindex( 'PONumber', '2005')
```

**Example 2**

```
setimageindex( 'PONumber', @(2,24,32) )
```

# SetLineWidth (procedure)

Sets the width of the pen used for drawing lines, boxes and other objects. When setting the line width, remember that the exact location of the pen is on the middle of the line, with the stroke spilling over equally on both sides.

**Syntax**

setlinewidth( width )

**Argument**

*width*SetLineWidth (procedure)

Measure value setting the width, in inches, of the drawing pen.

**Code Sample Examples**

**Example 1**

This example sets the pen width to 1/4 inch then draws a 2"x2" rectangle using that pen.

```
setlinewidth(0.25)
rectstroke(0,0,2,2)
```

## SetLPP(procedure)

Sets the number of lines per data page. This function is only useful when working with User-Defined Emulation, if you have a method of determining the number of LPP from within your data.

**Syntax**

setlpp( lines )

**Argument**

*lines*

Integer value specifying the number of lines per page.

## SetPDFBookmark (procedure)

Creates a PDF bookmark at the current page, for the PDF files PlanetPress Image generates.

>>> This function has no effect when the output of the document is not a PDF file generated through PlanetPress Image.

**Syntax**

setpdfbookmark( name )

**Arguments**

*name*

String value specifying the name of the bookmark that appears in the PDF.

**Code Sample Examples**

These examples illustrate **setpdfbookmark()**.

**Example 1**

```
setpdfbookmark('Table of Contents')
```

**Example 2**

```
setpdfbookmark( @(10,23,45) )
```

**Example 3**

```
setpdfbookmark( @(10,23,45) + '|' + @(11,23,45) )
```

## SetStrokeColor (procedure)

Sets the colour of the pen used for drawing lines, boxes and other objects.

**Syntax**

setstrokecolor( colour )

**Argument**

*colour*

Colour array that can be expressed expressed using one, three or four values ranging from 0 to 255. For Grayscale, a single value is used in the form [K]. For the RGB model, three values are used in the form [R, G, B]. For the CMYK model, four values are used in the form [C, M, Y, K].

**Code Sample Example**

This example draws two empty rectangles of different colors.

**Example**

```
setstrokecolor([0,0,100]) %Sets stroke color to medium blue using the RGB model
rectstroke(0,0,1,1) %Draws a medium blue rectangle
setstrokecolor([0,0,255,0]) %Sets stroke color to bright yellow using the CMYK model
rectstroke(1,1,1,1) %Draws a bright yellow rectangle
```

## SetStyle (procedure)

Sets the style to be used for all subsequent commands. The specified style must already exist in the document.

**SetStyle()** supercedes the **SetFont()** command. Although PlanetPress Talk still recognizes **SetFont()**, this behavior is unlikely to continue in future versions, and it is therefore highly recommended that you use the **SetStyle()** command in your scripts.

**Syntax**

setstyle( stylename )

**Argument**

*stylename*

Name of the style to use for all subsequent commands. The name of the style must be preceded by an ampersand. You can reference the bold, italic, or underlined versions of a font by appending the appropriate letter or letters to the name of the style. Append *.b* for the bold version, *.i* for the italic version, *.bi* for the bold italic version, and *.u* for the underlined version.

**Code Sample Example**

This example displays a line of text using two different existing styles called Blackfont and Bluefont.

**Example**

```
margin(1,1)
setstyle(&blackfont)
show('The word ')
setstyle(&bluefont.i)
show('blue')
setstyle(&blackfont)
show(' is now in blue italic')
```

## SetStyleExt (procedure)

Select an existing style and set the size, color, style property, or font ratio for its font. This eliminates the need to create several styles that use the same font. **setstyleext**() does not support bold, italic, or underline styles with double-byte character sets.

### Syntax

setstyleext(stylename, fontsize, styleproperty, fontcolor, fontratio)

### Arguments

*stylename*

The name of the style. The name of the style must be preceded by an ampersand. You can set the bold, italic, or bold italic property of the style by appending .b, .i, or .bi respectively to the name of the style.

The ability to append .b, .i, or .bi to style names that have double-byte character sets is not supported.

*fontsize*

Measure value representing the point size for the font. A value of -1 use the default font size of the style specified by stylename.

*styleproperty*

Integer value representing the style property for the font:

- 0: Normal
- 1: Underline
- 2: Outline

*fontcolor*

Colour array that can be expressed using one, three or four values ranging from 0 to 255. For Grayscale, a single value between 0 and 100 is used in the form [K], representing percentage of gray (100 is black, 0 is white). For the RGB model, three values are used in the form [R, G, B]. For the CMYK model, four values are used in the form [C, M, Y, K].

*fontratio*

Integer value representing the percentage by which to shrink or stretch the font spacing. This value adjusts both the width of each glyph and the spacing between glyphs. This is in contrast to kerning, which modifies the spacing between characters without modifying the width of characters.

Example

```
setstyleext(&Style2.bi, 25, [100], 100)
```

# Show / ShowCenter / ShowRight (procedure)

These three commands are used to display simple text on the document. They are identical in all aspects, except in the way they behave relative to the current margin settings: **show** uses the margin as a starting point to extend the string to the right; **showright** uses the margin as the ending point of the string; and **showcenter** extends the string equally on both sides of the margin. These commands are equivalent to *print*, *echo* or *write* commands in other scripting languages.

## Syntax

show( text )

showcenter( text )

showright( text )

## Argument

*text*

String value to be displayed.

## Code Sample Example

This example illustrates the difference between all three variations of using the SHOW command.

## Example

```
margin(3,1)
showright('This line is to the left of the margin')
crlf()
showcenter('This line is centered on the margin')
crlf()
show('This line is to the right of the margin')
```

# ShowBarCode (procedure)

**Note: This PressTalk command is deprecated.  Use instead the barcode-specific commands.**

Displays a specified string as a barcode.

## Syntax

showbarcode( string )

## Arguments

*string*

Value of type string to display as a barcode.

showbarcode needs 3 additional parameters before it can display the barcode. These parameters must be set before calling **showbarcode**, using the following variables:

*BarWidth*

Measure value setting the width of each bar, in inches.

*BarHeight*

Measure value setting the height of each bar, in inches.

*BarType*

Integer value specifying the type of barcode to print. Current possible values include:

| BarType: | Code: |
|----------|-------|
| 0 | Code 39 |
| 1 | Code 128 |
| 2 | UPC_A |
| 3 | UPC_E |
| 4 | UPC/EAN 8 |
| 5 | UPC/EAN 13 |
| 6 | PostNet |
| 7 | PDF-417 |

**Code Sample Example**

This example prints a Code 128 barcode.

The **BarWidth**, **BarHeight** and **BarType** variables must be created exactly as shown here, with the same case changes in the variable names.

**Example**

```
define(&BarWidth,measure,0.012)

define(&BarHeight,measure,1.000)

define(&BarType,integer,1)

%Set bar color to black and display barcode

setfillcolor([100,100,100,100])

showbarcode('Bar 128')
```

## ShowBarcode2of5(procedure)

Displays a specified string as a Standard 2 of 5 barcode.

**Syntax**

showbarcode2of5( str, barwidth, readable, bearers)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

*readable*

Boolean that enables display of the human readable portion of the barcode

*bearers*

Boolean that enables bearer bars surrounding the barcode

**Code Sample Example**

```
showbarcode2of5('123456789',12,true,true)
```

# ShowBarcodeAustPost (procedure)

Displays a specified string as an Australia Post barcode.

**Syntax**

showbarcodeaustpost( str)

**Arguments**

*str*

Value of type string to display as a barcode.

**Code Sample Example**

```
showbarcodeaustpost('56439111ABA 9')
```

# ShowBarcodeAztec (procedure)

Displays a specified string as an Aztec barcode.

**Syntax**

showbarcodeaztec( str, mode, errorcorrection, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*mode*

Mode to use for the barcode.

Value Mode

0      Normal

1    Compact

2    Full Range

*errorcorrection*

Error correction level to use in the barcode.

Mode Error Correction Range

0    0-99

1    0-4

2    0-32

*barwidth*

Measure value of the width a a single barcode bar, in mils (0.001 inch).

**Code Sample Example**

```
showbarcodeaztec('A 12345 B 434343',5,35,24)
```

# ShowBarcodeCodabar (procedure)

Displays a specified string as a Codabar barcode.

**Syntax**

showbarcodecodabar( str, start, stop, barwidth, readable)

**Arguments**

*str*

Value of type string to display as a barcode.

*Start*

The Start character of the barcode

A

B

C

D

*Stop*

The Stop character of the barcode

A

B

C

D

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

*readable*

Boolean that enables display of the human readable portion of the barcode

**Code Sample Example**

```
showbarcodecodabar('12345','A','A',12,true)
```

## ShowBarcodeCodablockF (procedure)

Displays a specified string as a Codablock F barcode.

**Syntax**

showbarcodecodablockf( str, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

**Code Sample Example**

```
showbarcodecodablockf('12345',12)
```

## ShowBarcodeCode11 (procedure)

Displays a specified string as a Code 11 barcode.

**Syntax**

showbarcodecode11( str, barwidth, checksum, readable, readchecksum)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

*Checksum*

Boolean allows the automatic calculation of the barcode checksum

*readable*

Boolean that enables display of the human readable portion of the barcode

*readchecksum*

Boolean that enables display of the checksum in the human readable portion of the barcode

## Code Sample Example

```
showbarcodecode11('123456789',12,true,true,false)
```


# ShowBarcodeCode128 (procedure)

Displays a specified string as a Code 128 barcode.

## Syntax

showbarcodecode128( str, subset, barwidth, readable)

## Arguments

*str*

Value of type string to display as a barcode.

*subset*

Integer specifying the subset of the barcode to use.

| Value | Subset |
|-------|--------|
| 0 | A |
| 1 | B |
| 2 | C |

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

*readable*

Boolean that enables display of the human readable portion of the barcode

## Code Sample Example

```
showbarcodecode128('123456789',0,12,true)
```

## ShowBarcodeCode16k (procedure)

Displays a specified string as a Code 16k barcode.

**Syntax**

showbarcodecode16k( str, mode, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*Mode*

Mode (0-7) of the Code 16k barcode

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

**Code Sample Example**

```
showbarcodecode16k('12345',4,12)
```

## ShowBarcodeCode39 (procedure)

Displays a specified string as a Code 3 of 9 barcode.

**Syntax**

showbarcodecode39( str, barwidth, checksum, readable, readchecksum)

**Arguments**

*str*

Value of type string to display as a barcode.

*BarWidth*

Measure value setting the width of each bar, in mils (0.001 inch).

*Checksum*

Boolean value that indicates if the checksum should automatically be calculated.

*Readable*

Boolean value that indicates if the human readable should be shown

*ReadChecksum*

Boolean value that indicates if the checksum value should be included in the human readable

**Code Sample Example**

```
showbarcodecode39('123456',12,true,true,false)
```

## ShowBarcodeCode49 (procedure)

Displays a specified string as a Code 49 barcode.

**Syntax**

showbarcodecode49( str, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

**Code Sample Example**

```
showbarcodecode49('12345', 12)
```

## ShowBarcodeCode93 (procedure)

Displays a specified string as a Code 93 barcode.

**Syntax**

showbarcodecode93( str, barwidth, readable)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

*readable*

Boolean that enables display of the human readable portion of the barcode

**Code Sample Example**

```
showbarcodecode93('123456789',12,true)
```

## ShowBarcodeDatamatrix (procedure)

Displays a specified string as a Datamatrix barcode.

### Syntax

showbarcodedatamatrix( str, mode, errorcorrection, barwidth)

### Arguments

*str*

Value of type string to display as a barcode.

*Mode*

Integer value of the Datamatrix Mode to use.

| Value | Mode |
|-------|------|
| 0 | Square |
| 1 | Rectangular |

*errorCorrection*

Error correction level to use with the barcode.

| Mode | Allowed Range |
|------|---------------|
| Square | 1-24 |
| Rectangular | 1-6 |

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

### Code Sample Example

```
showbarcodedatamatrix('12345', 0, 5, 12)
```

## ShowBarcodeEAN8 (procedure)

Displays a specified string as a EAN-8 barcode.

### Syntax

showbarcodeean8( str, subset,[readable, barwidth] )

## Arguments

*str*

Value of type string to display as a barcode.

*Subset*

The subset of EAN-8 to use

 0 EAN-8
 1 EAN-8 ext.2
 2 EAN-8 ext.5

*readable*

Optional boolean that enables display of the human readable portion of the barcode. Default = True.

*barwidth*

Optional width of one barcode bar, in mils (0.001 inch). Default = 12mils.

### Code Sample Example

```
showbarcodeean8('0133558',0)
```

## ShowBarcodeEAN13 (procedure)

Displays a specified string as a EAN-13 barcode.

### Syntax

showbarcodeean13( str, subset,[readable, barwidth] )

### Arguments

*str*

Value of type string to display as a barcode.

*Subset*

The subset of EAN-13 to use

 0 EAN-13
 1 EAN-13 ext.2
 2 EAN-13 ext.5

*readable*

Optional boolean that enables display of the human readable portion of the barcode. Default = True.

*barwidth*

Optional width of one barcode bar, in mils (0.001 inch). Default = 12mils.

**Code Sample Example**

```
showbarcodeean13('977147396801',0)
```

## ShowBarcodeFIM (procedure)

Displays a specified string as a FIM barcode.

**Syntax**

showbarcodefim( str)

**Arguments**

*str*

Value of type string to display as a barcode.

**Code Sample Example**

```
showbarcodefim('A')
```

## ShowBarcodeI2of5 (procedure)

Displays a specified string as an Interleaved 2 of 5 barcode.

**Syntax**

showbarcodei2of5( str, barwidth, checksum, readable, readchecksum, bearers)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

*Checksum*

Boolean to add checksum automatically to barcode

*readable*

Boolean that enables display of the human readable portion of the barcode

*readchecksum*

Boolean that enables display of the checksum in the human readable portion of the barcode

*bearers*

Boolean that enables bearer bars surrounding the barcode

**Code Sample Example**

```
showbarcodei2of5('123456789',12,true,true,false,true)
```

# ShowBarcodeISBN (procedure)

Displays a specified string as an ISBN barcode.

**Syntax**

showbarcodeisbn( str, subset,[readable, barwidth] )

**Arguments**

*str*

Value of type string to display as a barcode.

*Subset*

The subset of ISBN to use

0 ISBN
1 ISBN ext.2
2 ISBN ext.5

*readable*

Optional boolean that enables display of the human readable portion of the barcode. Default = True.

*barwidth*

Optional width of one barcode bar, in mils (0.001 inch). Default = 12mils.

**Code Sample Example**

```
showbarcodeisbn('978-1-86074-271-2-54495',2)
```

# ShowBarcodeJapanpost (procedure)

Displays a specified string as a Japan Post barcode.

**Syntax**

showbarcodejapanpost( str, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*BarWidth*

Measure value setting the width of each bar, in mils (0.001 inch).

**Code Sample Example**

```
showbarcodejapanpost('6540123789-A-K-Z',12)
```

# ShowBarcodeMaxicode (procedure)

Displays a specified string as a Maxicode barcode.

**Syntax**

showbarcodemaxicode( str, mode)

**Arguments**

*str*

Value of type string to display as a barcode.

*Mode*

Integer value (2-6) of the Maxicode Mode to use.

**Code Sample Example**

```
show-
bar-
codemaxicode('^059^042^041^059^040<RS>01^02996152382802^029840^029001^0291Z00004951^029UPSN^029
 ALPHA DR^029PITTSBURGH^029PA^030<ET>',2)
```

# ShowBarcodeMicroPDF (procedure)

Displays a specified string as a Micro PDF-417 barcode.

**Syntax**

showbarcodemicropdf( str, mode, columns, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*Mode*

Integer value of the Micro PDF mode to use.

Value Mode

0     Text

1     Numeric

*columns*

Integer value (1-4) of the number of columns to use.

*barwidth*

Integer value of the barwidth of a single bar in the Micro PDF

**Code Sample Example**

```
showbarcodemicropdf('123456',0,4,12)
```

# ShowBarcodeMicroQR (procedure)

Displays a specified string as a Micro QR Code barcode.

**Syntax**

showbarcodemicroqr( str, mode, errorcorrection, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*Mode*

Integer value of the Micro QR mode to use.

Value Mode

0     Alpha

1     Numeric

*errorcorrection*

Integer value of the error correction level to use.

Value Correction Level

0     L

1     S

2     Q

*barwidth*

Integer value of the barwidth of a single bar in the Micro QR

**Code Sample Example**

```
showbarcodemicroqr('123456',0,2,12)
```

## ShowBarcodeMSI (procedure)

Displays a specified string as a MSI barcode.

**Syntax**

showbarcodemsi( str, barwidth, readable)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Measure of the width of a single barcode bar, in mils (0.001 inch).

*readable*

Boolean value determining if the human readable portion of the barcode is visible.

**Code Sample Example**

```
showbarcodemsi('1234567')
```

## ShowBarcodeOnecode (procedure)

Displays a specified string as a Onecode/IMB barcode.

**Syntax**

showbarcodeonecode( str)

**Arguments**

*str*

Value of type string to display as a barcode.

**Code Sample Example**

```
showbarcodeonecode('01234567094987654321012345678910')
```

## ShowBarcodePDF417 (procedure)

Displays a specified string as a PDF 417 Code barcode.

**Syntax**

showbarcodepdf417( str, mode, columns, truncated, errorcorrection, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*Mode*

Integer value of the PDF 417 mode to use.

Value Mode
| Value | Mode |
|-------|------------|
| 0 | Text |
| 1 | ASCII Plus |
| 2 | Numeric |

*columns*

Integer value (1-20) of the number of columns to use.

*truncated*

Boolean value to decide to use the truncated version of the barcode

*errorcorrection*

Integer value (0-8) of the error correction level to use.

*barwidth*

Integer value of the barwidth of a single bar in the PDF 417

**Code Sample Example**

```
showbarcodepdf417('123456',0,2,false,5,12)
```


## ShowBarcodePlessey (procedure)

Displays a specified string as a plessey barcode.

**Syntax**

showbarcodeplessey( str, barwidth, readable)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Measure of the width of a single barcode bar, in mils (0.001 inch).

*readable*

Boolean value determining if the human readable portion of the barcode is visible.

**Code Sample Example**

```
showbarcodeplessey('1234567')
```

## ShowBarcodePostnet (procedure)

Displays a specified string as a Postnet barcode.

**Syntax**

showbarcodepostnet( str, subset)

**Arguments**

*str*

Value of type string to display as a barcode.

*subset*

Subset of the Postnet barcode to use

 Subset Zipcode Length
 0       5 digits
 1       9 digits
 2       11 digits

**Code Sample Example**

```
showbarcodepostnet('12345',0)
```

## ShowBarcodeQRCode (procedure)

Displays a specified string as a QR Code barcode.

**Syntax**

showbarcodeqrcode( str, mode, errorcorrection, barwidth)

**Arguments**

*str*

PLANETPRESS. SUITE

Value of type string to display as a barcode.

*Mode*

Integer value of the QR Code mode to use.

| Value | Mode |
|-------|---------|
| 0 | Numeric |
| 1 | Alpha |

*errorcorrection*

Integer value of the error correction level to use.

| Value | Mode |
|-------|------|
| 0 | L |
| 1 | S |
| 2 | Q |
| 3 | H |

*barwidth*

Integer value of the barwidth of a single bar in the QR Code

**Code Sample Example**

```
showbarcodeqrcode('123456',0,2,12)
```

## ShowBarcodeRoyalMail (procedure)

Displays a specified string as a Royal Mail barcode.

**Syntax**

showbarcoderoyalmail( str)

**Arguments**

*str*

Value of type string to display as a barcode.

**Code Sample Example**

```
showbarcoderoyalmail('LE28HS9Z')
```

## ShowBarcodeRSS (procedure)

Displays a specified string as a RSS barcode.

**Syntax**

showbarcoderss( str, mode, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*Mode*

Integer value of the RSS mode to use.

Value Mode
0     RSS-14
1     RSS-Truncated
2     RSS-Limited
3     RSS-Stacked
4     RSS-Stacked Omni
5     RSS-Expanded
6     RSS-Expanded Stacked

*barwidth*

Integer value of the barwidth of a single bar in the RSS Code

**Code Sample Example**

```
showbarcoderss('123456',0,12)
```

# ShowBarcodeUPCA (procedure)

Displays a specified string as a UPC-A barcode.

**Syntax**

showbarcodeupca( str, subset,[readable, barwidth] )

**Arguments**

*str*

Value of type string to display as a barcode.

*Subset*

The subset of UPC-A to use

0 UPC-A
1 UPC-A ext.2
2 UPC-A ext.5

*readable*

Optional boolean that enables display of the human readable portion of the barcode. Default = True.

*barwidth*

Optional width of one barcode bar, in mils (0.001 inch). Default = 12mils.

**Code Sample Example**

```
showbarcodeupca('123456789012',0)
```

## ShowBarcodeUPCE (procedure)

Displays a specified string as a UPC-E barcode.

**Syntax**

showbarcodeupce( str, subset,[readable, barwidth] )

**Arguments**

*str*

Value of type string to display as a barcode.

*Subset*

The subset of UPC-E to use

0 UPC-E
1 UPC-E ext.2
2 UPC-E ext.5

*readable*

Optional boolean that enables display of the human readable portion of the barcode. Default = True.

*barwidth*

Optional width of one barcode bar, in mils (0.001 inch). Default = 12mils.

**Code Sample Example**

```
showbarcodeupce('123456',0)
```

## ShowBitmap (procedure)

Displays a bitmap resource.

**Syntax**

showbitmap( resname, resolution, width, height[, transparent[, duotone]] )

**Arguments**

*resname*

String value containing either the name of the bitmap resource within the document, or the path to a bitmap file.

*resolution*

Integer value specifying the resolution, in pixels per inch, at which the bitmap displays. A larger number yields a smaller, clearer image, since more pixels are squeezed into a smaller space. But the bitmap then requires more memory. Except in some very specific applications, it is rarely desirable to use resolutions exceeding 100 ppi.

*width, height*

Measure values specifying the width/height, in inches, the bitmap occupies when it displays. Specifying a value of zero for one but not both of the two dimensions scales the bitmap to fit the non-zero dimension. Specifying both a width and a height scales the bitmap to the best fit possible with that width and height, while maintaining its aspect ratio. Specifying both a width and a height of zero displays the bitmap at its default resolution.

*transparent*

Boolean value specifying whether part of the bitmap is transparent. When set to true, the bitmap image is combined with the background; otherwise it is pasted on top.

*duotone*

Colour array specifying the colour to paint all non-white pixels in the bitmap.

**Code Sample Examples**

**Example 1**

This example displays the bitmap image street_photo at a resolution of 72 pixels per inch, and a width of 1 inch, as a transparent bitmap image with non-white pixels set to blue.

```
showbitmap('street_photo',72,1,0,true,[100,100,0,0])
```

**Example 2**

This example displays the bitmap image sunset at a resolution of 300 pixels per inch.

```
showbitmap('sunset',300,0,0)
```

**Example 3**

This example prints either an image, or, if the image cannot be found, the pathname to the image.

```
define( &image_paths, directory, 'c:\\images\\*.JPG' )

define( &i, integer, 1 )
```

```
if( resourcetype( &image_paths[&i] ) <> 0 )

  showbitmap( &image_paths[&i], 300, 0, 0)

  crlf( bitmapheight( &image_paths[&i] ) )

elseif()

  show( &image_paths[&i] )

  crlf( 0.16 )

endif()
```

## ShowCaptureUserArea (procedure)

Displays a Capture Field object.

**Syntax**

showcaptureuserarea( name: string, areatype: integer, left: measure,top: measure, width: measure, height: measure, requiredfield: integer, disablerewrite: boolean, groupid: integer )

**Arguments**

*name*

String that defines the name of the Capture Field object (aka "PressTalk ID"). Must be the same as defined in the metadata and can only be letters or numbers (no spaces or special characters).

*areatype*

Integer that defines what kind of area type to use, from the following list:

- 0 - Checkbox
- 1 - List Fields
- 2 - Text Area
- 3 - Multi-Area Field

*left, top*

Measure variables that define the position of the Capture Field, in inches, from the left and top of the page.

*width, height*

Measure variables that define the size, in inches, of the Capture Field.

*requiredfield*

Integer that defines what kind of requirement this field has, from the following list:

- 0 - Optional Field
- 1 - Mandatory Field
- 2 - Final Field

*disablerewrite*

Boolean that defines whether the field will accept new ink if some ink is already present in previous processing of the document. *True* prevents rewriting.

*groupid*

Integer value that defines a group for mandatory fields.

**Example**

```
definemeta('CapPatternSequence','', 0, 'job.group.document')
definemeta('CaptureField', 'Capture1;' + IntToStr(0) + ';2;' + FloatToStr((&physical.x *
72),4)  + ';' + FloatToStr((&physical.y*72),4) + ';' + FloatToStr(3.7861*72) + ';' +
FloatToStr(1.4223*72) + ';false;0', 2, 'job.group.document.datapage.page')
showcaptureuserarea('Capture1',2,&physical.x,&physical.y,&width,&height,0,false,0)
```

# ShowEPS (procedure)

Displays an EPS image resource.

## Syntax

showeps( resname, width, height )

## Argument

*resname*

String value containing either the name of the EPS resource within the document, or the path to an EPS file .

*width, height*

Measure values specifying the width/height, in inches, the EPS occupies when it displays. Specifying a value of zero for one but not both of the two dimensions scales the EPS to fit the non-zero dimension. Specifying both a width and a height scales the EPS to the best fit possible with that width and height, while maintaining its aspect ratio. Specifying both a width and a height of zero displays the EPS at its default resolution.

## Code Sample Example

This example displays the EPS image 'my_eps', scaling it as necessary to fit a width of 4.5 inches.

## Example

```
showeps('my_eps', 4.5, 0)
```

# ShowLeftRight (procedure)

Displays a string of characters, in inches, using a specified width.

## Syntax

showleftright( text, width )

**Arguments**

*text*

String value to be displayed.

*width*

Measure value representing the amount of horizontal space to use for displaying the string. showleftright adjusts the amount of spaces between characters to insure the string fits exactly within the specified width.

**Code Sample Example**

This example shows how to fit strings of different sizes within the same box that has a width of 3 inches.

**Example**

```
margin(0,0)

rectstroke(0,0,3,3)

moveto(0,0.5)

showleftright('This fits well inside the box',3)

crlf(0.1599)

showleftright('But this one is a little more tight',3)
```

# ShowPage (procedure)

Instructs the printer to output the page and move on to the next instruction.

This command has no effect on screen, because it is intended to be executed only inside the printer.

**Syntax**

showpage()

**Argument**

None

**Code Sample Example**

This example specifies the number of pages to print at line 1, between columns 1 and 5.

**Example**

```
Define(&NumberOfPages,integer,strtoint(trim(@(1,1,5))))

Define(&x,integer,1)

for(&x,1,1,&NumberOfPages)

  execpage('Page1')
```

```
    showpage()

endfor()
```

## ShowPDF (procedure)

Displays a page of a PDF image resource.

**Syntax**

showpdf( resname, pagenum, width, height )

**Argument**

*resname*

String value containing either the name of the PDF resource within the document, or the path to a PDF file .

*pagenum*

Integer value specifying the page number within the PDF image resource.

*width, height*

Measure values specifying the width/height, in inches, the PDF page occupies when it displays. Specifying a value of zero for one but not both of the two dimensions scales the PDF page to fit the non-zero dimension. Specifying both a width and a height scales the PDF page to the best fit possible with that width and height, while maintaining its aspect ratio. Specifying both a width and a height of zero displays the PDF page at its default resolution.

**Code Sample Example**

This example displays page 5 of the PDF image resource 'new_models', at its default resolution.

**Example**

```
showpdf('new_models', 5, 0, 0)
```

## ShowUTF8 (procedure)

This command is used to display UTF8 text on the document. It is designed to be used within a **beginUTF8paragraph...endUTF8paragraph** structure. Its uses the paragraph properties defined in the **beginUTF8paragraph** command.

**Syntax**

showUTF8( text )

**Argument**

*text*

Static text string or data selection to be displayed. Bear in mind that the information, wether static or variable, must be in UTF8. The **maputf8** function can be used to convert the information to UTF8.

**Code Sample Example**

This example displays a UTF8 string within a justified paragraph with the text running from right to left.

**Example**

```
setlinewidth(0.007)

moveto(0,0)

rlineto(&width,0)

rlineto(0,&height)

rlineto(neg(&width),0)

rlineto(0,neg(&height))

closepath()

stroke()

moveto(&width/2,&height/18)


BeginUTF8Paragraph(0.0000,&width,0.0000,'leftright',0.1667,'rtl')

   SetStyleExt(&Style1,12.0000,0,[100],100)

   ShowUTF8('\u0623\u0633\u0627\u0633\u064B\u0627\u060C')

EndUTF8Paragraph()
```

## ShowUTF8Left / ShowUTF8Right / ShowUTF8Center (procedure)

These three commands are used to display simple text strings on the document. They are identical in all aspects, except in the way they behave relative to the text insertion point: **showUTF8Left** displays the text from the insertion point to the left; **showUTF8Right** displays the text from the insertion point to the right; and **showUTF8Center** displays the text equally on both sides of the insertion point.

These commands cannot be used with a **beginUTF8paragraph…endUTF8paragraph** structure.

**Syntax**

showUTF8left( text, mode)

showUTF8right( text, mode )

showUTF8center( text, mode )

**Argument**

*text*

Static text string or data selection to be displayed. Bear in mind that the information, wether static or variable, must be in UTF8. The **maputf8** function can be used to convert the information to UTF8.

*mode*

String value specifying the contextual analysis is to be performed. Possible values are 'normal', 'reverse', and 'none' (for no contextual analysis).

**Code Sample Examples**

**Example 1**

This example illustrates a static text string (the name Saudi Arabia) escaped to its UTF8 reference form that will be displayed from the insertion point to the left.

```
setlinewidth(0.007)

moveto(0,0)

rlineto(&width,0)

rlineto(0,&height)

rlineto(neg(&width),0)

rlineto(0,neg(&height))

closepath()

stroke()

moveto(&width/2,&height/2)

SetStyleExt(&Style1,12.0000,0,[100],100)

showUTF8left('\u0627\u0644\u0633\u0639\u0648\u062F\u064A\u0629',        'normal')
```

**Example 2**

This example shows a variable text string that is taken from the data, converted from the MS-CP-1256 encoding to UTF8 and then displayed from the insertion point to the right.

```
setlinewidth(0.007)

moveto(0,0)

rlineto(&width,0)

rlineto(0,&height)

rlineto(neg(&width),0)

rlineto(0,neg(&height))

closepath()

stroke()
```

```
moveto(&width/2,&height/2)

SetStyleExt(&Style1,12.0000,0,[100],100)

showUTF8right(maputf8(@(2,15,35), 'MS-CP-1256', False),'normal')
```

## StopJob (procedure)

Terminates execution of the document and returns control to the PostScript interpreter.

### Syntax

stopjob()

## Store (procedure)

Stores a string of characters on a specific line in the current data page.

### Syntax

store( line, string )

### Arguments

*line*

Integer value specifying the line on which to store the string.

*string*

String value to store in the data page.

### Code Sample Example

This example is taken for an OnReadDataPage event for a user defined emulation. It shows how to store lines of text in the current data page. It searches for a formfeed, increases the current line, stores the string, executes the document, resets the page, and then ends the search.

### Example

```
search(&str,'\014')

   set(&current.line,&current.line+1)

   store(&current.line,&str)

   doform()

   clearpage()

endsearch()
```

## Stroke (procedure)

Outlines the current shape using the pen colour and width specified with the **setstrokecolor** and **setlinewidth** command. The shape is not filled. To fill and outline the shape, use **strokeandfill** instead. If you only want to fill the shape, use **fill**.

**Syntax**

stroke()

**Argument**

None

**Code Sample Example**

This example draws a blue triangle with a blue border.

**Example**

```
setstrokecolor([100,100,0,0]) %Set pen colour to blue

moveto(1,1)

lineto(1.5,2)

lineto(0,2)

closepath()

stroke()
```

## StrokeAndFill (procedure)

Outlines the current shape using the pen colour and width specified with the **setstrokecolor** and **setlinewidth** command. The shape is then filled using the colour specified with the **setfillcolor** command. To simply outline the shape, use **stroke** instead. If you only want to fill the shape, use **fill**.

**Syntax**

strokeandfill()

**Argument**

None

**Code Sample Example**

This example draws a yellow triangle with a blue outline.

**Example**

```
setstrokecolor([100,100,0,0])

setfillcolor([0,0,100,0])
```

```
moveto(1,1)

lineto(1.5,2)

lineto(0,2)

closepath()

strokeandfill()
```

## Translate (procedure)

Offsets an object's position from its original spot on the page. The point of origin of any object is (0,0) by default. All commands within an object are relative to that point of origin. Moving, or translating the point of origin allows you to reset the point of origin within the boundaries of the object.

### Syntax

translate( x, y )

### Arguments

*x, y*

Measure values representing the horizontal/vertical offset, in inches, of the new point of origin within the object. These values are relative to the current point of origin, not absolute.

### Code Sample Examples

### Example 1

This example moves the point of origin 1 inch right and down from the original point.

```
translate(1,1)

show('some text')
```

### Example 2

This example moves the point of origin 1 inch right and down from the current point.

```
translate(1,1)

show('some more text')
```

**Example 3**

This example moves the point of origin back to its original position. **Translate** sets the new point of origin for all subsequent commands. Consequently, **translate** commands are cumulative. Therefore, to restore the previous point of origin, you must issue a **translate** command that negates whatever offset was previously applied.

```
translate(-2,-2)
```

## Functions

### @ (function)

Returns a selection within the current data page for text emulations.

>>> The @ character can be the beginning of a data selection, a page call or a procedure or function call.

**Syntax**

@( line, startcolumn, endcolumn ) ➾ string value

**Arguments**

*line*

Integer value specifying the line on which to read the data in the current data page.

*startcolumn, endcolumn*

Integer values specifying the start/end columns of the chunk of data to read from the current data page.

**Code Sample Example**

The first line of code assigns data from the current data page to a variable. The second line of code checks for the presence of a value inside the data page and sets a Boolean variable according to the results.

```
define(&invnum,string,@(7,50,59))
define(&isfirstpage,boolean,(strtoint(@(1,60,70))=1))
```

### @name (function/procedure)

Executes a global function or procedure created using the function() command.

>>> The @ character can be the beginning of a data selection, a page call or a procedure or function call.

**Syntax**

@functionname( parameters ) ➾ integer, measure, currency, string, Boolean or no return value

## Arguments

*functionname*

String value specifying the name of the function or procedure.

*parameters*

Comma separated list of parameters required by the specified function or procedure.

## BitmapWidth/BitmapHeight (function)

Returns the width or height, in inches, of a bitmap image resource, at the specified resolution.

### Syntax

Bitmapwidth( name, resolution ) ⇒ measure value

Bitmapheight( name, resolution ) ⇒ measure value

### Argument

*name*

String value that specifies the name of the bitmap image resource.

*resolution*

Integer value that specifies the resolution at which you want to measure the width or height of the bitmap.

### Code Sample Example

This example sets the variable maxwidth to the width of the bitmap 'fingerprint' at a resolution of 150 DPI, and the variable maxheight to the height of the same bitmap at a resolution of 200 DPI.

### Example

```
&maxwidth := bitmapwidth( 'fingerprint', 150 )
&maxheight := bitmapheight( 'fingerprint', 200 )
```

## C128 (function)

>>> This function is deprecated and should be replaced by ShowBarcodeCode128 (procedure) whenever encountered. It is kept for backwards compability.

Converts a two-character string to Code 128 bar code, character set C data.

### Syntax

c128( string ) ⇒ string value

### Argument

*string*

String value composed of two characters.

**Code Sample Example**

This example converts the data on line 5, columns 12 to 13, to Code 128 bar code C character set data.

**Example**

```
c128(@(5,12,13))
```

# Ceil (function)

Returns the smallest integer greater than or equal to the specified measure value.

**Syntax**

ceil( value )  ⇒  integer value

**Argument**

*value*

Measure value. The absolute value of the measure value must be less than the maximum value of an integer in PostScript (2,147,483,647).

**Code Sample Examples**

These examples illustrate various applications of **ceil()**.

```
ceil( -2.8 ) %Returns -2
ceil( 2.8 ) %Returns 3
ceil( -5.0 ) %Returns -5
```

# Char (function)

Returns the character whose ASCII value is specified.

**Syntax**

char( value )  ⇒  string value

**Argument**

*value*

Integer value between 0 and 255, specifying the ASCII index of the character.

**Example**

```
show(char(84)+char(97)+char(108)+char(107))
% This displays the word "Talk"
```

## Date (function)

Returns the print date. Note that this function does not work in Printer-Centric mode, since it is impossible for the document to get the current date from a printer. So if you send your document to a printer and then simply send data with the appropriate trigger to that printer, the document will run on the printer and the function will return an empty string. Use the Run locally option, available in the PlanetPress Suite Workflow Tools, to ensure that the document runs on a computer rather than on a printer.

### Syntax

date(longdate)

### Argument

*longdate*

Boolean specifying whether to use short or long date. True returns the date in long format, False, in short format. The precise format of the short and long date value strings are set in the Windows Regional options.

### Code Sample Example

This example shows how to add the current date in long form on a document page.

### Example

```
show(date(true))
```

### Result

On a system set to French (Canada), returns .

## Definemeta (function)

Defines a field when running a PlanetPress Design Document with metadata file creation. In every cases, the name of the field is case-insensitive and must begin with a letter (A-Z, a-z), followed by any number of letters, numbers (0-9), underscore ('_') or dash ('-') characters.

### Syntax

definemeta(name: string, data: string, [flags: integer, path: string])

### Argument

*name*

The name of the metadata field to define. Unless the **path** argument specifies otherwise, the new field is defined at *page level*.

*data*

An initial string value of the defined metadata field. This value can also be a valid PlanetPress Talk expression.

*flags*

Optional integer value to define the behavior of the command when a field with the same name already exists in the current level. The default behavior uses 0 and overwrites the old value with the new one.

| Name | Value | Behavior |
|------|-------|----------|
| DefineOverwrite | 0 | The content of the **value** parameter overwrites the current value of the field if it exists, or creates it if it does not. |
| DefineAppend | 1 | The content of the **value** parameter is appended to the existing field. |
| DefineDuplicate | 2 | A new field with the same name is created, appending an index at the end of its name. |
| DefineError | 3 | The command fails and the job crashes, yielding an error dialog. |

*path*

Optional string value to specify the level where to create the metadata field. Path components cannot be indexed, and if no path is provided, the field is created at the current page level.

>>> The node names of the path argument can have an optional zero-based index, indicated between square brackets ('[ ]'), to access a specific node. If no index is provided, the first node is used.

**Code Sample Example**

This example illustrates how to assign data from the current line printer data file to a document-based metadata field, specifying the level where the field will be created.

