The PrintOMatic Xtra

Version 1.5.3

The PrintOMatic Xtra is the premier printing tool for Director. PrintOMatic adds a full set of page-layout, text and graphics printing features to Director 5.0 projects on Macintosh and Windows.

PrintOMatic includes commands for accurately specifying the position of any text or graphic element on the page. PrintOMatic documents can contain graphic files from disk, styled text, graphic primitives, cast member bitmaps, and snapshots of the Director stage.

Product Features

- Generates multi-page layouts with full control over the placement of text and graphic elements on the page.
- Print styled text: any combination of available fonts, sizes, or styles
- Print text, PICT, BMP or EPS files from disk, any portion of the Director stage, or Director cast members.
- Print object-oriented graphic primitives: lines, boxes, ovals and rounded rectangles.
- "Master Page" can contain any combination of text or graphic elements
- Automatic page numbering
- Customizable and hideable Print Progress dialog
- Paper-saving Print Preview feature
- Supports color and landscape-mode printing
- Supports all Macintosh and Windows compatible printers
- Fully compatible with MacOS™, Windows™ 3.1 and Windows™ 95

Topics in this document:

PrintOMatic Software Updates

PrintOMatic is updated and enhanced frequently. The latest version of PrintOMatic is always available, along with a selection of XObjects, XCMDs and other software on Electronic Ink's Web Site:

```
http://www.rahul.net/peterv/products.html
```

Before reporting a bug in the software, please make sure the version you are using is up to date. Users with dedicated FTP clients can access our FTP site directly:

```
ftp://ftp.rahul.net/pub/peterv/
```

Registered users of PrintOMatic will get announcements via e-mail when new major versions are released.

Purchasing PrintOMatic

The PrintOMatic Xtra is published and distributed worldwide by g/matter, inc. The registered edition of PrintOMatic is shipped on a cross-platform CD-ROM disc, along with demonstrations, sample code, and other tidbits. A cross-platform copy of PrintOMatic, with a royalty-free, unlimited use license to use PrintOMatic in your multimedia productions costs $299.00 US.

The PrintOMatic demonstration movie prints out an order form for the PrintOMatic Xtra, as well as other g/matter products. To order your copy of PrintOMatic, simply fill out this form and FAX, mail, or e-mail it with payment information to:

g/matter, inc.
300 Brannan Street, Suite 210
San Francisco, CA 94107

Tel: (800) 933-6223 or (415) 243-0394
FAX: (415) 243-0396
Email: sales@gmatter.com
http://www.gmatter.com/

Non-Profit Licensing

Products created by or for bona fide philanthropic non-profit organizations can get a copy of PrintOMatic free of charge. In lieu of a licensing agreement, non-profit organizations must send a "receipt of donation" for the commercial value of the product. Please contact g/matter for more information about non-profit licensing.

Using PrintOMatic

The PrintOMatic Xtra can be used in two ways. The first and simplest way to use PrintOMatic is by calling the global print command, passing the Director object(s) you would like to print as parameters:

```
print member "illustration" of castLib 1
print "Some example text for printing"
print sprite 1, sprite 5
print castLib "documentation"
```

If all you have to do is print something out quickly and easily, you can stop reading now. That's all there is to it.

Creating an Instance of the PrintOMatic Xtra

The second way of using PrintOMatic, which is much more powerful and customizable than simply using the global print command, is to create an instance of the Xtra using the new command:

```
set doc = new(xtra "PrintOMatic")
if not objectP(doc) then exit
```

An instance of the PrintOMatic Xtra is called a "document object ", also simply referred to as a "document". Be sure to check the validity of the document object using the objectP() function after creating it. Once you have created a document, you can change its attributes, such as the document's name, margins and landscape orientation:

```
setDocumentName doc, "My Document"
setMargins doc, Rect(36,36,36,36)
setLandscapeMode doc, TRUE
```

Master Page

Page 0 (zero) of a document is the document's "master page". The contents of the master page are drawn on every body page of the document, beneath the body page's contents. When you create a new document, the master page is the default page until you create a new page (see Adding Pages, below.) All items added before the first newPage command will appear on the document's master page.

A common item to place on the master page is a page number. Use the setPageNumSymbol command in combination with a text item on the master page to place a page number on every page of your document:

```
-- place a page number at the bottom right of the page
set pgNumSym = numToChar(166) -- paragraph symbol on mac
setPageNumSymbol doc, pgNumSym
setTextJust doc, "right"
drawText doc, Point(getPageWidth(doc),getPageHeight(doc)),
"page"&&pgNumSym
```

Adding Pages

For a PrintOMatic document to be printable, it must contain at least one page. Add new pages to

a document using the newPage command. A page added using newPage becomes the "current page", where all new graphic elements, "frames", and other items are placed. To make a previously added page (including the master page) the "current page", use the setPage command.

```
newPage doc -- add a new page
setPage doc, 0 -- return to the master page
```

PrintOMatic Coordinate System

PrintOMatic uses the same basic coordinate system as the Director stage, which places (0,0) at the top-left corner. Coordinate values increase towards the bottom and right sides of the page. All drawing coordinates are specified in points, with 72 points to the inch. All drawing coordinates for objects in PrintOMatic are relative to the page margins, which are set using the setMargins command. The drawing area defined by the margins applies to the entire document.