```
% Create simple 'InvoiceNumber' holding the value found on line 10,
% between columns 35 and 42 of a line printer emulation data file.
definemeta('InvoiceNumber', @(10,35,42))
% Create the same 'InvoiceNumber' field but duplicating it if it already exists.
definemeta('InvoiceNumber', @(10,35,42), 2)
% Create the same 'InvoiceNumber' field but at the Document level this time,
% and raising an error if it already exists.
definemeta('InvoiceNumber', @(10,35,42), 3, 'Job.Group.Document')
```

## EPSWidth/EPSHeight (function)

Returns the width or height, in inches, of an Encapsulated PostScript (EPS) image resource.

**Syntax**

epswidth( name ) ⇒ measure value

epsheight( name ) ⇒ measure value

**Argument**

*name*

String value that specifies the name of the EPS image resource.

**Code Sample Example**

This example sets the variable maxwidth to the width, and the variable maxheight to the height of the EPS image resource wing_nut.

**Example**

```
&maxwidth := EPSwidth('wing_nut')
&maxheight := EPSheight('wing_nut')
```

## ExecScriptFile (function)

Allows PlanetPress Design documents to execute the content of an external script file. The document waits until the script has completed before resuming its own execution.

**Note**: This function does not support printer centric mode; results are undefined.

### Syntax

ExecScriptFile(filename: string, parameters: string, scriptlanguage: integer) ⇒ integer

### Arguments

*filename*

Fully qualified path and name of the script to call and execute.

*parameters*

User-defined string value allowing a document to pass additional information to the script, such as data selections, PlanetPress Talk variables, etc..

*scriptlanguage*

The programming language in which the script is written. The valid values are:

0 Auto-detect based on file extension: VBScript (.vbs) | JavaScript (.js) | Perl (.pl) | Python (.py)
1 VBScript
2 JavaScript
3 Perl
4 Python

### Return value

Successful execution of the external script file will return 0 by default, although this value may be set by users through the "Script (system object)" (page 247) system variable. Note that some negative values are internally used to indicate different types of failure, always using negative values, which means conflicts may happen when users set the *Script.ReturnValue* variable with an existing negative value. The proper practice is thus to return only positive values.

Here are the existing negative values:

-1 Script file does not exist or is not accessible.
-2 Invalid script language.
-3 Error during script execution.

### Use of PlanetPress APIs within called scripts

When a script is called from within PlanetPress Design using ExecScriptFile(), all of the PlanetPress environment is available to the script, including the Watch, Metadata, Capture and Alambic objects.

> Using these objects from a script within a Design document is not supported and can lead to unexpected results. Furthermore, setting a Watch variable or job info from within the script will not return this value to the document, as these values are always passed once when the document is called, never during execution.

Accessing Parameters from the script

The parameters value is a string which is sent to the executed script. It is available using the Script.Paramstring system variable from within the code. If you want to send multiple parameters, you will have to separate them with a special character and use this character as a separator to split the string or create an array with it, such as (example in JavaScript):

```
params = Script.Paramstring.split(";")
```

**Code Example**

```
ExecScriptFile('c:\myScript.vbs', 'name;'+@(1,1,10), 1)
```

**Sample Scripts**

2 sample scripts are available with any PlanetPress Suite installation.
They are are located in *C:\Documents and Settings\All Users\Application Data\Objectif Lune\PlanetPress Suite N\Scripts*, where N is the PlanetPress Suite version number. These 2 sample scripts are used by PlanetPress Design's Excel Graph functionality, allowing to insert a business graphic from a Microsoft Excel file.

# ExpandString (function)

Returns a string that contains the data from a Workflow tool variable. Any variable available to the Workflow tool can be used here.

## Syntax

ExpandString(stringtoexpand: string) ➡ string

## Arguments

*stringtoexpand*

The Workflow tool variable string to expand. This can be local and global variables, job infos, as well as any other variable such as %o, %j, %y, etc.

## Return Value

Returns the resulting expanded string.

## Code Sample Example

```
define(&result, string, '')
&result := ExpandString('%{global.GlobalVar}')
show(&result)
% Also works with system variables
&result := ExpandString('%o')
show(&result)
```

# Field (function)

Returns the contents of the specified field for the current record. This function is only useful in database emulation mode.

## Syntax

field( fieldname[, recordnumber] ) ➡ string value

**Argument**

*fieldname*

String value specifying the name of the field whose value you want to retrieve.

*recordnumber*

Integer value specifying the number of the record, in the current record set, whose value you want to retrieve. If you omit this argument, the document uses the value of **&current.line**.

This argument is case sensitive. Thus the field names OrderNumber and ORDERNUMBER are not equivalent.

**Code Sample Example**

This example shows how to print the name and content of the current record in a generic fashion.

**Example**

```
define(&x,integer,1) //Define loop variable
define(&fn,string,'') //Define name variable
margin(0,0)
for(&x,1,1,fieldcount()) //Loop through all fields
  set(&fn, fieldname(&x)) //Store field name
  show(&fn) //Display field name
  moveto(2,&current.y) //Move right
  show(field(&fn)) //Display field contents
  crlf() //Move to next line
endfor()
```

## FieldCount (function)

Returns a count of fields in the current database record. Since all records always hold the same number of fields, this value will not change unless a new sample data file is used. This function is only useful in database emulation mode.

**Syntax**

fieldcount() ⇒ integer value

**Code Sample Example**

## FieldName (function)

Returns the name of the specified field for the current record. This function is only useful in database emulation mode.

**Syntax**

fieldname( index ) ⇒ string value

**Argument**

*index*

Integer value specifying the index of the field whose name should be retrieved. Correct values range from 1 to fieldcount().

**Code Sample Example**

# Find (function)

Checks for the presence of a string within a rectangular region of the current data page. If the specified string is found, the function returns true. This allows you to look for strings within a more general area, rather than at some precise location.

## Syntax

find( string1, column1, line1, column2, line2 ) ⇒ Boolean value

## Argument

*string1*

String value to look for.

*column1, line1*

Integer values for the top left corner of the rectangular region to search in the current data page.

*column2, line2*

Integer values for the bottom right corner of the rectangular region to search in the current data page.

## Code Sample Example

This example looks for the word 'Invoice' within a region of the data page.

## Example

```
if(find('Invoice',60,1,80,5))
  show('Invoice')
elseif()
  show('Sales Order')
endif()
```

# Get (function)

Returns an element of an array.

## Syntax

get( array, position ) array element

## Argument

*array*

Name of the array.

*position*

Integer specifying the position in the array, of the element you want to retrieve. Recall that the first position in an array is 0, the second is 1, the third 2, etc.

## Code Sample Examples

Examples 1 and 2 are equivalent representations of using the **get()** command.

### Example 1

```
&month := get(&MyArray,12)
```

### Example 2

```
&month := &MyArray[12]
```

# GetBlack (function)

Returns the value of the Black component of a color array.

### Syntax

getblack( color ) ⇒ integer

### Arguments

*color*

Color array for which you want to determine the Black value.

### Code Sample Example

```
set-
fill-
color([getcyan(&c),getmagenta(&c),getyellow(&c),if(getblack(&c)>50,100,getblack(&c)+20)])
```

# GetCyan (function)

Returns the value of the Cyan component of a color array.

### Syntax

getcyan( color ) ⇒ integer

### Arguments

*color*

Color array for which you want to determine the Cyan value.

**Code Sample Example**

```
set-
fill-
color([getcyan(&c),getmagenta(&c),getyellow(&c),if(getblack(&c)>50,100,getblack(&c)+20)])
```

# GetMagenta (function)

Returns the value of the Magenta component of a color array.

**Syntax**

getmagenta( color ) ⇒ integer

**Arguments**

*color*

Color array for which you want to determine the Magenta value.

**Code Sample Example**

```
set-
fill-
color([getcyan(&c),getmagenta(&c),getyellow(&c),if(getblack(&c)>50,100,getblack(&c)+20)])
```

# Getmeta (function)

Returns the value of a metadata attribute or field.

**Syntax**

getmeta(name: string; [flags : integer; path : string]) ⇒ string

**Arguments**

*name*

The name of the metadata value to retrieve. Unless the **GetAttribute** flag argument is specified, the function looks for the name in the *field* collection.

*flags*

Optional integer value to define the behavior of the command when a field with the same name already exists in the current level. The flag value to enter should be the sum of all desired flags. (Eg: GetAttribute and FailIfNotFound would give a flag value of '5').

The default behavior uses 0.

*Name*         *Value Behavior*

| | | |
|---|---|---|
| GetAttribute | 1 | Search for the **name** argument in the attribute collection instead of the default field collection. |
| NoCascade | 2 | Search only the level specified by the **path** argument (defaults to *Page* level when **path** argument is empty), instead of default behavior, going from the *Page* level to the *Job* level. |
| FailIfNotFound | 4 | Raise an error and crash the job is the specified **name** is not found instead of returning an empty string. |
| SelectedNodesOnly | 8 | Returns values from the selected nodes only. |

*path*

Optional string value to specify the level where to start looking for the field or attribute. Path components cannot be indexed, and if no path

is provided, the search is done from the the *Page* level.

**Note**: The node names of the path argument can have an optional zero-based index, indicated between square brackets ('[ ]'), to access a specific node. If no index is provided, the first node is used.

**Important Note**: If used from PlanetPress Suite Workflow Tools, every argument is **mandatory**.

**Code Sample Example**

These examples show the three different syntaxes for the GetMeta() function.

```
% Retrieve value for field "mydata". If that field exists in both the current group and
the current datapage collections, for instance,

% this call to the GetMeta() function will return the value of the field from the datap-
age.

GetMeta('mydata')
```

```
% Retrieve the second value of the current page, containing two fields called "address",
while the group collection also has four "address" fields.

GetMeta('address[1]')
```

```
% Note that using GetMeta('address[2]') would return the third value of the group level.
To prevent the function from cascading up to the group level,

% the NoCascade flag must be provided. And to retrieve the first two "address" fields of
the group collection, the third argument must be used, set to

% the proper path, thusly:

GetMeta('address[1]', 2, 'Job.Group')
```

## Getmetacount (function)

Returns the number of times an attribute or a field is encountered in the current metadata hierarchy. It can also be used to verify whether a field or attribute exists by comparing the returned value to zero. Note that attributes can only appear once, and thus can only yield two possible values, either zero or one.

**Syntax**

getmetacount(name: string[, flags: integer, path: string]) ⇒ integer

**Arguments**

*name*

The name of the metadata value to count.

*flags*

Optional integer value to define the behavior of the command.The default behavior uses 0.  The flag value to enter should be the sum of all desired flags. (Eg: GetAttribute and FailIfNotFound would give a flag value of '5').

| Name | Value | Behavior |
|------|-------|----------|
| GetAttribute | 1 | Search for the **name** argument in the attribute instead of the default field . |
| NoCascade | 2 | Search only the level specified by the **path** argument (defaults to *Page* level when **path** argument is empty), instead of default behavior, going from the *Page* level to the *Job* level. |
| FailIfNotFound | 4 | Raise an error and crash the job is the specified **name** is not found instead of returning an empty string. |

*path*

Optional string value to specify the level where to start counting the field or attribute. Path components cannot be indexed, and if no path

is provided, the search is done from the the *Page* level.

**Note**: The node names of the path argument can have an optional zero-based index, indicated between square brackets ('[ ]'), to access a specific node. If no index is provided, the first node is used.

Also note that , if no attribute nor field is found, the command returns zero, unless the **FailIfNotFound** flag is specified.

**Code Sample Example**

This example shows the syntax for the GetMetacount() function.

```
% Retrieve and count the number of iterations of "address" for the current datapage
Getmetacount('address')
```

PLANETPRESS. SUITE

# GetYellow (function)

Returns the value of the Yellow component of a color array.

## Syntax

getyellow( color ) ⇒ integer

## Arguments

*color*

Color array for which you want to determine the Yellow value.

## Code Sample Example

```
set-
fill-
color([getcyan(&c),getmagenta(&c),getyellow(&c),if(getblack(&c)>50,100,getblack(&c)+20)])
```

# IsNumber (function)

Tests a string and returns true if the string is a measure or integer value.

## Syntax

isnumber( string ) ⇒ Boolean value

## Argument

*string*

String value.

## Code Sample Example

This example displays the word Yes if the string is either a measure or an integer, and No if it is not.

## Example

```
show(if(isnumber(@(1,1,10)),'Yes','No'))
```

# IsPageEmpty (function)

Returns False if the current data page contains data, or True if it does not.

## Syntax

ispageempty() ⇒ Boolean

## Left (function)

Extracts the specified leftmost characters of a string.

**Syntax**

left( string, number ) ➡ string value

**Arguments**

*string*

String value from which to extract characters.

*number*

Integer value specifying how many characters to extract from the beginning of the string.

**Code Sample Example**

This example prints a string up to a certain position.

**Example**

```
define(&data,string,'This is a~sample string')

%Define string

define(&x,integer, pos('~', &data))

%Find tilde

%Print the string up to the tilde character, if found

if(&x>1)

   show(left(&data,&x-1))

endif()
```

## Length (function)

Returns the length of a string, or the number of elements in an array.

**Syntax**

length( element ) ➡ integer value

**Argument**

*element*

Either a string value, or an array variable.

**Code Sample Example**

**Example 1**

This example creates an a string array with the same number of elements as the &parts array.

```
define( &partnames, arraystring, length( &parts ) )
```

**Example 2**

This example prints part of a string, starting after a specific delimiter.

```
define(&data,string,'This is a~sample string')

define(&x,integer, pos('~', &data))

%Find tilde

if(&x>1)

  show(right(&data,length(&data)-&x))

  %Show remainder of string

endif()
```

# LowerCase (function)

Convert a string to all lower case characters.

## Syntax

lowercase( string ) ⇒ string

## Argument

*string*

String value you want to convert to lower case. The string may be a mix of upper and lower case characters.

## Code Sample Example

```
&partname := lowercase( @(30,16,39) )
```

# MapUTF8 (function)

Converts a text string from its original encoding to UTF8. This function can be used within a ShowUTF8, ShowUTF8Left, a ShowUTF8Right, and a ShowUTF8Center procedure.

## Syntax

maputf8( text, encoding, reversedmapping )

## Arguments

*text*

Static text string or data selection to be converted.

*encoding*

String identifying the original encoding or reference to a style defined in the document that identifies the original encoding.

*reversedmapping*

Boolean specifying whether the text to be converted should be mapped in regular or reversed order. Possible values are true (reversed mapping) and false (no reversed mapping). In the case of preformated Arabic data, this argument should be set to false.

## Supported Encodings

The following Arabic original encodings are supported for the conversion process:

AL-ARABI    ASMO-708   ASMO-708-UNIX IBM-864        ISO-8859-6 (Arabic)
ASMO-449+ ASMO-708+ HP-ARABIC8       IRAN SYSTEM MS-CP-1256

## Code Sample Examples

Example 1 illustrates a variable text string that is taken from the data, converted from the MS-CP-1256 encoding to UTF8 and then displayed from the insertion point to the right. Example 2 illustrates the same variable text string, but this time it is converted from the encoding specified in the style named "MyArabEncoding" to UTF8.

## Example 1

```
setlinewidth(0.007)

moveto(0,0)

rlineto(&width,0)

rlineto(0,&height)

rlineto(neg(&width),0)

rlineto(0,neg(&height))

closepath()

stroke()

moveto(&width/2,&height/2)

SetStyleExt(&Style1,12.0000,0,[100],100)

showUTF8right(maputf8(@(2,15,35), 'MS-CP-1256', False),'normal')
```

## Example 2

```
setlinewidth(0.007)

moveto(0,0)

rlineto(&width,0)
```

```
rlineto(0,&height)

rlineto(neg(&width),0)

rlineto(0,neg(&height))

closepath()

stroke()

moveto(&width/2,&height/2)

SetStyleExt(&Style1,12.0000,0,[100],100)

showUTF8right(maputf8(@(2,15,35), &MyArabEncoding, False),'normal')
```

## Mid (function)

Extracts a specified number of characters from a string, starting at a specific position.

### Syntax

mid( string, start, length ) ⇒ string value

### Arguments

*string*

String value from which to extract characters.

*start*

Integer value specifying where the first character to extract is located.

*length*

Integer value specifying how many characters to extract, starting from number onwards.

### Code Sample Example

This example extracts a bracket-delimited word from a sentence.

### Example

```
define(&data,string,'This is a [bracketed] word')

define(&i,integer, pos('[', &data))

%Find the '[' character

define(&j,integer, pos(']', &data))

%Find the ']' character

if((&i>0) and (&j>&i))

  %If both were found

  show(mid(&data,&i+1,(&j-&i)-1))

  %Print the word
```

```
endif()
```

## Mul (function)

Multiplies two expressions. This is the functional equivalent of the * operator.

**Syntax**

mul( expression, expression2 ) ➡ integer, measure, currency value

**Arguments**

*expression1, expression2*

Integer, measure or currency values. Both expressions must be of the same type. The returned value is set accordingly to the type of the parameters.

**Code Sample Examples**

Examples 1 and 2 show both ways of multiplying numbers.

**Example 1**

```
show(inttostr(2*2)) %Displays 4
```

**Example 2**

```
show(floattostr(mul(4.7,2.1))) %Displays 9.87
```

## Ord (function)

Returns the ASCII value of a character.

**Syntax**

ord( string ) ➡ integer value

**Argument**

*string*

String value whose ASCII value is returned. Only the first character is considered.

**Code Sample Example**

This example shows how to determine if a character is lowercase or uppercase.

**Example**

```
define(&data,string,'a') %Define sample string

if((ord(&data)>=97) and (ord(&data)<=122))
```

```
    %Test lowercase
    show(&data+' is lowercase')
elseif()
  if((ord(&data)>=65) and (ord(&data)<=90))
    %Test uppercase
    show(&data+' is uppercase')
  elseif()
    show(&data+' is not between A and Z or a to z')
  endif()
endif()
```

## PDFPageCount (function)

Returns the number of pages in the specified PDF file.

### Syntax

pdfpagecount( filename ) integer

### Argument

*filename*

String value specifying the path of the PDF file.

## PDFWidth/PDFHeight (function)

Returns the width or height, in inches, of a page of a Portable Document Format (PDF) image resource.

### Syntax

pdfwidth( name, page ) ⇒ measure value

pdfheight( name, page ) ⇒ measure value

### Argument

*name*

String value that specifies the name of the PDF image resource.

*page*

Integer value that specifies the page of the PDF

### Code Sample Example

This example sets the variable maxwidth to the width of page 3, and the variable maxheight to the height of page 4, of the PDF named parts_manual.

**Example**

```
&maxwidth := pdfwidth( 'parts_manual',3 )

&maxheight := pdfheight( 'parts_manual',4 )
```

## PixelHeight (function)

Returns the height, in pixels, of the specified image.

### Syntax

pixelheight( image ) ⇒ integer

### Argument

*image*

String value specifying the path of the file containing the image.

## PixelWidth (function)

Returns the width, in pixels, of the specified image.

### Syntax

pixelwidth( image ) ⇒ integer

### Argument

*image*

String value specifying the path of the file containing the image.

## Pos (function)

Returns the starting position of a specified string within another string, or 0 if the specified string cannot be found.

### Syntax

pos( string1, string2 ) ⇒ integer value

### Arguments

*string1*

String value to search for.

*string2*

String value in which to search.

### Code Sample Example

This example uses POS to look for a word in a data selection.

**Example**

```
define(&s,string,@(1,10,40)) %Initialize var. with data

if(pos('INVOICE',&s)>0) %Look for the word INVOICE

  show('Invoice') %if found, display some text

elseif()

  %Otherwise,display something else

  show('Sales order')

endif()
```

## Random (function)

Returns a measure value between 0 and 1, non-inclusive. Since this function uses the current system time when run inside the printer, it returns a true random value. On the computer, however, the function uses the current data page as its seed to ensure the returned value is constant when you navigate from page to page on your document.

**Syntax**

random() ⇒ measure value

**Argument**

None

**Code Sample Example**

This example displays 10 random numbers between 1 and 20.

**Example**

```
define(&x,integer,1)

for(&x,1,1,10)

  show(inttostr(floattoint(random()*20)+1))

  crlf()

endfor()
```

## Region (function)

Returns an array of strings, containing the text found inside left-top, right-bottom coordinates.
Note: This function only works in Windows Printing or Optimized PostScript Stream. If used in printer centric mode, it will

return an empty string.

**Syntax**

region(left, top, right, bottom) ⇒ array of strings

**Argument**

*left,top,right,bottom*

Measure values specifying the coordinates, in inches or centimeters, for the area on which to read the data in the current data page.
Note that since this function returns an array of string, using region to display data on a page does not make much sense; for better results, consider using regionline().

**Code Sample Example**

This example illustrates how to assign data from the current data page to a array of strings variable by using the *region()* command. Then, a *for()* loop is used to show every item in the array of strings **&detailarray**, followed by a carriage return.

**Example**

```
define(&detailarray, arraystring, region(1.92,3.53,4.65,4.39))
define(&i, integer, 0)

for(&i, 0, 1, Length(&detailarray)-1)
  show(&detailarray[&i])
  crlf()
endfor()
```

**Related topics**:

-

# Regionline (function)

Returns an array of strings, containing the text found inside left-top, right-bottom coordinates.
Note: This function only works in Windows Printing or Optimized PostScript Stream. If used in printer centric mode, it will return an empty string.

**Syntax**

regionline(x1, y1, x2, y2: measure; [index: integer = 0] ⇒ string

**Argument**

*x1, y1, x2, y2*

Measure values specifying the coordinates, in inches or centimeters, for the area on which to read the data in the current data page.

*index*

PLANETPRESS. SUITE

Integer value specifying the line number to return. If no <>index<> paramter is specified, the default value is 0.

**Code Sample Example**

This example illustrates how to assign data from the current data page to a string variable by using the regionline() command.

**Example**

```
define(&invnum, string, regionline(2.1807,0.3018,3.8843,0.9054, 0))
```

**Related topics**:

- ["Region (function)" (page 500)](navigation)

## ResourceType (function)

Returns the type of a resource. Recall that resources are either images or attachments.

**Syntax**

resourcetype( name ) ⇒ integer value

**Argument**

*name*

String value that specifies the name of the resource.

**Return Values**

| Return value: | Indicates: |
|---|---|
| 0 | The resource does not exist or could not be found. |
| 1 | A color bitmap. |
| 2 | A monochrome bitmap. |
| 3 | A grayscale bitmap. |
| 4 | An Encapsulated PostScript (EPS) image. |
| 5 | A PostScript file. |
| 6 | A Portable Document Format (PDF) file. |

**Code Sample Example**

This example sets the variable &resolution to 200 DPI if the resource is a color bitmap, and to 75 DPI if it is not.

**Example**

```
if( eq(resourcetype( 'employee' ), 1))

  &resolution := 200

elseif

  &resolution := 75

endif
```

## Right (function)

Extracts the specified rightmost characters of a string.

**Syntax**

right( string, number ) ⇒ string value

**Arguments**

*string*

String value from which to extract characters.

*number*

Integer value specifying how many characters to extract from the end of the string.

**Code Sample Example**

This example prints part of a string, starting after a specific delimiter.

**Example**

```
define(&data,string,'This is a~sample string')

define(&x,integer, pos('~', &data)) %Find tilde

if(&x>1)

  show(right(&data,length(&data)-&x))

  %Show remainder of string

endif()
```

## StringReplace (function)

Replaces all occurrences of a pattern within a string, with another pattern.

**Syntax**

stringreplace( string, old, new) ⇒ string

**Arguments**

*string*

String in which you want to replace a pattern.

*old*

String value that defines the pattern you want to replace.

*new*

String value that represents the replacement pattern.

**Code Sample Example**

This example replaces the dollar currency symbol with the pound currency symbol.

**Example**

```
stringreplace('1,000.00$','$','£')
```

# StringWidth (function)

Returns the physical display width, in inches, of a string.

**Syntax**

stringwidth( string ) ⇒ measure value

**Argument**

*string*

String value whose width is returned.

**Code Sample Example**

This example draws a tight box around a variable length data selection.

**Example**

```
margin(0,0)
%Define the data. This could be a data selection.
define(&data,string,'This is a sample string')
set(&data,trimright(&data))
%Store the length
define(&dw,measure,stringwidth(&data))
moveto(1,1)
%Display the string, then draw a box around it.
show(&data)
rectstroke(1,0.86,&dw,0.2)
```

## StringWidthUTF8 (function)

Returns the length, in inches, of the string as it would be displayed on a page, using the current font. Bear in mind that the combination of many bytes sometimes results in a single glyph and that text orientation can also affect the number of glyphs displayed.

**Syntax**

stringwidthutf8(text, mode)

**Arguments**

*text*

UTF8 text string to be measured.

*mode*

String specifying how the contextual analysis is to be performed. Possible values are 'normal', 'reverse', and 'none' (for no contextual analysis).

## Strip (function)

Removes all occurrences of a string within another string.

**Syntax**

strip( string1, string2 ) ➡ string value

**Arguments**

*string1*

String value to be removed from string2.

*string2*

String value to change.

**Code Sample Example**

This example strips all dollar signs from a variable.

**Example**

```
define(&data,string,'$11.95')

show(strip('$',&data))
```

## Sub (function)

Subtracts one expression from another. This is the functional equivalent of the - operator.

**Syntax**

sub( expression, expression2 ) ➡ integer, measure, currency value

**Arguments**

*expression1, expression2*

Integer, measure or currency values. Both expressions must be the same type. The returned value is set according to the type of the parameters.

**Code Sample Example**

Example 1 and Example 2 illustrates both ways of subtracting numbers.

**Example 1**

```
show(inttostr(2-2))
```

**Example 2**

```
show(floattostr(sub(4.7,2.1)))
```

# SubRecCount (function)

Returns the number of records per data page. Since data pages can hold a variable number of records, this value may change whenever you skip through data pages. Every data page always has at least one child, otherwise it wouldn't be stored in the database. Subreccount can return 0 only if the emulation is set to something other than database mode, or if the converted database is invalid. This function is only useful in database emulation mode.

**Syntax**

subreccount ➡ integer value

**Code Sample Example**

This example prints the line items from an invoice. It defines the loop and total variables, loops through all records, shows and positions invoice information, and finally shows the invoice total. The fields listed here are for illustration purposes only.

**Example**

```
define(&x,integer,1)

define(&tot,measure,0.0)

for(&x,1,1,subreccount())

  show(field('PartNum'))
```

PLANETPRESS. SUITE

```
    moveto(1.5,&current.y)

    show(field('PartDesc'))

    moveto(4.5,&current.y)

    show(field('Qty'))

    moveto(5.5,&current.y)

    show(field('UnitPrice'))

    moveto(6.5,&current.y)

    show(field('Total'))

    %Add line total to tot variable

    set(&tot,&tot+ strtofloat(field('Total')))

    moveto(0,&current.y+.25)
endfor()
moveto(6.5,8)
show(floattostr(&tot))
```

## Time (function)

Returns the print time. Note that this function *will only work if the document runs on a computer*, since it is impossible for the document to get the current time from a printer. So if you send your document to a printer and then simply send data with the appropriate trigger to that printer, the document will run on the printer and the function will return an empty string. Use the Run locally option, available in the PlanetPress Suite Workflow Tools, to ensure that the document runs on a computer rather than on a printer.

**Syntax**

time(displayseconds)

**Argument**

*displayseconds*

Boolean specifying whether to display the time using seconds. True returns the time with seconds, False, without. The precise format of the of the time value strings are set in the Windows Regional options.

**Code Sample Example**

This example shows how to add the current time with seconds (11:14:46 AM, as opposed to 11:14 AM, for example) on a document page.

**Example**

```
show(time(true))
```

## Trim (function)

Removes both leading and trailing spaces from a string.

**Syntax**

trim( string ) ➡ string value

**Argument**

*string*

String value from which you want to trim leading and trailing spaces.

**Code Sample Example**

This example trims leading and trailing spaces from the data selection on line 3, columns 8 to 24.

**Example**

```
trim(@(3,8,24))
```

## TrimLeft (function)

Removes leading spaces from a string.

**Syntax**

trimleft( string ) ➡ string value

**Argument**

*string*

String value to trim.

**Code Sample Example**

This example makes sure * delimiters are added immediately before and after a data selection.

**Example**

```
show('*'+trimleft(trimright(@(12,10,35)))+'*')
```

## TrimRight (function)

Removes trailing spaces from a string.

**Syntax**

trimright( string ) ⇒ string value

**Argument**

*string*

String value to trim.

**Code Sample Example**

## UpperCase (function)

Convert a string to all upper case characters.

**Syntax**

uppercase( string ) ⇒ string

**Argument**

*string*

String value you want to convert to upper case. The string may be a mix of upper and lower case characters.

**Code Sample Example**

This example illustrates **uppercase()**.

**Example**

```
&month := uppercase( @(1,60,70) )
```

## xmlCount()

Counts the number of children for a specific element, according to standard XPath syntax. The return value is an integer and always returns 0 if the specified XPath is invalid or if no value is found. The XPath is always relative to the XML root path.

**Syntax**

xmlCount(path: string) ⇒ integer value

**Arguments**

*path*

The XML Path of the value you want to retrieve. The path must correspond to a single entry in the XML, thus is not a complete XPath with regular expressions and variables.

**Code Sample**

The following gets the number of items in an invoice, loops through each item and displays the item's description with show().

```
define(&i, integer, 0)
define(&currentItemDescription, string, '')

define(&numberOfItems, integer, xmlcount('/MyData/CUSTOMER/INVOICES/INVOICE/ITEM'))

for(&i, 1, 1, &numberOfItems)
      &currentItemDescription := xmlget('-
/MyData/CUSTOMER/INVOICES/INVOICE/ITEM['+inttostr(&i)+']/Description')
      show(&currentItemDescription)
      crlf()
endfor()
```

# xmlGet()

Retrieves a data selection value through the structure of an XML file. This function uses a single parameter specifying the Path of the value to be retrieved. This function returns a string. The return value is an empty string if no value is found or if the specified Element is invalid or not found.

**Syntax**

xmlGet(path: string) ⇒ string value

**Arguments**

*path*

The XML Path of the value you want to retrieve. The path must correspond to a single entry in the XML, thus is not a complete XPath with regular expressions and variables.

**Code Sample**

These are examples of xmlGet() commands returning data from an XML file. Note that these will most likely not work with your own XML even if you tried, as the path is completely dependent on the exact structure of your XML Data file.

```
define(&myVar, string, '')
&myVar := xmlget('/PLANETPRESS_DATA_FILE[1]-
/CUSTOMER[1]/INVOICES[1]/INVOICE[1]/ITEM[1]/Description[1]')
&myVar := xmlget('/MyData/CUSTOMER[1]/INVOICES[1]/INVOICE[1]/ITEM[1]/Description[1]')
% This next line uses the &i variable to dynamically select an item in the invoice. Use-
ful in a repeat & overflow
% configuration, or a manual PlanetPress Talk loop through te data. Note that the var-
iable must be converted to a string.
&myVar := xmlget('-
/MyData/CUSTOMER[1]/INVOICES[1]/INVOICE[1]/ITEM['+inttostr(&i)+']/Description[1]')
```

Example 2

The following gets the number of items in an invoice, loops through each item and displays the item's description with show().

```
define(&i, integer, 0)
define(&currentItemDescription, string, '')

define(&numberOfItems, integer, xmlcount('/MyData/CUSTOMER/INVOICES/INVOICE/ITEM'))

for(&i, 1, 1, &numberOfItems)
      &currentItemDescription := xmlget('-
/MyData/CUSTOMER/INVOICES/INVOICE/ITEM['+inttostr(&i)+']/Description')
      show(&currentItemDescription)
      crlf()
endfor()
```

# System Variables (by data type)

"Integer" (page 511)

"Measure" (page 511)

"String" (page 512)

"Boolean" (page 512)

## Integer

&Current.DataPage (see "Current (system object)" (page 376))

&Current.Line (see "Current (system object)" (page 376))

&Current.LPP (see "Current (system object)" (page 376))

&Current.MediaWeight (see "Current (system object)" (page 376))

&Current.Orientation (see "Current (system object)" (page 376))

&Current.PrintPage (see "Current (system object)" (page 376))

"&PrinterMode (system variable)" (page 375)

"SubRecCount (function)" (page 526)

&Metamode (variable)

&System.FormVersion (see "&system (system object)" (page 378))

## Measure

&Current.MinFeature (see "Current (system object)" (page 376))

&Current.PageHeight (see "Current (system object)" (page 376))

&Current.PageWidth (see "Current (system object)" (page 376))

&Current.X (see "Current (system object)" (page 376))

&Current.Y (see ["Current (system object)" (page 376)](#))

["&Height (system variable)" (page 374)](#)

&Physical.X (see ["Physical (system object)" (page 378)](#))

&Physical.Y (see ["Physical (system object)" (page 378)](#))

["&Width (system variable)" (page 376)](#)

## String

&Current.MediaColor (see ["Current (system object)" (page 376)](#))

&Current.MediaType (see ["Current (system object)" (page 376)](#))

["&Str (system variable)" (page 375)](#)

&System.Product (see ["&system (system object)" (page 378)](#))

&System.Version (see ["&system (system object)" (page 378)](#))

## Boolean

["&EOJob (system variable)" (page 374)](#)

["&FirstSide (system variable)" (page 374)](#)


# Functions (by return value data type)

This section contains a reference to all PlanetPress Talk functions, including operator functions, by the data type of their return values.

## Currency

### Add (function)

Adds two numerical expressions. This is the functional equivalent to the + operator, when you use the + operator with numerical expressions.

**Syntax**

add( expression, expression2 ) ⇒ integer, measure, currency value

**Arguments**

*expression1, expression2*

Integer, measure, or currency values. Both expressions must be the same type. The returned value is set according to the type of the parameters.

**Code Sample Examples**

Examples 1 and 2 show both ways of adding numbers.

**Example 1**

```
show(inttostr(2+2)) %Displays 4
```

**Example 2**

```
show(floattostr(add(4.7,2.1))) %Displays 6.8
```

# FloatToCur (function)

Converts a measure expression into a currency value.

**Syntax**

floattocur( expression ) ⇒ currency value

**Argument**

*expression*

Measure value to be converted. The decimal part is rounded down to 2 decimal points.

**Code Sample Example**

This example converts a floating point value into a currency value.

```
floattocur(3.263457) %Returns 3.26
```

# IntToCur (function)

Converts an integer expression into a measure value.

**Syntax**

inttocur( expression ) ⇒ currency value

**Argument**

*expression*

Integer value to be converted.

# StrToCur (function)

Converts a string value into a currency value.

**Syntax**

strtocur( string ) ⇒ currency value

PLANETPRESS. SUITE

**Argument**

*string*

String value to convert.

## Integer

### Add (function)

Adds two numerical expressions. This is the functional equivalent to the + operator, when you use the + operator with numerical expressions.

**Syntax**

add( expression, expression2 ) ➡ integer, measure, currency value

**Arguments**

*expression1, expression2*

Integer, measure, or currency values. Both expressions must be the same type. The returned value is set according to the type of the parameters.

**Code Sample Examples**

Examples 1 and 2 show both ways of adding numbers.

**Example 1**

```
show(inttostr(2+2)) %Displays 4
```

**Example 2**

```
show(floattostr(add(4.7,2.1))) %Displays 6.8
```

### Ceil (function)

Returns the smallest integer greater than or equal to the specified measure value.

**Syntax**

ceil( value ) ➡ integer value

**Argument**

*value*

Measure value. The absolute value of the measure value must be less than the maximum value of an integer in PostScript (2,147,483,647).

**Code Sample Examples**

These examples illustrate various applications of **ceil()**.

```
ceil( -2.8 ) %Returns -2
ceil( 2.8 ) %Returns 3
ceil( -5.0 ) %Returns -5
```

## Div (function)

Divides an expression with another. This is the functional equivalent to the / operator.

### Syntax

div( expression, expression2 ) ➡ integer, measure, currency value

### Arguments

*expression1, expression2*

Integer, measure or currency values. Both expressions must be the same type. The returned value is set according to the type of the parameters.

### Code Sample Examples

Examples 1 and 2 show both ways of dividing numbers.

### Example 1

```
show(inttostr(2/2)) %Displays 1
```

### Example 2

```
show(floattostr(div(4.7,2.1))) %Displays 2.2381
```

## FieldCount (function)

Returns a count of fields in the current database record. Since all records always hold the same number of fields, this value will not change unless a new sample data file is used. This function is only useful in database emulation mode.

### Syntax

fieldcount() ➡ integer value

### Code Sample Example

## FloatToInt (function)

Converts a measure expression into an integer value.

### Syntax

floattoint( expression ) ➡ integer value

### Argument

*expression*

PLANETPRESS. SUITE

Measure value to be converted. The decimal part is rounded down to a whole number.

**Code Sample Examples**

These examples illustrate **floattoint()**.

**Example 1**

```
floattoint(3.2) %Returns 3
```

**Example 2**

```
floattoint(11.6) %Returns 12
```

# Get (function)

Returns an element of an array.

**Syntax**

get( array, position ) array element

**Argument**

*array*

Name of the array.

*position*

Integer specifying the position in the array, of the element you want to retrieve. Recall that the first position in an array is 0, the second is 1, the third 2, etc.

**Code Sample Examples**

Examples 1 and 2 are equivalent representations of using the **get()** command.

**Example 1**

```
&month := get(&MyArray,12)
```

**Example 2**

```
&month := &MyArray[12]
```

# GetBlack (function)

Returns the value of the Black component of a color array.

**Syntax**

getblack( color ) ➡ integer

**Arguments**

*color*

Color array for which you want to determine the Black value.

**Code Sample Example**

```
set-
fill-
color([getcyan(&c),getmagenta(&c),getyellow(&c),if(getblack(&c)>50,100,getblack(&c)+20)])
```

## GetCyan (function)

Returns the value of the Cyan component of a color array.

**Syntax**

getcyan( color ) ➡ integer

**Arguments**

*color*

Color array for which you want to determine the Cyan value.

**Code Sample Example**

```
set-
fill-
color([getcyan(&c),getmagenta(&c),getyellow(&c),if(getblack(&c)>50,100,getblack(&c)+20)])
```

## GetMagenta (function)

Returns the value of the Magenta component of a color array.

**Syntax**

getmagenta( color ) ➡ integer

**Arguments**

*color*

Color array for which you want to determine the Magenta value.

**Code Sample Example**

```
set-
fill-
color([getcyan(&c),getmagenta(&c),getyellow(&c),if(getblack(&c)>50,100,getblack(&c)+20)])
```

# GetYellow (function)

Returns the value of the Yellow component of a color array.

**Syntax**

getyellow( color ) ⇒ integer

**Arguments**

*color*

Color array for which you want to determine the Yellow value.

**Code Sample Example**

```
set-
fill-
color([getcyan(&c),getmagenta(&c),getyellow(&c),if(getblack(&c)>50,100,getblack(&c)+20)])
```

# Getmetacount (function)

Returns the number of times an attribute or a field is encountered in the current metadata hierarchy. It can also be used to ver-ify whether a field or attribute exists by comparing the returned value to zero. Note that attributes can only appear once, and thus can only yield two possible values, either zero or one.

**Syntax**

getmetacount(name: string[, flags: integer, path: string]) ⇒ integer

**Arguments**

*name*

The name of the metadata value to count.

*flags*

Optional integer value to define the behavior of the command.The default behavior uses 0.  The flag value to enter should be the sum of all desired flags. (Eg: GetAttribute and FailIfNotFound would give a flag value of '5').

| Name | Value | Behavior |
| --- | --- | --- |
| GetAttribute | 1 | Search for the **name** argument in the attribute instead of the default field . |

| NoCascade | 2 | Search only the level specified by the **path** argument (defaults to *Page* level when **path** argument is empty), instead of default behavior, going from the *Page* level to the *Job* level. |
|-----------|---|---|
| FailIfNotFound | 4 | Raise an error and crash the job is the specified **name** is not found instead of returning an empty string. |

*path*

Optional string value to specify the level where to start counting the field or attribute. Path components cannot be indexed, and if no path

is provided, the search is done from the the *Page* level.

**Note**: The node names of the path argument can have an optional zero-based index, indicated between square brackets ('[ ]'), to access a specific node. If no index is provided, the first node is used.

Also note that , if no attribute nor field is found, the command returns zero, unless the **FailIfNotFound** flag is specified.

**Code Sample Example**

This example shows the syntax for the GetMetacount() function.

```
% Retrieve and count the number of iterations of "address" for the current datapage
Getmetacount('address')
```

## If (function)

Evaluates an expression and returns one of two specified values according to the results of the evaluation.

**Syntax**

if( expression, trueresult, falseresult ) ⇒ any type

**Arguments**

*expression*

Boolean value. If it evaluates to true, the function returns the contents of trueresult, otherwise it returns the contents of falseresult.

*trueresult, falseresult*

Values of any type, as long as both are of the same type. The return value of the function is therefore of the same type as trueresult and falseresult.

**Code Sample Example**

This example changes the text to display according to the current value of a variable.

**Example**

```
show(if(&pagenumber=1,'Customer Copy','Store Copy'))
```

## Length (function)

Returns the length of a string, or the number of elements in an array.

**Syntax**

length( element ) ⇒ integer value

**Argument**

*element*

Either a string value, or an array variable.

**Code Sample Example**

**Example 1**

This example creates an a string array with the same number of elements as the &parts array.

```
define( &partnames, arraystring, length( &parts ) )
```

**Example 2**

This example prints part of a string, starting after a specific delimiter.

```
define(&data,string,'This is a~sample string')

define(&x,integer, pos('~', &data))

%Find tilde

if(&x>1)

  show(right(&data,length(&data)-&x))

  %Show remainder of string

endif()
```

## Mod (function & procedure)

Returns the remainder resulting from the division of two numbers.

**Syntax**

mod( number1, number2 ) ⇒ integer value

number1 mod number2 ⇒ integer value

**Arguments**

*number1*

Integer value to be divided.

*number2*

Integer value with which to divide number1.

**Code Sample Example**

This example uses MOD to distinguish between odd and even numbers.

**Example**

```
define(&i,integer, 1)

%Define loop variable

for(&i,1,1,10) %Set loop

  show(inttostr(&i)+' is an') %Display number

  if(mod(&i,2)=1)

  %Determine if number is odd or even

    show(' odd ')

  elseif()

    show(' even ')

  endif()

  show('number.')

  crlf()

endfor()
```

## Mul (function)

Multiplies two expressions. This is the functional equivalent of the * operator.

**Syntax**

mul( expression, expression2 ) ⇒ integer, measure, currency value

**Arguments**

*expression1, expression2*

Integer, measure or currency values. Both expressions must be of the same type. The returned value is set accordingly to the type of the parameters.

**Code Sample Examples**

Examples 1 and 2 show both ways of multiplying numbers.

**Example 1**

```
show(inttostr(2*2)) %Displays 4
```

**Example 2**

```
show(floattostr(mul(4.7,2.1))) %Displays 9.87
```

# Neg (function)

Returns the negative value of the specified expression. This is the functional equivalent to the - sign.

**Syntax**

neg( expression1 ) ⇒ integer, measure, currency value

**Argument**

*expression1*

Integer, measure or currency value. The returned value is the same type as expression1.

**Code Sample Examples**

Examples 1and 2 show both ways of negating a number. Negating negative numbers yields a positive result.

**Example 1**

```
-12 %Returns -12

-&max %Returns negative of the value of the variable &max

neg(-12.45) %Returns 12.45
```

# Ord (function)

Returns the ASCII value of a character.

**Syntax**

ord( string ) ⇒ integer value

**Argument**

*string*

String value whose ASCII value is returned. Only the first character is considered.

**Code Sample Example**

This example shows how to determine if a character is lowercase or uppercase.

**Example**

```
define(&data,string,'a') %Define sample string

if((ord(&data)>=97) and (ord(&data)<=122))
```

```
      %Test lowercase
      show(&data+' is lowercase')
elseif()
   if((ord(&data)>=65) and (ord(&data)<=90))
      %Test uppercase
      show(&data+' is uppercase')
   elseif()
      show(&data+' is not between A and Z or a to z')
   endif()
endif()
```

## PDFPageCount (function)

Returns the number of pages in the specified PDF file.

### Syntax

pdfpagecount( filename ) integer

### Argument

*filename*

String value specifying the path of the PDF file.

## PixelHeight (function)

Returns the height, in pixels, of the specified image.

### Syntax

pixelheight( image ) ⇒ integer

### Argument

*image*

String value specifying the path of the file containing the image.

## PixelWidth (function)

Returns the width, in pixels, of the specified image.

### Syntax

pixelwidth( image ) ⇒ integer

## Argument

*image*

String value specifying the path of the file containing the image.

## Pos (function)

Returns the starting position of a specified string within another string, or 0 if the specified string cannot be found.

### Syntax

pos( string1, string2 ) ⇒ integer value

### Arguments

*string1*

String value to search for.

*string2*

String value in which to search.

### Code Sample Example

This example uses POS to look for a word in a data selection.

### Example

```
define(&s,string,@(1,10,40)) %Initialize var. with data

if(pos('INVOICE',&s)>0) %Look for the word INVOICE

  show('Invoice') %if found, display some text

elseif()

  %Otherwise,display something else

  show('Sales order')

endif()
```

## ResourceType (function)

Returns the type of a resource. Recall that resources are either images or attachments.

### Syntax

resourcetype( name ) ⇒ integer value

### Argument

*name*

String value that specifies the name of the resource.

**Return Values**

| Return value: | Indicates: |
|---|---|
| 0 | The resource does not exist or could not be found. |
| 1 | A color bitmap. |
| 2 | A monochrome bitmap. |
| 3 | A grayscale bitmap. |
| 4 | An Encapsulated PostScript (EPS) image. |
| 5 | A PostScript file. |
| 6 | A Portable Document Format (PDF) file. |

**Code Sample Example**

This example sets the variable &resolution to 200 DPI if the resource is a color bitmap, and to 75 DPI if it is not.

**Example**

```
if( eq(resourcetype( 'employee' ), 1))

  &resolution := 200

elseif

  &resolution := 75

endif
```

# StrToInt (function)

Converts a string into an integer value.

**Syntax**

strtoint( string ) ⇒ integer value

**Argument**

*string*

String value to convert.

**Code Sample Example**

This example reads a column of 6 values then converts the values to integers and then adds the integer to the &total variable. The total value is subsequently converted back to a string and the result is printed.

**Example**

```
define(&total,integer,0)

define(&x,integer,0)

for(&x,1,1,6)
```

```
set(&total,&total+strtoint(@(&x,10,18)))
```

```
endfor()
```

```
show(inttostr(&total))
```

## Sub (function)

Subtracts one expression from another. This is the functional equivalent of the - operator.

### Syntax

sub( expression, expression2 ) ➡ integer, measure, currency value

### Arguments

*expression1, expression2*

Integer, measure or currency values. Both expressions must be the same type. The returned value is set according to the type of the parameters.

### Code Sample Example

Example 1 and Example 2 illustrates both ways of subtracting numbers.

### Example 1

```
show(inttostr(2-2))
```

### Example 2

```
show(floattostr(sub(4.7,2.1)))
```

## SubRecCount (function)

Returns the number of records per data page. Since data pages can hold a variable number of records, this value may change whenever you skip through data pages. Every data page always has at least one child, otherwise it wouldn't be stored in the database. Subreccount can return 0 only if the emulation is set to something other than database mode, or if the converted database is invalid. This function is only useful in database emulation mode.

### Syntax

subreccount ➡ integer value

**Code Sample Example**

This example prints the line items from an invoice. It defines the loop and total variables, loops through all records, shows and positions invoice information, and finally shows the invoice total. The fields listed here are for illustration purposes only.

**Example**

```
define(&x,integer,1)
define(&tot,measure,0.0)
for(&x,1,1,subreccount())
  show(field('PartNum'))
  moveto(1.5,&current.y)
  show(field('PartDesc'))
  moveto(4.5,&current.y)
  show(field('Qty'))
  moveto(5.5,&current.y)
  show(field('UnitPrice'))
  moveto(6.5,&current.y)
  show(field('Total'))
  %Add line total to tot variable
  set(&tot,&tot+ strtofloat(field('Total')))
  moveto(0,&current.y+.25)
endfor()
moveto(6.5,8)
show(floattostr(&tot))
```

# xmlCount()

Counts the number of children for a specific element, according to standard XPath syntax. The return value is an integer and always returns 0 if the specified XPath is invalid or if no value is found. The XPath is always relative to the XML root path.

**Syntax**

xmlCount(path: string) ⇒ integer value

**Arguments**

*path*

The XML Path of the value you want to retrieve. The path must correspond to a single entry in the XML, thus is not a complete XPath with regular expressions and variables.

**Code Sample**

The following gets the number of items in an invoice, loops through each item and displays the item's description with show().

```
define(&i, integer, 0)
define(&currentItemDescription, string, '')

define(&numberOfItems, integer, xmlcount('/MyData/CUSTOMER/INVOICES/INVOICE/ITEM'))

for(&i, 1, 1, &numberOfItems)
      &currentItemDescription := xmlget('-
/MyData/CUSTOMER/INVOICES/INVOICE/ITEM['+inttostr(&i)+']/Description')
      show(&currentItemDescription)
      crlf()
endfor()
```

# Measure

## Add (function)

Adds two numerical expressions. This is the functional equivalent to the + operator, when you use the + operator with numerical expressions.

**Syntax**

add( expression, expression2 ) ⟹ integer, measure, currency value

**Arguments**

*expression1, expression2*

Integer, measure, or currency values. Both expressions must be the same type. The returned value is set according to the type of the parameters.

**Code Sample Examples**

Examples 1 and 2 show both ways of adding numbers.

**Example 1**

```
show(inttostr(2+2)) %Displays 4
```

**Example 2**

```
show(floattostr(add(4.7,2.1))) %Displays 6.8
```

## BitmapWidth/BitmapHeight (function)

Returns the width or height, in inches, of a bitmap image resource, at the specified resolution.

**Syntax**

Bitmapwidth( name, resolution ) ⇒ measure value

Bitmapheight( name, resolution ) ⇒ measure value

**Argument**

*name*

String value that specifies the name of the bitmap image resource.

*resolution*

Integer value that specifies the resolution at which you want to measure the width or height of the bitmap.

**Code Sample Example**

This example sets the variable maxwidth to the width of the bitmap 'fingerprint' at a resolution of 150 DPI, and the variable maxheight to the height of the same bitmap at a resolution of 200 DPI.

**Example**

```
&maxwidth := bitmapwidth( 'fingerprint', 150 )
&maxheight := bitmapheight( 'fingerprint', 200 )
```

## Cos (function)

Returns the cosine value of the specified angle.

**Syntax**

cos( value ) ⇒ measure value

**Argument**

*value*

Measure value specifying the angle whose cosine is returned.

**Code Sample Example**

This example displays a sinusoidal graph.

**Example**

```
moveto(0,0)
define(&i,integer,0)
for(&i,1,10,360)
        lineto(cos(inttofloat((&i)/ 4)),sin(inttofloat(&i)))
endfor()
closepath()
```

## CurToFloat (function)

Converts a currency expression into a measure value.

**Syntax**

curtofloat( expression ) ⇒ measure value

**Argument**

*expression*

Currency value to convert.

## Div (function)

Divides an expression with another. This is the functional equivalent to the / operator.

**Syntax**

div( expression, expression2 ) ⇒ integer, measure, currency value

**Arguments**

*expression1, expression2*

Integer, measure or currency values. Both expressions must be the same type. The returned value is set according to the type of the parameters.

**Code Sample Examples**

Examples 1 and 2 show both ways of dividing numbers.

**Example 1**

```
show(inttostr(2/2)) %Displays 1
```

**Example 2**

```
show(floattostr(div(4.7,2.1))) %Displays 2.2381
```

## EPSWidth/EPSHeight (function)

Returns the width or height, in inches, of an Encapsulated PostScript (EPS) image resource.

**Syntax**

epswidth( name ) ⇒ measure value

epsheight( name ) ⇒ measure value

**Argument**

*name*

String value that specifies the name of the EPS image resource.

## Code Sample Example

This example sets the variable maxwidth to the width, and the variable maxheight to the height of the EPS image resource wing_nut.

## Example

```
&maxwidth := EPSwidth('wing_nut')
&maxheight := EPSheight('wing_nut')
```

# Get (function)

Returns an element of an array.

## Syntax

get( array, position ) array element

## Argument

*array*

Name of the array.

*position*

Integer specifying the position in the array, of the element you want to retrieve. Recall that the first position in an array is 0, the second is 1, the third 2, etc.

## Code Sample Examples

Examples 1 and 2 are equivalent representations of using the **get()** command.

## Example 1

```
&month := get(&MyArray,12)
```

## Example 2

```
&month := &MyArray[12]
```

# If (function)

Evaluates an expression and returns one of two specified values according to the results of the evaluation.

## Syntax

if( expression, trueresult, falseresult ) ⇒ any type

**Arguments**

*expression*

Boolean value. If it evaluates to true, the function returns the contents of trueresult, otherwise it returns the contents of falseresult.

*trueresult, falseresult*

Values of any type, as long as both are of the same type. The return value of the function is therefore of the same type as trueresult and falseresult.

**Code Sample Example**

This example changes the text to display according to the current value of a variable.

**Example**

```
show(if(&pagenumber=1,'Customer Copy','Store Copy'))
```

# IntToFloat (function)

Converts an integer expression into a measure value.

**Syntax**

inttofloat( expression ) ⇨ measure value

**Argument**

*expression*

Integer value to be converted.

**Code Sample Example**

This example uses a loop to draw a series of blue rectangles.

**Example**

```
define(&x,integer,1)

%Define loop variable

setfillcolor([100,100,0,0])

%Fill blue

for(&x,1,1,10)

   %Set up loop

   %We must convert &x because rectfill

   %expects measure values

   rectfill(inttofloat(&x)/2,1,0.1,1)

   %Draw rectangle
```

```
endfor()
```

## Mul (function)

Multiplies two expressions. This is the functional equivalent of the * operator.

**Syntax**

mul( expression, expression2 ) ⇒ integer, measure, currency value

**Arguments**

*expression1, expression2*

Integer, measure or currency values. Both expressions must be of the same type. The returned value is set accordingly to the type of the parameters.

**Code Sample Examples**

Examples 1 and 2 show both ways of multiplying numbers.

**Example 1**

```
show(inttostr(2*2)) %Displays 4
```

**Example 2**

```
show(floattostr(mul(4.7,2.1))) %Displays 9.87
```

## Neg (function)

Returns the negative value of the specified expression. This is the functional equivalent to the - sign.

**Syntax**

neg( expression1 ) ⇒ integer, measure, currency value

**Argument**

*expression1*

Integer, measure or currency value. The returned value is the same type as expression1.

**Code Sample Examples**

Examples 1and 2 show both ways of negating a number. Negating negative numbers yields a positive result.

**Example 1**

```
-12 %Returns -12

-&max %Returns negative of the value of the variable &max

neg(-12.45) %Returns 12.45
```

## PDFWidth/PDFHeight (function)

Returns the width or height, in inches, of a page of a Portable Document Format (PDF) image resource.

**Syntax**

pdfwidth( name, page ) ⇒ measure value

pdfheight( name, page ) ⇒ measure value

**Argument**

*name*

String value that specifies the name of the PDF image resource.

*page*

Integer value that specifies the page of the PDF

**Code Sample Example**

This example sets the variable maxwidth to the width of page 3, and the variable maxheight to the height of page 4, of the PDF named parts_manual.

**Example**

```
&maxwidth := pdfwidth( 'parts_manual',3 )

&maxheight := pdfheight( 'parts_manual',4 )
```

## Random (function)

Returns a measure value between 0 and 1, non-inclusive. Since this function uses the current system time when run inside the printer, it returns a true random value. On the computer, however, the function uses the current data page as its seed to ensure the returned value is constant when you navigate from page to page on your document.

**Syntax**

random() ⇒ measure value

**Argument**

None

**Code Sample Example**

This example displays 10 random numbers between 1 and 20.

**Example**

```
define(&x,integer,1)

for(&x,1,1,10)

   show(inttostr(floattoint(random()*20)+1))

   crlf()

endfor()
```

## Sin (function)

Returns the sine value of the specified angle.

**Syntax**

sin( value ) ➡ measure value

**Argument**

*value*

Measure value specifying the angle whose sine is returned.

**Code Sample Example**

This example draws a somewhat sinusoidal graph.

**Example**

```
moveto(0,0)

define(&i,integer,0)

for(&i,1,10,360)

   lineto(cos(inttofloat((&i)/ 4)),sin(inttofloat(&i)))

endfor()

closepath()
```

## StringWidth (function)

Returns the physical display width, in inches, of a string.

**Syntax**

stringwidth( string ) ➡ measure value

**Argument**

*string*

PLANETPRESS. SUITE

String value whose width is returned.

**Code Sample Example**

This example draws a tight box around a variable length data selection.

**Example**

```
margin(0,0)

%Define the data. This could be a data selection.

define(&data,string,'This is a sample string')

set(&data,trimright(&data))

%Store the length

define(&dw,measure,stringwidth(&data))

moveto(1,1)

%Display the string, then draw a box around it.

show(&data)

rectstroke(1,0.86,&dw,0.2)
```

## StringWidthUTF8 (function)

Returns the length, in inches, of the string as it would be displayed on a page, using the current font. Bear in mind that the combination of many bytes sometimes results in a single glyph and that text orientation can also affect the number of glyphs displayed.

**Syntax**

stringwidthutf8(text, mode)

**Arguments**

*text*

UTF8 text string to be measured.

*mode*

String specifying how the contextual analysis is to be performed. Possible values are 'normal', 'reverse', and 'none' (for no contextual analysis).

## StrToFloat (function)

Converts a string into a measure value.

**Syntax**

strtofloat( string ) ⇒ measure value

**Argument**

*string*

String value to convert.

**Code Sample Example**

This example reads a column of 6 values then converts the values to floating point values and then adds the ir values to the &total variable. The total value is subsequently converted back to a string and the result is printed.

**Example**

```
define(&total,measure,0.0)

define(&x,integer,0)

for(&x,1,1,6)

  set(&total,&total+strtofloat(@(&x,10,18)))

endfor()

show(floattostr(&total))
```

# Sub (function)

Subtracts one expression from another. This is the functional equivalent of the - operator.

**Syntax**

sub( expression, expression2 ) ➡ integer, measure, currency value

**Arguments**

*expression1, expression2*

Integer, measure or currency values. Both expressions must be the same type. The returned value is set according to the type of the parameters.

**Code Sample Example**

Example 1 and Example 2 illustrates both ways of subtracting numbers.

**Example 1**

```
show(inttostr(2-2))
```

**Example 2**

```
show(floattostr(sub(4.7,2.1)))
```

## String

### @ (function)

Returns a selection within the current data page for text emulations.

>>> The @ character can be the beginning of a [data selection](), a [page call]() or a [procedure or function call]().

**Syntax**

@( line, startcolumn, endcolumn ) ⇒ string value

**Arguments**

*line*

Integer value specifying the line on which to read the data in the current data page.

*startcolumn, endcolumn*

Integer values specifying the start/end columns of the chunk of data to read from the current data page.

**Code Sample Example**

The first line of code assigns data from the current data page to a variable. The second line of code checks for the presence of a value inside the data page and sets a Boolean variable according to the results.

```
define(&invnum,string,@(7,50,59))
define(&isfirstpage,boolean,(strtoint(@(1,60,70))=1))
```

### C128 (function)

>>> This function is deprecated and should be replaced by [ShowBarcodeCode128 (procedure)]() whenever encountered. It is kept for backwards compability.

Converts a two-character string to Code 128 bar code, character set C data.

**Syntax**

c128( string ) ⇒ string value

**Argument**

*string*

String value composed of two characters.

**Code Sample Example**

This example converts the data on line 5, columns 12 to 13, to Code 128 bar code C character set data.

### Example

```
c128(@(5,12,13))
```

## Char (function)

Returns the character whose ASCII value is specified.

### Syntax

char( value ) ⇒ string value

### Argument

*value*

Integer value between 0 and 255, specifying the ASCII index of the character.

### Example

```
show(char(84)+char(97)+char(108)+char(107))
% This displays the word "Talk"
```

## CurToStr (function)

Converts a currency expression into a string value.

### Syntax

curtostr( expression ) ⇒ string value

### Argument

*expression*

Currency value to convert.

## ExpandString (function)

Returns a string that contains the data from a Workflow tool variable. Any variable available to the Workflow tool can be used here.

### Syntax

ExpandString(stringtoexpand: string) ⇒ string

### Arguments

*stringtoexpand*

The Workflow tool variable string to expand. This can be local and global variables, job infos, as well as any other variable such as %o, %j, %y, etc.

### Return Value

Returns the resulting expanded string.

**Code Sample Example**

```
define(&result, string, '')
&result := ExpandString('%{global.GlobalVar}')
show(&result)
% Also works with system variables
&result := ExpandString('%o')
show(&result)
```

# Field (function)

Returns the contents of the specified field for the current record. This function is only useful in database emulation mode.

**Syntax**

field( fieldname[, recordnumber] ) ⇒ string value

**Argument**

*fieldname*

String value specifying the name of the field whose value you want to retrieve.

*recordnumber*

Integer value specifying the number of the record, in the current record set, whose value you want to retrieve. If you omit this argument, the document uses the value of **&current.line**.

This argument is case sensitive. Thus the field names OrderNumber and ORDERNUMBER are not equivalent.

**Code Sample Example**

This example shows how to print the name and content of the current record in a generic fashion.

**Example**

```
define(&x,integer,1) //Define loop variable
define(&fn,string,'') //Define name variable
margin(0,0)
for(&x,1,1,fieldcount()) //Loop through all fields
  set(&fn, fieldname(&x)) //Store field name
  show(&fn) //Display field name
  moveto(2,&current.y) //Move right
  show(field(&fn)) //Display field contents
  crlf() //Move to next line
endfor()
```

# FieldName (function)

Returns the name of the specified field for the current record. This function is only useful in database emulation mode.

**Syntax**

fieldname( index ) ⇒ string value

**Argument**

*index*

Integer value specifying the index of the field whose name should be retrieved. Correct values range from 1 to fieldcount().

**Code Sample Example**

# FloatToStr (function)

Converts a measure expression into a string value.

**Syntax**

floattostr( expression[, precision]) ⇒ string value

**Argument**

*expression*

Measure value to be converted.

*precision*

The number of decimal places to include in the result string.

**Code Sample Examples**

These examples illustrate **floattostr()**.

**Example 1**

```
floattostr(3.14) %Returns 3.14
```

**Example 2**

```
floattostr(11) %Returns 11.0
```

**Example 3**

```
floattostr(34.6453455, 4) %Returns 34.6453
```

# Get (function)

Returns an element of an array.

**Syntax**

get( array, position ) array element

**Argument**

*array*

Name of the array.

PLANETPRESS. SUITE

*position*

Integer specifying the position in the array, of the element you want to retrieve. Recall that the first position in an array is 0, the second is 1, the third 2, etc.

**Code Sample Examples**

Examples 1 and 2 are equivalent representations of using the **get()** command.

**Example 1**

```
&month := get(&MyArray,12)
```

**Example 2**

```
&month := &MyArray[12]
```

# If (function)

Evaluates an expression and returns one of two specified values according to the results of the evaluation.

**Syntax**

if( expression, trueresult, falseresult ) ⇒ any type

**Arguments**

*expression*

Boolean value. If it evaluates to true, the function returns the contents of trueresult, otherwise it returns the contents of falseresult.

*trueresult, falseresult*

Values of any type, as long as both are of the same type. The return value of the function is therefore of the same type as trueresult and falseresult.

**Code Sample Example**

This example changes the text to display according to the current value of a variable.

**Example**

```
show(if(&pagenumber=1,'Customer Copy','Store Copy'))
```

# IntToStr (function)

Converts an integer expression into a string value.

**Syntax**

inttostr( expression ) ➡ string value

**Argument**

*expression*

Integer value to be converted.

**Code Sample Example**

This example prints the iterations of a loop.

**Example**

```
define(&x,integer,1)

%Define loop variable

for(&x,1,1,10)

  %Set up loop

  show('Iteration ' + inttostr(&x) )

  %Show text

  crlf()

  %Skip line

endfor()
```

## Left (function)

Extracts the specified leftmost characters of a string.

**Syntax**

left( string, number ) ➡ string value

**Arguments**

*string*

String value from which to extract characters.

*number*

Integer value specifying how many characters to extract from the beginning of the string.

**Code Sample Example**

This example prints a string up to a certain position.

**Example**

```
define(&data,string,'This is a~sample string')
```

```
%Define string
define(&x,integer, pos('~', &data))
%Find tilde
%Print the string up to the tilde character, if found
if(&x>1)
   show(left(&data,&x-1))
endif()
```

## LowerCase (function)

Convert a string to all lower case characters.

**Syntax**

lowercase( string ) ⇒ string

**Argument**

*string*

String value you want to convert to lower case. The string may be a mix of upper and lower case characters.

**Code Sample Example**

```
&partname := lowercase( @(30,16,39) )
```

## Mid (function)

Extracts a specified number of characters from a string, starting at a specific position.

**Syntax**

mid( string, start, length ) ⇒ string value

**Arguments**

*string*

String value from which to extract characters.

*start*

Integer value specifying where the first character to extract is located.

*length*

Integer value specifying how many characters to extract, starting from number onwards.

**Code Sample Example**

This example extracts a bracket-delimited word from a sentence.

**Example**

---

```
define(&data,string,'This is a [bracketed] word')

define(&i,integer, pos('[', &data))

%Find the '[' character

define(&j,integer, pos(']', &data))

%Find the ']' character

if((&i>0) and (&j>&i))

  %If both were found

  show(mid(&data,&i+1,(&j-&i)-1))

  %Print the word

endif()
```

---

# Right (function)

Extracts the specified rightmost characters of a string.

**Syntax**

right( string, number ) ⇒ string value

**Arguments**

*string*

String value from which to extract characters.

*number*

Integer value specifying how many characters to extract from the end of the string.

**Code Sample Example**

This example prints part of a string, starting after a specific delimiter.

**Example**

---

```
define(&data,string,'This is a~sample string')

define(&x,integer, pos('~', &data)) %Find tilde

if(&x>1)

  show(right(&data,length(&data)-&x))
```

```
   %Show remainder of string

endif()
```

## StringReplace (function)

Replaces all occurrences of a pattern within a string, with another pattern.

### Syntax

stringreplace( string, old, new) ⇒ string

### Arguments

*string*

String in which you want to replace a pattern.

*old*

String value that defines the pattern you want to replace.

*new*

String value that represents the replacement pattern.

### Code Sample Example

This example replaces the dollar currency symbol with the pound currency symbol.

### Example

```
stringreplace('1,000.00$','$','£')
```

## Strip (function)

Removes all occurrences of a string within another string.

### Syntax

strip( string1, string2 ) ⇒ string value

### Arguments

*string1*

String value to be removed from string2.

*string2*

String value to change.

**Code Sample Example**

This example strips all dollar signs from a variable.

**Example**

```
define(&data,string,'$11.95')

show(strip('$',&data))
```

# Trim (function)

Removes both leading and trailing spaces from a string.

**Syntax**

trim( string ) ➡ string value

**Argument**

*string*

String value from which you want to trim leading and trailing spaces.

**Code Sample Example**

This example trims leading and trailing spaces from the data selection on line 3, columns 8 to 24.

**Example**

```
trim(@(3,8,24))
```

# TrimLeft (function)

Removes leading spaces from a string.

**Syntax**

trimleft( string ) ➡ string value

**Argument**

*string*

String value to trim.

**Code Sample Example**

This example makes sure * delimiters are added immediately before and after a data selection.

**Example**

```
show('*'+trimleft(trimright(@(12,10,35)))+'*')
```

## TrimRight (function)

Removes trailing spaces from a string.

**Syntax**

trimright( string ) ⇒ string value

**Argument**

*string*

String value to trim.

**Code Sample Example**

"TrimLeft (function)" (page 547)

## UpperCase (function)

Convert a string to all upper case characters.

**Syntax**

uppercase( string ) ⇒ string

**Argument**

*string*

String value you want to convert to upper case. The string may be a mix of upper and lower case characters.

**Code Sample Example**

This example illustrates **uppercase()**.

**Example**

```
&month := uppercase( @(1,60,70) )
```

## xmlGet()

Retrieves a data selection value through the structure of an XML file. This function uses a single parameter specifying the Path of the value to be retrieved. This function returns a string. The return value is an empty string if no value is found or if the specified Element is invalid or not found.

**Syntax**

xmlGet(path: string) ➡ string value

**Arguments**

*path*

The XML Path of the value you want to retrieve. The path must correspond to a single entry in the XML, thus is not a complete XPath with regular expressions and variables.

**Code Sample**

These are examples of xmlGet() commands returning data from an XML file. Note that these will most likely not work with your own XML even if you tried, as the path is completely dependent on the exact structure of your XML Data file.

```
define(&myVar, string, '')
&myVar := xmlget('/PLANETPRESS_DATA_FILE[1]-
/CUSTOMER[1]/INVOICES[1]/INVOICE[1]/ITEM[1]/Description[1]')
&myVar := xmlget('/MyData/CUSTOMER[1]/INVOICES[1]/INVOICE[1]/ITEM[1]/Description[1]')
% This next line uses the &i variable to dynamically select an item in the invoice. Use-
ful in a repeat & overflow
% configuration, or a manual PlanetPress Talk loop through te data. Note that the var-
iable must be converted to a string.
&myVar := xmlget('-
/MyData/CUSTOMER[1]/INVOICES[1]/INVOICE[1]/ITEM['+inttostr(&i)+']/Description[1]')
```

Example 2

The following gets the number of items in an invoice, loops through each item and displays the item's description with show().

```
define(&i, integer, 0)
define(&currentItemDescription, string, '')

define(&numberOfItems, integer, xmlcount('/MyData/CUSTOMER/INVOICES/INVOICE/ITEM'))

for(&i, 1, 1, &numberOfItems)
      &currentItemDescription := xmlget('-
/MyData/CUSTOMER/INVOICES/INVOICE/ITEM['+inttostr(&i)+']/Description')
      show(&currentItemDescription)
      crlf()
endfor()
```

# Boolean

# Add (function)

Adds two numerical expressions. This is the functional equivalent to the + operator, when you use the + operator with numerical expressions.

## Syntax

add( expression, expression2 ) ⇒ integer, measure, currency value

## Arguments

*expression1, expression2*

Integer, measure, or currency values. Both expressions must be the same type. The returned value is set according to the type of the parameters.

## Code Sample Examples

Examples 1 and 2 show both ways of adding numbers.

## Example 1

```
show(inttostr(2+2)) %Displays 4
```

## Example 2

```
show(floattostr(add(4.7,2.1))) %Displays 6.8
```

## Eq (function)

Compares two expressions of any type and returns true if both are equal, false otherwise. This is the functional equivalent of the = operator.

## Syntax

eq( expression1, expression2 ) ⇒ boolean value

## Arguments

*expression1, expression2*

Values of any type. Both expressions must be of the same type.

## Code Sample Examples

Examples 1 and 2 shows both ways of comparing two expressions.

## Example 1

```
if( eq(&current.line, 33) )
  show('Middle of page')
endif()
```

## Example 2

```
if(&current.line = 33)
  show('Middle of page')
endif()
```

# Find (function)

Checks for the presence of a string within a rectangular region of the current data page. If the specified string is found, the function returns true. This allows you to look for strings within a more general area, rather than at some precise location.

## Syntax

find( string1, column1, line1, column2, line2 ) ⇒ Boolean value

## Argument

*string1*

String value to look for.

*column1, line1*

Integer values for the top left corner of the rectangular region to search in the current data page.

*column2, line2*

Integer values for the bottom right corner of the rectangular region to search in the current data page.

## Code Sample Example

This example looks for the word 'Invoice' within a region of the data page.

## Example

```
if(find('Invoice',60,1,80,5))
  show('Invoice')
elseif()
  show('Sales Order')
endif()
```

# GE (function)

Compares two expressions of any type and returns true if the first is greater than or equal to the second, false otherwise. This is the functional equivalent to the >= operator.

## Syntax

ge( expression1, expression2 ) ⇒ Boolean value

## Arguments

*expression1, expression2*

Values of any type. Both expressions must be of the same type.

## Code Sample Examples

Example 1 and Example 2 show both ways of comparing two expressions. Example 2 illustrates the >= operator.

## Example 1

```
if( ge(&current.line, 50) )
  show('Bottom of page')
endif()
```

**Example 2**

```
if(&current.line >= 50)
  show('Bottom of page')
endif()
```

## GT (function)

Compares two expressions of any type and returns true if the first is greater than the second, false otherwise. This is the functional equivalent to the > operator.

**Syntax**

gt( expression1, expression2 ) ⇒ Boolean value

**Arguments**

*expression1, expression2*

Values of any type. Both expressions must be of the same type.

**Code Sample Examples**

Examples 1 and 2 illustrate both ways of comparing two expressions.

**Example 1**

```
if( gt(&current.line, 50) )

  show('Bottom of page')

endif()
```

**Example 2**

```
if(&current.line > 50)

  show('Bottom of page)

endif()
```

## If (function)

Evaluates an expression and returns one of two specified values according to the results of the evaluation.

**Syntax**

if( expression, trueresult, falseresult ) ⇒ any type

**Arguments**

*expression*

Boolean value. If it evaluates to true, the function returns the contents of trueresult, otherwise it returns the contents of falseresult.

*trueresult, falseresult*

Values of any type, as long as both are of the same type. The return value of the function is therefore of the same type as trueresult and falseresult.

**Code Sample Example**

This example changes the text to display according to the current value of a variable.

**Example**

```
show(if(&pagenumber=1,'Customer Copy','Store Copy'))
```

# IsNumber (function)

Tests a string and returns true if the string is a measure or integer value.

**Syntax**

isnumber( string ) ⇒ Boolean value

**Argument**

*string*

String value.

**Code Sample Example**

This example displays the word Yes if the string is either a measure or an integer, and No if it is not.

**Example**

```
show(if(isnumber(@(1,1,10)),'Yes','No'))
```

# IsPageEmpty (function)

Returns False if the current data page contains data, or True if it does not.

**Syntax**

ispageempty() ⇒ Boolean

---

## LE (function)

Compares two expressions of any type and returns true if the first is less than or equal to the second, false otherwise. This is the functional equivalent to the <= operator.

**Syntax**

le( expression1, expression2 ) ⇒ Boolean value

**Arguments**

*expression1, expression2*

Values of any type. Both expressions must be of the same type.

**Code Sample Example**

This example shows both ways of comparing two expressions.

**Example**

```
if(le(&current.line, 15) )

  show('Top of page')

endif()

%Same statement, using the <= operator

if(&current.line <= 50)

  show('Top of page')

endif()
```

## LT (function)

Compares two expressions of any type and returns true if the first is less than the second, false otherwise. This is the functional equivalent to the < operator.

**Syntax**

lt( expression1, expression2 ) ⇒ Boolean value

**Arguments**

*expression1, expression2*

Values of any type. Both expressions must be of the same type.

**Code Sample Example**

This example shows both ways of comparing two expressions.

**Example**

```
if(lt(&current.line, 15))
```

```
    show('Top of page')

endif()

%Same statement, using the < operator

if(&current.line < 50)

    show('Top of page')

endif()
```

## NE (function)

Compares two expressions of any type and returns true if they are not equal, false otherwise. This is the functional equivalent to the <> operator.

**Syntax**

ne( expression1, expression2 ) ➡ Boolean value

**Arguments**

*expression1, expression2*

Values of any type. Both expressions must be of the same type.

**Code Sample Examples**

Examples 1 and 2 show both ways of comparing two expressions.

**Example 1**

```
if( ne(&current.line, 1) )

    show('Not a new page')

endif()
```

**Example 2**

```
if(&current.line <> 1) %Same statement, using the <> operator

    show('Not a new page')

endif()
```

## Not (Boolean operator function)

Negates the specified expression. Not to be confused with the **neg** function: not is a Boolean operator function, not a mathematical one.

**Syntax**

not( expression1 ) ➡ Boolean value

**Argument**

*expression1*

Boolean value.

**Code Sample Example**

This example shows how NOT can be used in place of operators.

**Example**

```
define(&x,integer,1) %Define loop variable

for(&x,1,1,10) %Set loop

  if(not(&x=5)) %Same as if(&x<>5)

    show('This is NOT the 5th line') %Show some text

  elseif()

    show('This IS the 5th line') %Show alternate

  endif()

endfor()
```

## Or (Boolean operator function)

Returns true if either or both specified expressions are true, false otherwise.

**Syntax**

or( expression1, expression2 ) ➡ Boolean value

expression1 or expression2 ➡ Boolean value

**Arguments**

*expression1, expression2*

Boolean values.

**Code Sample Examples**

These examples illustrate the possible values for OR.

**Example 1**

---

```
or(true, false) %Returns true
```

---

**Example 2**

---

```
or(false, true) %Returns true
```

---

**Example 3**

---

```
or(false, false) %Returns false
```

---

**Example 4**

---

```
or(true, true) %Returns true
```

---

## XOr (Boolean operator function)

Returns true if only one of two specified expressions is true, false otherwise.

**Syntax**

xor( expression1, expression2 ) ⇒ ' boolean value

expression1 xor expression2 ⇒ boolean value

**Arguments**

*expression1, expression2*

Boolean values.

**Code Sample Examples**

These examples illustrate the possible values for XOR.

**Example 1**

---

```
xor(true, false) %Returns true
```

---

### Example 2

```
xor(false, true) %Returns true
```

### Example 3

```
xor(false, false) %Returns false
```

### Example 4

```
xor(true, true) %Returns false
```

# Procedures (by category)

## Debugging

## Breakpoint (procedure)

Stops the execution of a PlanetPress Talk object when the specified expression is True. This breakpoint is effective only when running in debug mode in the PlanetPress Talk editor.

### Syntax

breakpoint( expression1 )

### Argument

*expression1*

Boolean value.

### Code Sample Example

This example sets up a loop, stops the loop on its 5th iteration, and then displays the value.

### Example

```
define(&x,integer,1)for(&x,1,1,10)
      breakpoint(&x = 5)
      show(inttostr(&x))
endfor()
```

## OutputDebugString (procedure)

Outputs a string to the PlanetPress Talk Messages window in PlanetPress.

### Syntax

outputdebugstring(message)

### Argument

*message*

String value to display in the PlanetPress Talk Messages window.

## Variables

## Define (procedure)

This command creates a new local variable, or redefines an existing one. The define procedure must appear before the variable is actually used. PlanetPress Talk scripts require all variables to be defined either as a global variable in your document, or a local scope variable within your PlanetPress Talk script.

### Syntax

define( name, type, value )

### Arguments

*name*

The name of the variable, prefixed with the ampersand (&) character. The name must conform to the rules for names described in <u>"Names" (page 219)</u>.

*type*

Constant value specifying the variable type, either integer, measure, string, Boolean, or an array type. The array types are color, arrayinteger, arraymeasure, arraycurrency, arrayBoolean, arraystring or directory. Once a type is assigned to a variable, it cannot be changed, unless the variable is redefined.

*value*

An initial value for the variable, of the same type as the variable (for example if the variable is of type integer, this value must be an integer). This value can also be a valid PlanetPress Talk expression. In the case of any array variable other than a color array or an array of type directory, the initial value can be either the number of elements you want the array to contain, or the set of values you want the array to contain. In the former case you assign values to the array elements in subsequent commands. In the latter case, you separate each value by a comma, and enclose the complete set of elements in square brackets. In the case of a string array, you must also quote each of the array elements. In an array of type directory, you specify the pathname to the folder and the file name filter you want to apply to each of the files in that folder; this in turn determines the number of elements in the array.

**Code Sample Example**

```
define( &customer_name, string, 'Smith' )
define( &invoice_number, integer, strtoint(@(1,1,10)) )
define( &left_margin, measure, 1.5 )
define( &is_bottom, boolean, (&current.line > 50) )
define( &my_array,arrayinteger,[12,3,76,109,4] )
define( &my_array,arrayinteger, 5 )
define( &prices,arraycurrency, 6)
define( &greetings,arraystring,['hello','bonjour','ola'] )
define( &mustard,color,[0,12,84,16] )
define( &image_paths, directory, 'c:\\images\\*.JPG' )
define( &cost, currency, 0.00 )
```

## Put (procedure)

Assigns a value to an element of an array.

**Syntax**

put( &array, index, value )

**Arguments**

*&array*

Array variable containing the element whose value you want to change.

*index*

Integer value representing the position of the element in the array.

*value*

The value you want to assign to the element. The value must be of the same the type as the array. Thus if the array is of type string, the value must be of type string.

## Set (procedure)

This command assigns a new value to a variable. Only user-defined variables can be set. System variables are read-only. Note that you can also use the assignment operator to assign a value to a variable. See "= (operator)" (page 391).

**Syntax**

set( name, value )

**Arguments**

*name*

Variable name, prefixed with the ampersand (&) character. The variable name must already exist.

*value*

New value to store in the variable. This value can be any valid PlanetPress Talk expression, as long as its type is the same as the variable's original type.

**Code Sample Example**

```
set( &invoice_number, (&invoice_number + 1) )

set( &is_bottom, (not(&is_top) and not(&is_middle)) )

&tax_rates[2] :=32

put( &tax_rates[2], @(1,3,5) )

set( &months, ['JAN','FEB','MAR','APR','MAY'] )

set( &mustard_color,[0,20,90,16] )

set( &prices, [12.5632,18.9932,23.6651,27.0300] )

put( &imagefiles[0], 'c\\images\\sushi.png' )
```

## Global Functions

## @name (function/procedure)

Executes a global function or procedure created using the function() command.

> The @ character can be the beginning of a data selection, a page call or a procedure or function call.

**Syntax**

@functionname( parameters ) ⇒ integer, measure, currency, string, Boolean or no return value

**Arguments**

*functionname*

String value specifying the name of the function or procedure.

*parameters*

Comma separated list of parameters required by the specified function or procedure.

## Function @name (procedure)

Defines a new PlanetPress Talk function or procedure. If you define a function that you want to return a value, you assign that value to the predefined variable **&result** on the last line of the function definition. You call the function or procedure you define using the **@name()** command. See "@name (function/procedure)" (page 561).

You can reference a global function from within another global function only if the global function you are referencing already exists.

> You cannot reference a function from within itself, so recursive functions are not possible within PlanetPress Talk.

**Syntax**

**Syntax for a function that returns a value:**

function @name( arglist ): type

...

&result := return-_value

endfunction()

**Syntax for a function that does not return a value (procedure):**

function @name( arglist )

...

endfunction()

**Arguments**

*name*

Name to use for the function, prefixed with the @ character. The name must conform to the rules for names described in "Names" (page 219). It is good practice to be careful naming functions that you define in different documents, to ensure that if at some point you want to copy code between documents, the names of functions do not conflict.

*arglist*

List of arguments the function requires, where each element in the list has the syntax varname : type. For example, &date : string. An ampersand always precedes the first letter of varname. A comma separates each element in the list. Note that all arguments are pass by value (as opposed to pass by reference). Pass by value means that the function you define cannot modify any of the arguments it receives. Any modifications you make to an argument within the body of the function are made to a local copy, and do not affect the original.

*type*

Type of value (integer, measure, currency, string, Boolean) the function returns.

**Code Sample Example**

This example displays a string.

```
function @showtext(&mystring: string)
  moveto(1,1)
  show(&mystring)
endfunction()
```

**Code Sample Example**

This example illustrates a procedure that creates a 3D pie chart.

**Example**

```
function @_3DPie( &W:measure, &H:measure, &HB:measure, &S:measure, &E:measure,
&L:string, &c:color )
  %==================================================
  % &W is the total width of the pie chart
  % &H is the total height of the pie chart
  % &HB is the height of the lower band
  % &S is the starting angle (0..360)
  % &E is the ending angle (0..360 and bigger than &S)
  % &L is the label text
  %==================================================
  %Local variables
  %==================================================
  define(&SB,measure,0)
  define(&EB,measure,0)
  define(&Scaling,measure,0)
  define(&R,measure,&W/2)
  %==================================================
  &Scaling := (&H-&HB)/&W
  gsave()
  if(&Scaling<0.01)
    &Scaling := 0.01
  endif()
  if(&Scaling>1)
    &Scaling := 1
  endif()
  scale(1,&Scaling)
  setstyle(&Style1)
  &HB := &HB / &Scaling
  translate(&R,&R)
  if(&E > 180)
    if(&E > 360)
      &EB := 360
    elseif()
      &EB := &E
    endif()
```

```
  %=================================================
  if(&S < 180)
    &SB := 180
  elseif()
    &SB := &S
  endif()
  %=================================================
  set-
fill-
color([getcyan(&c),getmagenta(&c),getyellow(&c),if(getblack(&c)>50,100,getblack(&c)+20)])
    moveto(cos(&SB) * &R,neg(sin(&SB) * &R))
    rlineto(0,&HB)
    arc(0,&HB,&R,&SB,&EB)
    rlineto(0,neg(&HB))
    closepath()
    strokeandfill()
  endif()
  %=================================================
  setfillcolor(&c)
  pie(0,0,&R,&S,&E)
  strokeandfill()
  &SB := (&S+&E)/2
  %=================================================
  if(gt(&SB,180))
    margin(cos(&SB) * &R,neg(sin(&SB) * &R)+&HB+(0.2/&Scaling))
  elseif()
    margin(cos(&SB) * &R,neg(sin(&SB) * &R)-(0.05/&Scaling))
  endif()
  %=================================================
  scale(1,1/&Scaling)
  if(and(gt(&SB,90),lt(&SB,270)))
    showright(&L+' ')
  elseif()
    show(' '+&L)
  endif()
  grestore()
```

## Comments

## % (procedure)

Comments out the line. This is useful for embedding documentation in your code, making it easier to read and maintain. It is also useful during debugging, for commenting out lines of code you do not want to execute.

Comments must not appear in the first line of the script you enter in the PlanetPress Talk Editor.

### Syntax

% … comment ….

---

**Argument**

None

**Code Sample Example**

This example illustrates comments in PlanetPress Talk code.

```
define(&x,integer,1)
% Creates a new local variable named x that is an
% integer and has a value of 1. The variable is used
% in a loop
for(&x,1,1,10)
        show(inttostr(&x))
endfor()
```

# Graphics State

## GRestore (procedure)

This command restores all system parameters previously saved with the **gsave** command. If no **gsave** command was issued previously, the results are unpredictable. For more information, refer to the **gsave** command.

**Syntax**

grestore()

**Argument**

None

**Code Sample Example**

See .

## GSave (procedure)

This command saves all current system parameters, which can later be restored using the **grestore** command. By bracketing parts of your PlanetPress Talk programs with **gsave**/**grestore** commands, you can make sure the current system state is preserved and remains unaffected by whatever operations your program executes. For instance, each PlanetPress object, once converted to its PlanetPress Talk scripting language equivalent, begins with a **gsave** command and ends with a **grestore**, thus ensuring objects do not interfere with each other, or with the system.

The system parameters gsave saves include current line width, current stroke color, and current fill color.

**Syntax**

gsave()

**Argument**

None

**Code Sample Example**

In this example, the current position is set first and saved.

**gsave** and **grestore** are required to be issued in pairs, meaning that you need the equal number of **gsave** and **grestore** commands to run in any given object. However, an uneven number of commands can be located in the code, if some of them are within IF statements, so that when the code is actually run, the pairing occurs correctly in all instances.

**Example**

```
moveto( 1, 1 )

%Position is now 1,1

gsave()

%Save current state

lineto( 2, 1 )

%Position is now 2,1

lineto( 2, 2 )

%Position is now 2,2

grestore()

%Restore previous state: position is now 1,1
```

## Scale (procedure)

Scales the result of all commands that follow the scale() command.

**Syntax**

scale(width, height)

**Argument**

*width*

Measure value specifying the factor by which to scale the width.

*height*

Measure value specifying the factor by which to scale the height.

**Code Sample Example**

```
scale( 2,0.5 )
```

## SetAngle (procedure)

This command rotates all subsequent commands by the angle specified. Most commands and objects that are displayed/printed on the document are affected by the **setangle** command.

**Syntax**

setangle( angle )

**Argument**

*angle*

Measure value specifying the angle of rotation in degrees for all subsequent commands. Use a positive value for counter-clockwise motion and a negative value for clockwise motion. This value is relative to the current angle, not absolute.

**Code Sample Example**

This example illustrates **setangle()**.

**setangle()** sets a rotation for all subsequent commands. Consequently, **setangle** commands are cumulative. Therefore, to restore the previous angle of rotation, you must issue a **setangle** command that negates whatever rotation applied.

**Example**

```
setangle(45.00) %Rotate subsequent commands by 45 degrees

show('Some text') %Display at a 45 degree angle

setangle(45.00) %Rotate subsequent commands by 45 degrees

show('Some text') %Display at a 90 degree angle

setangle(-90) %Cancel all rotations
```

## SetDash(procedure)

Set the properties of a dashed line. Any drawing done after **setdash()** draws dashed instead of solid lines. You can revert to solid lines by calling **setdash()** with an empty array as a parameter.

**Syntax**

setdash( [dashlength, spacelength] )

**Argument**

*dash*

Measure value specifying the length, in inches, of each of the dashes in the dashed line.

*space*

PLANETPRESS. SUITE

Measure value specifying the length, in inches, of the space between each of the dashes in the dashed line.

**Code Sample Examples**

These examples illustrate **setdash()**.

**Example 1**

```
setdash( [0.5,0.3] )
```

**Example 2**

```
setdash( [] )
```

## SetFillColor (procedure)

Sets the colour used to paint the insides of any closed shapes when issuing a **fill** or a **strokeandfill** command.

**Syntax**

setfillcolor( colour )

**Argument**

*colour*

Colour array that can be expressed expressed using one, three or four values ranging from 0 to 255. For Grayscale, a single value is used in the form [K]. For the RGB model, three values are used in the form [R, G, B]. For the CMYK model, four values are used in the form [C, M, Y, K].

**Code Sample Example**

This example draws three filled rectangles.

**Example**

```
setfillcolor([255]) %Sets fill colour to black using the Grayscale model

rectfill(1, 1, 1, 1) %Draws a black square

setfillcolor([125,0,0]) %Sets fill colour to medium red using the RGB model

rectfill(1, 1, 1, 1) %Draws a medium red square

setfillcolor([0,0,25,0]) %Sets fill colour to light yellow using the CMYK model

rectfill(0, 0, 1, 1) %Draws a light yellow square
```

## SetLineWidth (procedure)

Sets the width of the pen used for drawing lines, boxes and other objects. When setting the line width, remember that the exact location of the pen is on the middle of the line, with the stroke spilling over equally on both sides.

**Syntax**

setlinewidth( width )

**Argument**

*width*[SetLineWidth (procedure)](#)

Measure value setting the width, in inches, of the drawing pen.

**Code Sample Examples**

**Example 1**

This example sets the pen width to 1/4 inch then draws a 2"x2" rectangle using that pen.

```
setlinewidth(0.25)
rectstroke(0,0,2,2)
```

## SetStrokeColor (procedure)

Sets the colour of the pen used for drawing lines, boxes and other objects.

**Syntax**

setstrokecolor( colour )

**Argument**

*colour*

Colour array that can be expressed expressed using one, three or four values ranging from 0 to 255. For Grayscale, a single value is used in the form [K]. For the RGB model, three values are used in the form [R, G, B]. For the CMYK model, four values are used in the form [C, M, Y, K].

**Code Sample Example**

This example draws two empty rectangles of different colors.

**Example**

```
setstrokecolor([0,0,100]) %Sets stroke color to medium blue using the RGB model
rectstroke(0,0,1,1) %Draws a medium blue rectangle
setstrokecolor([0,0,255,0]) %Sets stroke color to bright yellow using the CMYK model
rectstroke(1,1,1,1) %Draws a bright yellow rectangle
```

## Translate (procedure)

Offsets an object's position from its original spot on the page. The point of origin of any object is (0,0) by default. All commands within an object are relative to that point of origin. Moving, or translating the point of origin allows you to reset the point

of origin within the boundaries of the object.

**Syntax**

translate( x, y )

**Arguments**

*x, y*

Measure values representing the horizontal/vertical offset, in inches, of the new point of origin within the object. These values are relative to the current point of origin, not absolute.

**Code Sample Examples**

**Example 1**

This example moves the point of origin 1 inch right and down from the original point.

```
translate(1,1)
show('some text')
```

**Example 2**

This example moves the point of origin 1 inch right and down from the current point.

```
translate(1,1)
show('some more text')
```

**Example 3**

This example moves the point of origin back to its original position. **Translate** sets the new point of origin for all subsequent commands. Consequently, **translate** commands are cumulative. Therefore, to restore the previous point of origin, you must issue a **translate** command that negates whatever offset was previously applied.

```
translate(-2,-2)
```

# Path

# Arc and ArcN (procedures)

Arc() draws an arc in a counter-clockwise direction, while ArcN() draws an arc in a clockwise direction. If there is a current point set, the command draws a straight line from the current point to the start point of the arc. Whether or not a current point is set when the command executes, after execution the current point is the end point of the arc.

You define the arc you want to draw by specifying the x and y coordinates of the center of the circle, the radius of the circle, and the start and end points for the arc. You specify each of the start and end points of the arc as an angle; the command uses the angle to position the point. For example, to position the start point, the command draws an invisible line at the specified start angle from the center of the circle, the length of the radius; it positions the start point at the end of that line.



Example using Arc()                                    Example using ArcN().

Example of an arc drawn from start point D to end point F: A. Center of circle (x,y) B. Radius C. Start angle D. Start point of arc E. End angle F.

**Syntax**

arc( x, y, arc_length, start_angle, end_angle )

**Arguments**

*x, y*

Measure values representing the x and y coordinates respectively of the center of the circle.

*arc_length*

Measure value representing the length of the arc.

*start_angle*

Measure value representing the start angle, in degrees, for the arc. Degrees are relative to a standard X, Y axis, with a value of 0 degrees lying flush with the X axis. The value of the start angle can be either positive or negative.

*end_angle*

Measure value representing the end angle, in degrees, for the arc. Degrees are relative to a standard X, Y axis, with a value of 0 degrees lying flush with the X axis. The value of the end angle can be either positive or negative.

**Code Sample Examples**

**Example 1**

Example 1 draws a 180 degree arc 3 inches in length. The center of the circle is at (0,0). The **rmoveto()** command that precedes the **arc()** command ensures the current point is set at the center of the circle and prevents a line from being drawn from the current point to the start point of the arc.

```
rmoveto(3,0)
arc(0,0,3,0,180)
stroke()
```

**Example 2**

Example 2 draws an arc 2.5 inches in length. The center for the arc is four inches from the left edge of the page and three inches from the top edge. The arc starts at 30 degrees and ends at 60 degrees. The **rmoveto()** command that precedes the **arc()** command moves the current point to the center of the circle, resulting in a line that is drawn from the center of the circle to the start point of the arc.

```
rmoveto(4,3)
arc(4,3,2.5,30,60)
stroke()
```

# ClosePath (procedure)

Closes any open path and completes the current shape, if need be, by drawing a line from the last point in the shape to the starting point. This ensures the shape is closed, thus enabling filling to take place if specified. Only closed shapes can be filled.

**Syntax**

closepath()

**Argument**

None

**Code Sample Example**

The first line of code sets the starting point of a triangle, the second and third lines of code draw the first and second lines (or sides) of the triangle, the fourth line of code closes the triangle shape by implicitly issuing a **moveto** command.

**Example**

```
moveto(1.0,1.0)lineto(1.5,2.0)
lineto(0.5,2.0)
closepath()
fill()
```

# CurveTo/RCurveTo (procedure)

Creates a Bezier curve. Bezier curves are shapes defined using 4 points: the starting and ending points are physical positioning points while the second and third points are control points on the curve. In PlanetPress Talk, the starting point is implicit

and automatically set to the current cursor position. The first control point defines the direction the curve takes when moving from the starting point, and before shifting to the second control point, which defines a second curve before reaching the ending point.

A complete explanation of Bezier curves is beyond the scope of this document. Feel free, however, to try out this command using the PlanetPress Talk Editor in PlanetPress in order to study its possibilities.

To draw a simple curve, use the same (x,y) coordinates for both control points.

**Syntax**

curveto( x1, y1, x2, y2, x3, y3 )

**Arguments**

*x1, y1, x2, y2, x3, y3*

Pairs of measure values representing the horizontal/vertical position, in inches, of each point defining the Bezier curve. When using rcurveto, these values are relative to the current point of origin. When using curveto, they are absolute values.

**Code Sample Example**

This example draws a triangle with a twisted bottom side. The triangle has a blue outline and is filled with yellow.

**Example**

```
setstrokecolor([100,100,0,0])
setfillcolor([0,0,100,0])
moveto(1,1)
lineto(1.5,2)
curveto(1,1,1,3,0.5,2)
closepath()
strokeandfill()
```

**Result:**



# Fill (procedure)

Fills the current closed shape, using the colour specified with the **setfillcolor** command. Only the inside of the shape is filled, not its outline. To fill and outline the shape, use **strokeandfill** command instead. If you only want to outline the shape, use

**stroke**. Only closed shapes can be filled.

**Syntax**

fill()

**Argument**

None

**Code Sample Example**

This example draws a yellow filled triangle.

**Example**

```
setfillcolor([0,0,100,0])
moveto(1,1)
lineto(1.5,2)
lineto(0,2)
closepath()
fill()
```

## LineTo/RLineTo (procedure)

Draws a line starting from the current point up to the specified coordinates.

**Syntax**

lineto( x, y )

**Arguments**

*x, y*

Measure values representing the horizontal/vertical position, in inches, of the ending point of the line. When using **rlineto**, these values are relative to the current point of origin. When using **lineto**, they are absolute values.

**Code Sample Example**

This example draws an empty triangle with a blue outline.

**Example**

```
setstrokecolor([100,100,0,0])   %Set pen colour to blue

moveto(1,1) %Set starting point

lineto(1.5,2) %Draw first line

lineto(0.5,2) %Draw second line

closepath() %Close shape

stroke() %Draw shape
```

## MoveTo/RMoveTo (procedure)

Moves the current point to the specified coordinates.

**Syntax**

moveto( x, y )

**Arguments**

*x, y*

Measure values representing the new horizontal/vertical position, in inches, of the current point. In **rmoveto**, these values are relative to the current point of origin. In **moveto**, they are absolute values.

**Code Sample Example**

This example displays a triangle within a rectangle.

**Example**

```
rectstroke(.5,.5,2,2) %Draw rectangle

moveto(1,1) %Reset current point

lineto(1.5,2) %Draw first line

lineto(0.5,2) %Draw second line

closepath() %Close shape

stroke() %Draw shape
```

## Pie (procedure)

Creates a pie slice shape, which can then be rendered using **stroke**, **fill**, or **strokeandfill**.

**Syntax**

pie( x, y, radius, startangle, endangle )

**Arguments**

*x, y*

Measure values representing the horizontal/vertical position, in inches, of a circle's center representing the whole pie from which to draw a slice.

*radius*

Measure value representing the length, in inches, of the segments of the pie, i.e. its radius.

*startangle, endangle*

Measure values representing the angles, in degrees, of both segments of the pie slice. Values increase counter clock wise, with 0 extending right from the center of the circle.

**Code Sample Example**

This example draws a whole pie chart with a shadow effect.

**Example**

```
setfillcolor([0,0,0,50]) %Set shadow colour to gray

pie(1.5,1.5,1,60,290) %Create shadow for first slice

translate(0.2,0.2) %Separate shadows

pie(1.5,1.5,1,290,420) %Create shadow for second slice

fill() %Display shadows

translate(-0.3, 0.3) %Offset chart from its shadow

setstrokecolor([0,0,100,0]) %Set pen colour to yellow

setfillcolor([0,100,100,0]) %Set fill colour to red

pie(1.5,1.5,1,60,290) %Create first slice

strokeandfill() %Draw first slice

translate(0.2,0.2) %Separate slices

setfillcolor([100,100,0,0]) %Set fill colour to blue

pie(1.5,1.5,1,290,420) %Create second slice

strokeandfill() %Draw second slice
```

## Rectangle (procedure)

Creates and draws a rectangle shape. The rectangle can optionally be filled using the colour specified with the **setfillcolor** command. The border can optionally be drawn using the colour specified with the **setstrokecolor** command.

**Syntax**

rectangle( x, y, x1, y1, filled, stroked )

**Arguments**

*x, y*

Measure values representing the horizontal/vertical position, in inches, of the starting point.

*x1, y1*

Measure values representing the horizontal/vertical position, in inches, of the bottom right corner of the rectangle.

*filled*

Boolean value specifying whether the shape should be filled using the colour specified with the setfillcolor command.

*stroked*

Boolean value specifying whether the shape should be outlined using the colour specified with the setstrokecolor command.

**Code Sample Example**

This example draws a yellow rectangle with a black border.

**Example**

```
setstrokecolor([0,0,0,100])

setfillcolor([0,0,50,0])

rectangle(0,0,3,3,true,true)
```

# RectFill (procedure)

Creates and draws a filled rectangle shape. The colour used to fill the shape can be specified using the **setfillcolor** command.

**Syntax**

rectfill( x, y, width, height )

**Arguments**

*x, y*

Measure values representing the horizontal/vertical position, in inches, of the starting point.

*width, height*

Measure values representing the width/height, in inches, of the rectangle shape.

**Code Sample Example**

This example draws two identical rectangles.

To draw a rectangle that is both stroked and filled, use **rectstroke()** and **rectfill()** in succession, or **rectfillstroke()**.

**Example**

```
setfillcolor([0,100,100,0])

%Draw first rectangle using rectfill with red

rectfill(1,1,2,2)

%Offset starting position slightly to distinguish

%between each rectangle and fill second one with blue.

translate(.2,.2)

setfillcolor([100,100,0,0])

%Draw second rectangle using conventional commands

moveto(1,1) %Reset current point
```

PLANETPRESS. SUITE

```
lineto(3,1) %Draw first line

lineto(3,3) %Draw second line

lineto(1,3) %Draw third line

closepath() %Close shape

fill() %Draw shape
```

## RectFillStroke (procedure)

Creates a rectangle that has both an outline and a fill color. You set the color for the outline using setstrokecolor and the fill color using setfillcolor.

**Syntax**

rectfillstroke( x, y, width, height )

**Arguments**

*x,y*

Measure values representing the x and y coordinates respectively of the top left corner of the rectangle.

*width, height*

Measure values specifying the width and height respectively of the rectangle.

## RectStroke (procedure)

Creates and draws an empty rectangle shape. The colour of the pen used to draw the rectangle can be set using **setstrokecolor**.

**Syntax**

rectstroke( x, y, width, height )

**Arguments**

*x, y*

Measure values representing the horizontal/vertical position, in inches, of the starting point.

*width, height*

Measure values representing the width/height, in inches, of the rectangle shape.

**Code Sample Example**

This example draws two identical rectangles.

To draw a rectangle that is both stroked and filled, use **rectstroke** and **rectfill** in succession.

**Example**

```
rectstroke(1,1,2,2) %Draw first rectangle
```

```
%Offset starting position slightly to distinguish

%between each rectangle; the second rectangle is blue.

translate(.2,.2)

setstrokecolor([100,100,0,0])

%Draw second rectangle using conventional commands

moveto(1,1) %Reset current point

lineto(3,1) %Draw first line

lineto(3,3) %Draw second line

lineto(1,3) %Draw third line

closepath() %Close shape

stroke() %Draw shape
```

## Stroke (procedure)

Outlines the current shape using the pen colour and width specified with the **setstrokecolor** and **setlinewidth** command. The shape is not filled. To fill and outline the shape, use **strokeandfill** instead. If you only want to fill the shape, use **fill**.

**Syntax**

stroke()

**Argument**

None

**Code Sample Example**

This example draws a blue triangle with a blue border.

**Example**

```
setstrokecolor([100,100,0,0]) %Set pen colour to blue

moveto(1,1)

lineto(1.5,2)

lineto(0,2)

closepath()

stroke()
```

## StrokeAndFill (procedure)

Outlines the current shape using the pen colour and width specified with the **setstrokecolor** and **setlinewidth** command. The shape is then filled using the colour specified with the **setfillcolor** command. To simply outline the shape, use **stroke** instead. If you only want to fill the shape, use **fill**.

**Syntax**

strokeandfill()

**Argument**

None

**Code Sample Example**

This example draws a yellow triangle with a blue outline.

**Example**

```
setstrokecolor([100,100,0,0])

setfillcolor([0,0,100,0])

moveto(1,1)

lineto(1.5,2)

lineto(0,2)

closepath()

strokeandfill()
```

## Paragraphs and Text

## BeginParagraph ... EndParagraph (procedure)

Delimits a paragraph of formatted text.

The **beginparagraph** ... **endparagraph** structure can contain only the following commands: **setstyle()**, **setstyleext()**, **show()**, and **%** (indicating a commented line). Any other command included within this structure will yield unpredictable results. A **margin()** command should precede the **beginparagraph...endparagraph** structure, and each variable you reference within the structure must have its own procedure., as demonstrated in the second of the examples below. Also note that the first command within a **beginparagraph...endparagraph** structure must be the **setstyle()** command.

No **crlf** procedures are permitted inside a paragraph structure, since **crlfs** are paragraph delimiters.

If you do not need to change fonts or use variables within a paragraph, it is strongly recommended you use the text object in PlanetPress, as PlanetPress optimizes text objects to improve performance.

When a long string containing no spaces between its characters appears in the context of a **show()** command and the **begin-paragraph** ... **endparagraph()** command, the line of text will not wrap. The same is true for a text object; no wrapping occurs when a long string has no spaces between its characters.

Note that when a Text object is converted to PlanetPress Talk, an extra argument is is added to the **beginparagraph** ... **end-paragraph** syntax. Ignore this argument since it is strictly for internal use.

**Syntax**

beginparagraph( lmargin, rmargin, firstindent, align, leading,[opt, forcedwordwrap])

...

endparagraph()

**Arguments**

*lmargin*

Measure value specifying the left margin, in inches, of the text relative to the left border of the object.

*rmargin*

Measure value specifying the right margin, in inches, of the text relative to the left border of the object.

*firstindent*

Measure value specifying the indent, in inches, of the first line of the paragraph, relative to the lmargin parameter.

*align*

String value specifying the text alignment within the paragraph. Possible values are 'left', 'right', 'center' and 'leftright' (for both left and right justification of text).

*leading*

Measure value specifying the amount of vertical space, in inches, between each line of the paragraph.

*opt*

Optional boolean value used internally.  Default value is false.

*forcedwordwrap*

Optional boolean value that forces wordwrap at the defined paragraph size.  Default value is false.

**Code Sample Examples**

Example that displays a paragraph aligned to the right.

**Example 1**

```
beginparagraph(1,3,0,'right',0.16)
      setstyle(&Default)
      show('This long line of text should wrap around')
      show('This long line of text should wrap around')
endparagraph()
```

Example that shows how to include a variable within a formatted paragraph.

PLANETPRESS. SUITE

**Example 2**

```
define(&var,string,'very')
beginparagraph(1,3,0,'left',0.16)
       setstyle(&Default)
       show('This ')
       show(&var)
       show(' long line of text should wrap around')
endparagraph()
```

## BeginUTF8Paragraph ... EndUTF8Paragraph (procedure)

Delimits a paragraph of formatted UTF8 text.

This procedure is based on BeginParagraph ... EndParagraph (procedure), with the following differences:

- ShowUTF8 (procedure) must be used instead of show()
- SetStyleExt (procedure) must use a style that has been associated with the True Type Font "TT Host UTF8Arabic".
- Only UTF8 text can be displayed in a PlanetPress Talk object defined using the **beginUTF8paragraph** ... **endUTF8paragraph** structure. Non-UTF8 static text should therefore be converted to its UTF8 reference using the Ansi/UTF8 Converter (place the cursor within the string and press CTRL+N). Variable text should be converted using MapUTF8 (function).

### Syntax

beginUTF8paragraph( lmargin, rmargin, firstindent, align, leading, majormode, [forcedwordwrap] )

% Your Paragraph data here

endUTF8paragraph()

### Arguments

Same as BeginParagraph ... EndParagraph (procedure).

### Code Sample Example

This example displays a justified paragraph with the UTF8 text running from right to left.

### Example

```
setlinewidth(0.007)moveto(0,0)
rlineto(&width,0)
rlineto(0,&height)
rlineto(neg(&width),0)
rlineto(0,neg(&height))
closepath()
stroke()
moveto(&width/2,&height/18)
BeginUTF8Paragraph(0.0000,&width,0.0000,'leftright',0.1667,'rtl')
       SetStyleExt(&Style1,12.0000,0,[100],100)
       ShowUTF8('\u0623\u0633\u0627\u0633\u064B\u0627\u060C')
EndUTF8Paragraph()
```

## CRLF (procedure)

Moves the current point by simulating a carriage return/line feed combination. The horizontal position is reset to the value of the current left margin, while the vertical position is offset by the leading value specified.

### Syntax

crlf( [leading] )

### Argument

*leading*

Measure value setting the vertical distance, in inches, between two lines. If you do not provide this argument, the value by default is 0.167 inches, which is roughly equivalent to a 6 LPI (lines per inch) setting.

### Code Sample Example

This example demonstrates how to change the vertical spacing of lines within an object.

### Example

```
margin(1,1)
show('These lines are displayed')
crlf(0.125)
show('at 8 LPI.')
crlf(0.5)
show('While these lines are displayed')
crlf(0.5)
show('at 2 LPI')
```

## Margin (procedure)

Offsets text within a series of commands, relative to the original top left position of the object. Margin also sets the horizontal position to be used when **crlf** commands are encountered. All text-rendering functions that make use of margins use this setting.

### Syntax

margin( x, y )

### Arguments

*x, y*

Measure values representing the horizontal/vertical position, in inches, of the starting point for the text within an object.

### Code Sample Example

This example displays shadowed text by repeating the same text commands twice, using styles of different colours and slightly offsetting the margin.

### Example

```
margin(1,1) %Set margins to ( 1,1 )

setstyle(&blackfont) %Blackfont must already exist
```

```
show('This is shadowed text')

margin(.98,.98) %Offset margins slightly

setstyle(&bluefont) %Bluefont must already exist

show('This is shadowed text')
```

## ShowLeftRight (procedure)

Displays a string of characters, in inches, using a specified width.

**Syntax**

showleftright( text, width )

**Arguments**

*text*

String value to be displayed.

*width*

Measure value representing the amount of horizontal space to use for displaying the string. showleftright adjusts the amount of spaces between characters to insure the string fits exactly within the specified width.

**Code Sample Example**

This example shows how to fit strings of different sizes within the same box that has a width of 3 inches.

**Example**

```
margin(0,0)

rectstroke(0,0,3,3)

moveto(0,0.5)

showleftright('This fits well inside the box',3)

crlf(0.1599)

showleftright('But this one is a little more tight',3)
```

## Show / ShowCenter / ShowRight (procedure)

These three commands are used to display simple text on the document. They are identical in all aspects, except in the way they behave relative to the current margin settings: **show** uses the margin as a starting point to extend the string to the right; **showright** uses the margin as the ending point of the string; and **showcenter** extends the string equally on both sides of the margin. These commands are equivalent to *print*, *echo* or *write* commands in other scripting languages.

**Syntax**

show( text )

showcenter( text )

showright( text )

**Argument**

*text*

String value to be displayed.

**Code Sample Example**

This example illustrates the difference between all three variations of using the SHOW command.

**Example**

```
margin(3,1)
showright('This line is to the left of the margin')
crlf()
showcenter('This line is centered on the margin')
crlf()
show('This line is to the right of the margin')
```

## Styles

### SetStyle (procedure)

Sets the style to be used for all subsequent commands. The specified style must already exist in the document.

**SetStyle()** supercedes the **SetFont()** command. Although PlanetPress Talk still recognizes **SetFont()**, this behavior is unlikely to continue in future versions, and it is therefore highly recommended that you use the **SetStyle()** command in your scripts.

**Syntax**

setstyle( stylename )

**Argument**

*stylename*

Name of the style to use for all subsequent commands. The name of the style must be preceded by an ampersand. You can reference the bold, italic, or underlined versions of a font by appending the appropriate letter or letters to the name of the style. Append *.b* for the bold version, *.i* for the italic version, *.bi* for the bold italic version, and *.u* for the underlined version.

**Code Sample Example**

This example displays a line of text using two different existing styles called Blackfont and Bluefont.

**Example**

```
margin(1,1)
setstyle(&blackfont)
show('The word ')
setstyle(&bluefont.i)
show('blue')
setstyle(&blackfont)
show(' is now in blue italic')
```

## SetStyleExt (procedure)

Select an existing style and set the size, color, style property, or font ratio for its font. This eliminates the need to create several styles that use the same font. **setstyleext**() does not support bold, italic, or underline styles with double-byte character sets.

**Syntax**

setstyleext(stylename, fontsize, styleproperty, fontcolor, fontratio)

**Arguments**

*stylename*

The name of the style. The name of the style must be preceded by an ampersand. You can set the bold, italic, or bold italic property of the style by appending .b, .i, or .bi respectively to the name of the style.

> The ability to append .b, .i, or .bi to style names that have double-byte character sets is not supported.

*fontsize*

Measure value representing the point size for the font. A value of -1 use the default font size of the style specified by stylename.

*styleproperty*

Integer value representing the style property for the font:

- 0: Normal
- 1: Underline
- 2: Outline

*fontcolor*

Colour array that can be expressed using one, three or four values ranging from 0 to 255. For Grayscale, a single value between 0 and 100 is used in the form [K], representing percentage of gray (100 is black, 0 is white). For the RGB model, three values are used in the form [R, G, B]. For the CMYK model, four values are used in the form [C, M, Y, K].

*fontratio*

Integer value representing the percentage by which to shrink or stretch the font spacing. This value adjusts both the width of each glyph and the spacing between glyphs. This is in contrast to kerning, which modifies the spacing between characters without modifying the width of characters.

Example

```
setstyleext(&Style2.bi, 25, [100], 100)
```

## Object $name()... EndObject (procedure)

Creates an object. You can reference the object you create using the $ operator. See "$element (procedure)" (page 611).

This command can only be used from within the PressTalk Before and PressTalk After of a PlanetPress Design document, and cannot exist within a page's properties or the PressTalk of any object, including

**Syntax**

```
object $name( top, left, width, height, condition, angle[, setsnappingpoint[, snap-
toprevious]] )
 %Object Contents
endobject()
```

**Argument**

*$name* (PressTalk Name)

Name to use for the object, preceded by the dollar sign character ($). The name must conform to the rules for names described in "Names" (page 219).

*top* (Measure)

Measure value specifying the distance, in inches, to offset the top edge of the picture object, from the top edge of the document page. If you set the snaptoprevious argument to a value other than 0, the value you set for the top argument becomes the vertical offset for the Snap to previous snapping point.

*left* (Measure)

Measure value specifying the distance, in inches, to offset the left edge of the picture object, from the left edge of the document page. If you set the snaptoprevious argument to a value other than 0, the value you set for the left argument becomes the horizontal offset for the Snap to previous snapping point.

*width, height* (Measure)

Measure values specifying the width and height respectively, in inches, of the object.

*condition* (Boolean)

Boolean value, or PlanetPress Talk expression that resolves to a Boolean value, specifying a condition on the object.

*angle* (Measure)

Measure value specifying the angle of rotation, in degrees. The pivot point for the rotation is the bottom left corner of the object.

*setsnappingpoint* (optional, Boolean)

Boolean value (True or False) specifying whether the object has its Set snapping point property set. If you set this argument to True, the last line of the code for the object must be a **moveto()** command that moves the current point to the position at which you want to set the object's snapping point.

*snaptoprevious* (optional, Integer)

Integer value specifying whether the object has its Snap to previous property set, and if so, the position of that snapping point. If you set the Snap to previous property, you can use the top and left arguments to specify, respectively, a vertical and horizontal offset for the snapping point.

- 0 = do not set the Snap to previous snapping point
- 1 = top left
- 2 = top middle
- 3= top right
- 4= middle left
- 5 = middle middle
- 6= middle right
- 7 = bottom left
- 8 = bottom middle
- 9 = bottom right

**Code Sample Example**

The following creates a rectangle object (2 inches wide by 3 inches high), rotates it 45 degrees, and sets its Snap to previous snapping point at the top left of the object with a vertical offset of 1 inch and a horizontal offset of 1.5 inches.

```
object $Box2(1.0,1.5,2.0,3.0,true,45.0,false,1)
  setstyle(&Style1)
  MoveTo(0,0)
  SetLineWidth(0.0070)
  SetStrokeColor([0,0,0,100])
  SetFillColor([0,0,0,0])
  LineTo(&width,0.0)
  LineTo(&width,&height)
  LineTo(0.0,&height)
  LineTo(0.0,0.0)
  ClosePath()
  Stroke()
endobject()
```

## Bar Codes

## ShowBarCode (procedure)

**Note: This PressTalk command is deprecated.  Use instead the barcode-specific commands.**

Displays a specified string as a barcode.

**Syntax**

showbarcode( string )

**Arguments**

*string*

Value of type string to display as a barcode.

showbarcode needs 3 additional parameters before it can display the barcode. These parameters must be set before calling **showbarcode**, using the following variables:

*BarWidth*

Measure value setting the width of each bar, in inches.

*BarHeight*

Measure value setting the height of each bar, in inches.

*BarType*

Integer value specifying the type of barcode to print. Current possible values include:

| BarType: | Code: |
|---|---|
| 0 | Code 39 |
| 1 | Code 128 |
| 2 | UPC_A |
| 3 | UPC_E |
| 4 | UPC/EAN 8 |
| 5 | UPC/EAN 13 |
| 6 | PostNet |
| 7 | PDF-417 |

### Code Sample Example

This example prints a Code 128 barcode.

The **BarWidth**, **BarHeight** and **BarType** variables must be created exactly as shown here, with the same case changes in the variable names.

### Example

```
define(&BarWidth,measure,0.012)

define(&BarHeight,measure,1.000)

define(&BarType,integer,1)

%Set bar color to black and display barcode

setfillcolor([100,100,100,100])

showbarcode('Bar 128')
```

## ShowBarcode2of5(procedure)

Displays a specified string as a Standard 2 of 5 barcode.

### Syntax

showbarcode2of5( str, barwidth, readable, bearers)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

*readable*

Boolean that enables display of the human readable portion of the barcode

*bearers*

Boolean that enables bearer bars surrounding the barcode

**Code Sample Example**

```
showbarcode2of5('123456789',12,true,true)
```

# ShowBarcodeAustPost (procedure)

Displays a specified string as an Australia Post barcode.

**Syntax**

showbarcodeaustpost( str)

**Arguments**

*str*

Value of type string to display as a barcode.

**Code Sample Example**

```
showbarcodeaustpost('56439111ABA 9')
```

# ShowBarcodeAztec (procedure)

Displays a specified string as an Aztec barcode.

**Syntax**

showbarcodeaztec( str, mode, errorcorrection, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*mode*

Mode to use for the barcode.

Value Mode
0       Normal
1       Compact
2       Full Range

*errorcorrection*

Error correction level to use in the barcode.

Mode Error Correction Range
0       0-99
1       0-4
2       0-32

*barwidth*

Measure value of the width a a single barcode bar, in mils (0.001 inch).

## Code Sample Example

```
showbarcodeaztec('A 12345 B 434343',5,35,24)
```

## ShowBarcodeCodabar (procedure)

Displays a specified string as a Codabar barcode.

### Syntax

showbarcodecodabar( str, start, stop, barwidth, readable)

### Arguments

*str*

Value of type string to display as a barcode.

*Start*

The Start character of the barcode

A
B
C
D

*Stop*

The Stop character of the barcode

A
B
C
D

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

*readable*

Boolean that enables display of the human readable portion of the barcode

**Code Sample Example**

```
showbarcodecodabar('12345','A','A',12,true)
```

# ShowBarcodeCodablockF (procedure)

Displays a specified string as a Codablock F barcode.

## Syntax

showbarcodecodablockf( str, barwidth)

## Arguments

*str*

Value of type string to display as a barcode.

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

**Code Sample Example**

```
showbarcodecodablockf('12345',12)
```

# ShowBarcodeCode11 (procedure)

Displays a specified string as a Code 11 barcode.

## Syntax

showbarcodecode11( str, barwidth, checksum, readable, readchecksum)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

*Checksum*

Boolean allows the automatic calculation of the barcode checksum

*readable*

Boolean that enables display of the human readable portion of the barcode

*readchecksum*

Boolean that enables display of the checksum in the human readable portion of the barcode

**Code Sample Example**

```
showbarcodecode11('123456789',12,true,true,false)
```

# ShowBarcodeCode128 (procedure)

Displays a specified string as a Code 128 barcode.

**Syntax**

showbarcodecode128( str, subset, barwidth, readable)

**Arguments**

*str*

Value of type string to display as a barcode.

*subset*

Integer specifying the subset of the barcode to use.

| Value | Subset |
|-------|--------|
| 0 | A |
| 1 | B |
| 2 | C |

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

*readable*

Boolean that enables display of the human readable portion of the barcode

**Code Sample Example**

```
showbarcodecode128('123456789',0,12,true)
```

# ShowBarcodeCode16k (procedure)

Displays a specified string as a Code 16k barcode.

**Syntax**

showbarcodecode16k( str, mode, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*Mode*

Mode (0-7) of the Code 16k barcode

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

**Code Sample Example**

```
showbarcodecode16k('12345',4,12)
```

# ShowBarcodeCode39 (procedure)

Displays a specified string as a Code 3 of 9 barcode.

**Syntax**

showbarcodecode39( str, barwidth, checksum, readable, readchecksum)

**Arguments**

*str*

Value of type string to display as a barcode.

*BarWidth*

Measure value setting the width of each bar, in mils (0.001 inch).

*Checksum*

Boolean value that indicates if the checksum should automatically be calculated.

*Readable*

Boolean value that indicates if the human readable should be shown

*ReadChecksum*

Boolean value that indicates if the checksum value should be included in the human readable

**Code Sample Example**

```
showbarcodecode39('123456',12,true,true,false)
```

# ShowBarcodeCode49 (procedure)

Displays a specified string as a Code 49 barcode.

**Syntax**

showbarcodecode49( str, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

**Code Sample Example**

```
showbarcodecode49('12345', 12)
```

# ShowBarcodeCode93 (procedure)

Displays a specified string as a Code 93 barcode.

**Syntax**

showbarcodecode93( str, barwidth, readable)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

*readable*

Boolean that enables display of the human readable portion of the barcode

**Code Sample Example**

```
showbarcodecode93('123456789',12,true)
```

# ShowBarcodeDatamatrix (procedure)

Displays a specified string as a Datamatrix barcode.

**Syntax**

showbarcodedatamatrix( str, mode, errorcorrection, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*Mode*

Integer value of the Datamatrix Mode to use.

| Value | Mode |
|-------|------|
| 0 | Square |
| 1 | Rectangular |

*errorCorrection*

Error correction level to use with the barcode.

| Mode | Allowed Range |
|------|---------------|
| Square | 1-24 |
| Rectangular | 1-6 |

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

**Code Sample Example**

```
showbarcodedatamatrix('12345', 0, 5, 12)
```

# ShowBarcodeEAN8 (procedure)

Displays a specified string as a EAN-8 barcode.

**Syntax**

showbarcodeean8( str, subset,[readable, barwidth] )

**Arguments**

*str*

Value of type string to display as a barcode.

*Subset*

The subset of EAN-8 to use

 0 EAN-8
 1 EAN-8 ext.2
 2 EAN-8 ext.5

*readable*

Optional boolean that enables display of the human readable portion of the barcode. Default = True.

*barwidth*

Optional width of one barcode bar, in mils (0.001 inch). Default = 12mils.

**Code Sample Example**

```
showbarcodeean8('0133558',0)
```

# ShowBarcodeEAN13 (procedure)

Displays a specified string as a EAN-13 barcode.

**Syntax**

showbarcodeean13( str, subset,[readable, barwidth] )

**Arguments**

*str*

Value of type string to display as a barcode.

*Subset*

The subset of EAN-13 to use

 0 EAN-13
 1 EAN-13 ext.2
 2 EAN-13 ext.5

*readable*

Optional boolean that enables display of the human readable portion of the barcode. Default = True.

*barwidth*

Optional width of one barcode bar, in mils (0.001 inch). Default = 12mils.

**Code Sample Example**

```
showbarcodeean13('977147396801',0)
```

# ShowBarcodeFIM (procedure)

Displays a specified string as a FIM barcode.

**Syntax**

showbarcodefim( str)

**Arguments**

*str*

Value of type string to display as a barcode.

**Code Sample Example**

```
showbarcodefim('A')
```

# ShowBarcodeI2of5 (procedure)

Displays a specified string as an Interleaved 2 of 5 barcode.

**Syntax**

showbarcodei2of5( str, barwidth, checksum, readable, readchecksum, bearers)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Width of one barcode bar, in mils (0.001 inch).

*Checksum*

Boolean to add checksum automatically to barcode

*readable*

Boolean that enables display of the human readable portion of the barcode

*readchecksum*

Boolean that enables display of the checksum in the human readable portion of the barcode

*bearers*

Boolean that enables bearer bars surrounding the barcode

**Code Sample Example**

```
showbarcodei2of5('123456789',12,true,true,false,true)
```

# ShowBarcodeISBN (procedure)

Displays a specified string as an ISBN barcode.

**Syntax**

showbarcodeisbn( str, subset,[readable, barwidth] )

**Arguments**

*str*

Value of type string to display as a barcode.

*Subset*

The subset of ISBN to use

 0 ISBN
 1 ISBN ext.2
 2 ISBN ext.5

*readable*

Optional boolean that enables display of the human readable portion of the barcode. Default = True.

*barwidth*

Optional width of one barcode bar, in mils (0.001 inch). Default = 12mils.

**Code Sample Example**

```
showbarcodeisbn('978-1-86074-271-2-54495',2)
```

# ShowBarcodeJapanpost (procedure)

Displays a specified string as a Japan Post barcode.

**Syntax**

showbarcodejapanpost( str, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*BarWidth*

Measure value setting the width of each bar, in mils (0.001 inch).

**Code Sample Example**

```
showbarcodejapanpost('6540123789-A-K-Z',12)
```

# ShowBarcodeMaxicode (procedure)

Displays a specified string as a Maxicode barcode.

**Syntax**

showbarcodemaxicode( str, mode)

**Arguments**

*str*

Value of type string to display as a barcode.

*Mode*

Integer value (2-6) of the Maxicode Mode to use.

**Code Sample Example**

```
show-
bar-
codemaxicode('^059^042^041^059^040<RS>01^02996152382802^029840^029001^0291Z00004951^029UPSN^029(
 ALPHA DR^029PITTSBURGH^029PA^030<ET>',2)
```

# ShowBarcodeMicroPDF (procedure)

Displays a specified string as a Micro PDF-417 barcode.

**Syntax**

showbarcodemicropdf( str, mode, columns, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*Mode*

Integer value of the Micro PDF mode to use.

Value Mode

0    Text

1    Numeric

*columns*

Integer value (1-4) of the number of columns to use.

*barwidth*

Integer value of the barwidth of a single bar in the Micro PDF

**Code Sample Example**

```
showbarcodemicropdf('123456',0,4,12)
```

## ShowBarcodeMicroQR (procedure)

Displays a specified string as a Micro QR Code barcode.

**Syntax**

showbarcodemicroqr( str, mode, errorcorrection, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*Mode*

Integer value of the Micro QR mode to use.

Value Mode

0    Alpha

1    Numeric

*errorcorrection*

Integer value of the error correction level to use.

Value Correction Level

0    L

1    S

2    Q

*barwidth*

Integer value of the barwidth of a single bar in the Micro QR

**Code Sample Example**

```
showbarcodemicroqr('123456',0,2,12)
```

# ShowBarcodeMSI (procedure)

Displays a specified string as a MSI barcode.

**Syntax**

showbarcodemsi( str, barwidth, readable)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Measure of the width of a single barcode bar, in mils (0.001 inch).

*readable*

Boolean value determining if the human readable portion of the barcode is visible.

**Code Sample Example**

```
showbarcodemsi('1234567')
```

# ShowBarcodeOnecode (procedure)

Displays a specified string as a Onecode/IMB barcode.

**Syntax**

showbarcodeonecode( str)

**Arguments**

*str*

Value of type string to display as a barcode.

**Code Sample Example**

```
showbarcodeonecode('01234567094987654321012345678 91')
```

# ShowBarcodePDF417 (procedure)

Displays a specified string as a PDF 417 Code barcode.

## Syntax

showbarcodepdf417( str, mode, columns, truncated, errorcorrection, barwidth)

## Arguments

*str*

Value of type string to display as a barcode.

*Mode*

Integer value of the PDF 417 mode to use.

| Value | Mode |
|-------|------------|
| 0 | Text |
| 1 | ASCII Plus |
| 2 | Numeric |

*columns*

Integer value (1-20) of the number of columns to use.

*truncated*

Boolean value to decide to use the truncated version of the barcode

*errorcorrection*

Integer value (0-8) of the error correction level to use.

*barwidth*

Integer value of the barwidth of a single bar in the PDF 417

## Code Sample Example

```
showbarcodepdf417('123456',0,2,false,5,12)
```

# ShowBarcodePlessey (procedure)

Displays a specified string as a plessey barcode.

**Syntax**

showbarcodeplessey( str, barwidth, readable)

**Arguments**

*str*

Value of type string to display as a barcode.

*barwidth*

Measure of the width of a single barcode bar, in mils (0.001 inch).

*readable*

Boolean value determining if the human readable portion of the barcode is visible.

**Code Sample Example**

```
showbarcodeplessey('1234567')
```


# ShowBarcodePostnet (procedure)

Displays a specified string as a Postnet barcode.

**Syntax**

showbarcodepostnet( str, subset)

**Arguments**

*str*

Value of type string to display as a barcode.

*subset*

Subset of the Postnet barcode to use

| Subset | Zipcode Length |
|--------|----------------|
| 0 | 5 digits |
| 1 | 9 digits |
| 2 | 11 digits |

**Code Sample Example**

```
showbarcodepostnet('12345',0)
```

## ShowBarcodeQRCode (procedure)

Displays a specified string as a QR Code barcode.

### Syntax

showbarcodeqrcode( str, mode, errorcorrection, barwidth)

### Arguments

*str*

Value of type string to display as a barcode.

*Mode*

Integer value of the QR Code mode to use.

| Value | Mode |
|-------|---------|
| 0 | Numeric |
| 1 | Alpha |

*errorcorrection*

Integer value of the error correction level to use.

| Value | Mode |
|-------|------|
| 0 | L |
| 1 | S |
| 2 | Q |
| 3 | H |

*barwidth*

Integer value of the barwidth of a single bar in the QR Code

### Code Sample Example

```
showbarcodeqrcode('123456',0,2,12)
```

## ShowBarcodeRoyalMail (procedure)

Displays a specified string as a Royal Mail barcode.

### Syntax

showbarcoderoyalmail( str)

### Arguments

*str*

Value of type string to display as a barcode.

**Code Sample Example**

```
showbarcoderoyalmail('LE28HS9Z')
```

# ShowBarcodeRSS (procedure)

Displays a specified string as a RSS barcode.

**Syntax**

showbarcoderss( str, mode, barwidth)

**Arguments**

*str*

Value of type string to display as a barcode.

*Mode*

Integer value of the RSS mode to use.

| Value | Mode |
|-------|------|
| 0 | RSS-14 |
| 1 | RSS-Truncated |
| 2 | RSS-Limited |
| 3 | RSS-Stacked |
| 4 | RSS-Stacked Omni |
| 5 | RSS-Expanded |
| 6 | RSS-Expanded Stacked |

*barwidth*

Integer value of the barwidth of a single bar in the RSS Code

**Code Sample Example**

```
showbarcoderss('123456',0,12)
```

# ShowBarcodeUPCA (procedure)

Displays a specified string as a UPC-A barcode.

**Syntax**

showbarcodeupca( str, subset,[readable, barwidth] )

**Arguments**

*str*

Value of type string to display as a barcode.

*Subset*

The subset of UPC-A to use

 0 UPC-A
 1 UPC-A ext.2
 2 UPC-A ext.5

*readable*

Optional boolean that enables display of the human readable portion of the barcode. Default = True.

*barwidth*

Optional width of one barcode bar, in mils (0.001 inch). Default = 12mils.

## Code Sample Example

```
showbarcodeupca('123456789012',0)
```

## ShowBarcodeUPCE (procedure)

Displays a specified string as a UPC-E barcode.

## Syntax

showbarcodeupce( str, subset,[readable, barwidth] )

## Arguments

*str*

Value of type string to display as a barcode.

*Subset*

The subset of UPC-E to use

 0 UPC-E
 1 UPC-E ext.2
 2 UPC-E ext.5

*readable*

Optional boolean that enables display of the human readable portion of the barcode. Default = True.

*barwidth*

Optional width of one barcode bar, in mils (0.001 inch). Default = 12mils.

## Code Sample Example

```
showbarcodeupce('123456',0)
```

## Resources

### InStream... EndInStream (procedure)

Ceate an alias for a resource file (image, data file, attachment, etc.). You can then reference the resource file using the alias instead of the path. This eliminates the need to repeat a path in more than one place in your code, and thus makes your code easier to maintain. For example, if you change the resource, you need only modify the path in the **instream()** command, rather than everywhere you reference the original resource.

You cannot use the **instream()** command inside any function you define using **function @name()**.

**Syntax**

instream external resname

path

endinstream()

If you want to create a reference to a text document (for example a sample data file), replace "external" by "cleartext."

**Argument**

*resname*

String value specifying the alias to use to refer to this resource file.

*path*

String value specifying the path of the resource file.

**Code Sample Example**

This example creates the alias 'photo' that refers to the resource file *c:\images\employees\JAdler.bmp*.

**Example**

```
instream external photo

  c:\images\employees\JAdler.bmp

endinstream()

%You can then reference the resource using the alias

showbitmap('photo',72,1,1)
```

# ShowBitmap (procedure)

Displays a bitmap resource.

## Syntax

showbitmap( resname, resolution, width, height[, transparent[, duotone]] )

## Arguments

*resname*

String value containing either the name of the bitmap resource within the document, or the path to a bitmap file.

*resolution*

Integer value specifying the resolution, in pixels per inch, at which the bitmap displays. A larger number yields a smaller, clearer image, since more pixels are squeezed into a smaller space. But the bitmap then requires more memory. Except in some very specific applications, it is rarely desirable to use resolutions exceeding 100 ppi.

*width, height*

Measure values specifying the width/height, in inches, the bitmap occupies when it displays. Specifying a value of zero for one but not both of the two dimensions scales the bitmap to fit the non-zero dimension. Specifying both a width and a height scales the bitmap to the best fit possible with that width and height, while maintaining its aspect ratio. Specifying both a width and a height of zero displays the bitmap at its default resolution.

*transparent*

Boolean value specifying whether part of the bitmap is transparent. When set to true, the bitmap image is combined with the background; otherwise it is pasted on top.

*duotone*

Colour array specifying the colour to paint all non-white pixels in the bitmap.

## Code Sample Examples

### Example 1

This example displays the bitmap image street_photo at a resolution of 72 pixels per inch, and a width of 1 inch, as a transparent bitmap image with non-white pixels set to blue.

```
showbitmap('street_photo',72,1,0,true,[100,100,0,0])
```

### Example 2

This example displays the bitmap image sunset at a resolution of 300 pixels per inch.

```
showbitmap('sunset',300,0,0)
```

**Example 3**

This example prints either an image, or, if the image cannot be found, the pathname to the image.

```
define( &image_paths, directory, 'c:\\images\\*.JPG' )

define( &i, integer, 1 )

if( resourcetype( &image_paths[&i] ) <> 0 )

  showbitmap( &image_paths[&i], 300, 0, 0)

  crlf( bitmapheight( &image_paths[&i] ) )

elseif()

  show( &image_paths[&i] )

  crlf( 0.16 )

endif()
```

## ShowEPS (procedure)

Displays an EPS image resource.

**Syntax**

showeps( resname, width, height )

**Argument**

*resname*

String value containing either the name of the EPS resource within the document, or the path to an EPS file .

*width, height*

Measure values specifying the width/height, in inches, the EPS occupies when it displays. Specifying a value of zero for one but not both of the two dimensions scales the EPS to fit the non-zero dimension. Specifying both a width and a height scales the EPS to the best fit possible with that width and height, while maintaining its aspect ratio. Specifying both a width and a height of zero displays the EPS at its default resolution.

**Code Sample Example**

This example displays the EPS image 'my_eps', scaling it as necessary to fit a width of 4.5 inches.

**Example**

```
showeps('my_eps', 4.5, 0)
```

## ShowPDF (procedure)

Displays a page of a PDF image resource.

### Syntax

showpdf( resname, pagenum, width, height )

### Argument

*resname*

String value containing either the name of the PDF resource within the document, or the path to a PDF file .

*pagenum*

Integer value specifying the page number within the PDF image resource.

*width, height*

Measure values specifying the width/height, in inches, the PDF page occupies when it displays. Specifying a value of zero for one but not both of the two dimensions scales the PDF page to fit the non-zero dimension. Specifying both a width and a height scales the PDF page to the best fit possible with that width and height, while maintaining its aspect ratio. Specifying both a width and a height of zero displays the PDF page at its default resolution.

### Code Sample Example

This example displays page 5 of the PDF image resource 'new_models', at its default resolution.

### Example

```
showpdf('new_models', 5, 0, 0)
```

## Elements

## $element (procedure)

Executes the specified document element (object, page, resource, etc.). Note that in the case of a page, this procedure executes both the content of the page and the paper handling properties of the page. The height and width of the *calling* page are therefore set to the same values as those set by the *called* page.

Recall that every element in a document has a unique name. You can therefore call any element in the document at any time. For example, the first page of a medical record document may contain a header listing all of the patient information, while other pages in the document scale that same header to fit in the lower left corner of each page. Any modification you subsequently make to the header is automatically reflected throughout the document.

### Syntax

*$element*

### Arguments

*element*

String value specifying the name of the element you want to execute.

**Examples**

Example 1 illustrates basic usage of the syntax.

```
$my_square
$page1
$box5
```

Example 2 scales the *$header* element to one quarter of its original size, and rotates it 90 degrees.

```
scale( 0.25, 0.25 )
setangle( 90 )
$header
```

Example 3 creates a thumbnail of each of the first two pages and positions the thumbnails side-by-side.

```
scale( 0.10, 0.10 )
$page1
translate( 8.5, 0 )
$page2
```

## Emulation, Data File, and Data Pages

## ClearPage (procedure)

Clears the current data page and loads the next one. This command should only be used before reading the next page of data from the input stream. It is strictly for use within a User-Define Emulation and generally follows DoForm (procedure).

**Syntax**

clearpage()

**Argument**

None

Example:

This example the default User-Defined Emulation that corresponds to a Line Printer data file.

```
search(&str,'\014')
      set(&current.line,&current.line + 1)
      store(&current.line,&str)
      doform()
      clearpage()
endsearch()
set(&current.line,&current.line + 1)
store(&current.line,&str)
```

```
if(ge(&current.line,&current.lpp))
        doform()
        clearpage()
endif()
```

## DefineData (procedure)

Change the active data file to the one specified in the *path* parameter.

> Changing the data file in the middle of document processing can have unexpected consequences and should only be used if you understand the implications of doing so.

### Syntax

definedata( path )

### Argument

*path*

String value specifying the path of the data file.

## DoForm (procedure)

Runs the current document. This command is usually issued once a complete data page has been received and committed to buffer. This command should only be used before reading the next page of data from the input stream. It is strictly for use within a User-Define Emulation.

### Syntax

doform()

### Argument

None

### Code Sample Example

This example is an extract from a user-defined emulation. The search() command looks for a formfeed and adds its line number. The line number is stored as a string and the document is run. The data page is cleared and the search is over.

### Example

```
search(&str,'\014')
        set(&current.line,&current.line + 1)
        store(&current.line,&str)
        doform()
        clearpage()
endsearch()
```

# GetNextDataPage(procedure)

Advance to the next data page of the sample data file. Note that this command may have unpredicted behavior if not used correctly in documents that use a user-defined emulation.

This command should only be executed when printermode is not in design mode. The result will be incorrect on screen, but there will be no
unexpected behaviors. Therefore, the only solution is for end-users to make sure the command doesn't get executed while in Design mode by bracketing it within a if(&printermode<>0) ... endif()
structure.

## Syntax

getnextdatapage()

# SetDataPage(procedure)

Specify the data page of the sample data file.

## Syntax

setdatapage( pagenum )

## Argument

*pagenum*

Integer value representing the page number of the data page.

# SetEmulation(procedure)

Sets the emulation to use.

## Syntax

setemulation( emulation )

## Argument

*emulation*

Integer value indicating the type of emulation.

| Value: | Emulation: |
|--------|------------|
| 0 | Line printer |
| 1 | ASCII |
| 2 | Comma Separated Value (CSV) |
| 3 | Channel skip |
| 4 | Database |
| 5 | XML |
| 6 | PDF |

## SetLPP(procedure)

Sets the number of lines per data page. This function is only useful when working with User-Defined Emulation, if you have a method of determining the number of LPP from within your data.

### Syntax

setlpp( lines )

### Argument

*lines*

Integer value specifying the number of lines per page.

## Store (procedure)

Stores a string of characters on a specific line in the current data page.

### Syntax

store( line, string )

### Arguments

*line*

Integer value specifying the line on which to store the string.

*string*

String value to store in the data page.

### Code Sample Example

This example is taken for an OnReadDataPage event for a user defined emulation. It shows how to store lines of text in the current data page. It searches for a formfeed, increases the current line, stores the string, executes the document, resets the page, and then ends the search.

### Example

```
search(&str,'\014')

  set(&current.line,&current.line+1)

  store(&current.line,&str)

  doform()

  clearpage()

endsearch()
```

## Data Destined for PlanetPress Image, PlanetPress Fax and PlanetPress Search

# DefineImageIndex (procedure)

Defines a PlanetPress Search index term. PlanetPress Image uses this information to generate the .PDI file it creates for each PDF file it creates. You use SetImageIndex (procedure) to associate a data value with the index term.

**Syntax**

defineimageindex( name, [length] )

**Arguments**

*name*

String value specifying a name for the index term.

*length*

String value specifying a length for the index term.

**Code Sample Examples**

**Example 1**

This example defines the index term 'PONumber' and assigns it a length of 8 characters. The PDI file generated by PlanetPress Image contains the line: ~IndexName=PONumber,8.

```
defineimageindex( 'PONumber', '8')
```

**Example 2**

This example defines the same index term, this time without defining a length for the term. In this case the line in the PDI file becomes: ~IndexName=PONumber

```
defineimageindex( 'PONumber' )
```

# SetBodyText (procedure)

Defines the text that appears as the body of an email message sent by PlanetPress Image.

**Syntax**

setbodytext( bodytext )

**Arguments**

*bodytext*

String value specifying the text to use as the body of the email message.

# SetEmailAddress (procedure)

Defines an email address for PlanetPress Image.

**Syntax**

setemailaddress( address )

## Arguments

*address*

String value specifying the email address.

# SetEmailSubject (procedure)

Defines the subject line of an email message sent by PlanetPress Image.

## Syntax

setemailsubject( subjectline )

## Arguments

*subjectline*

String value specifying the subject line.

# SetFaxInformation (procedure)

Defines the fax description that appears in both the PlanetPress Fax dialog box and the PlanetPress Fax log file.

## Syntax

setfaxinformation( info )

## Arguments

*info*

String value specifying a description of the fax.

## Code Sample Example

```
setfaxinformation( 'purchase confirmation' )
```

# SetFaxNumber (procedure)

Defines a fax number for PlanetPress Fax.

## Syntax

setfaxnumber( faxnumber )

## Arguments

*faxnumber*

String value specifying the fax number.

## Code Sample Example

```
setfaxnumber( '(514)276-7633' )
```

## SetImageIndex (procedure)

Associates a data value with a PlanetPress Search index term defined using the **DefineImageIndex()** command. PlanetPress Image uses this information when it generates the .PDI file it creates for each PDF file it creates.

>>> This function has no effect when the output of the document is not a PDF file generated through PlanetPress Image.

**Syntax**

setimageindex( name, data )

**Arguments**

*name*

String value specifying the name of the index term with which you want to associate a data value.

*data*

String value specifying the data value you want to associate with the index term.

**Code Sample Examples**

These examples illustrate **setimageindex()**.

**Example 1**

```
setimageindex( 'PONumber', '2005')
```

**Example 2**

```
setimageindex( 'PONumber', @(2,24,32) )
```

## SetPDFBookmark (procedure)

Creates a PDF bookmark at the current page, for the PDF files PlanetPress Image generates.

>>> This function has no effect when the output of the document is not a PDF file generated through PlanetPress Image.

**Syntax**

setpdfbookmark( name )

**Arguments**

*name*

String value specifying the name of the bookmark that appears in the PDF.

**Code Sample Examples**

These examples illustrate **setpdfbookmark()**.

**Example 1**

```
setpdfbookmark('Table of Contents')
```

**Example 2**

```
setpdfbookmark( @(10,23,45) )
```

**Example 3**

```
setpdfbookmark( @(10,23,45) + '|' + @(11,23,45) )
```

# Document Pages

## @page (procedure)

Executes the content of the specified document page, as if it were part of the calling page. The content of the page in this case excludes the paper handling properties of the page. The height and width of the page content thus becomes the height and width of the group that would result if you created a single group of all of the page elements. If you want to include the paper handling properties in the page execution, use the *$element* syntax instead.

> The @ character can be the beginning of a <u>data selection</u>, a <u>page call</u> or a <u>procedure or function call</u>.

> The *@page()* and *execpage()* commands are functionally equivalent. However, the *<u>execpage()</u>* command has the added ability of calling variable page names.

**Syntax**

@pagename()

**Arguments**

*pagename*

String value specifying the name of the page you want to execute.

## Example

The following two lines of code are functionally equivalent and display the content of a page named *MyFirstPage*:

```
@MyFirstPage
execpage('MyFirstPage')
```

## ExecPage (procedure)

Executes the content of the specified document page, as if it were part of the calling page. The content of the page in this case excludes the paper handling properties of the page. If you want to include the paper handling properties in the page execution, see **"$element (procedure)" (page 611)**.

The **execpage()** and **@page()** commands are equivalent.

### Syntax

execpage( pagename )

### Argument

*pagename*

String value specifying the name of the page to run. The page must exist, and its name is case insensitive.

### Code Sample Example

In this example, the current page executes pages 2, 3 or 4 according to a data selection. This allows you to create pages of 'subroutines' that can be called from any other page.

### Example

```
define(&x,integer,strtoint(@(7,4,8)))
if(&x < 1000)
  execpage('PAGE1')
elseif()
  if(&x > 1000)
      %This is equivalent to execpage('PAGE3')
    @PAGE3
  elseif()
    execpage('PAGE2')
  endif()
endif()
```

## ShowPage (procedure)

Instructs the printer to output the page and move on to the next instruction.

This command has no effect on screen, because it is intended to be executed only inside the printer.

### Syntax

showpage()

**Argument**

None

**Code Sample Example**

This example specifies the number of pages to print at line 1, between columns 1 and 5.

**Example**

```
Define(&NumberOfPages,integer,strtoint(trim(@(1,1,5))))

Define(&x,integer,1)

for(&x,1,1,&NumberOfPages)

  execpage('Page1')

  showpage()

endfor()
```

# PPDs and PostScript

## CallPPD (procedure)

Calls a section of the PostScript Printer Description (PPD) file and executes it on the printer. Use this to set printer options such as the input tray or duplexing options. The tokens and subtokens you use as arguments with this function must be defined in the PPD file associated with your document or page in the document's or page's Basic Attributes. PlanetPress searches the PPD file for the token-subtoken combination and then prompts the printer to execute the corresponding code.

**Syntax**

callppd( token, subtoken )

**Arguments**

*token*

This is the string to search for in the PPD file (for example, Duplex).

*subtoken*

This is string to search for that is a setting for the token (for example, None).

**Example**

```
if((&Condition1) or (&Condition2))
      callppd("Tray", "2")
elseif()
      if((&Condition3) or (&Condition4))
             callppd("Tray", "3")
```

```
        elseif()
                callppd("Tray", "4")
        endif()
endif()
```

## PassThrough (procedure)

Sends a literal PostScript command to the printer.

**Syntax**

passthrough( string, [mode])

**Argument**

*string*

String value to be sent to the printer. This string is sent as is, with no further modification.

*mode*

Integer used as a bitfield to specify when to send the passthrough. Use a value of 1 to send the passthrough in PostScript-printer centric mode only, a value of 2 to send the passthrough in PostScript-Optimized PostScript Stream only, a value of 3 to send the passthrough in both PostScript-printer centric AND PostScript-Optimized PostScript Stream modes (default behavior if the parameter is not present), and a value of 4 to send the passthrough in Windows printing mode only.

**Code Sample Example**

Refer to your PostScript manual for valid commands.

## RunPS (procedure)

This command is typically used to call resources, usually images, downloaded either with the Image Downloader in PlanetPress or Send Images to Printer action tasks in the PlanetPress Suite Workflow Tools. *Note that this command does not work with any other PostScript files.*

**Syntax**

runps( filename )

**Argument**

*filename*

String value. Name of the resource file.

## SelectMedia (procedure)

On screen, displays the physical page. On a PostScript printer, calls the specified paper.

Variables used as arguments in this command are not parsed when printing using the Optimized PostScript Stream mode.

**Syntax**

selectmedia( width, height, [media], [color], [weight], orientation, mode )

**Arguments**

*width, height*

Measure values specifying the width and height of the physical page in inches.

*media*

A string value specifying the type of paper to use.

*color*

String value specifying the color of the paper.

*weight*

Measure value specifying the weight of the paper (in grams per square meter). You can convert weight in pounds to weight in grams by multiplying by the weight in pounds by 3.76.

*orientation*

Integer representing the orientation of the page (0=Portrait, 1=Landscape, 2=Reverse portrait, and 3=Reverse landscape).

*mode*

Reserved for future use. Specify 0 for now.

**Code Sample Example**

```
define(&mcolor,string,'')

&mcolor := 'White'

selectmedia(8.5,11,'Danny',&mcolor,20,0,0)
```

The code above will generate the following PostScript code (notice how the variable was not replaced by its value).

```
612 792 0 ^SM (Danny) &mcolor 20 ^SPM
```

## Program Control

## Exit (procedure)

Exits from a for...endfor or repeat...until loop.

**Syntax**

exit()

**Argument**

None

**Code Sample Example**

This example shows how to break from a loop when a specific word, in this case 'Invoice,' is found somewhere on the data page.

```
define(&x,integer,1)
for(&x,1,1,&current.lpp)
  if(pos('Invoice',@(&x,1,80))>0)
    exit()
  endif
endfor()
```

## For... EndFor (procedure)

This command structure allows a series of nested commands to be repeated a specified number of times. Each **for** statement must have a corresponding **endfor** statement in order for PlanetPress Talk to know which commands to include in the loop. Note that you cannot increment the counter for a **for()** loop, inside the loop.

If you open a document created a with a version of PlanetPress earlier than 5.2.0, and if that document includes a **for()** loop with an increment greater than 1, you will notice that the loop is executed one less time than with your older version software both in PlanetPress itself and when you use the Optimized PostScript Stream option. The behavior on printers (using the Printer Centric option), on the other hand, remains unchanged.

### Syntax

for( varname, startvalue, increment, stopvalue )

...

endfor()

### Arguments

*varname*

Name of the variable to use for iterations. The variable must already exist.

*startvalue*

Integer value to initialize varname.

*increment*

Integer value used to increment varname after each iteration.

*stopvalue*

Integer value after which iterations stop.

### Code Sample Example

This example simply prints 5 lines of text.

To cycle backwards through values, make **startvalue** larger than **stopvalue**, and specify a negative increment.

### Example

```
define(&x,integer,1)
for(&x,1,1,5)
  show('Line n° '+inttostr(&x))
```

```
    crlf()
endfor()
```

## If ... ElseIf... EndIf (procedure)

This command structure can define up to two conditional blocks. Statements nested within the first block are executed only if the condition specified in the if statement returns true. If it returns false, the program branches to the first statement in the second block, if one has been defined using the elseif statement. Otherwise, control flows to the first statement following the endif command.

With PlanetPress Suite version 7.0, the elseif statement has been optimized to evaluate an argument, exactly like an if statement, and elseif statements from other programming languages. In order to complete the if...elseif...endif command structure, however, an else statement was also added, acting exactly like the previous, empty elseif. Note that the use of the else statement is strongly recommended, as it is optimized by comparison to the elseif statement. Also note that documents created with previous versions and using the if...elseif...endif structure will not be updated, for full backward compatibility.

### Syntaxes

Prior to version 7.0

    if( condition )

    ...

    elseif()

    ...

    endif()

Version 7.0 and above

    if( condition1 )

    ...

    elseif( condition2 )

    ...

    else()

    ...

    endif()

### Argument

*condition, condition1, condition2*

Value of type Boolean to evaluate.

### Code Sample Example

This example simply prints 5 lines of text, selecting fonts according to line numbers.

### Example prior to version 7.0

```
define(&x,integer,1)
```

PLANETPRESS. SUITE

```
%Define loop variable and then set up loop

for(&x,1,1,5)

  if((&x mod 2)=0)

    %If &x is an even number

    setstyle(&bluefont)

    %use the blue font

  elseif() %otherwise

    setstyle(&Default)

    %use the default font

  endif()

  show('Some text to display')

  %Display the text

  crlf()

  %Skip to the next line

endfor()
```

## Example with version 7.0

```
define(&x, integer, inttostr(@(1, 1, 10)))

if(&x = 1)

  setstyle(&bluefont)

elseif(&x = 2)

  setstyle(&redfont)

else()

  setstyle(&default)

endif()
```

**Related topics:**

## Repeat... Until (procedure)

Repeat a sequence of commands until a condition is true. It is similar to a for... endfor loop in that you can nest the loops. It is different from a **for... endfor** loop in that the loop always executes at least once, and that you tie the number of times the loop repeats to a Boolean condition. For example you might tie the number of times the loop repeats to the occurrence of a piece of data in the data stream.

### Syntax

repeat

...

until( expression )

### Argument

*expression*

Boolean value. If it evaluates to true, the loop ends. If it evaluates to false, the loop repeats.

### Code Sample Example

This example prints the string 'Cheers' ten times, one underneath the other, with a dashed line above and below the ten instances.

### Example

```
show('---------------')

crlf(0.16)

&count := 1

repeat

  &count := &count + 1

  show('Cheers')

  crlf(0.16)

until(&count = 10)

show('---------------')

crlf(0.16)
```

## Search ... EndSearch (procedure)

This command structure allows a series of nested commands to be repeated as long as a specific string of characters (the search string) is found within another string (the target string). After each iteration, the target string contains all the data between occurrences of the search string, excluding the search string. Once the loop is over, target contains the remaining characters of the original data.

---

**Syntax**

search( target, search )

...

endsearch()

**Arguments**

*target*

Name of a variable of type string in which to perform the search.

*search*

String value to search for.

**Code Sample Example**

This example splits a string delimited with semi-colons.

In the **OnReadDataPage** event of the user-defined emulation, the **&str** system variable contains each new line of data being fed to the document. Also, do not assign anything to the variable while inside the **search-endsearch** loop; it will get overwritten.

**Example**

```
define(&v1,string,'This;is;a;test') %Create the variable

search(&v1,';') %Start loop

  show(&v1) %Show token

  crlf() %Skip line

endsearch()

show(&v1) %Show remaining text
```

# StopJob (procedure)

Terminates execution of the document and returns control to the PostScript interpreter.

**Syntax**

stopjob()

# Conversion Tables

This appendix contains an ASCII conversion table, a table for converting from points to inches, another for converting from points to centimeters, and a table for determining the height of a line of text given the number of Lines Per Unit (LPU).

This appendix contains the following conversion tables:

# ASCII Conversion Table

| Dec | Hex | Oct | Char |
|-----|-----|-----|------|
| 0 | 00 | 000 | NUL (null) |
| 1 | 01 | 001 | SOH (start of heading) |
| 2 | 02 | 002 | STX (start of text) |
| 3 | 03 | 003 | ETX (end of text) |
| 4 | 04 | 004 | EOT (end of transmission) |
| 5 | 05 | 005 | ENQ (enquiry) |
| 6 | 06 | 006 | ACK (acknowledge) |
| 7 | 07 | 007 | BEL (bell) |
| 8 | 08 | 010 | BS (backspace) |
| 9 | 09 | 011 | TAB (horizontal tab) |
| 10 | 0A | 012 | LF (NL line feed, new line) |
| 11 | 0B | 013 | VT (vertical tab) |
| 12 | 0C | 014 | FF (NP form feed, new page) |
| 13 | 0D | 015 | CR (carriage return) |
| 14 | 0E | 016 | SO (shift out) |
| 15 | 0F | 017 | SI (shift in) |
| 16 | 10 | 020 | DLE (data link escape) |
| 17 | 11 | 021 | DC1 (device control 1) |
| 18 | 12 | 022 | DC2 (device control 2) |
| 19 | 13 | 023 | DC3 (device control 3) |
| 20 | 14 | 024 | DC4 (device control 4) |
| 21 | 15 | 025 | NAK (negative acknowledge) |
| 22 | 16 | 026 | SYN (synchronous idle) |
| 23 | 17 | 027 | ETB (end of trans. block) |
| 24 | 18 | 030 | CAN (cancel) |
| 25 | 19 | 031 | EM (end of medium) |
| 26 | 1A | 032 | SUB (substitute) |
| 27 | 1B | 033 | ESC (escape) |
| 28 | 1C | 034 | FS (file separator) |
| 29 | 1D | 035 | GS (group separator) |
| 30 | 1E | 036 | RS (record separator) |
| 31 | 1F | 037 | US (unit separator) |
| 32 | 20 | 040 | SPACE |
| 33 | 21 | 041 | ! |
| 34 | 22 | 042 | " |
| 35 | 23 | 043 | # |
| 36 | 24 | 044 | $ |
| 37 | 25 | 045 | % |
| 38 | 26 | 046 | & |
| 39 | 27 | 047 | ' |
| 40 | 28 | 050 | ( |
| 41 | 29 | 051 | ) |

| Dec | Hex | Oct | Char |
|-----|-----|-----|------|
| 42 | 2A | 052 | * |
| 43 | 2B | 053 | + |
| 44 | 2C | 054 | , |
| 45 | 2D | 055 | - |
| 46 | 2E | 056 | . |
| 47 | 2F | 057 | / |
| 48 | 30 | 060 | 0 |
| 49 | 31 | 061 | 1 |
| 50 | 32 | 062 | 2 |
| 51 | 33 | 063 | 3 |
| 52 | 34 | 064 | 4 |
| 53 | 35 | 065 | 5 |
| 54 | 36 | 066 | 6 |
| 55 | 37 | 067 | 7 |
| 56 | 38 | 070 | 8 |
| 57 | 39 | 071 | 9 |
| 58 | 3A | 072 | : |
| 59 | 3B | 073 | ; |
| 60 | 3C | 074 | < |
| 61 | 3D | 075 | = |
| 62 | 3E | 076 | > |
| 63 | 3F | 077 | ? |
| 64 | 40 | 100 | @ |
| 65 | 41 | 101 | A |
| 66 | 42 | 102 | B |
| 67 | 43 | 103 | C |
| 68 | 44 | 104 | D |
| 69 | 45 | 105 | E |
| 70 | 46 | 106 | F |
| 71 | 47 | 107 | G |
| 72 | 48 | 110 | H |
| 73 | 49 | 111 | I |
| 74 | 4A | 112 | J |
| 75 | 4B | 113 | K |
| 76 | 4C | 114 | L |
| 77 | 4D | 115 | M |
| 78 | 4E | 116 | N |
| 79 | 4F | 117 | O |
| 80 | 50 | 120 | P |
| 81 | 51 | 121 | Q |
| 82 | 52 | 122 | R |
| 83 | 53 | 123 | S |
| 84 | 54 | 124 | T |
| 85 | 55 | 125 | U |
| 86 | 56 | 126 | V |
| 87 | 57 | 127 | W |

PLANETPRESS. SUITE

| Dec | Hex | Oct | Char |
|-----|-----|-----|------|
| 88 | 58 | 130 | X |
| 89 | 59 | 131 | Y |
| 90 | 5A | 132 | Z |
| 91 | 5B | 133 | [ |
| 92 | 5C | 134 | \ |
| 93 | 5D | 135 | ] |
| 94 | 5E | 136 | ^ |
| 95 | 5F | 137 | _ |
| 96 | 60 | 140 |  |
| 97 | 61 | 141 | a |
| 98 | 62 | 142 | b |
| 99 | 63 | 143 | c |
| 100 | 64 | 144 | d |
| 101 | 65 | 145 | e |
| 102 | 66 | 146 | f |
| 103 | 67 | 147 | g |
| 104 | 68 | 150 | h |
| 105 | 69 | 151 | i |
| 106 | 6A | 152 | j |
| 107 | 6B | 153 | k |
| 108 | 6C | 154 | l |
| 109 | 6D | 155 | m |
| 110 | 6E | 156 | n |
| 111 | 6F | 157 | o |
| 112 | 70 | 160 | p |
| 113 | 71 | 161 | q |
| 114 | 72 | 162 | r |
| 115 | 73 | 163 | s |
| 116 | 74 | 164 | t |
| 117 | 75 | 165 | u |
| 118 | 76 | 166 | v |
| 119 | 77 | 167 | w |
| 120 | 78 | 170 | x |
| 121 | 79 | 171 | y |
| 122 | 7A | 172 | z |
| 123 | 7B | 173 | { |
| 124 | 7C | 174 | | |
| 125 | 7D | 175 | } |
| 126 | 7E | 176 | ~ |
| 127 | 7F | 177 | DEL |

## Points to Inches or Centimeters

Points are PostScript points. A PostScript point measures 1/72 of an inch, or 0.035278 of a centimeter.

## Points to Inches

| Points | Inches | Points | Inches |
|---|---|---|---|
| 1 | 0.013889 | 2 | 0.027778 |
| 3 | 0.041667 | 4 | 0.055556 |
| 5 | 0.069445 | 6 | 0.083334 |
| 7 | 0.097223 | 8 | 0.111112 |
| 9 | 0.125001 | 10 | 0.13889 |
| 11 | 0.152779 | 12 | 0.166668 |
| 13 | 0.18055 | 14 | 0.194446 |
| 15 | 0.208335 | 16 | 0.222224 |
| 17 | 0.236113 | 18 | 0.250002 |
| 19 | 0.263891 | 20 | 0.27778 |
| 21 | 0.291669 | 22 | 0.305558 |
| 23 | 0.319447 | 24 | 0.333336 |
| 25 | 0.347225 | 26 | 0.361114 |
| 27 | 0.375003 | 28 | 0.388892 |
| 29 | 0.402781 | 30 | 0.41667 |
| 31 | 0.430559 | 32 | 0.444448 |
| 33 | 0.458337 | 34 | 0.472226 |
| 35 | 0.486115 | 36 | 0.500004 |
| 37 | 0.513893 | 38 | 0.527782 |
| 39 | 0.54167 | 40 | 0.55556 |
| 41 | 0.569449 | 42 | 0.583338 |
| 43 | 0.597227 | 44 | 0.611116 |
| 45 | 0.625005 | 46 | 0.638894 |
| 47 | 0.652783 | 48 | 0.666672 |
| 49 | 0.680561 | 50 | 0.69445 |
| 51 | 0.708339 | 52 | 0.722228 |
| 53 | 0.736117 | 54 | 0.750006 |
| 55 | 0.763895 | 56 | 0.777784 |
| 57 | 0.791673 | 58 | 0.805562 |
| 59 | 0.81945 | 60 | 0.83334 |
| 61 | 0.847229 | 62 | 0.861118 |
| 63 | 0.875007 | 64 | 0.888896 |
| 65 | 0.902785 | 66 | 0.916674 |
| 67 | 0.930563 | 68 | 0.944452 |
| 69 | 0.95834 | 70 | 0.97223 |
| 71 | 0.98611 | 72 | 1.000008 |
| 73 | 1.013897 | 74 | 1.027786 |
| 75 | 1.04167 | 76 | 1.055564 |
| 77 | 1.06945 | 78 | 1.083342 |
| 79 | 1.09723 | 80 | 1.11112 |
| 81 | 1.125009 | 82 | 1.138898 |
| 83 | 1.15787 | 84 | 1.166676 |
| 85 | 1.18056 | 86 | 1.194454 |
| 87 | 1.208343 | 88 | 1.222232 |

| Points | Inches | Points | Inches |
|--------|--------|--------|--------|
| 89 | 1.236121 | 90 | 1.25001 |
| 91 | 1.263899 | 92 | 1.277788 |
| 93 | 1.291677 | 94 | 1.305566 |
| 95 | 1.319455 | 96 | 1.333344 |
| 97 | 1.347233 | 98 | 1.361122 |
| 99 | 1.375011 | 100 | 1.3889 |

## Points to Centimeters

| Points | Centimeters | Points | Centimeters |
|--------|-------------|--------|-------------|
| 1 | 0.035278 | 2 | 0.070556 |
| 3 | 0.105834 | 4 | 0.141112 |
| 5 | 0.176389 | 6 | 0.211667 |
| 7 | 0.246945 | 8 | 0.282222 |
| 9 | 0.3175 | 10 | 0.352778 |
| 11 | 0.388056 | 12 | 0.423334 |
| 13 | 0.458611 | 14 | 0.493889 |
| 15 | 0.529167 | 16 | 0.564445 |
| 17 | 0.599723 | 18 | 0.635 |
| 19 | 0.670278 | 20 | 0.705556 |
| 21 | 0.740834 | 22 | 0.776112 |
| 23 | 0.811389 | 24 | 0.846667 |
| 25 | 0.881945 | 26 | 0.917222 |
| 27 | 0.952500 | 28 | 0.987778 |
| 29 | 1.023056 | 30 | 1.058333 |
| 31 | 1.093611 | 32 | 1.128889 |
| 33 | 1.164167 | 34 | 1.199444 |
| 35 | 1.234722 | 36 | 1.270000 |
| 37 | 1.305278 | 38 | 1.340556 |
| 39 | 1.375833 | 40 | 1.411111 |
| 41 | 1.446389 | 42 | 1.481667 |
| 43 | 1.516945 | 44 | 1.552222 |
| 45 | 1.587500 | 46 | 1.622778 |
| 47 | 1.658056 | 48 | 1.693333 |
| 49 | 1.728611 | 50 | 1.763889 |
| 51 | 1.799167 | 52 | 1.834445 |
| 53 | 1.869722 | 54 | 1.905000 |
| 55 | 1.940278 | 56 | 1.975556 |
| 57 | 2.010833 | 58 | 2.046111 |
| 59 | 2.081389 | 60 | 2.116667 |
| 61 | 2.151945 | 62 | 2.187222 |
| 63 | 2.222500 | 64 | 2.257778 |
| 65 | 2.293056 | 66 | 2.328333 |
| 67 | 2.363611 | 68 | 2.398889 |
| 69 | 2.434167 | 70 | 2.469445 |
| 71 | 2.504722 | 72 | 2.540000 |
| 73 | 2.575278 | 74 | 2.610555 |
| 75 | 2.645833 | 76 | 2.681111 |
| 77 | 2.716389 | 78 | 2.751667 |
| 79 | 2.786945 | 80 | 2.822222 |
| 81 | 2.857500 | 82 | 2.892778 |
| 83 | 2.928056 | 84 | 2.963333 |
| 85 | 2.998611 | 86 | 3.033889 |
| 87 | 3.069167 | 88 | 3.104445 |

| Points | Centimeters | Points | Centimeters |
|---|---|---|---|
| 89 | 3.139722 | 90 | 3.175000 |
| 91 | 3.210278 | 92 | 3.245556 |
| 93 | 3.280833 | 94 | 3.316111 |
| 95 | 3.351389 | 96 | 3.386667 |
| 97 | 3.421945 | 98 | 3.457222 |
| 99 | 3.492500 | 100 | 3.527778 |

# Line Height as a Function of Lines Per Unit (LPU)

Use these tables to determine the height of an individual line, in inches, given a specific Lines Per Unit (LPU) value.

## Line Height as a Function of Lines Per Inch

Use this table to determine the height, in inches, of an individual line when the unit of the LPU is inches.

| LPI | Line Height (inches) | LPI | Line Height (inches) |
|---|---|---|---|
| 1 | 1 | 2 | 0.5 |
| 3 | 0.333333333 | 4 | 0.25 |
| 5 | 0.2 | 6 | 0.166666667 |
| 7 | 0.142857143 | 8 | 0.125 |
| 9 | 0.111111111 | 10 | 0.1 |
| 11 | 0.090909091 | 12 | 0.083333333 |
| 13 | 0.076923077 | 14 | 0.071428571 |
| 15 | 0.066666667 | 16 | 0.0625 |
| 17 | 0.058823529 | 18 | 0.055555556 |
| 19 | 0.052631579 | 20 | 0.05 |
| 21 | 0.047619048 | 22 | 0.045454545 |
| 23 | 0.043478261 | 24 | 0.041666667 |
| 25 | 0.04 | 26 | 0.038461538 |
| 27 | 0.037037037 | 28 | 0.035714286 |
| 29 | 0.034482759 | 30 | 0.033333333 |
| 31 | 0.032258065 | 32 | 0.03125 |
| 33 | 0.03030303 | 34 | 0.029411765 |
| 35 | 0.028571429 | 36 | 0.027777778 |
| 37 | 0.027027027 | 38 | 0.026315789 |
| 39 | 0.025641026 | 40 | 0.025 |
| 41 | 0.024390244 | 42 | 0.023809524 |
| 43 | 0.023255814 | 44 | 0.022727273 |
| 45 | 0.022222222 | 46 | 0.02173913 |
| 47 | 0.021276596 | 48 | 0.020833333 |
| 49 | 0.020408163 | 50 | 0.02 |

## Line Height as a Function of Lines Per Centimeter

Use this table to determine the height, in inches, of an individual line when the unit of the LPU is centimeters.

| LPC | Line Height (inches) | LPC | Line Height (inches) |
|---|---|---|---|
| 1 | 0.3937008 | 2 | 0.196850394 |
| 3 | 0.181233596 | 4 | 0.098425197 |
| 5 | 0.078740157 | 6 | 0.065616798 |
| 7 | 0.056242970 | 8 | 0.049212598 |
| 9 | 0.043744532 | 10 | 0.039370079 |
| 11 | 0.035790980 | 12 | 0.032808399 |
| 13 | 0.030284676 | 14 | 0.028121485 |
| 15 | 0.026246719 | 16 | 0.024606299 |
| 17 | 0.023158870 | 18 | 0.021872266 |
| 19 | 0.020721094 | 20 | 0.019685039 |
| 21 | 0.018747656 | 22 | 0.017895490 |
| 23 | 0.017117425 | 24 | 0.016404199 |
| 25 | 0.015748031 | 26 | 0.015142338 |
| 27 | 0.014581511 | 28 | 0.014060742 |
| 29 | 0.013575889 | 30 | 0.013123360 |
| 31 | 0.012700025 | 32 | 0.012303150 |
| 33 | 0.011930327 | 34 | 0.011579435 |
| 35 | 0.011248594 | 36 | 0.010936133 |
| 37 | 0.010640562 | 38 | 0.010360547 |
| 39 | 0.010094892 | 40 | 0.009842520 |
| 41 | 0.009602458 | 42 | 0.009373828 |
| 43 | 0.009155832 | 44 | 0.008947745 |
| 45 | 0.008748906 | 46 | 0.008558713 |
| 47 | 0.008376612 | 48 | 0.008202100 |
| 49 | 0.008034710 | 50 | 0.007874016 |

# Tools and Utilities

This chapter describes specific tools and utilities that are available in the PlanetPress Design interface.

## The Image Downloader

It is important to understand that the copy of the image file the Image Downloader downloads or copies is a PostScript file. More precisely, the Image Downloader takes the image file you want to download or copy, applies any optimizations you defined for it, and creates a PostScript file that contains the optimized image (since PostScript is ASCII, the image is represented as a sequence of hexadecimal values). Thus, although the downloaded or copied image may have the same filename extension as the original file, it is *not* a copy of the original file.

To copy images to a folder on the host:

1. In Windows locate the external image files the dynamic image references. These images can be in any of the supported formats.
2. Copy the images to the folder on the host. This is the folder you specified in the Image box of the picture object that contains the dynamic image. Recall that this folder must be accessible to the document at runtime.

To download images to one or more printers and/or to the local PlanetPress Design Suite virtual disk:

1. Choose **Tools** | **Application** | **Image Downloader**.
2. In the Image Downloader dialog box, select the image files you want to copy. All image files you select must be in one of the supported image formats (BMP, EPS, JPEG, PDF, PNG, TIFF). If you want to download bitmapped images with a mix of image qualities (some are line art, some are photo), you should download each type of image quality separately. This is because you can set only one image quality for all images in a download.
3. In the **Copy to** list, select the destinations to which you want to copy the images. You can select one or more printers, and/or the **PlanetPress Design Suite virtual disk**.
4. Adjust the image settings for the download. The Image Downloader applies these settings to all of the images in the Files to download list. Note that they affect only the copy of each image; the original image remains unchanged.
   **Scan orientation**: Select the scanline orientation for all bitmapped images in the Files to download list.
   **Color conversion**: Define any color depth conversion you want PlanetPress Design to perform on any of the bitmapped images in the Files to download list, during the download. Select Leave as is to leave the color depth of each image unchanged. Select Convert to grayscale to convert each color bitmapped image to grayscale. Select Convert to monochrome to convert each color or grayscale bitmapped image to monochrome.
   **Naming convention**: Specify the case and file name extension conventions you want PlanetPress Design to use for the file names of the copied images.Select File name original to have the name of each copied file appear in upper case, with the file name extension as it is on the original. Select File name no extension to have the name of each copied file appear in upper case, with the file name extension dropped.
   **Naming convention**: Specify the case and file name extension conventions you want PlanetPress Design to use for the file names of the copied images. Select Filename [original] to have the name of each copied file appear in upper case, and the file name extension to remain as it is on the original. Select Filename w/o extension to have the name of each copied file appear in upper case, and to have the file name extension dropped. Select Filename lowercase to have the name of the copied file appear in lower case, and the file name extension to remain as it is on the original. Select Filename w/o extension lowercase to have the name of the copied file appear in lower case, and to have the file name extension dropped.
   **Printer path**: Specify the path on the printer where you want to download the copy of the images. This path can be either a path on the hard disk or a path in flash memory. This path must exist on all of the printers you select in the Copy to list.
   **Image quality**: Select the image quality for the bitmapped images in the Files to download list. Recall that image quality in PlanetPress Design refers to the type of content a bitmapped image contains. Select Line art if edges in the image

are clearly defined and sensitive to any loss of image information. The compression PlanetPress Design applies to line art images is lossless. Note that Line art quality only works with PostScript Level 3. If you select Line art for images, and the PPD you select for the document is not for a PostScript Level 3 printer, PlanetPress Design automatically switches the image quality from Line art to Photo, and applies the compression level set for Photo quality. In most cases this results in a noticeable degradation of quality. One solution is to adjust the compression level to 100%. A compression level of 100% means PlanetPress Design does not perform any compression. This preserves all of the image information in the line art images, at the expense of not performing any compression on any of the images in the Files to download list.

**Image compression level**: Set the compression level you want the Image Downloader to use when you select Photo in the Image quality box. Legal values are 1 to 100.

**Force PostScript mode**: Select this option when the destination printer is used for printing jobs in other formats such as PJL, and when PlanetPress Suite Workflow Tools is not sending the output to the printer's hard disk. When this option is selected, PlanetPress Design inserts the following command statement [Esc]%-12345X@PJL ENTER LANGUAGE = POSTSCRIPT to precede the document as well as any other command statement.

5. Click **Download**.

For each of the image files in the Files to download list, the Image Downloader makes a copy of the file. It then applies any resolution, scan orientation, color, or file name modifications you specified, to the copy. Finally it converts the copy to PostScript and downloads it to each of the destinations selected in the Copy to list. If it cannot download some or all of the images, it displays an error message to this effect.

To create a file of images, along with the PostScript code to install them on a printer:

1. Choose **Tools** | **Application** | **Image Downloader**.
2. In the **Image Downloader** dialog box, select the image files you want to copy. All image files you select must be in one of the supported image formats (BMP, EPS, JPEG, PDF, PNG, TIFF). If you want to download bitmapped images with a mix of image qualities, you should download each type of image quality separately. This is because you can set only one image quality for all images in a download.
3. Adjust the image settings for the download. The Image Downloader applies these settings to all of the images in the Files to download list.
4. Click **Save to file**.
5. In the **Save As** dialog box, navigate to folder in which you want to save the file, and enter a name for the file.
6. Click **Save**.

To select the images to download or save in a file:

1. In the **Image Downloader** dialog box, click the **Browse** button at the top right of the Files to download list to display the Open dialog box.
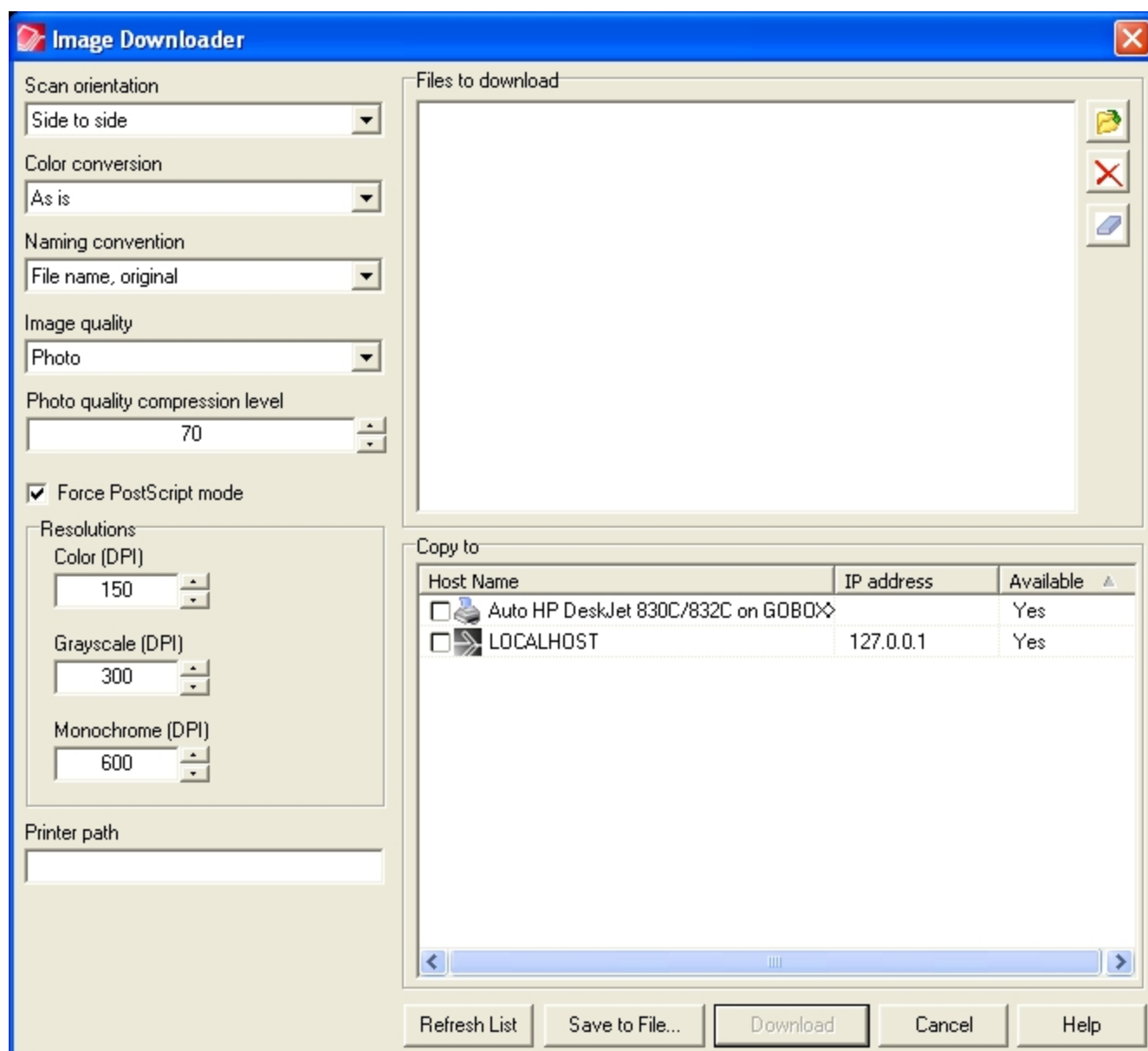
A. Browse  B. Delete  C. Clear

2.  In the **Open** dialog box, navigate to the folder containing the images, and select the images you want to download and/or copy. To select more than one image file, click the first file you want to add, then **CTRL**+click each additional file you want to add to the selection. **CTRL**+click a file a second time to remove it from the selection. You can also **SHIFT**+click to select a range of files. Click **Open** to exit the Open dialog box.

3.  If necessary, you can edit the content of the Files to download list as follows:
    -   step 1
    -   To delete one or more images from the list, select the images then click **Delete**. To select more than one image file, click the first file, then **CTRL**+click each additional file to add it to the selection. **CTRL**+click a file a second time to remove it from the selection. You can also **SHIFT**+click to select a range of files.
    -   To remove all images from the list, click **Clear**.

## Adjust the Image Quality Options

To set the default image quality and the photo quality compression level:

---

1. Double-click on the **Document** node.
2. In the **Document** properties dialog box, click **Resource options** and adjust the compression options.
   **Image quality**: Select the image quality for this image resource.
   **Photo quality compression level**: Set the compression level you want PlanetPress Design to use for image resources of Photo image quality. Legal values are 1 to 100. Compression may affect print speed because each image on the printer must be uncompressed at print time, which may affect print quality. Some formats such as JPEG compression may cause image degradation, whereas LZW compression does not. The user can select the type and degree of depression PlanetPress Design uses to send to the printer.
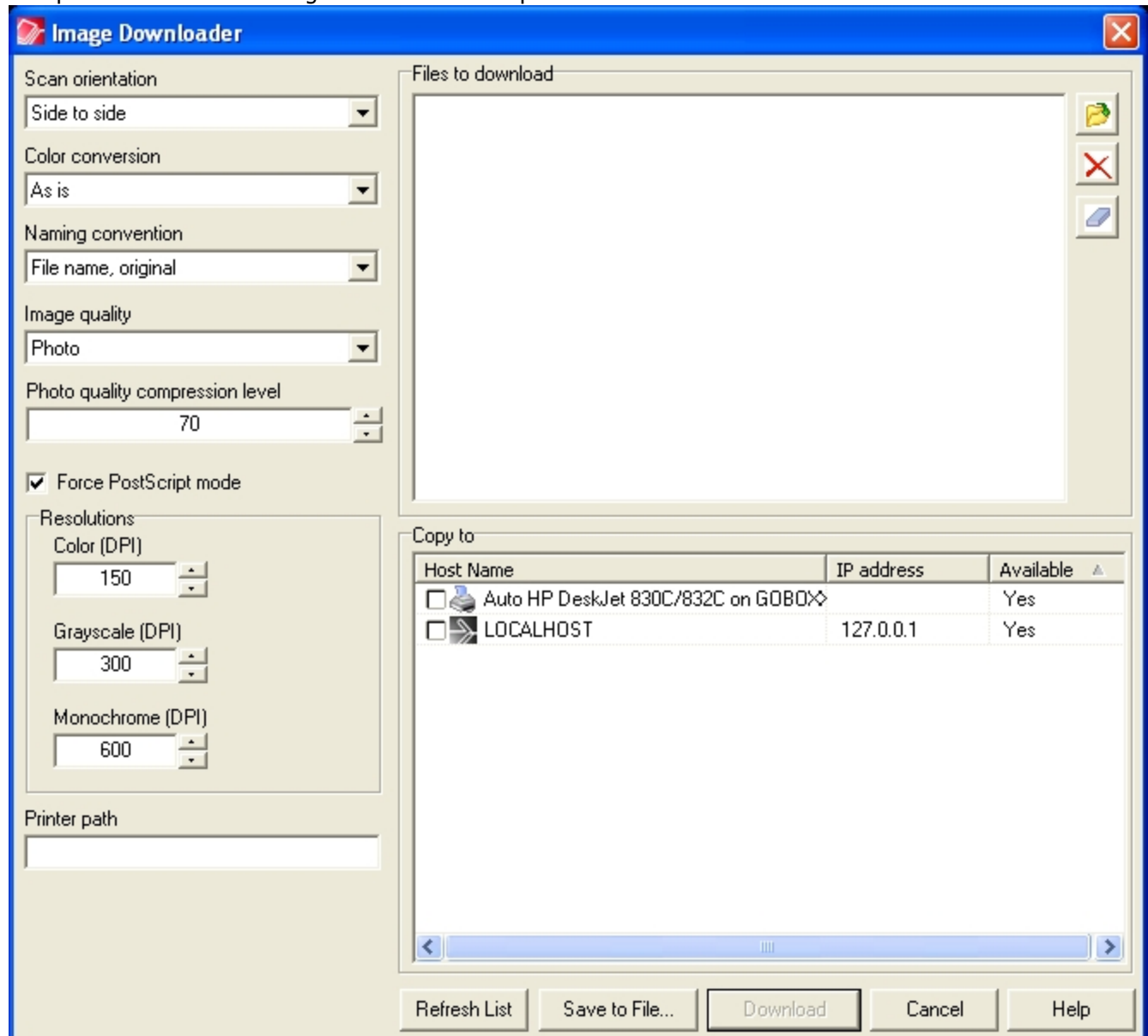
A. Force PostScript mode option; B. Refresh List button; C. Save to File button

**Force PostScript mode**: Select this option when the destination printer is used for printing jobs in other formats such as PJL, and when PlanetPress Suite Workflow Tools is not sending the output to the printer's hard disk. When this option is selected, PlanetPress Design inserts the following command statement [Esc]%-12345X@PJL ENTER LANGUAGE = POSTSCRIPT to precede the document as well as any other command statement.

**Refresh List**: Click this button to refresh the list of available hosts to which the images may be downloaded.

**Save to File**: Click this button to save all the image files in the Files to download list to the filename you specify with a PS file extension. The Image Downloader makes a copy of each image, applies the image settings to the copy, and then converts the copy to PostScript.

3. Click **OK**.
4. When you add an image resource to the document, PlanetPress Design associates the specified image quality with that image resource. If you selected Photo as the image quality, it uses the specified compression to compress the image resource when it converts the document.

To adjust the compression algorithm for an individual image resource in the document:

- Edit the Image quality property of the image resource whose image quality you want to change.

## Virtual Drive Manager

When you use the Send images to printer action in a given process, you have the option of, at the same time, sending the images to the virtual drive (a local storage folder used by PlanetPress Suite applications) of any computer included in your network. You need to do this, for instance, if you plan to run documents that contain dynamic images on those computers (using the **Optimized PostScript Stream** option). You can then use the Virtual Drive Manager to see the images that were downloaded to your computer as well as to delete them from your virtual drive.

**To add images to the virtual drive, use either of the following methods**:

- Send a single resource file to the printer: see Download to Printer Action Task Properties.
- Send one or more images to the printer: see Send Images to Printer Action Task Properties.
- Use The Image Downloader.

**To delete images from your virtual drive**:

1. In the PlanetPress Suite Ribbon, Go to the **Tools** tab, then click on **Virtual Drive Manager**.
   The Virtual Drive Manager dialog box is displayed. It lists all the images currently stored in your computer's virtual drive.
2. Select the images you want to delete.
3. Press the Delete key.

# Access Manager

Use the Access Manager to control access to those services running on your computer.
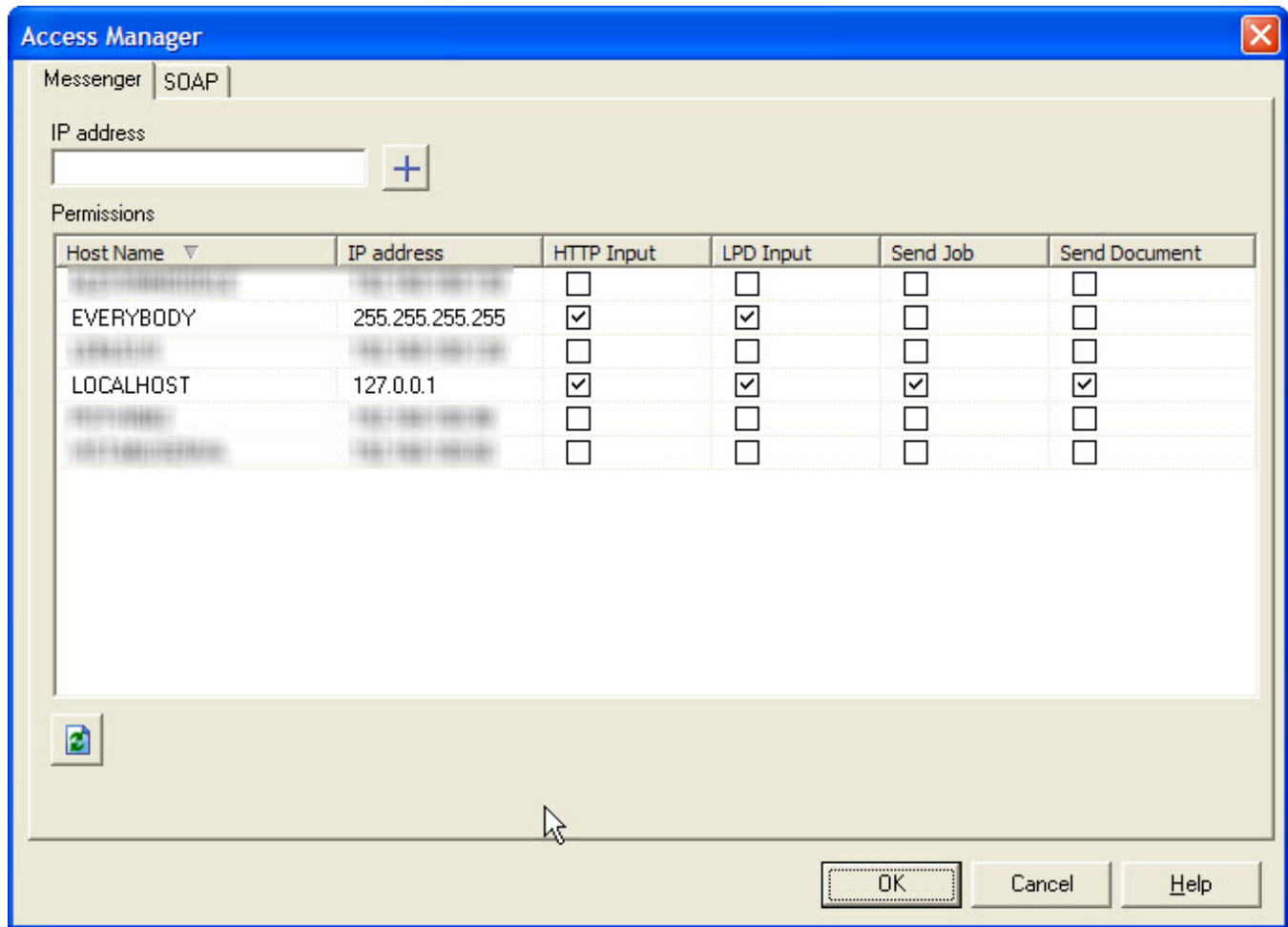
## To open the Access Manager

- Open **PlanetPress Design** or **PlanetPress Workflow Configuration Tool**.
- In the Ribbon, go in **Tools** | **Managers** | **Access Manager**.

The **Access Manager** dialog box is displayed. It lists all IP and IP ranges that have PlanetPress Design, PlanetPress Suite Workflow Tools, PlanetPress Fax or PlanetPress Image installed in the same network.

## To add a new entry in the list

- Open the *Access Manager*
- Make sure you are in the *Messenger* tab.
- In the **IP address** box, enter the IP address of the remote machine.
- Click on the ⊞ button.

The format of the IP address must be one of the following:

- **127.0.0.1**: The local computer. Typically this IP should have all the accesses checked.
- **255.255.255.255**: Everyone on the same subnet. This is equivalent to hard-coding the current subnet, such as 192.168.1.255 or 10.0.0.255.
- **192.168.0.42** : A single IP address. This can be any valid address on the same subnet.
- **10.0.255.255**: Any IP in the 10.0.X.X range.

The following considerations are to be understood when using the Access Manager to configure IP limitations:

---

- Each permission type (column) is evaluated from top to bottom (column per column) in the order they are visible in the Access Manager window. This means that wide ranges should always be at the top to increase performance. For example, if you accept HTTP connections from any IP, the first entry should be 255.255.255.255 with the Allow checkmark in the HTTP Input box. PlanetPress does not continue processing after it has found an "Allow" checkmark. There is no concept of "Deny", meaning if any "Allow" permission is given, there is no way to later remove it for certain IPs or IP ranges.
- Messenger communications (between different part of the PlanetPress Suite) was limited to within the same Class C subnet. This means that PlanetPress Workflow on 192.168.1.23 cannot send a job to a PlanetPress Image on 192.168.100.54. This limitation has been removed in PlanetPress Suite 7.3
- HTTP, FTP and SOAP communication is not limited to the local subnet on any version where these plugins appear.

## To modify permissions

Permissions are given simply by adding and removing options in the permission grid. Access to the services installed on this computer is granted or denied by checking the corresponding boxes next to the listed IP ranges. For each IP range, the following information is displayed:

- **Host name**: The name of those computers on which PlanetPress Suite software are currently installed or which have been manually added.
- **IP address**: The IP address or IP address range to give permission to.
- **Permissions**
  - **HTTP Input**:Grants access to send HTTP Requests to this server.
  - **LPD Input**: Grants access to send LPD Queue jobs to this server.
  - **Send Job**: Grants access to the selected computer or server to send jobs to PlanetPress Fax and PlanetPress Image installed on this server.
  - **Send Document**: Grants access to the remote computer to send new or updated PlanetPress Design Documents to this server.

> In order for the changes made here to be effective, you will need to restart the PlanetPress Messenger service. This can be done via the PlanetPress Suite Service Console.

# SOAP Access

The SOAP tab of the Access Manager controls access from SOAP clients to local processes and SOAP processes. Each username entered in this dialog can have access to one or more processes.
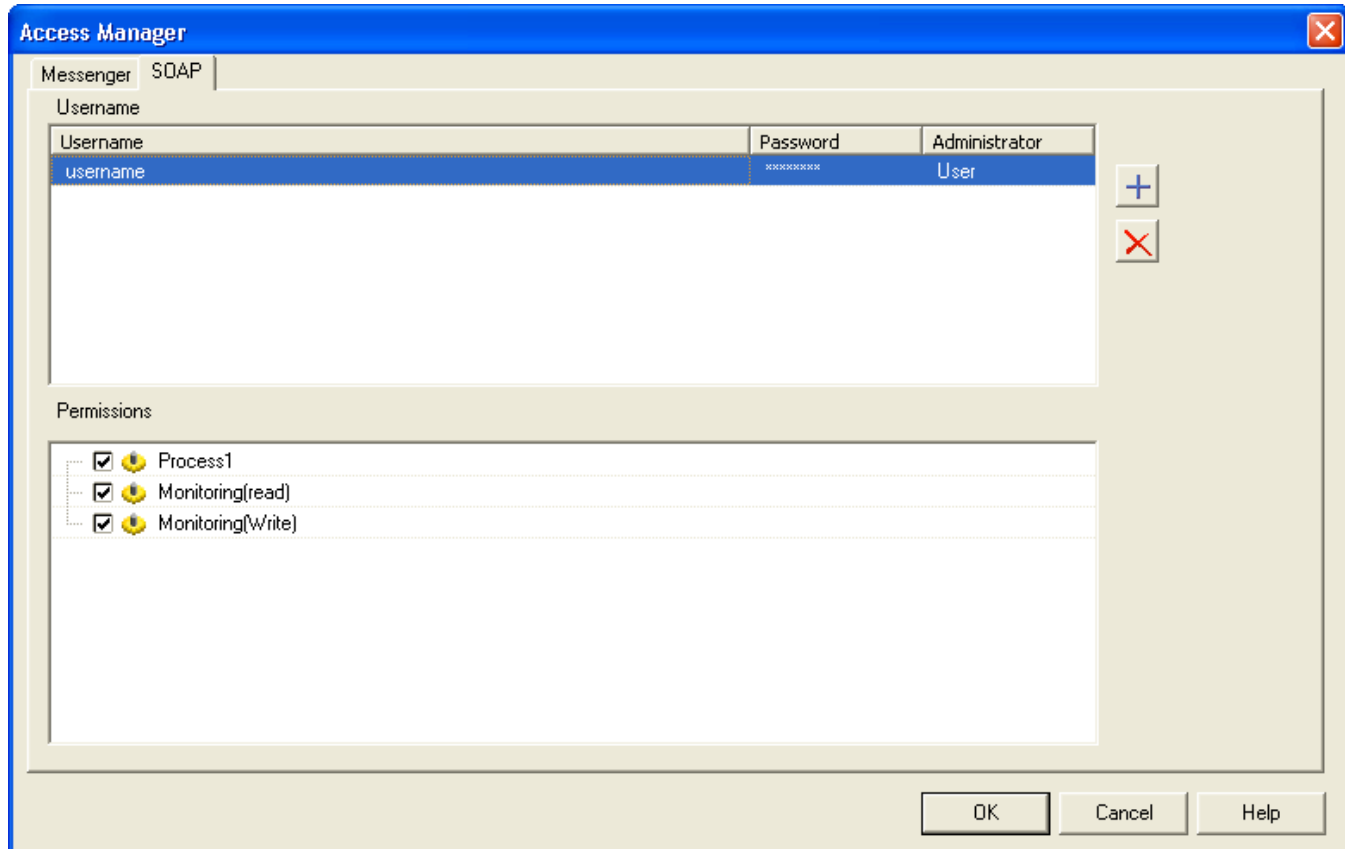
## To add a new SOAP user

- Click on the ⊞ button.
- Enter the following information in the Username section for the new entry that was created:
  - Username: An alphanumerical username for the user.
  - Password: A password to protect the user. Note that the password will always revert to ******** (8 stars) when clicking outside of this box - that is normal and is meant to protect the length of the password as much as its contents.
  - Administrator: Choose the permission type
    - User: Can access none, some, or all of the processes, selected individually in the Permissions section.
    - Admin: Has access to all processes and features. When this option is selected, the Permission section is greyed out an all options are selected in it.
    - Disabled: Has access to nothing. The result is the same as not having this user defined at all, but has the advantage that a disabled user can be reactivated with a simple click.
- Define the permissions for the user (see below).
- Click OK to save the changes.

## To define or change the permissions for a SOAP User

The Permissions section of the SOAP tab displays all of the processes that are available in the live configuration (the one that the PlanetPress service uses). To change or define the permissions:

- In the top Username section, click on the username of which to modify permissions.
- Place a checkmark in each process that the user should have access to.
- Click OK to save the changes.

In order for the changes made here to be effective, you will need to restart the PlanetPress Messenger service. This can be done via the PlanetPress Suite Service Console.

# Managing Documents and Printers

This section presents the features of PlanetPress Design that you can use to manage document on a printer, and to adjust printer settings.

## Obtain Information from a Printer

To obtain information from a printer:

1. Choose **Tools** | **Managers** | **Printer Utilities**.
2. In the **Select information to request** list, select the information you require.
   **PlanetPress Design printer status page**: Select to request a PlanetPress Design printer status page.
   **Printer variable content document listing**: Select to request a list of all variable content documents installed on the printer. The list includes all PlanetPress Design documents on the hard drive, in RAM, and in the flash memory of the printer. The information for each document includes the comments you defined in the Notes box of the Document properties dialog box.
   **Printer files listing**: Select to request a list of all files on the printer.
3. In the **Select printers** list, select the printers for which you want to obtain the selected information.
4. Select **Send to file** if you want to save the information as a PostScript file rather than output it on the printer(s).
5. Click **OK**.
   If you selected Send to file, for each of the selected information sheets you requested PlanetPress Design prompts you to specify the name of the file to which you want to save the information. Thus if you selected all three information sheets, PlanetPress Design prompts you three times.
   If you did not select Send to file, the information prints on each of the selected printers.

## Delete Documents or Files on the Printer

To delete files on a printer:

1. Choose **Tools** | **Managers** | **Printer Utilities** to display the Printer Information dialog box.
2. In the **Select information to request** list, select **Delete files from printer's file system**.
3. In the **Name of document to delete** box, enter the path of the first document or file you want to delete. The path syntax depends on whether the file is located on the hard disk or in flash memory.

| Path syntax: | Deletes a file: |
|---|---|
| %<DISKNAME>%<DOC_OR_FILE_NAME> | On the hard disk. If the printer has more than one hard disk, you must specify the one on which the document or file you want to delete resides. For example:<br>%sales%sale_flyer<br>deletes the document or file "sale_flyer" on the hard disk named "sales." |
| %FLASH%<DOC_OR_FILE_NAME> | In flash memory. For example:<br>%flash%tax_bill<br>deletes the document or file "tax_bill" from the flash memory of the printer. |
| <DOC_OR_FILE_NAME> | On a printer that has either only flash memory or only a single hard disk. |

4. If you want to delete additional documents or files, at the end of the path, press **ENTER** and enter the path for the next document or file.
5. Repeat step 4 for each additional document or file you want to delete.
6. Select either **Send to file** or select one or more printers in the Select printers list:
   - If you want to save the request to a file and perform the deletion at a later time, select **Send to file**.
   - If you want to perform the deletion at this time, in the **Select printers** list, select the printer(s) from which you want to delete the document or file.
7. Click **OK**.

If you selected Send to file, PlanetPress Design prompts you to specify the name of the file to which you want to save the deletion request. Once you enter the file name and click **Save**, PlanetPress Design saves the request as a Post-Script file.

If you did not select Send to file, PlanetPress Design attempts to delete the specified files and/or documents from the selected printer(s) and prints a confirmation sheet. The confirmation sheet either confirms the printer deleted the files/documents or reports that it could not delete the file.

## Printer Firmware Version

What can prevent my printer from printing PlanetPress Design documents?

Some older PostScript printers do not integrate the full PostScript Level 2 language. This may cause PlanetPress Design print jobs to crash and the printer to display the following error message:

- *Error: undefined; OffendingCommand: execform.*

To fix this problem, you should upgrade your printer's firmware. If this cannot be done, you will need to disable caching in the document options.

## Control Versions of a Document

You can assign a version number to a document and increment the number each time you update and reinstall the document. When you execute a document that uses a version number, the document verifies its version number against the one you specify in the trigger, and it proceeds with execution only if the two version numbers match.

You might use the version option in the following situations:

- You have a single document installed on several different printers, and you must update it on all printers. If you are unable to install the updated document on all the printers, different versions of the document exist. To ensure you don't accidentally print a job with the wrong version of the document, you include a version number in the document. When you send a trigger to the printer, in the trigger you specify the version number of the document you want to use. The document checks its version number against the one in the trigger. If it determines it is not the latest version, it prints a page reporting it is not the latest version and does not execute.
- You want to install multiple versions of a document on a printer and have a means of controlling which one you execute at any given time.

To set the version number of a document:

1. Double-click on the **Document** node in the Document Structure area.
2. Click **Conversionoptions** and locate the version options.
3. Adjust the version options.
   **Use document versioning at printer level**: Select to use version numbers with the document. Use the Document version box to enter the version number for this document.
   **Document version**: Enter the version number of the document, or use the spin buttons to adjust the value. In a document that uses version numbers, you normally increment the version number each time you update and reinstall the document.
4. Install the document. You must include the version number of the document in the trigger.

To execute a document that uses version numbers:

To execute a document that uses version numbers, insert the version number of the document in the trigger, just before the "run" command. The following trigger examples all execute version 3 of the document named PAY.

| Trigger: | Document location: |
| --- | --- |
| <CRTL-D> | Hard drive of a printer that has a single hard drive. |

| Trigger: | Document location: |
| --- | --- |
| %!PS-Adobe<br><CR><LF><br>(PAY) run 3 PAY<br><CR><LF><br><CRTL-D> | In this example, the document named PAY is loaded from the printer's hard drive to RAM. The trigger executes version 3 of the PAY document which is stored in RAM. |
| %!PS-Adobe<br><CR><LF><br>(Accounting\PAY)<br>run 3 PAY<br><CR><LF><br><CRTL-D> | Accounting folder of a printer with a single hard drive. This example uses a relative path (a back slash does not precede Accounting). Whether you use an absolute or relative path depends on your printer configuration. For example, a printer may not have access to the root folder of its disk when it is running in PostScript mode, and would thus require a relative path. |
| %!PS-Adobe<br><CR><LF><br>(%admin%PAY)<br>run 3 PAY<br><CR><LF><br><CRTL-D> | Hard drive named "admin" of a printer. Printer may or may not have more than one hard drive. |
| %!PS-Adobe<br><CR><LF><br>3 PAY<br><CR><LF><br><CRTL-D> | Random Access Memory (RAM) of a printer. |
| %!PS-Adobe<br><CR><LF><br>(%flash%PAY)<br>run 3 PAY<br><CR><LF> | Flash memory of a printer. |

## Adjust Printer Settings

There are five printer settings you can set from PlanetPress Design: Form cache, Manual feed timeout, Wait timeout, Print PostScript error, and Do Sys/Start.

To adjust printer settings:

1. Choose **Tools** | **Managers** | **Printer Utilities**.
2. In the **Select information to request** list, select **Set printer's advanced options**.
3. In the **Select printers** list, select the printer(s) for which you want to adjust the options.
4. In the **Advanced options** area, adjust the options you want to set for the selected printer(s).
   **Max form items**: Set the size, in bytes, of the largest single EPS, PDF, or bitmapped image that the form cache can contain. This option applies to all documents that execute on the printer. You use the size of the largest and most frequently used image in your document to determine an appropriate value for this option.
   **Max form cache**: Set the size, in bytes, of the PostScript printer form cache. This sets the cache size for all documents that execute on the printer. You base the setting for this option on the number of images in your documents, their sizes, and how frequently each image repeats in a document.
   **Manual feed timeout**: Enter the maximum amount of time, in seconds, you want the printer to wait for paper from a manual feed before terminating the current job.
   **Wait timeout**: Enter the maximum amount of time, in seconds, you want the printer to wait for input before terminating the current job.
   **Do Sys/Start**: Select to have the printer execute its Sys/Start file at boot time.
   **Print PostScript error**: If the printer encounters a PostScript error during execution of a job, it terminates the job. Select this option to have the printer print an error page that reports the offending PostScript command and the status of the print stack. This can be useful in debugging a document.
5. Click **OK**.

## Form Cache

What is the form cache? What options can I adjust with respect to the form cache?

The form cache is an amount of RAM set aside to hold images that the document references during execution. The cache improves performance by providing faster access to images that the document uses more than once.

The size of the cache and the order in which the document executes the images determine the contents of the cache at any given point in time. As the cache fills, the printer may have to remove images to make room for the most recent image. When it must remove an image, the printer removes the least recently referenced one.

In PlanetPress Design, you can set both the size of the printer form cache and the maximum size of an individual cache item. You can set these options for all documents that execute on the printer.

When you set the cache options, you should know that the images in the cache have already undergone part of the rasterization process and that this may have increased their size, sometimes quite substantially. Since this difference in size depends on the individual image the best approach for determining appropriate values for the cache options is empirical: if document execution is slow, adjust the value and see if performance improves.

### PlanetPress Design Printer Status Page

The PlanetPress Design printer status page contains the following information about the printer.

- The serial number and the full version number of the copy of PlanetPress Design making the request
- Printer information
- Information on the current job
- Information on the installed devices
- Memory size information
- Input and output tray information
- Paper handling and finishing information
- System settings

## Remove Background Color

In the Color Picker, a new option called Remove background color is available with the CMYK color model.

The main purpose of this feature is to reduce the amount of colored ink used by the printer.

Bear in mind that this option will modify the colors as they come out at the printer. Also note that colors on the printed page may differ depending on various factors, such as the color itself, the printer and the paper.
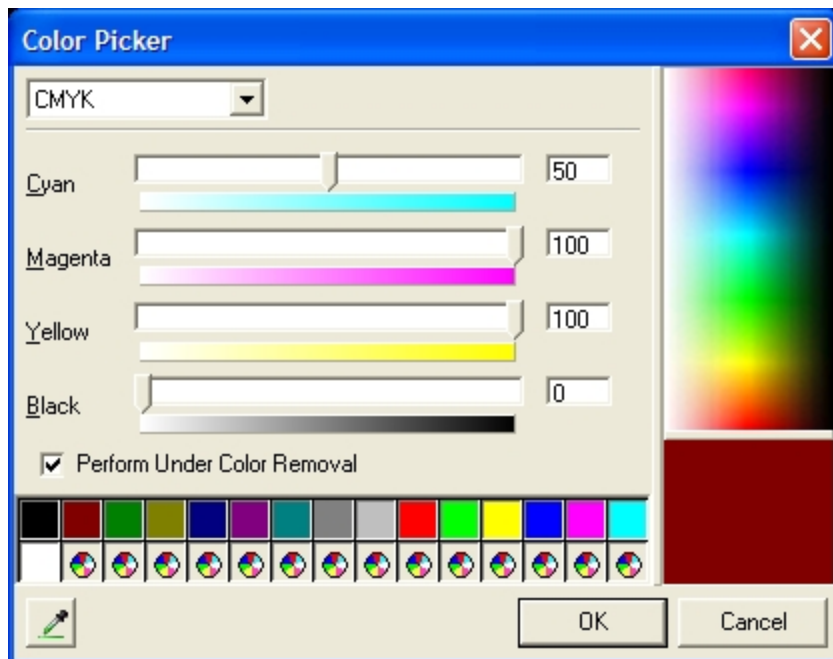
A. The new under color removal feature; B. The amount of ink the printer will use to print the selected color; C. The selected color

In the case of the red selected in the previous illustration, using the Remove background color feature saves a considerable amount of cyan, magenta and yellow ink, as shown below.
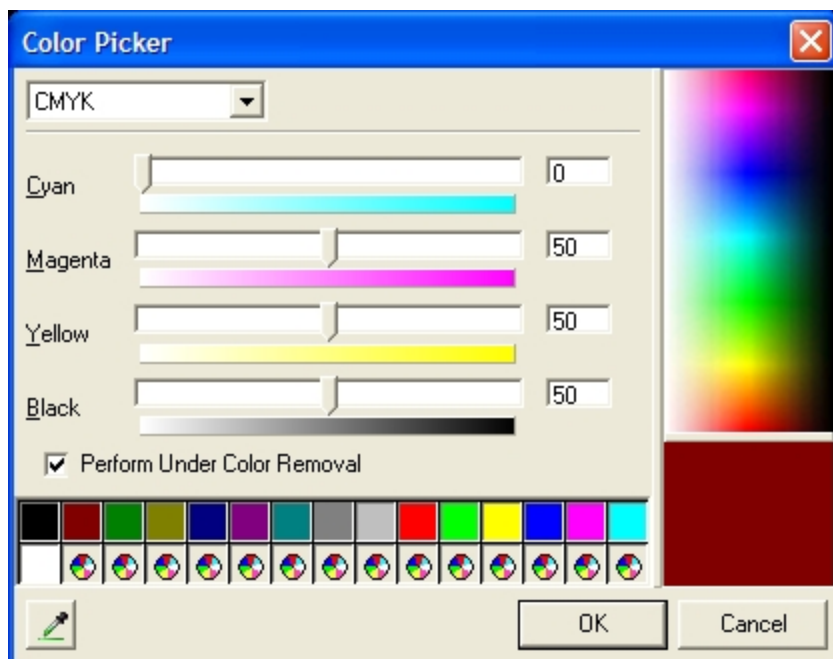


A. The Remove background color feature selected, colored ink is replaced by black ink when possible; B. The amount of cyan, magenta and yellow ink used to print the selected color is now 50% less and the amount of black ink is now 50%

## Named Colors

To display the Color Picker:

- In the properties dialog box of an object, click 🔳 .

To select a color using the color box:

1. In the color spectrum box, click on a point, or click and drag a point to a new color.
2. Release when the pointer is over the color you want.

To select a color from the palette:

- Click in the color pot that contains the color you require.

To select a color using a CMYK or RGB settings:

1. Choose the color model you want to use in the drop-down list.
2. Click and drag the sliders for each one of the basic colors.

To select a grayscale value:

1. Choose the GRAY color model.
2. Click and drag the slider to the new value.

To select a color that appears anywhere on the screen:

1. In the Color Picker, click 🖊️ .
2. Position the crosshair over the color you want to select and click.

To create a custom palette:

You create a custom palette whenever you make any change to the colors displayed by default in the color palette.

1. If necessary, change the current color to one you want to add to the custom palette.
2. Click in the current color and drag it to one of the color pots. The color pot may or may not be empty. A color wheel (●) indicates an empty color pot.
3. Enter the name of the color and click OK.
4. Repeat step 1 through step 3 for each color you want to add to your custom palette.

To reset the color palette to its default colors:

1. Right-click in any of the color pots in the palette.
2. Click **Reset Custom Colors to Defaults**.

## Create a Graybar Report

You use the Graybar Wizard to:

- Enter a name for the graybar report document.
- Associate a data file with the document.
- Select the PPD you want the document to use.
- Define the properties of the graybar page. These properties are the page format, page orientation, duplexing, and the height, color, and border of each bar in the band.

In PlanetPress Design, a graybar report document contains at most one graybar page. It can contain as many other pages as you need. You can create the graybar page using the Graybar Wizard, and any other pages you need by adding pages to the document once you exit the Graybar Wizard.

You use the Graybar Wizard once, and only once, to set up the graybar document. Once you exit the Graybar Wizard, you can design the rest of the graybar report document using any of the features available in PlanetPress Design. Note that the

graybar page you created using the Graybar Wizard contains a set of box objects, where each box object is a bar on the gray-bar page. You can edit the properties of, or delete, any of these box objects. If you want to change any of the bar and/or page properties you set in the Graybar Wizard, you must start a new document and begin a new session with the Graybar Wizard.

To create a graybar report document:

1. From the **PlanetPress Design Button**, choose **New**.
2. Choose **Tools** | **Application** | **Graybar Wizard**.
3. In the **Graybar Wizard**, click **Introduction** and enter a name for the graybar report document.
    **Document name**: Enter a name for the document. PlanetPress Design uses the name you enter here as the value of the Name box in the Document properties dialog box for this graybar report document. If you install the document on a printer, this is the name under which the printer stores the converted document. Note that when you convert a doc-ument for PlanetPress Suite Workflow Tools, the converted document bears the name of the PP4 file for the document, not the name you enter here.
4. In the **Graybar Wizard**, click **Select data file** and select the sample data file for the document.
    **Data file**: Displays the path of the sample data file associated with the graybar report document. Click the **Browse** button to the right of the box to use the Data Selector to select a sample data file. You can also set up the emulation you want to use with the graybar report in the Data Selector.
5. In the **Graybar Wizard**, click **Page setup**, select a PostScript Printer Definition (PPD) file, and define the default paper format and orientation for the graybar report document.
    **Printers**: Select the PPD for the graybar report document. PlanetPress Design uses the PPD you select here as the value of the Designed for box in the Document properties dialog box for this graybar report document. The contents of the PPD subfolder in the PlanetPress program folder determine the contents of this list. The PPD that appears by default here is the one selected in the Default printer box of the User Options dialog box.
    **Paper type**: Select the default paper format for the graybar page. The format that appears here by default is the one set in the User Options dialog box. The formats available depend on the PPD you selected in the Printers box. Note that you should not adjust the paper format for the graybar page once you exit the Graybar Wizard; the bands of the gray-bar page do not adjust to reflect any change.
    If you are printing in 2-up mode, the Default page size box in the Document properties dialog box determines the size of the paper on which the two pages print; the page size you select here determines the scaling required to fit the two pages on that paper size.
    **Paper orientation**: Select the orientation for the graybar page (Portrait, Landscape, Rotated portrait, Rotated land-scape). PlanetPress Design uses the paper orientation you select here as the value of the Paper orientation box in the Page properties dialog box for the graybar page. Rotated options rotate the paper 180 degrees, and can be useful when you are working with perforated paper or using an attachment created in another application. You rotate the document page instead of the attachment. Note that you should not adjust the paper orientation for the graybar page once you exit the Graybar Wizard; the bands of the graybar page do not adjust to reflect a change.
    **Duplexing**: Select the duplexing options for the document. The duplexing options available depend on the PPD you selected for the document. Note that duplexing options apply only to normal pages since these are the only pages that can print. If you want to print simplex and duplex in the same document, and your printer supports switching between simplex and duplex in the same job, you can set the duplexing options on a page-by-page basis using the Duplexing option in the Page properties dialog box. If your printer does not support switching between simplex and duplex in the same job, you can simulate the switch by setting the duplexing option here and printing a blank page on the back of each page that you want to print simplex. If you select Default, no command is issued to the printer regarding duplex-ing for this document; the document uses the printer setting in effect at execution time. Consult the PlanetPress Talk lesson related to duplex mode for an example of the latter approach.
6. In the **Graybar Wizard**, click **Margins** and set the top, bottom, right, and left page margins, and the border options for the graybar page.
    **Left**: Enter the left margin, relative to the left edge of the physical page.
    **Right**: Enter the right margin, relative to the right edge of the physical page.
    **Top**: Enter the top margin, relative to the top edge of the physical page.

**Bottom**: Enter the bottom margin, relative to the bottom edge of the physical page.

**Border**: Select to display a border along the margins of the graybar page. Use the Line width box to specify the width of the line the border uses.

**Line width**: Enter the width of the line you want to use as the border. Units are typographical points. This option is available only when you select Border.

7. Click **Bar definitions** and define each bar that makes up the band that repeats to create the graybar page.
   **Bar list**: Lists all the bars that make up the band, in the order in which they appear in the band. The band can contain a maximum of ten bars. The default band contains a gray and white bar. The height of a band is the sum of the heights of all bars. PlanetPress Design creates the graybar page by repeating this band from top to bottom on the page.

8. In the **Graybar Wizard**, click **Finish**.
   The graybar page contains a set of box objects, where each box object is a bar on the graybar page.
   The graybar page also contains a data selection object that displays a page of the sample data file or a set of data selection objects that together reference a record set. If the single data page fits neatly on to the graybar page, you may not need to adjust these data selection objects. If this is not the case you can adjust or delete any or all of these data selections, as well as add new data selection objects to the page.

9. Design the rest of the graybar report document using any of the features available in PlanetPress Design.

To add a bar:

1. If necessary, in the **Graybar Wizard**, click **Bar definitions**.
2. Click **Add Bar**.
3. In the Bar list, select the newly added bar, and adjust the properties for that bar.
   Bar Properties
   **Height**: Displays the height of the bar currently selected in the Bar list.
   **Color button**: Click to select the color for the bar using the Color Picker.
   **Color box**: View the current color for the bar.
   **Line**: Select to display a black border around the bar.
   **Line width**: Enter the width of the line the border uses.

To edit the properties of a bar:

1. In the **Bar list**, select the bar you want to edit.
2. Edit the height, color, and border properties for that bar.

To delete a bar:

1. In the Bar list, select the bar you want to delete.
2. Click **Remove Bar**.

## The Hex Viewer

The Hex Viewer is a tool for viewing the characters in the sample data file as hexadecimal values. ASCII characters appear on the right side of the Hex Viewer, and the corresponding hexadecimal values on the left side. This is useful when you are selecting an emulation or fine-tuning the size and structure of the data page.

> The Hex Viewer is also a standalone tool and can also be used outside of PlanetPress Design. It is located in the following folder:
> *C:\Program Files\Common Files\Objectif Lune\PlanetPress Suite 7\Tools*

**Working with the Hex Viewer**

**To open the Hex Viewer from PlanetPress Design:**

- From the Tools tab in the PlanetPress Suite Ribbon, click on **Hex Viewer**. The default active data file is automatically opened.
- In the Data Selector, click the Hex Viewer button. Note that this button is not available if you select a database emulation.

**To exit the Hex Viewer:**

- In the Hex Viewer, choose **File** | **Exit**.
- Click the X button at the top-right corner of the Hex Viewer.
- Press ALT+F4 on your keyboard.

**To open a different file in the Hex Viewer:**

- In the Hex Viewer, choose **File** | **Open** then navigate to the file you want to examine and click **Open**.

**To save a file from the Hex Viewer:**

- Choose **File** | **Save**.

**To save the sample data file as a binary or hexadecimal text file:**

1. Choose **File** | **Save As** and choose one of the following:
   - **Bin file**: Choose to save the sample data file as a binary file.
   - **Hex file**: Choose to save the sample data file as a text file, where each two-character text string is the hexadecimal code for a byte of data.
2. Use the **Save As** dialog box that appears to save the file.

**To adjust the view:**

- Choose **View** and set the option:
   - **Line size**: Choose the number of bytes of input data you want to display on each line. You can choose 16, 32, or 64 bytes per line.
   - **Column width**: Choose the number of bytes of input data you want to represent in each column of the hexadecimal display area. You can choose 1, 2, or 4 bytes per column.
   - **Caret style**: Choose a style for the pointer. You can choose among an empty rectangle (Full block), a left line (Left line), and an underscore (Bottom line). You can also choose Auto to switch from one style to another. When you switch from one style to another, the Hex Viewer also switches between insert and overwrite mode.
   - **Offset display**: Choose the representation for the byte offset numbers that appear on the left of the Hex Viewer. You can choose among Hexadecimal, Decimal, Octal, or None if you want to hide the byte offset numbers.
   - **Translation**: Choose the character set you want to use to represent the input data in the Hex Viewer. You can choose among ANSI, ASCII 7 bit, DOS 8 bit, Mac, and IBM EBCDIC CP 38.
   - **Grid**: Choose to toggle the Hex Viewer grid on or off.
   - **Show markers**: Choose to show or hide all markers. Markers appear in a column between the hexadecimal line number and the first hexadecimal value of the line. You create markers to make it easy to jump to specific lines in the file.
   - **Swap nibbles**: Choose to swap the nibbles in the hexadecimal representation of each byte of input data.
   - **Mask whitespaces**: Choose to turn the marking of carriage returns in the input data on or off. When it is on, an empty rectangle appears around each carriage return in the input data.

**To navigate through the input data:**

- Scroll through the file using the scrollbar on the right.

**To get a hard copy of the Hex Viewer presentation of the file:**

PLANETPRESS. SUITE

1. Be certain that you want to print the complete sample data file as it appears in the Hex Viewer. If it is quite large and you only want the first few pages, you may want to cancel the print job after those pages print. Alternatively, you might copy and paste the portion of interest into a separate Hex Viewer file and get a hard copy of that file.
2. Choose **File | Print Layout** and choose the representation you want to use for the values that represent the input data.
   - **Hex**: Choose to print the values that represent the input data as hexadecimal values.
   - **Decimal**: Choose to print the values that represent the input data as decimal values.
   - **Octal**: Choose to print the values that represent the input data as octal values.
3. Choose **File | Print Setup**.
4. Choose **File | Print**.

**To determine the hexadecimal value of a byte of input data or vice-versa:**

- Click on a value.
   - If you clicked on a hexadecimal value, a rectangle appears around the corresponding byte of input data.
   - If you clicked on a byte of input data, a rectangle appears around the first nibble of the corresponding hex-adecimal value.

**To search for a hexadecimal value or text string:**

1. Position the pointer in the Hex Viewer at the point at which you want to start the search.
2. Choose **Edit | Find**.
3. In the **Find Data** dialog box, enter the hexadecimal value or text string. If you are searching for a text string, you can specify the case sensitivity of the search.
   - To enter a hexadecimal value, type the value. You can also precede the value with a dollar sign ($).
   - To enter a text string, enter the letter t or T, followed by the text string. A lower case t specifies a case-sensitive search. An upper case T specifies a case-insensitive search.
4. Click **OK**.
5. If necessary, choose **Edit | FindNext** to find the next occurrence of the search term.

**To edit the sample data file:**

- Type data directly in the Hex Viewer. Click in the input data to enter byte values and in the hexadecimal values to enter nibble values. When you enter nibble values, each byte of data requires two keystrokes.

**To copy and paste a portion of the input data into a separate Hex Viewer file:**

1. Select the portion of the input data you want to paste into a separate file.
2. Choose **Edit | Copy**.
3. Choose **Edit | New** to display an empty file in the Hex Viewer.
4. Choose **Edit | Paste**.

**To undo commands:**

1. Choose **Edit | Undo**.
2. Repeat step 1 as many times as necessary. The Hex Viewer supports multiple Undo commands.

## Date and Time Format

What format should I use to enter dates and times?

To simplify things and to prevent errors, date and time formats have been standardized.

- Date are entered and displayed as yyyy/MM/dd (2007/06/13, for example).
- Times are entered and displayed using the 24 hour format as HH:mm:ss (3:38:54 PM, for example, is entered and

displayed as 15:38:54).

# Document Output and Preview

This chapter will explain the different methods of outputting a document. It covers previewing and soft-proofing your document before output, as well as printing to a printer and installing documents on a printer, from PlanetPress Design. It also covers sending a document to PlanetPress Workflow for further processing in a workflow.

It does not describe platform-specific procedures for inserting triggers, or how to create or run a PlanetPress Suite process. Consult the **Trigger and Data Capture Guide** and the **PlanetPress Workflow Tools User Guide,** respectively, for information on these topics.

## About Previewing and Printing

PlanetPress Design is a WYSIWYG (**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et) interface, which means the result in print should be very similar, if not identical, to what you see on the screen. You can also preview your documents using the Print Preview function (see "Preview a Document On Screen" (page 661)), print a soft proof (see "Generate a Soft Proof" (page 668)) or a print whole job using PlanetPress Design.

Once your design is stable, you can also send your document to the PlanetPress Suite Workflow Tools in order for it to be used in an automated workflow (see the PlanetPress Workflow Tools User Guide).

### Preview a Document On Screen

It is common to preview a document often during the creation process and it is strongly recommended you preview the final document before you install it.

When you preview a document, you preview one output type at a time. If your document contains more than one output type, you must preview each separately. You can preview the document with all data pages, or a range of the data pages in the sample data file for the document.

An on-screen preview displays the generated document as a PDF file on-screen. The Preview command gives you two options to generate PDFs: you can use the PlanetPress Design internal interpreter or you can use your system's default PostScript interpreter. Note that if you choose the second option, you must make sure that the system's default PostScript interpreter is correctly configured (if you are using Adobe Distiller, for example, the Output options, such as the Ask for PDF file destination, must not be selected).

If you are previewing a document that uses dynamic images that reference external images, remember that to see the referenced images in the preview, those images must be accessible from the environment in which the preview executes. Thus the PlanetPress Talk expression you entered in the Image box for each of the dynamic images that reference external images, must resolve to the correct pathnames for the external images in the preview environment.

The minimum time required to run a document for a preview is approximately 10 times longer on Windows 2000 than on Windows XP/2003.

Note that if you look at a document that uses one of the 35 standard PostScript fonts and that includes the Euro glyph, you are likely to find out that although this glyph is displayed at design time it does not show up in previews. If this is the case, faxes and archives generated using PlanetPress Suite Workflow Tools will also lack this glyph. Try printing to see if the printer is able to recognize and print this glyph.

To perform an on-screen preview:

1. From the **PlanetPress Design Button**, choose **Preview**.
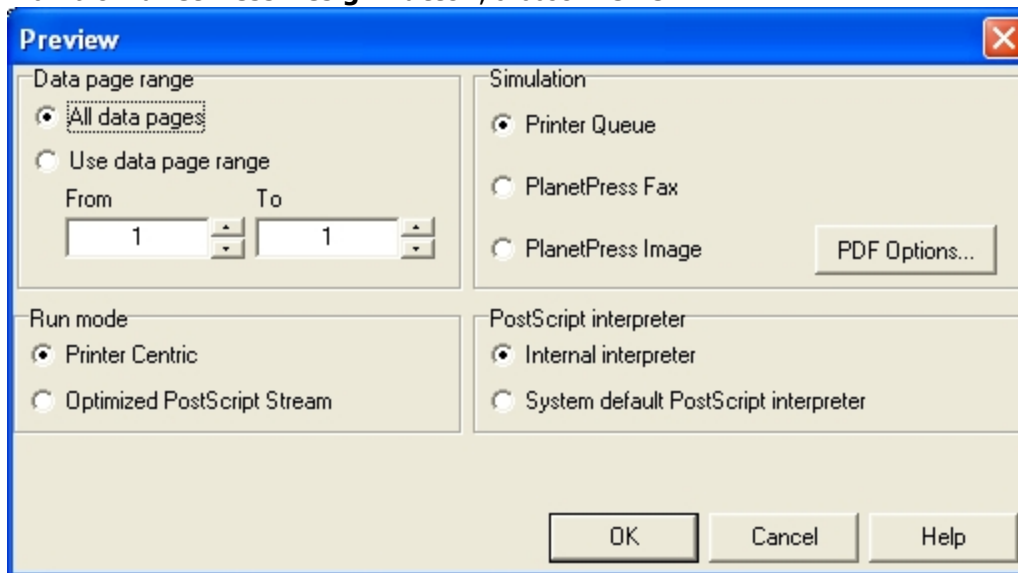


A. PlanetPress Image Options button

2. Select the data pages you want to use for the preview in the **Data page range** group.

   **All data pages**: Select to use all **data** pages (or in the case of a database emulation, all record sets) in the sample data file. It is important to understand that this refers to data pages and not document pages. The presence or absence of an individual **document** page in the preview depends on the output type (printer, PlanetPress Image, PlanetPress Fax) and the condition, if any, set on the page.

   **Use data page range**: Select to specify the range of data pages in the sample data file that you want to use for this preview. Use the From and To spin boxes to enter the range. It is important to understand that this refers to data pages and not document pages. The presence or absence of an individual **document** page in the preview depends on the output type and the condition, if any, set on the page.

   **From**: Enter the page number of the first data page in the range of data pages you want to use for the preview. You can type the page number or use the spin buttons to adjust the value. Remember that if you start document execution at a given data page, any effects the preceding data pages may have had on execution up to that point, are not available. This option is available only when you select Use data page range.

   **To**: Enter the page number of the last data page in the range of data pages you want to use for the preview. You can type the page number or use the spin buttons to adjust the value. This option is available only when you select Use data page range.

3. Select the type of output you want to preview in the **Simulation** group.

   **Printer Queue**: Select to preview the result of executing the document on a printer.

   **PlanetPress Fax**: Select to preview the result of executing the document in PlanetPress Fax.

   **PlanetPress Image**: Select to preview the result of executing the document in PlanetPress Image.

4. If you selected PlanetPress Image, click the **PDF Options** button to display the PDF Options dialog and set the display options for the PDF files.

**Zoom factor**: Select the zoom factor at which you want Adobe Acrobat or Adobe Reader to open the PDF file generated by the preview. Select Fit in window to adjust the zoom factor such that each page of the PDF occupies a full screen, or select a specific zoom factor. This is the same option available in the PDF options for the PlanetPress Image output task in PlanetPress Suite Workflow Tools. Consult the PlanetPress Suite Workflow Tools *User Guide* for more complete information.

**View**: Select the information you want Adobe Acrobat or Adobe Reader to display along with the pages of the PDF file generated by the preview. Select Page only to leave the tabs area to the left of the PDF pages empty. Select Bookmarks and page to display the contents of the Bookmarks tab alongside the PDF pages. Select Pages tab and page to display the content of the Pages tab alongside the PDF pages. Select Full screen to hide all screen contents except the PDF page, and expand the PDF page to the maximum size it can occupy on screen. You can press **ESCAPE** to restore the screen contents. Note that Full screen overrides the setting in the Zoom factor box. This is the same option available in the PDF options for the PlanetPress Image output task in PlanetPress Suite Workflow Tools. Consult the *PlanetPress Workflow Tools User Guide* for more complete information.

5.  Select your preferred run mode in the **Run mode** group. Note that this option is important if your document contains dynamic images that reference external images.
    **Printer centric**: Select to have PlanetPress Design execute the preview as if the document were run on the printer itself. Note that for screen previews, documents are always physically run on the host on which you are running PlanetPress Design.
    **Optimized PostScript Stream**: Select to have PlanetPress Design execute the preview using the Optimized PostScript Stream option. Note that the external images included in the document must reside in a folder on the host on which PlanetPress Design is running. Obviously the PlanetPress Talk expression you entered in the Image box for each of the dynamic images that reference external images, must resolve to the correct pathnames for the external images in the preview environment.

6.  Select the PostScript interpreter you want to use in the **PostScript interpreter** group.
    **Internal interpreter**: Select to use the PostScript interpreter built in to PlanetPress Design.
    **System default**: Select to use the PostScript interpreter your system uses as its default. Note that this option will use whatever application is associated with the **.ps** extension to create the preview file. If this application is not able to perform postscript conversions and to generate PDF files, the preview process will fail. Refer to your Windows documentation for information on how to associate filename extensions and applications.
    To verify the program associated with a PS file:
    - Open Windows Explorer.
    - Choose **Tools** | **Folder Options**.
    - Click the**File Types** tab.
    - Scroll to select the PS file type and the associated program. In this example, the associated program is Adobe® Distiller®. This program is appropriate for converting PS files to PDF files. Programs such as Microsoft® Word and Notepad cannot convert PS files to PDF files.

7.  If your document uses database emulation, set the refresh option.

**Refresh data from database**: Select to have PlanetPress Design refresh the sample data file by repeating the SQL query before creating the preview. Clear to have PlanetPress Design use the sample data file in its current state. This option is available only when your document uses database emulation.

**Refresh Metadata**: Select to have PlanetPress Design refresh the metadata file before creating the preview. Clear to have PlanetPress Design use the metadata file in the current state, without regenerating it.

8. Click **OK**.
   PlanetPress Design closes the Preview dialog box, performs the preview, and displays the resulting PDF file.

> The Objectif Lune watermark appears on all on-screen previews, unless an activated installation of PlanetPress Production can be found on the same computer or another computer on the same network subnet.

If you are using color management, recall that color in on-screen previews may not match color in printed output. The color settings of the PDF viewer you use for on-screen previews determine the color in an on-screen preview of a document, and thus whether that color matches that in the printed output.

## Previews of Documents that Use ASCII Emulation

What should I know about previews of documents that use ASCII emulation?

If your document uses the ASCII emulation, the appearance of any hard copy and on-screen previews you perform may differ from the appearance of the document in the Page area of the PlanetPress Design Program window. The following describes why and when this discrepancy can occur.

Discrepancies can occur when the PostScript interpreter that performs the preview and the internal code that displays the document in the Page area, do not have the same Read in binary mode setting. Thus discrepancies can occur in any of the following three cases.

| If you perform a: | And you: | A discrepancy occurs because: |
|---|---|---|
| Hard copy preview on a printer running in binary mode | Clear Optimized PostScript Stream in the Print dialog box. Clear Read in binary mode. | PlanetPress Design displays the document in the Page area with data in which it has performed end of line character replacements, while the printer prints the document with data that has not had any end of line character replacements. |
| Hard copy preview on a printer that does not run in binary mode | Clear Optimized PostScript Stream in the Print dialog box. Select Read in binary mode. | PlanetPress Design displays the document in the Page area with data that has not had any end of line character replacements, while the printer prints the document with data that has end of line character replacements. |
| On-screen preview | Clear Optimized PostScript Stream in the Print dialog box. Clear Read in binary mode. | PlanetPress Design displays the document in the Page area with data that has not had any end of line character replacements, while the PostScript interpreter performs the preview with data that has end of line character replacements. These PostScript interpreters always run in binary mode. |

The PlanetPress Suite Access Manager is available in both the PlanetPress and PlanetPress Suite Workflow Tools applications to facilitate the transfer of files among computer systems connected over a network. An exclusive service component called PlanetPress Suite Messenger allows PlanetPress Suite components to work together across the network and identifies work-

stations on which PlanetPress Suite products are installed (note that PlanetPress Suite Messenger 6 only works with version 6 components). In this way, users can share image files, documents, and jobs.

Use the Access Manager to assign permission for other computers to access your workstation.

## Preview a Capture-Ready document

PlanetPress Capture-Ready documents can be previewed using the regular Preview functions of PlanetPress Design. When you preview and print a Capture-Ready document, it contains a special reserved pattern for this purpose.

When signing a Capture-Ready document and docking the pen, the PGC file form the pen is placed in a special location in the PlanetPress Design folder. To view the result of signing the document, simply use the Preview function again and the ink data will be present on the preview.

Some important notes on previewing Capture-ready documents:

- Only one pattern is available and will be used. Even when printing multiple pages, the same pattern will be used on all the pages.
- When doing the second preview with the pen docked, only the first page will contain ink. In a multi-page preview, if you write on more than one page, all of the ink (of every page) will appear on the first page of the preview.
- Capture patterns have to be printed in full-quality at a minimum of 600DPI on a Laser printer. Ink savers, draft versions and fast printing features may cause patterns not to print correctly and not to be visible by the Anoto Digital Pen.
- When printing, disable any scaling or resizing feature of your PDF software.
- Because of the resolution change (96dpi on screen, 600dpi on the printer) and the way in which patterns are displayed, it is expected to see a different in size between the field as displayed from a PDF on screen and the printed version on paper. In order to test your design, make sure to actually print a preview to see the final result.
- Further considerations are present in the PlanetPress Workflow Tools User Guide.

For more information on PlanetPress Capture, see the following topics:

- "Capture Field Object" (page 166)
- PlanetPress Capture (PlanetPress Workflow User Guide)

## Print a Document Preview

It is common to preview a document often during the creation process and it is strongly recommended you preview the final document before you install it.

A hard copy preview prints the output the document yields when it executes on a printer. When you perform a hard copy preview, PlanetPress converts the document, adds a trigger to the PostScript code, and sends the document to the printer along with the contents of the sample data file. In a hard copy preview, you cannot preview PlanetPress Image or PlanetPress Fax output. You can preview the document on any printer available on your local system. The printer you use for the preview and the printer on which the document is destined to execute must both use the same, or a compatible, PPD file for the preview to be reliable. In a hard copy preview, you can also save the output destined for the printer to a file instead of sending it to the printer.

If you are previewing a document that uses dynamic images that reference external images, remember that to see the referenced images in the preview, those images must be accessible from the environment in which the preview executes. Thus the PlanetPress Talk expression you entered in the Image box for each of the dynamic images that reference external images, must resolve to the correct pathnames for the external images in the preview environment.

To print a document preview:

1. From the **PlanetPress Design Button**, choose **Print**.
2. In the **Select printers** list, select the printer or printers on which you want to perform the preview.
3. If you also want to save a copy of the converted document to a file, select **Print to file**.
4. Select the data pages you want to use for the preview.
   **All data pages**: Select to use all **data** pages in the sample data file. It is important to understand that this refers to data pages and not document pages. The presence or absence of an individual **document** page in the preview depends on the output type of the page and the condition, if any, set on the page.
   **Use data page range**: Select to perform the preview using a range of the data pages in the sample data file. Use the From and To spin boxes to enter the range. It is important to understand that this refers to data pages and not document pages.
   **From**: Enter the page number of the first data page in the range of data pages you want to use for the preview. You can type the page number or use the spin buttons to adjust the value. This option is available only when you select Use data page range.
   **To**: Enter the page number of the last data page in the range of data pages you want to use for the preview. You can type the page number or use the spin buttons to adjust the value. This option is available only when you select Use data page range.
5. Specify whether you want to execute the preview host-based or printer-based. This option is important if your document contains dynamic images that reference external images.
   **Optimized PostScript Stream**: Select to have PlanetPress Design execute the preview host-based. Clear to have PlanetPress Design execute the preview printer-based. If you clear Optimized PostScript Stream, the external images must reside on the printer on which you are printing the preview. If you select Optimized PostScript Stream, the external images must reside either on the host on which PlanetPress Design is running, or on the printer on which you are executing the preview. Obviously the PlanetPress Talk expression you entered in the Image box for each of the dynamic images that reference external images, must resolve to the correct pathnames for the external images in the preview environment. Executing a document host-based decreases the time the printer spends executing the document and can be useful if your concern is minimizing the time the document requires to execute on the printer.
6. In the **Number of copies** box, enter the number of copies of the preview you want to print, or use the spin buttons to adjust the value. This box is available only if you have at least one printer selected in the Select printers list.
7. If your document uses database emulation, set the refresh option.
   **Refresh data from database**: Select to have PlanetPress Design refresh the sample data file by repeating the SQL query before creating the soft proof preview. Clear to have PlanetPress Design use the sample data file in its current state. This option is available only when your document uses database emulation.
   **Refresh Metadata**: Select to have PlanetPress Design refresh the metadata file before creating the soft proof preview. Clear to have PlanetPress Design use the metadata file in the current state, without regenerating it.
8. Click **OK** to exit the Print dialog box and execute the preview.
   PlanetPress Design converts the document and adds a trigger to the PostScript code. It then sends the document to each printer you selected in the Select printers list, along with the pages (or record sets) of the sample data file you specified for the preview. The preview prints on each of those printers. If you selected a Host based execution, PlanetPress Design performs a partial execution of the document before sending it to each of the printers.
   If you selected **Print to File**, PlanetPress Design prompts you to specify the name of the file to which you want to save the converted document. Once you enter a file name and click **Save**, PlanetPress Design saves a copy of the converted document (a PostScript file).

## Print Using a Windows Driver

PlanetPress Design generates the job file and hands it over with the available print options to the Windows print driver, which takes the relay for the actual printing part.

When printing using a Windows Driver, PDF image resources will lose their transparency, because they are

rasterized before being printed. If you place a PDF with transparency on top of another object, the object in the back may be partially or completely hidden in your output.

To print using a Windows driver:

1. From the **PlanetPress Design Button**, choose **Print Using a Windows Driver**.
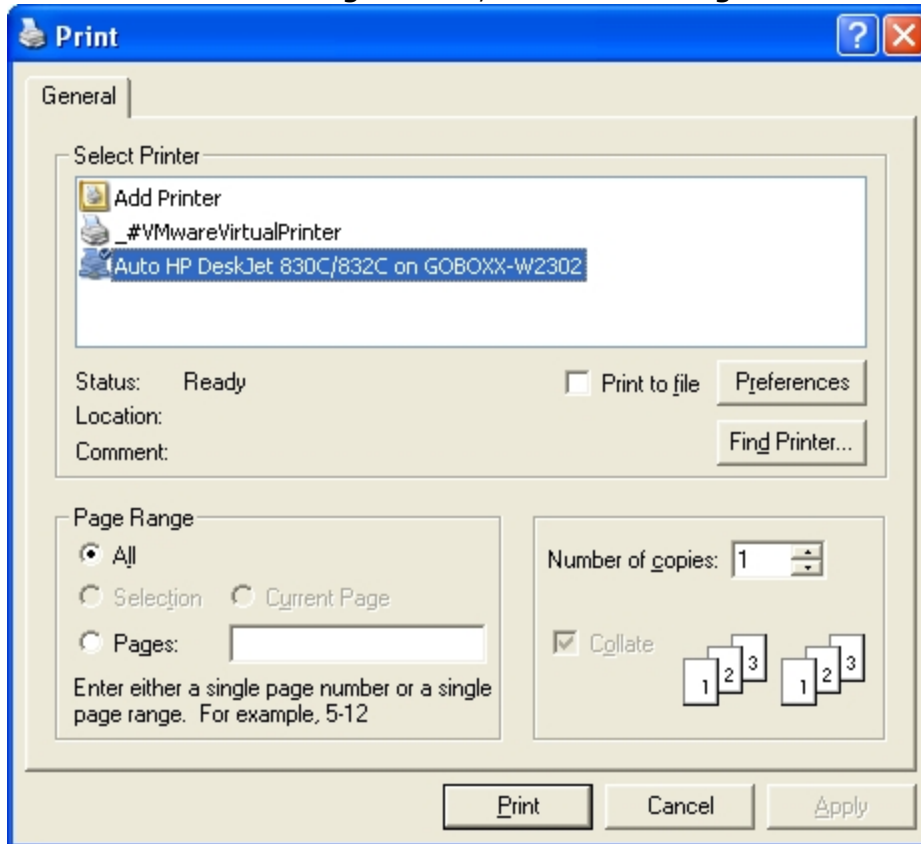


Windows displays the **Print** dialog box. Those options which are greyed out are not available when printing from PlanetPress Design. Any options, such as the paper size or media type, you may set by clicking the **Preferences** button are managed by the Windows print driver. Although PlanetPress Design provides available to the Windows print driver, it cannot ensure that these options will be correctly applied.

2. From the **Select Printer** box, select the printer to which you want to print.
3. To print to a file, select the **Print to file** option.
4. Select the data pages you want to use for the printout in the **Page range** group. Bear in mind that you are selecting data pages, so the number of actual pages being printed will vary based on such things as the number of data pages selected, the amount of data on each page, on the number of pages in the document, as well as page conditions.
   **All**: Select to use all **data** pages (or in the case of a database emulation, all record sets) in the sample data file. It is important to understand that this refers to data pages and not document pages. The presence or absence of an individual **document** page in the printout depends on the condition, if any, set on the page.
   **Pages**: Select to specify the range of data pages in the sample data file that you want to use for this printout. In the edit box to the right of the radio button, enter the number of the first data page followed by a hyphen and then by the number of the last data page in the range of data pages you want to use for the printout.
5. To print multiple copies, enter a value in the **Number of copies** edit box or use the spin buttons.
6. Click **OK**.
   The dialog box is closed and the print job is sent to the selected printer

Because of the way in which Windows Driver Output works, if a partial job has already been sent to the printer and an

> error occurs that stops the output in PlanetPress Design, the partial job will continue to print. This is because PlanetPress has no way to tell Windows to stop its printing and remove the job from its printer queue.

## Printing Using a Windows Driver

Can I print using a Windows driver and how does it differ from other printing methods?

PlanetPress Design lets you print your document and data using any Windows printer driver. Note that this option generates large print files and that it therefore does not provide the same level of speed and performance as the Optimized PostScript Stream and Printer Centric methods (the latter being the fastest and most efficient). See "Print Using a Windows Driver" (page 666).

## Generate a Soft Proof

Soft proofs are document previews that show no watermarks. Each soft proof has a maximum of 10 pages, so if your PlanetPress Design document has only 1 page, the soft proof can show as many as 10 records, but if your document has 10 pages, the soft proof can show only a single record.

As with document previews, soft proofs must be generated one output type at a time. If your document contains more than one output type, you must preview each separately.

Note that if you look at a document that uses one of the 35 standard PostScript fonts and that includes the Euro glyph, you are likely to find out that although this glyph is displayed at design time it does not show up in previews. If this is the case, faxes and archives generated using PlanetPress Suite Workflow Tools will also lack this glyph. Try printing to see if the printer is able to recognize and print this glyph.

To generate a soft proof:

1. From the **PlanetPress Design Button**, choose **Soft Proof**.



2. Select the data pages you want to use for the preview in the **Data page range** group.
   **All data pages**: Select to use all **data** pages (or in the case of a database emulation, all record sets) in the sample data file. It is important to understand that this refers to data pages and not document pages. The presence or absence

of an individual **document** page in the preview depends on the output type (printer, PlanetPress Image, PlanetPress Fax) and the condition, if any, set on the page.

**Use data page range**: Select to specify the range of data pages in the sample data file that you want to use for this preview. Use the From and To spin boxes to enter the range. It is important to understand that this refers to data pages and not document pages. The presence or absence of an individual **document** page in the preview depends on the output type and the condition, if any, set on the page.

**From**: Enter the page number of the first data page in the range of data pages you want to use for the preview. You can type the page number or use the spin buttons to adjust the value. Remember that if you start document execution at a given data page, any effects the preceding data pages may have had on execution up to that point, are not available. This option is available only when you select Use data page range.

**To**: Enter the page number of the last data page in the range of data pages you want to use for the preview. You can type the page number or use the spin buttons to adjust the value. This option is available only when you select Use data page range.
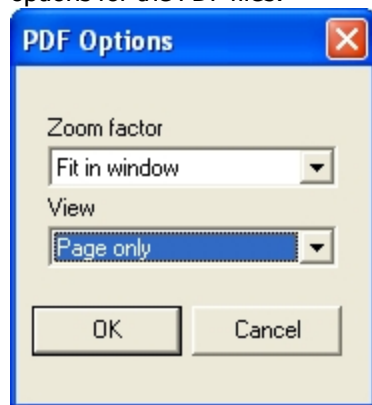
3. Select the type of output you want to preview in the **Simulation** group.

**Printer Queue**: Select to preview the result of executing the document on a printer.

**PlanetPress Fax**: Select to preview the result of executing the document in PlanetPress Fax.

**PlanetPress Image**: Select to preview the result of executing the document in PlanetPress Image.

4. If you selected PlanetPress Image, click the **PDF Options** button to display the PDF Options dialog and set the display options for the PDF files.

**Zoom factor**: Select the zoom factor at which you want Adobe Acrobat or Adobe Reader to open the PDF file generated by the preview. Select Fit in window to adjust the zoom factor such that each page of the PDF occupies a full screen, or select a specific zoom factor. This is the same option available in the PDF options for the PlanetPress Image output task in PlanetPress Suite Workflow Tools. Consult the *PlanetPress Suite Workflow Tools User Guide* for more complete information.

**View**: Select the information you want Adobe Acrobat or Adobe Reader to display along with the pages of the PDF file generated by the preview. Select Page only to leave the tabs area to the left of the PDF pages empty. Select Bookmarks and page to display the contents of the Bookmarks tab alongside the PDF pages. Select Pages tab and page to display the content of the Pages tab alongside the PDF pages. Select Full screen to hide all screen contents except the PDF page, and expand the PDF page to the maximum size it can occupy on screen. You can press **ESCAPE** to restore the screen contents. Note that Full screen overrides the setting in the Zoom factor box. This is the same option available in the PDF options for the PlanetPress Image output task in PlanetPress Suite Workflow Tools. Consult the *PlanetPress Suite Workflow Tools User Guide* for more complete information.

5. Select your output options in the **Soft Proof Options** group. Note that you may select all three available outputs for a single soft proof.

**Open in PDF viewer**: Select to have PlanetPress Design display the generated soft proof using your PDF viewer.

**Save to file**: Select to have PlanetPress Design save the generated soft proof. You can use the box to the right to specify a name and location. If no name is provided, the document's name is used. If no location is provided, a dialog box will be generated when the soft proof is actually generated.

**Send to Email recipient**: Select to have PlanetPress Design attach the generated soft proof to an Email message created within your default Email program.

6. If your document uses database emulation, set the refresh option.
   **Refresh data from database**: Select to have PlanetPress Design refresh the sample data file by repeating the SQL query before creating the preview. Clear to have PlanetPress Design use the sample data file in its current state. This option is available only when your document uses database emulation.
   **Refresh Metadata**: Select to have PlanetPress Design refresh the metadata file before creating the preview. Clear to have PlanetPress Design use the metadata file in the current state, without regenerating it.
7. Click **OK**.
   PlanetPress Design closes the Soft Proof dialog box and generates the soft proof. If you are using color management, recall that color in on-screen previews may not match color in printed output. The color settings of the PDF viewer you use for on-screen previews determine the color in an on-screen preview of a document, and thus whether that color matches that in the printed output.

## Convert a Document and Save It to a File

Compiling a document and saving it to a file can also be useful for debugging purposes or for acquainting yourself with the contents of a converted document. Before you convert a document, you may want to optimize it to create a smaller and more efficient converted document.

To convert a document with the PostScript code to install it on a printer:

1. Verify the Compilation options and Resource options settings in the Document properties dialog box are the ones you want for this document.
2. From the **PlanetPress Design Button**, choose **Send to | Printer**.
3. Select the printer location you want the converted document to reference.
   **Printer storage**: Select the printer location. When PlanetPress Design converts the document, it includes the PostScript code to install the document in the selected location.
4. If you want to include the PostScript code for producing a confirmation page in the converted document, in the **Confirmation page** box, select **Embedded confirmation page (less accurate)**. If you want to save this PostScript code to a separate file, select **Confirmation page**.
   Embedded confirmation page is useful if the printer on which you eventually want to install the document is not connected to the computer running PlanetPress Design.
5. Click **Save Printer File**.
6. Enter the name of the file in which you want to save the converted document and click **Save**.
   If you selected Confirmation page in the Confirmation page box, PlanetPress Design prompts you again, this time to specify the name of the file to which you want to save the PostScript code for producing the confirmation page.
7. In the **Send to Printers** dialog box, in the **Select printers** list, clear any selected printers. If any printers are selected when you exit the Send to Printers dialog box, PlanetPress Design installs the document on each of those printers.
8. Click **OK**.

To convert a document without the PostScript code to install it on a printer:

1. Verify the Compilation options and Resource options settings in the Document properties dialog box are the ones you want for this document.
2. From the **PlanetPress Design Button**, choose **Send to | Host**.
3. Click **Save Host File**.
4. Enter the name of the file in which you want to save the converted document and click **Save**.

To send a document to a UNIX or Linux print spooler system:

1. Verify the Compilation options and Resource options settings in the Document properties dialog box are the ones you want for this document.
2. From the **PlanetPress Design Button**, choose **Send to | Host**.
3. In the Host Type box, select Codehost BrightQ.
4. Set options for document output.

   **Host name or IP address**: Enter the host name or the IP address of the UNIX or Linux workstation. PlanetPress Design transfers the document to the UNIX or Linux workstation and print spooler system using TCP/IP. Be sure to specify the accompanying password.

   **Port:** The default port number is 5662. The port number may be modified under the specification of the network administrator. PlanetPress Design must connect to the specified TCP port and send a document conforming to the following specifications.

   **Password**: Enter the password for the workstation.

   **Document name**: Enter the file name for the converted document. This must be a file name, not a path. The name must be *no longer* than 10 characters.

   **Printer storage**: Select the printer location of the installed document. PlanetPress Design sends both the trigger and compiled document.

   **Force PostScript mode**: Select this option when the destination printer is used for printing jobs in other formats such as PJL, and when PlanetPress is not sending the output to the printer's hard disk. When this option is selected, PlanetPress Design inserts the following command statement [Esc]%-12345X@PJL ENTER LANGUAGE = POSTSCRIPT to precede the document as well as any other command statement.

   **Save Host File button**: When clicked, saves the document as a postscript file in a specified folder.

   **Send to Host button**: When clicked, sends the document from PlanetPress Design to BrightQ.

   **Sent to Printer button**: When clicked, sends the document only to the printer. Once you choose that option the window remains open.
5. To output the document, click **Save Host File**, **Send to Host**, or **Send to Printer**.

   A message appears indicating the document has been sent successfully, otherwise an error message appears. If you want a printer based document sent through Codehost BrightQ, first send the document to the printer, and then send the document to Codehost BrightQ.

# Install a Document

Once a document is ready to be used, it must be sent to a location where it will be used for merging with a data file and output. A document can be sent (or installed) in multiple locations.

Before sending your document, verify that the Conversion options and Resource options settings in the Document properties dialog box are the ones you want for this document.

## Install a document on one or more printers

1. From the **PlanetPress Design Button**, choose **Send to | Printer…**.
2. In the **Send to Printers** dialog box, set the options for the install.
   - **Select printers list**: Select each of the printers on which you want to install the document. If the printers on which you want to install the document require either a password, a hard drive path, or both, be sure you have set these properly in the Document properties dialog box (see Set Up a Document).If two or more of the printers you select require different passwords and/or hard drive paths, you must perform separate installs for each, making the necessary adjustments in the Document properties dialog box before performing each install.
   - **Printer storage**: Select the printer location. When PlanetPress Design converts the document, it includes the PostScript code to install the document in the selected location. The locations are Hard Drive, Flash and RAM.

- **Confirmation page**: Select the type of confirmation you want to receive when PlanetPress Design performs the installation. When you select either of these two options, you should be sure that the location where you want to install the document in fact exists.
  - Select **None** to prevent a confirmation page from printing after the install. In this case, nothing confirms the installation of the document. You can of course subsequently print a listing of all documents installed on a printer to determine whether or not the document installed.
  - Select **Confirmation page**, to have each of the selected printers print a confirmation sheet reporting the success or failure of the installation. PlanetPress Design sends two PostScript jobs to each printer. The first installs the document, the second verifies the installation and prints a confirmation page.
  - Select **Embedded confirmation page** to send the document installation and the confirmation printout as a single job instead of two separate postscript job.
3. Click **OK** to immediately send the document to the printer(s).
   You can also click on **Save Printer File**. PlanetPress Design will instead prompt you for a name of a file to which you want to save a copy of the converted document. Once you enter a file name and click **Save**, PlanetPress Design saves a copy of the converted document. If you also selected *Confirmation page* in the *Confirmation page* box, PlanetPress Design prompts you again, this time to specify the name of the file to which you want to save the PostScript code for producing the confirmation page.

## Send the document to one or more local PlanetPress Workflow servers

>>> The PlanetPress Messenger service must be running on both the PlanetPress Design and PlanetPress Workflow computers for this procedure to work.

1. From the **PlanetPress Design Button**, choose **Send to** | **PlanetPress Suite Workflow**.
   The Send to PlanetPress Suite Workflow interface appears.
2. The Host Name box displays a list of available PlanetPress Workflow servers. Put a checkmark next to each **Host Name** of the PlanetPress Suite Workflow server(s) you want to send your document to. If a Host Name is grayed out, it means that your computer has not been given permission to send documents to it (see Control Access to Your Locally Installed Services).
3. Click **OK**.

In this dialog, you can also do the following actions:

- Click the **Refresh** button at the bottom-left to scan the local network for PlanetPress Workflow servers other that the local machine.
- Click on the **Add Host** button at the bottom-left to add a new host if it is not detected or if the host resides outside of the current subnet. Enter the IP address of the server and click Refresh to make it available.
- Put a checkmark in Refresh Metadata to refresh the document's metadata for each of the document's data files before it is sent to PlanetPress Workflow.
- Put a checkmark in Attach a document preview to include a 10-page PDF preview when sending the document to PlanetPress Worfklow.

Depending on the version of PlanetPress Workflow, either a PTK or PTZ file will be sent. Note that the file format is determined by the version of the PlanetPress Workflow server that the document is being sent to, not the version of the local PlanetPress Design installation.

- In PlanetPress Suite 7.0 and older, a PTK file is sent. PTK files only include the document itself, they do not include any peripheral files.
- In PlanetPress Suite 7.1.0 up to 7.2.4, a PTZ file is sent to PlanetPress Workflow. The PTZ is a compressed format that includes the document in PTK format along with its data files, image resources and any metadata file that has been generated during the creation of the document.

- In PlanetPress Suite 7.3.0 and higher, the PTZ also contains a PDF preview of the document's output, multiple data files with their associated metadata and an XML file detailing the emulation used by the document.

You can verify that a given installation succeeded by navigating to the Documents folder of the appropriate PlanetPress Suite Workflow Tool installation and locating the **.ptk** file for the document. The **.ptk** file bears the same name as the PP7 file for the document. For example, if the name of your document is **invoice**, the name of the PP7 file is **invoice.PP7** and the name of the **.ptk** file is **invoice.ptk**.

## Save a PTK or PTZ file to send to a remote PlanetPress Workflow installation

1. From the **PlanetPress Design Button**, choose **Send to | PlanetPress Suite Workflow** .
2. In the Send to PlanetPress Suite Workflow Tools dialog, click **Save to file...** (previously Save PTK File ).
3. In the save dialog, a button lets you choose between the PTK and the PTZ format. Enter the name of the file in which you want to save the document and click **Save**. The produced PTK or PTZ file can be sent via any method and imported into PlanetPress Workflow.

## Send a document to PlanetPress iWatch or a UNIX or Linux CodeHost BrightQ print spooler system:

1. From the **PlanetPress Design Button**, choose **Send to | Host**.
2. In the Host Type box, select **Codehost BrightQ** or **PlanetPress iWatch**.
3. Set options for document output.
   - **Host name or IP address**: Enter the host name or the IP address of the target host. Be sure to specify the accompanying password.
   - **Port:** The default port number is 5662. The port number may be modified under the specification of the network administrator. PlanetPress Design must connect to the specified TCP port and send a document conforming to the following port specifications.
   - **Password**: Enter the password for the workstation.
   - **Document name**: Enter the file name for the converted document. This must be a file name, not a path. The name must be *no longer* than 10 characters.
   - **Send Group**: Defines what to send to the target host
     - **Trigger**: Sends the trigger definition to be used when printing.
       - **Printer storage**: Select the location where to send the file. In the case of CodeHost BrightQ, this is the only option enabled in the Send group, and both the trigger and document are sent together.
     - **Document**: Only sends the document to the host.
       - **Force PostScript mode**: Select this option when the destination printer is used for printing jobs in other formats such as PJL, and when PlanetPress is not sending the output to the printer's hard disk. When this option is selected, PlanetPress Design inserts the following command statement `[Esc]%-12345X@PJL ENTER LANGUAGE = POSTSCRIPT` to precede the document as well as any other command statement.
4. **Save Host File button**: When clicked, saves the document as a postscript file in a specified folder.
5. **Send to Host button**: When clicked, sends the document from PlanetPress Design to BrightQ.
6. **Sent to Printer button**: When clicked, opens the Send to Printer dialog (see above in this topic).

>>> If you want a printer based document sent through Codehost BrightQ, first send the document to the printer, and then send the document to Codehost BrightQ.

# Perform a Batch Conversion and/or Installation

You can have PlanetPress Design perform one or more of the following in a single operation:

- Install each document on one or more printers.
- Install each document in one or more installations of a PlanetPress Suite Workflow Tool.
- Install each document in a PlanetPress Design iWatch installation.
- Convert each document and save it to a file, along with the PostScript code for installing the document on the printer.
- Save the PTZ (or PTK) version of each document to a file.

To start a batch conversion and/or installation, from the **PlanetPress Design Button**, choose **Batch Send To**.

## To use the Batch Send To wizard

- In the Introduction screen, simply click **Next >>**.
- In **Select Documents**, select the documents you want to convert and/or install. To add a document, simply browse to its location on the left and drag it to the right part of the screen. Click **Next >>**.
  All of the subsequent conversion and/or installation operations you define in the wizard are performed on all of the documents you select here.
- If you want to install the documents on one or more printers, place a checkmark next to each printer you want to send to in the **Select printers** screen. Refer to Install a Document for the Send to Printer settings. Click **Next >>**.
- In **PlanetPress Suite Workflow**, select any PlanetPress Workflow workstation where you want to send the document(s). Refer to Install a Document for the Send to PlanetPress Workflow settings. Click **Next >>**.
- In PlanetPress iWatch, enter the host of any iSeries printer where you want to send the document(s). Refer to Install a Document for the Send to PlanetPress iWatch settings. Click **Next >>**.
- In **Send**, a list of operations will appear. Click **Next >>** to start the Batch Send To process.
  PlanetPress Design performs the batch conversion and/or installation you defined in the wizard, displaying status messages as it progresses through the operations.
- Click **Finish** to exit the wizard. The Batch Send To window disappears and returns to PlanetPress Design.

> ⏵⏵⏵ Batch Send To does not support sending to BrightQ hosts. To send to a BrightQ host, see Install a Document.

## Move a Document between PlanetPress Design Installations

*To move a document:*

1. Locate the PP7 file for the document you want to move.
2. If the document contains dynamic images that reference external images, verify it can access those images from the new location. If necessary, perform the necessary steps to move those images available to the document in its new location.
3. Copy the PP7 file to the new location.

## Trigger

What is a trigger?

A trigger is two lines of PostScript that immediately precede the input data, and "triggers" the execution of a document. The trigger puts the printer in PostScript mode, and tells the printer which document to launch.

You print a document installed on a printer by sending a trigger to the printer, followed by the input data. If you run your document into your PlanetPress Suite Workflow Tool, it takes care of inserting the trigger. If you run your document directly on a printer, you must manually insert the trigger at the head of the data stream.

# How a Variable Content Document Runs on a Printer

How does a variable content document runs on a printer?

If you run a document installed on a printer directly, you send a trigger, followed by the input data, to the printer. If you run a document into your PlanetPress Suite Workflow Tool, the document may or may not be installed in the printer, depending on the PlanetPress Suite process you created for the document. If it is not installed in the printer, the PlanetPress Suite Workflow Tool sends the document, followed by the trigger, followed by the input data, to the printer. If it is already installed in the printer, the PlanetPress Suite Workflow Tool sends the trigger, followed by the input data, to the printer.

When the trigger arrives at the printer, it puts the printer in PostScript mode, gives it the name and location of the document in the printer, and tells it to launch the document. The document launches, takes control of the printer, initializes all its global variables and starts processing its data stream. The document then cycles through three distinct processing phases for each data page in the data stream until it reaches the end of the data stream, at which point it relinquishes control of the printer.

The three phases are:

- "Phase 1: Data Reading" (page 675)
- "Phase 2: Global Condition Resolution" (page 675)
- "Phase 3: Document Page Printing" (page 675)

## Phase 1: Data Reading

In phase 1, the document reads a complete data page and loads it in the data page buffer.

## Phase 2: Global Condition Resolution

In phase 2, the document runs any attachments you selected to run before the document prints, and then resolves all **global** conditions for the current data page.

## Phase 3: Document Page Printing

In phase 3, the document processes each page of the document. It uses the conditions it resolved in phase 2 to determine whether to run and print the page and, if it prints the page, which objects to include in the output. In this phase it also evaluates all local conditions as it encounters them. Thus the document evaluates conditions at two distinct points in time. It evaluates all global conditions immediately after it fills the data page buffer, and maintains the results in memory for reference as it merges the data page. It evaluates all other conditions as it prints each page of the document.

When it finishes processing all pages of the document, it re-initializes, empties the buffer and reads the next data page.

# Techniques for Inserting Triggers

What are the common techniques for inserting a trigger?

How you create and insert a trigger is operating system dependent. There are four common techniques to insert a trigger manually:

1. Manually concatenating two files
   With this technique, you add a trigger by concatenating two files where the first contains the trigger and the second contains the input data. You send the concatenated file to the printer using the DOS Copy command or the file transfer protocol (FTP).
2. Setting up the print server to automatically insert triggers
   This technique works with a print server running either Novell or Windows NT. You create a print queue or print device for each document installed in the printer, and associate the appropriate trigger for the document with the queue you set up for it. All queues point to the same physical printer. When you send a job to that queue, the server automatically inserts the trigger associated with that queue before it forwards the printer job to the printer.
3. Setting up the host to automatically insert triggers
   This is the same technique as setting up the print server to automatically insert triggers. The only difference here is that you set up the queues on the host on which the input data resides. The host inserts the trigger ahead of the spool file when it sends the print job to the printer. This technique does not work with all hosts.
4. Including the trigger in application output
   With this technique, you modify the output of the application that generates the print file so that it adds a trigger for the appropriate document. It is important to understand that this hampers your ability to print these jobs using other printers since two additional lines are added to the print file.

**Printer-Specific Control Characters**

You can precede a trigger with printer-specific control characters. The most common reason to do so is to ensure the printer receives the job you send as a new job.

A printer expects each job that it handles to end with a special character that tells the printer it has reached the end of the input data. Until the printer receives this special character, it continues to process all input it receives as part of that job. If there is no input, the printer waits for a defined period of time, then times out and proceeds to the next job. If a new job arrives during the period of time the printer is waiting for input, the printer does not recognize it as a new job; rather it processes it as input for the current job.

It is thus common practice to include an end of job character at the beginning of the trigger to ensure that the printer recognizes your job as a new print job. For certain printers, <CTRL D> or ASCII 04 is a valid end of job character, while more recent printers require a Printer Job Language (PJL) sequence such as <ESC>%-12345X<CR><LF>.

As an example, the following trigger includes <CTRL D> as an end of job character:

<CRTL-D>%!PS-Adobe <CR><LF>

run INVOICE <CR><LF>

# Trigger Syntax

What is the syntax of a trigger?

In all syntax descriptions in this section, italics denote a variable, square brackets indicate the element is optional, <CR> denotes a carriage return and <LF> denotes a line feed.

The general syntax for the first line of the trigger is the same for all triggers.

[ printer-specific_commands ] %!PS-Adobe <CR><LF>

---

The first line of the trigger uses the string **"%!PS-Adobe"** to put the printer in PostScript mode. It may also include printer-specific commands.

The syntax of the second line of the trigger depends on where the document is installed in the printer: on the hard drive, in RAM, or in Flash memory. The second line, written in PostScript, tells the printer the name and location of the document, and it launches the document. If you are using version numbers in your documents, this line also contains the version number.

## Trigger Syntax for Documents Installed on a Hard Disk

The general syntax of a trigger for a document that resides on the printer's hard drive is:

[ printer-specific_commands ] %!PS-Adobe <CR><LF>

[[ (location_of_document) ] run ] name_of_document <CR><LF>

In this example, a document named INVOICE resides on the printer's single hard drive:

%!PS-Adobe <CR><LF>

(INVOICE) run INVOICE <CR><LF>

If the printer has more than one hard disk, you must include the name of the hard disk. In this example, the name of the hard disk is "presswork" and the name of the document is INVOICE.

%!PS-Adobe <CR><LF>

(%presswork%INVOICE) run INVOICE <CR><LF>

## Trigger Syntax for Documents Installed in RAM

The general syntax of a trigger for a document that resides in the printer's RAM is:

[ printer-specific_commands ] %!PS-Adobe <CR><LF>

name_of_document <CR><LF>

An example of the trigger for a document named FORMLETTER that resides in the printer's RAM:

%!PS-Adobe <CR><LF>

FORMLETTER <CR><LF>

## Trigger Syntax for Documents Installed in Flash Memory

The general syntax of a trigger for a document that resides in the printer's Flash memory is:

[ printer-specific_commands ] %!PS-Adobe <CR><LF>

(%flash%name_of_document) run name_of_document <CR><LF>

An example of the trigger for a document named PAYROLL that resides in the printer's Flash memory:

%!PS-Adobe <CR><LF>

(%flash%PAYROLL) run PAYROLL <CR><LF>

Note that using this syntax on some printers that have only Flash memory (no hard disk) may return an error messages saying that the file or the Flash device cannot be found. This can usually be fixed by using the trigger syntax for documents

installed on the printer's hard disk (see above). With these types of printers, the syntax for documents that reside in the printer's Flash memory must only be used if the printer has *both* a hard disk *and* Flash memory.

## Run a Document Installed on a Printer

If you installed your document directly on a printer, you print it by sending a trigger to the printer followed by the input data. The **Trigger and Data Capture Guide** provides platform-specific procedures for creating the appropriate trigger and inserting it at the head of the data stream you send to the printer.

Most PostScript printers also support Printer Control Language (PCL). For this reason, you should configure your printer to detect the language of the incoming job and automatically switch to the appropriate language. This is particularly important in environments where people send both PCL and PS jobs to the printer. Consult your printer manual for help configuring the printer.

To run a document installed on a printer:

1. Create the trigger.
   To create the trigger, you must know the operating system of the host on which the input data resides, and how that host is connected to the printer (direct connection, serial line, print server).
   You also need to determine the technique you intend to use to create the trigger and insert it at the head of the data stream.
2. Send the trigger, followed by the input data, to the printer.

## Run a Document Installed in a PlanetPress Suite Workflow Tool

Consult the *PlanetPress Workflow Tools User Guide* for complete information on creating and executing a PlanetPress Suite process.

To run a document in a PlanetPress Suite Workflow Tool:

1. Install the document into the PlanetPress Suite Workflow Tool.
2. Create a PlanetPress Suite process to run your document.
3. Run the PlanetPress Suite process.

When you install a document on a PlanetPress Suite Workflow Tool server, you should make sure the fonts included in the document are actually available on the PlanetPress Suite Workflow Tool server. To install TrueType fonts, use the standard Windows procedure. To install PostScript fonts, use the **Install PostScript Font** command in your PlanetPress Suite Workflow Tool.

## Run a Document that Uses a Database Emulation

You cannot install a document that uses database emulation on a printer and send a trigger along with the data to have the document print. Only PlanetPress Design and PlanetPress Suite Workflow Tools can perform the necessary conversion of the data before sending it to the document.

In most cases you run a document that uses database emulation in a PlanetPress Suite Workflow Tool. This makes it possible to automate the execution of the document, and ensures that the data sent to the document always reflects the contents of the database at the time the document prints. Consult the *PlanetPress Workflow Tools User Guide* for help printing a document that uses a database emulation.

You can also use the hard copy preview capability in PlanetPress Design to print the document with all or some of the database. In this case, in the hard copy preview options, you should select Refresh data from database to update the contents of the sample data file before executing the preview. This ensures the preview reflects the latest changes to the database.

## Run Several Documents as a Single Job

If you have several documents to run on a printer, and you are not using the PlanetPress Suite Workflow Tools to print your documents, you may find it convenient to send a single job to the printer that contains the triggers and input data for all those documents. The single job has the format:

**trigger1data1trigger2data2trigger3data3**...

Embedded within this single job are End Of Job (EOJ) strings that permit each document to detect the end of its input data. These strings prevent each document in the job from treating everything that follows it as its own input data.

You define the EOJ string you want a given document to use. You may choose to define a different EOJ string for each document, or to use the same EOJ string for all documents or for all documents of a certain category. The choice you make depends on your environment and preferred way of working. If you choose a different EOJ string for each document or category of document, you must pay close attention to how you construct the single job to ensure you specify the proper EOJ string for a given document.

You can embed the EOJ strings in the single job in one of two ways: either within the triggers or at the end of input data.

This section describes how to construct a job by embedding EOJ strings within a trigger and how to construct a job by appending EOJ strings to the end of input data. It also includes a procedure that explains how to define an EOJ string for a document.

## Troubleshoot Execution Problems

You should also be aware that there are two printer settings that can help you troubleshoot a problematic document.

1. Print PostScript Error
   If you enable this setting, the printer prints an error page if it encounters an error during document execution. The error page reports the offending PostScript command and the status of the print stack. Although these errors are terse, they are sometimes all you need to pinpoint and resolve the problem. You can set this printer setting from PlanetPress Design.
2. Print in Hexadecimal Mode
   Most printers let you print the incoming data exactly as it arrives at the printer, as hexadecimal values. This is useful for determining exactly what characters the printer receives and for spotting unexpected characters in the data stream, or characters that are not arriving in the proper order. The following table shows the hexadecimal codes the printer dumps for a carriage return followed by a line feed, and a line feed followed by a carriage return, respectively. The equivalent ASCII codes appear on the right.

   | Hexadecimal | ASCII |
   | --- | --- |
   | 0A 0D | LF CR |
   | 0D 0A | CR LF |

   Because a document page can generate hundreds of pages of hexadecimal code, if you print in hexadecimal mode, you should be prepared to cancel the print job once the first few pages have printed.

**Objectif Lune Printer Driver (PS)**

**Introduction**

The Objectif Lune Printer Driver (PS) allows end-users to print directly to the PlanetPress Suite Workflow Tools from any Windows application, by using the familiar **File|Print** option. At the other end, PlanetPress Office and PlanetPress Production specifically can capture the incoming stream and convert it internally into a PDF file along with its metadata.

Although it is available with every PlanetPress Suite Workflow Tools, this feature becomes even more useful in environments

where the Document Input emulation is available (with PlanetPress Office or PlanetPress Production).

### Install a Objectif Lune Printer Driver (PS)

The Objectif Lune Printer Driver (PS) is automatically installed during the PlanetPress Suite setup, along with a default Windows Printer Queue called *PlanetPress Printer*.

### Install a Windows Printer Queue using the Objectif Lune Printer Driver (PS)

A Windows Printer Queue using the Objectif Lune Printer Driver (PS) can be installed from the PlanetPress Design Data Capture window.

Creating a new Windows printer queue from PlanetPress Design:

1. Start PlanetPress Design.
2. Choose **Tools | Data Capture | Windows Queue Input**.
3. Click **New**.
4. Enter a **Name** for the printer queue.
5. Click **OK**.

Every new Windows printer queue using the Objectif Lune Printer Driver (PS) is shared by default. Once such a shared queue is created, end-users can install it on their own computer by going through the same steps they would when installing a new remote printer in their Operating System. By default, connecting to a shared printer will automatically result in the Objectif Lune Printer Driver being downloaded to the connecting host.

### Printer Properties setup

A printer queue using the Objectif Lune Printer Driver (PS) can be configured to produce one of 3 different types of datafiles: EMF, PostScript, or PDF.

Possible printer properties settings, along with the datafile type it will produce:

### Spool Print Jobs in EMF Format:

- This will create an EMF datafile.
- This format is usally reserved for use with the Windows Print Converter action plugin (see PlanetPress Suite Workflow Tools documentation).
- This format can be obtained using any PlanetPress Suite Product; PlanetPress Design or any of the PlanetPress Suite Workflow Tools.

### Spool Print Jobs in RAW Format:

- This will create a PostScript datafile when the option **Create PDF (with Medatada)** is *unchecked*.
    - This format can be obtained using any PlanetPress Suite Product; PlanetPress Design or any of the PlanetPress Suite Workflow Tools.
- This will create a PDF datafile when the option **Create PDF (with Medatada)** is *checked*.
    - This format can be obtained using PlanetPress Design and PlanetPress Office or PlanetPress Production.

By default, the **Create PDF** option is:

- Checked if the incoming stream has been produced with the Objectif Lune Printer Driver.
- Unchecked if the incoming stream comes from some other PostScript Driver.
- Grayed out and unchecked if the incoming stream is not PostScript.

**Data Capture from PlanetPress Design**

Once a shared Windows printer queue using Objectif Lune Printer Driver (PS) is installed on both the server and the client sides, data capture can be achieved the same way as with any other Windows printer queues.

**Data Capture from PlanetPress Design**:

1. Start PlanetPress Design.
2. Choose **Tools | Data Capture | Windows Queue Input**.
3. Select a Windows print queue using the Objectif Lune Printer Driver (PS) from the drop-down list.
4. Click **Capture**.
5. Start the windows application from which you want to capture data.
6. Open your selected document.
7. Click **File | Print**.
8. Choose the same Windows print queue as in step 4.

Note that steps 5-7 can be performed at any time, even if PlanetPress Design is not yet in data capture mode. This is because every Windows printer queue using Objectif Lune Printer Driver (PS) is paused by default. Once the data capture has started, it captures the first queued job. If there are more than 1 job, the user will have to use the data capture button again.

**PDF Creation Parameters**

PDF files retrieved from a Windows print queue using Objectif Lune Printer Driver (PS) have the following properties:

- PDF 1.4
- Optimized PDF (subject to change)
- No down-sampling of images

These settings are pre-configured and cannot be changed by the user.

**About Metadata**

Metadata files are files containing information on the job itself rather than containing the job per se. A job sent to the Objectif Lune Printer Driver (PS) creates its own metadata, allowing users to retrieve relevant information, such as, for instance, the time and date the print request was sent. For more on this, see the **"Working with Metada" (page n)**documentation pages.

# Keyboard Shortcuts

This appendix provides a complete list of all keyboard shortcuts available in PlanetPress Design6. Most Common User Access (CUA) shortcuts, the shortcuts available on most personal computers, also work in PlanetPress Design6. Where a CUA short-cut conflicts with a PlanetPress Design shortcut, the PlanetPress Design shortcut takes precedence.

## PlanetPress Design General

"Exit PlanetPress Design" (page 683)

"Use the Help System" (page 683)

"Adjust the Zoom" (page 686)

"Show or Hide Areas of the Program Window" (page 684)

"Work with Hierarchies" (page 684)

"Work in the Document Structure Area" (page 684)

"Work with Documents" (page 684)

"Preview and Install Documents" (page 685)

"Work with Pages" (page 685)

"Use Basic Editing Commands" (page 686)

"Work in the Data Pane" (page 686)

"Work with the Data File" (page 687)

"Work with Data Selections" (page 687)

"Work with Objects" (page 689)

"Work in the Text Properties of a Text Object" (page 690)

"Use the Hex Viewer" (page 688)

### Exit PlanetPress Design

| Use: | To: |
|------|-----|
| CTRL+Q | Quit PlanetPress Design. |

### Use the Help System

| Use: | To: |
|------|-----|
| F1 | Display the online help manual. |

**PLANETPRESS. SUITE**

## Show or Hide Areas of the Program Window

| Use: | To: |
|---|---|
| SHIFT+CTRL+T | Show/hide the Document Structure area. |
| SHIFT+CTRL+I | Show/hide the Object Inspector. |
| SHIFT+CTRL+M | Show/hide the PlanetPress Talk Messages area. |
| SHIFT+CTRL+P | Show/hide the Data Pane. |

## Work with Hierarchies

| Use: | To: |
|---|---|
| CTRL+T | Move the focus to the hierarchical view. For example, in the Program window, the focus moves to the Structure area. In a properties dialog box, the focus moves to the hierarchical view on the left side of the dialog box. In the PlanetPress Talk Editor, it moves to the Commands area. |
| UP ARROW, DOWN ARROW | Move up or down in the hierarchy. For example, move up or down in the Structure area, in the Commands area of the PlanetPress Talk Editor, or in the hierarchy on the left side of a properties dialog box. |
| LEFT ARROW, RIGHT ARROW | Expand an item or move left one level (LEFT ARROW), or collapse an item or move right one level (RIGHT ARROW). This works in the Structure area and in the Commands area of the PlanetPress Talk Editor. |

## Work in the Document Structure Area

| Use: | To: |
|---|---|
| SHIFT+UP ARROW, SHIFT+DOWN ARROW (when at least one element is selected) | Add the next element above or below the currently selected elements, to the selection. Recall that you can only add elements to the selection that are at the same level in the hierarchy. |
| CTRL+A | Select all elements at the same level in the hierarchy. |
| F2 (after selecting an element) | Rename the element. |

## Work with Documents

| Use: | To: |
|---|---|
| CTRL+N | Create a new document. |
| CTRL+O | Open an existing document. |
| CTRL+S | Save the current document. |
| CTRL+SHIFT+S | Save the current document under a new name. |
| F4 | Display the Document properties dialog box. |
| F5 | Refresh the document. PlanetPress Design reads anew the contents of the PP4 file and updates the Program window. This includes a refresh of the Structure area (collapses all open elements into the default view of the Structure area). The read of the PP4 file includes a read of the sample data file. |

## Preview and Install Documents

| Use: | To: |
|---|---|
| F9 | Perform an on-screen preview. |
| CTRL+P | Perform a hard copy preview. |
| CTRL+ALT+P | Install the document on one or more printers. |
| CTRL+ALT+W | Install the document in one or more installations of PlanetPress Suite Workflow Tools. |
| CTRL+ALT+H | Install the document on one or more hosts. |

## Work with Pages

| Use: | To: |
|---|---|
| F6 | Display the Page properties dialog box for the current page. |
| UP ARROW, DOWN ARROW (in the Page area) | Scroll the page up or down in the Page area. |
| MOUSE WHEEL | Scroll the page up or down in the Page area. |
| SHIFT+MOUSE WHEEL | Scroll the page left or right in the Page area. |
| CTRL+F10 | Switch to the Work tool. |
| CTRL+F11 | Switch to the Hand tool. |
| CTRL+F12 | Switch to the Zoom tool. |
| CTRL+SPACEBAR (in the Page area) | Switch to the next tool in the sequence for the duration of the press on the spacebar. For example, if the current tool is the Work tool, CTRL+SPACEBAR switches to the Hand tool. You could then use the Hand tool to move the page. When you release the spacebar, you return to the Work tool. Note that you can only move to the next tool in the sequence; you cannot cycle through the sequence by pressing CTRL+SPACEBAR repeatedly. |

## Adjust the Zoom

| Use: | To: |
|---|---|
| Plus sign (+) sign on numeric keypad | Zoom in on the Page area using the Zoom factor set in the User Options dialog box. If the pointer is over the Page area, PlanetPress Design centers the zoom around the pointer. |
| SHIFT+ plus sign (+) key on numeric keypad | Zoom in on the Page area using the Fine zoom factor set in the User Options dialog box. If the pointer is over the Page area, PlanetPress Design centers the zoom around the pointer. |
| Minus sign (-) key on numeric keypad | Zoom out on the Page area using the Zoom factor set in the User Options dialog box. If the pointer is over the Page area, PlanetPress Design centers the zoom around the pointer. |
| SHIFT+minus sign (-) key on numeric keypad | Zoom out on the Page area using the Fine zoom factor set in the User Options dialog box. If the pointer is over the Page area, PlanetPress Design centers the zoom around the pointer. |
| CTRL+SHIFT+HOME | Adjust the zoom and page position such that the width of the page fits the current width of the Page area. |
| CTRL+HOME | Adjust the zoom and page position such that the entire page is visible within the current size of the Page area. |
| Right-click/left-click (in the Page area when using the Zoom tool) | Zoom in (left-click) or zoom out (right-click) on the page using the Zoom factor set in the User Options dialog box. |
| SHIFT+right-click, SHIFT+left-click (in the Page area when using the Zoom tool) | Zoom in (left-click) or zoom out (right-click) on the page using the Fine zoom factor set in the User Options dialog box. |
| CTRL+F12 | Switch to the Zoom tool. |

## Use Basic Editing Commands

| Use: | To: |
|---|---|
| CTRL+Z | Undo the most recent command. Repeat to move backwards through the sequence of commands entered to date, undoing one command at a time. |
| CTRL+SHIFT+Z | Reverse the effect of the most recent undo command. Repeat to move backwards through the sequence of undo commands entered to date, reversing the effect of each command. |
| CTRL+X | Cut the currently selected items. |
| CTRL+C | Copy the currently selected items. |
| CTRL+V | Paste the currently copied or cut items. |
| ESC | Cancel the current cut. |
| DELETE | Delete the currently selected items. |

## Work in the Data Pane

**Note**: Some of these shortcuts may not apply when using the PDF emulation.

| Use: | To: |
|---|---|
| HOME END | Move the currently active cell to the first cell of the current line (HOME), or the last cell of the current line (END). This shortcut collapses the current data selection to include only the currently active cell. |
| PAGE DOWN PAGE UP | Move the currently active cell forward (PAGE DOWN) or backward (PAGE UP) one screen, where the size of a screen is defined as the number of lines currently visible in the Data Pane. Thus you can adjust the size of a screen by resizing the Data Pane. This shortcut collapses the current data selection to only the currently active cell. |
| SHIFT+ARROW | Add cells to, or remove cells from, a data selection. |

| Use: | To: |
|---|---|
| ALT+ARROW or CTRL+SHIFT+ ARROW | Move the current data selection region within the Data Pane. Note that this does not work with database emulation. |
| ARROW | Move the currently active cell. This shortcut collapses the current data selection to only the currently active cell. |

## Work with the Data File

| Use: | To: |
|---|---|
| SHIFT+PAGE UP SHIFT+PAGE DOWN (anywhere in the Data Selector, or after you click in a Data page box) | Move forward (SHIFT+PAGE UP) or backward (SHIFT+PAGE DOWN) one page (or record set in the case of a database emulation) in the data file. |

## Work with Data Selections

| Use: | To: |
|---|---|
| ALT+ARROW (after selecting one or more data selection objects on the document page) | Move the data selection in a data selection object. The size of the data selection remains stable; the shortcut changes the position of the selection on the data page. This adjusts the values of the line and/or column properties (or, in the case of a database emulation, the Field name, and/or the From/To record properties) of the data selection. Use the UP ARROW to move the selection up one line (or record), the DOWN ARROW to move the selection down one line (or record), the LEFT ARROW to move the selection one column (or field) to the left, and the RIGHT ARROW to move the selection one column (or field) to the right. The Data Pane also updates to reflect the changes to the data selections. |
| SHIFT+ARROW (after selecting one or more data selection objects on the document page) | Resize the data selection in a data selection object. This adjusts the values of the line and/or column properties (or, in the case of a database emulation, the Field name, and/or the From/To record properties) of the data selection. Use the DOWN ARROW or UP ARROW to respectively increase or decrease the size of the data selection by one line (or record in the case of a database emulation), along its bottom edge. Use the RIGHT ARROW or LEFT ARROW to respectively increase or decrease the size of the data selection by one column (or field in the case of a database emulation), along its right edge. The Data Pane also updates to reflect the changes to the data selections. |

## Use the Hex Viewer

| Use: | To: |
|---|---|
| CTRL+P | Print a hexadecimal dump of the input data. |
| CTRL+Z | Undo the most recent command. Repeat to move backwards through the sequence of commands entered to date, reversing the effect of each command. Not all commands can be undone. |
| CTRL+X | Cut the currently selected data. |
| CTRL+C | Copy the currently selected data. |
| CTRL+V | Paste the currently copied or cut data. |
| CTRL+SHIFT+V | Paste the hexadecimal values of the currently copied or cut data into the input data as text values. |
| CTRL+SHIFT+I | Insert a nibble at the current pointer position. Each bit of the nibble is a 0. |
| CTRL+SHIFT+D | Delete the nibble at the current pointer position. Each bit of the nibble is a 0. |
| CTRL+G | Go to a specific byte offset. |
| CTRL+J | Jump forward the number of bytes set in the Jump Amount dialog box. |
| CTRL+SHIFT+J | Jump backward the number of bytes set in the Jump Amount dialog box. |
| CTRL+F | Search for a string of data. |
| F3 | Find the next occurrence of the search string. |

## Work with Objects

| Use: | To: |
|---|---|
| F2<br>(in the Structure area, after selecting an object or a group) | Rename the object or group. |
| CTRL+click<br>(on an object or a group in the Structure area) | Add the object or group to the current selection, or, if it is already selected, remove it from the selection. |
| SHIFT+click<br>or<br>CTRL+click<br>(on an object or a group in the Page area) | Add the object or group to the current selection, or, if it is already selected, remove it from the selection. |
| ALT+click<br>(on an object or a group in the Page area) | Select all objects and groups that lie under the current pointer position. |
| CTRL+A<br>(in the Structure area or in the Page area) | Select all objects and groups at the same level in the Structure area hierarchy. |
| CTRL+click+drag<br>(on a selection of one or more objects and/or groups) | Move the selected object(s) and/or group(s) along only the X axis or only the Y axis. |
| CTRL+ARROW | Move the selected object(s) and/or group(s). |
| SHIFT+ARROW<br>(in the Page area) | Resize the selected object(s) and/or group(s). You set the magnitude of the resize that occurs with each press of the ARROW key, in the User Options dialog box, Nudge factor box. |
| CTRL+L | Display the Align dialog box to define the alignment for the selected objects and/or groups. |
| CTRL+G | Group the selected objects and/or groups. |
| CTRL+U | Ungroup the selected group(s). |
| CTRL+D | Duplicate the currently selected objects and/or groups using the displacement and data page offsets set in the User Options dialog box. |
| CTRL+SHIFT+D | Duplicate the currently selected objects and/or groups and align the copy along the X axis, to the right of, and flush with, the most recent copy. This overrides the Duplicate style setting in the User Options dialog box. It uses the data page offsets set in the User Options dialog box. |
| CTRL+ALT+D | Duplicate the currently selected objects and/or groups and align the copy along the Y axis, under and flush with, the most recent copy. This overrides the Duplicate style setting in the User Options dialog box. It uses the data page offsets set in the User Options dialog box. |
| CTRL+SHIFT+ UP ARROW | Move the currently selected objects and/or groups backward one layer. |
| CTRL+SHIFT+ DOWN ARROW | Move the currently selected objects and/or groups forward one layer. |
| CTRL+SHIFT+ PAGE UP | Move the currently selected objects and/or groups to the bottom layer. |
| CTRL+SHIFT+ | Move the currently selected objects and/or groups to the top layer. |

| Use: | To: |
|---|---|
| PAGE DOWN | |

## Work in the Text Properties of a Text Object

| Use: | To: |
|---|---|
| CTRL+Z | Undo the most recently entered editing operation. Repeat to move backwards through the sequence of editing operations entered to date, reversing the effect of each command. |
| CTRL+SHIFT+Z | Reverse the effect of the most recent Undo command. Repeat to move backwards through the sequence of undo commands entered to date, reversing the effect of each command. |
| CTRL+Z | Undo the most recently entered editing operation. Repeat to move backwards through the sequence of editing operations entered to date, reversing the effect of each command. |
| CTRL+SHIFT+Z | Reverse the effect of the most recent Undo command. Repeat to move backwards through the sequence of undo commands entered to date, reversing the effect of each command. |
| CTRL+A | Select all of the text in the Text area. The text can include variables. |
| SHIFT+ARROW | Increase or decrease the selected region one line at a time (up or down arrow), or one character at a time (left or right arrow). |
| CTRL+SHIFT+ ARROW | Increase or decrease the selected region one paragraph at a time (up or down arrow), or one word at a time (left or right arrow). |
| ALT+SHIFT+ARROW | Increase or decrease the region one line at a time (up or down arrow), or one character at a time (left or right arrow). |
| CTRL+X | Cut the currently selected region. |
| CTRL+C | Copy the currently selected region. |
| CTRL+V | Paste the last region copied. |
| DELETE | Delete the currently selected region. |
| CTRL+B | Toggle the bold property for the currently selected style on and off. |
| CTRL+I | Toggle the italic property for the currently selected style on and off. |
| CTRL+U | Toggle the underline property for the currently selected style on and off. |
| CTRL+W | Turn word wrap on or off. |
| SHIFT+CTRL+M | Display the Margins dialog box. |
| CTRL+D | Create a local variable using the Data Selector. |
| CTRL+E (after you click on a variable) | Display the Variables dialog box. |
| F2 (in the Variables dialog box) | Rename the currently selected local variable. You cannot rename global variables. |
| DELETE (in the Local Variable dialog box) | Delete the currently selected local variable. You cannot delete global variables. |

# PlanetPress Talk Editor

"General" (page 691)

"Show or Hide Areas of the Editor" (page 691)

"Print the Script" (page 694)

## General

| Use: | To: |
| --- | --- |
| ALT+F10 (in either Commands or Code area) | Display the menu that appears when you right-click in that area. |

## Show or Hide Areas of the Editor

| Use: | To: |
| --- | --- |
| CTRL+SHIFT+T | Show/hide the Commands area. |
| CTRL+SHIFT+S | Show/hide the Spy list. |
| CTRL+SHIFT+P | Show/hide the Object Preview. |

## Expand or Collapse Groups in the Commands Area

| Use: | To: |
| --- | --- |
| Multiplication symbol (*) followed by minus sign (−) (on numeric keypad) | Expand or collapse the selected command group. |
| LEFT ARROW, RIGHT ARROW | Expand (LEFT ARROW), or collapse (RIGHT ARROW) the selected command group. |
| CTRL+SHIFT+O | Collapse all command groups. |
| CTRL+SHIFT+E | Expand all command groups. |

## Work in the Code Area

| Use: | To: |
|---|---|
| CTRL+N | Insert a carriage return. |
| INSERT | Toggle between Insert and Overwrite mode. |
| CTRL+ LEFT or RIGHT ARROW | Move forward or backward in the code, one word at a time. |
| CTRL+HOME | Move to the first character of the first line of the Code area. |
| CTRL+END | Move to the last character of the last line of the Code area. |

## Use Command Name Completion/Argument Insertion

| Use: | To: |
|---|---|
| CTRL+SPACE | Launch command name completion or argument insertion. The position of the pointer determines which operation the shortcut launches. Press ESCAPE to abort the operation. See "Use Command and Variable Name Completion" on page 405 and "To enter arguments:" on page 403 for more complete explanations of this shortcut. |
| CTRL+SHIFT+SPACE | Display the list of arguments the command requires. This shortcut functions only when you use it between the opening and closing parentheses of the argument list. The current argument (the one at the current insertion point in the Code area) appears in bold. You can move the insertion point forward or backward by pressing the right and left arrows respectively; the current argument changes to reflect the move from one argument to another. |

## Undo Commands

| Use: | To: |
|---|---|
| CTRL+Z or ALT+BACKSPACE | Undo the last command or group of editing commands entered, depending on the setting of the Group undo check box in the User Options dialog box. If you selected Group undo, the Editor undoes the last group of editing commands entered. If you cleared Group undo, the Editor undoes only the last editing command entered. Repeat to continue undoing each preceding editing command or group of editing commands. |

## Work with Selections

| Use: | To: |
|---|---|
| CTRL+A | Select the complete contents of the Code area. |
| SHIFT+ARROW | Increase or decrease the size of the current selection one line at a time (up or down arrow), or one character at a time (left or right arrow). |
| CTRL+SHIFT+ LEFT/RIGHT ARROW | Increase or decrease the size of the current selection one word at a time. |
| ALT+SHIFT+ARROW | Increase or decrease the size of the current selection one character at a time (left or right arrow) or one line at a time (up or down arrow). One line in this case refers to only that portion of the line currently in the selection. For example, if a selection contains only characters 3 through 10 of the line and you press the down arrow, the selection adds only characters 3 through 10 of the next line. This is often referred to as a "block select". |
| CTRL+X | Cut the currently selected region of code. |
| CTRL+C | Copy the currently selected region of code to the clipboard. |
| CTRL+V | Paste the contents of the clipboard to the current pointer position. |
| DELETE or BACKSPACE | Delete the currently selected region of code. |

## Add/Remove Comments

| Use: | To: |
|---|---|
| CTRL+SHIFT+/ | Add the comment character (%) to the start of each line of the selected block of code, or if no block is currently selected, to the start of the line indicated by the current pointer position. |
| CTRL+SHIFT+. (period) | Remove the comment character (%) from the start of each line of the selected block of code, or if no block is currently selected, from the start of the line indicated by the current pointer position. |

## Indent Code

| Use: | To: |
|---|---|
| CTRL+SHIFT+I | Indent the current line or selected region of code. You set the number of spaces to advance with each indent in the Block indent box of the User Options dialog box. |
| CTRL+SHIFT+U | Move the current line or selected region of code towards the left one indent. You set the number of spaces to move with each indent in the Block indent box of the User Options dialog box. |

## Search

| Use: | To: |
|---|---|
| CTRL+F | Display the Find Text dialog box. |
| F3 | Find again. |
| CTRL+R | Replace. |

## Jump to a Specific Line

| Use: | To: |
|---|---|
| ALT+G | Display the Go to Line dialog box. This shortcut functions only when the current pointer position is in the Code area. |

## Use Bookmarks

| Use: | To: |
|---|---|
| CTRL+SHIFT+ **digit** | Set or remove a bookmark, where **digit** is the number (0 through 9) of that bookmark. If the bookmark is already set, this sequence removes it. Otherwise it sets the bookmark at the current pointer position. |
| CTRL+**digit** | Jump to a bookmark, where **digit** is the number (0 through 9) of the bookmark to which you want to jump. |

## Execute a Program

| Use: | To: |
|---|---|
| F9 | Run the program to completion. If breakpoints occur before the end of the program, the properties of those breakpoints may mean execution stops before reaching the end of the program. |
| F4 | Execute the program forward to the line indicated by the current cursor position. If breakpoints occur before the program reaches the pointer position, the properties of those breakpoints may mean execution stops before reaching the pointer position. |
| F8 | Execute only the current line of the program. |
| CTRL+F2 | Exit the current execution of the program. |

**Debug Code**

| Use: | To: |
|---|---|
| F5 | Add or remove a breakpoint at the line indicated by the current pointer position. |
| CTRL+F7 | Display the Evaluate dialog box in order to evaluate an expression, edit the value of a variable, or create a spy. |
| CTRL+F5 | Create a spy. |

**Print the Script**

| Use: | To: |
|---|---|
| CTRL+P | Print the contents of the Code area. |

**Converted Document**

A converted document is the final format of a document in PlanetPress Design: a PostScript program. A single document can contain pages destined to execute on a printer, pages destined to execute in PlanetPress Image, and pages destined to execute in PlanetPress Fax. For discussion purposes, we refer to each of these as an output type. When PlanetPress Design or a PlanetPress Suite Workflow Tool converts a document, it converts it for only one type of output at a time: printer, PlanetPress Image, or PlanetPress Fax.

PlanetPress Design converts a document automatically when you preview it on-screen, or install it on a printer or in PlanetPress Design iWatch. It uses the information in the document's PP7 file to generate the PostScript code of the converted document.

If you install a document in a PlanetPress Suite Workflow Tool, PlanetPress Design uses the information in the document's PP7 file to generate a PlanetPress Talk version of the document, a PTK file, that it then installs in a PlanetPress Suite Workflow Tool. The PTK file contains all of the information PlanetPress Suite Workflow Tools require to generate the PostScript code for any output type. When you execute a PlanetPress Suite process that references the document PlanetPress Suite Workflow Tools use the PTK file to generate the PostScript code for the specific output type.

You can also manually convert a document that executes on a printer.

The file name extension on a converted document indicates the type of output it produces.

| Extension: | Indicates: |
|---|---|
| PS | The converted document produces printer output. |
| PSI | The converted document produces PlanetPress Image output. The converted document in this case includes PostScript code to generate the PDI file PlanetPress Search uses to search archives created by PlanetPress Image. |
| PSF | The converted document produces PlanetPress Fax output. |

The font information included in a converted document depends on the type of output the document produces. If it produces PlanetPress Image or PlanetPress Fax output, the converted document contains font information for all fonts that the document uses. If it produces printer output, it consults the PPD you selected in the Document properties dialog box, and includes font information only for fonts that the document uses that are not printer-resident.

# About Documents

A document is essentially anything such as an invoice, financial statement, monthly report, brochure, booklet, cheque, form letter, catalogue, price list, graybar report, survey, shipping label, cheque, insurance policy, tax return, bank statement, receipt, notice, price list, and direct mail material.

A variable content document is one that can dynamically change its content and appearance based on the data it receives at runtime.

## About Document Elements

While *Objects* are generally placed in the Workspace Area (on your page), *Elements* are a broader term that include conditions, variables, functions and metadata fields. Here is a list of all the elements available in PlanetPress Design.

### Sample Data Files

A Sample Data file is a file that, while it follows the same structure as the one you will use when actually processing your document, only contains part of the actual data. It is used to create your document while having an example of the data file, and can even be fake, computer-generated data in some cases. For more information about data, please see the chapter on "Data in PlanetPress Design" (page 83).

### Document

A *Document* is generally described as any number of pages that are destined for one recipient. For example, a document can be an invoice (or multiple invoices) sent to a client, a receipt or a report on one particular department. In a PlanetPress Design document, the term *Document* is the element that includes all the pages of your document and controls the global document properties. For more information about documents, see the chapter on "Setting Up a Document" (page 105).

### Styles

Styles are basically specific font configurations can be applied to an object in any of your document's pages. You need at least one style in your document, but can have multiple styles applied to different objects. The styles contain properties that control the font type, size, color, spacing, etc. For more information about styles, see the chapter on "Fonts and Styles" (page 29).

### Conditions

Conditions can be applied to objects to make them appear or not, and can also be used within objects as PlanetPress Talk Boolean variables to display one value or another. For more information about conditions, see the chapter on "Conditions" (page 185).

### Metadata Fields

Metadata is generally defined as *data about data*, and in the PlanetPress Suite world is more precisely information about your document that is not part of your sample data file. Metadata includes things like page counts, page size properties, etc. It is divided in multiple levels including the Job, Group, Document, Data Page and Page levels, and can contain properties and fields at any of those levels. Metadata fields are user-defined strings that can be added at different levels of your metadata. For more information on Metadata and Metadata fields, please see Working with Metadata.

### Global Variables

Global variables are variables that can be shared between pages and datapages within your document. Global variables will not change unless you change them using a PlanetPress Talk definition. For more information on global variable, see Global Variables.

### Global Functions

Global functions are PlanetPress Talk functions that are available from any page or datapage within your document. Global functions cannot be changed during runtime so they are static. Functions can accept multiple parameters and return different types of variables. For more information on global functions, see "Create a Global Condition" (page 188).

**PostScript Attachments**

PostScript attachments are PostScript files (.ps) that are imported into your document and can be used as resources by your pages and will print after the page where you attach the postscript file. For more information on PostScript attachments, see the chapter on "Document Resources" (page 171).

**Picture Resources**

Picture resources are images of different formats that are imported into your document and can be used as image resources in your pages, generally using the Picture object. Picture resources, however, can also be external to the document such as on a PlanetPress Suite Workflow Tools server or a printer hard drive. For more information on images, see the chapter on "Document Resources" (page 171).

**Pages**

Pages are the canvas of your document and are where objects are added to build your document. Each page in your document generally represents one printed page on your printer (or one PDF page depending on your output). Conditions, overflows and PlanetPress Talk objects, however, can modify when and where pages appear. For more information on pages, see the chapter on "Setting Up Pages" (page 117).

# About Data Selections

A data selection could be compared to an address. It indicates a location within a data file using coordinates. PlanetPress Design includes a tool called the Data Selector that helps you make data selections. The Data Selector does two things:

- It uses the current emulation to format the data.
- It displays the formatted data to let you make selections easily using the mouse pointer.

# About Objects

*Objects* are the pieces of your document that are inserted in your page, such as text boxes, barcodes, images and data selections. Each object has its own properties, some being common to all objects and others being unique to each type of object.

Objects may be both variable or static. Static objects are generally design information such as addresses, table lines and logos. Variable objects are those that will change within your design, such as conditional objects, data selections, etc.

To learn more about each object, please see the chapter on "PlanetPress Design Objects" (page 123).

# PPD File

PostScript Printer Definition. A PPD is a file ending with .ppd which is generally associated with a printer model or family. It contains the PostScript commands that the printer recognizes and can use in its operation.

**PP7 File**

A PP7 file is a file that contains all the information PlanetPress Design requires to convert and execute the document. It includes a description of the visual layout of the document, the sample data file, the PPD for the document, all the static images for the document, and all page and document attachments.

By default it also includes the sample data file.

---

You create a PP7 file for a document the first time you save the document. You can also set a password on a PP7 file.

## PTZ File

A PlanetPress Document Package, or PTZ, is a file format consisting of a ZIP file, which contains all the document's data and resources. With a PTZ, resources are extracted and uncompressed only once when the PlanetPress Suite Workflow Tool receives the file. All the resource files are stored in a sub-folder from where they are used as needed by the PlanetPress Suite Workflow Tool, and a smaller PTK file is placed in the *Documents* folder.

## PTK File

A PTK file is a file that is destined to execute in PlanetPress Suite Workflow Tools. It is the file PlanetPress Design copies into the Documents folder of the PlanetPress Suite Workflow Tool installation when it installs a document into the PlanetPress Suite Workflow Tool.

The content of the PTK file is a PlanetPress Talk program that describes the document and contains all of the information PlanetPress Suite Workflow Tools require to convert the document for any output type. Thus it includes all of the information contained in the PP7 file for the document.

## About Resources

A resource is a PostScript Attachment or an image that is added to your document. Once you add a resource, it becomes available for re-use throughout the document. All resources in the document appear in the Resources area of the document's Structure area. For more information about resources, see the chapter on "Document Resources" (page 171).

PLANETPRESS. SUITE