Calling pageSetup or setLandscapeMode may change the size of the page, thereby changing the coordinate system, as well as the values returned by getPageWidth and getPageHeight. If you store these values in Lingo variables, be sure to reset them after calling pageSetup or setLandscapeMode.

Creating Frames

The most common method of placing text or graphics on a PrintOMatic page is to create one or more rectangular "frames" on the page, then append contents to these frames. The contents of multiple "linked" frames will flow from one frame to the next.

For example, to create a two-column layout, create two frames side by side on the page, and link them together so the text and graphics flow from one frame to the next:

```
-- create a new document
set doc = new(xtra "PrintOMatic")
-- get width and height
set w = getPageWidth(doc)
set h = getPageHeight(doc)
-- create a new page
newPage doc
-- create the first column
newFrame doc, Rect(0,0,(w/2)-18,h), FALSE
-- create the second column, linked to the first
newFrame doc, Rect((w/2)+18,0,w,h), TRUE
```

The last parameter in the newFrame command specifies whether or not the new frame is linked to the previous frame.

Appending to Frames

Once you have created one or more "frames" to append to, you can add items such as sprites and cast members to the document using the append and appendFile commands:

```
append doc, member "title" of castLib 1, TRUE
append doc, sprite 1, TRUE
appendFile doc, the pathName&"myFile.eps", FALSE
```

The last parameter of append and appendFile calls specifies whether pages are automatically

added to the document as content is inserted. Please see the append command for a detailed description of this "autoAppend" feature of PrintOMatic.

Text field cast members will be printed with all their fonts and styles intact. Rich text cast members' bitmap images will be printed. However, text *strings* have no inherent style data associated with them. You can use setTextFont, setTextSize and setTextStyle to set the attributes of text strings that you append.

```
setTextFont doc, "Helvetica"
setTextSize doc, 10
setTextStyle doc, "normal"
append doc, copyrightInfo
```

Drawing Graphic Elements

A second way of placing items on a PrintOMatic page is to explicitly position them as graphic elements in the exact location you want them to appear. You can place pictures, single lines of text, rectangles, rounded rectangles, lines and ovals on the page in this manner. Many of the graphic element drawing routines take a Lingo `Rect` as the second parameter, to specify their position and size:

```
drawRect doc, Rect(0,0,50,50), TRUE
drawRoundRect doc, Rect(0,0,50,50), 25, FALSE
drawOval doc, Rect(100,100,500,500), TRUE
```

The drawLine routine takes a starting `Point` and ending `Point` as its parameters:

```
drawLine doc, Point(0,0), Point(500,0)
```

The drawPicture routine can position a bitmapped image on the page using a `Point` or a `Rect` to specify size and location. If you use a `Point`, the image will be positioned at its normal size (72 dpi for cast members) with its top left corner at the specified point:

```
drawPicture doc, member "illustration", Point(0,100)
```

If you specify the image location using a `Rect`, the image will be scaled to fit at the largest possible size within the `Rect` *without distorting the image*. Unless the provided `Rect` has the exact same proportions as the image, the entire `Rect` will not be filled with the image data:

```
drawPicture doc, member "illustration", Rect(0,0,500,300)
```

Drawing the Contents of the Stage

The contents of the Director stage can be placed on a PrintOMatic page using the drawStagePicture routine. The size and positioning rules for drawStagePicture are the same as for drawPicture: if a `Point` is specified, the top left corner of the Stage picture will be placed there; if a `Rect` is specified, the image will be scaled to fit within the `Rect` *without distortion.*

```
drawStagePicture doc, Point(0,0)
drawStagePicture doc, Rect(0,0,the width of the stage, the height of
the stage)
```

The drawStagePicture routine can print a cropped portion of the stage by specifying the area you want to capture after specifying the image's position on the page:

```
drawStagePicture doc, Rect(0,0,100,100), Rect(50,50,150,150)
```

Finally, you can capture the stage picture from Director's off-screen buffer by adding a `TRUE` to the end of any of the above specified forms of `drawStagePicture`. Capturing from the offscreen buffer will prevent the contents of any windows in front of the Stage from being captured along with the stage image.

```
drawStagePicture doc, Point(0,0), Rect(50,50,100,100), TRUE
```

Printing and Print Preview

When you have appended all the elements you want to print to the document, show the user the job setup dialog, and print the document if `doJobSetup` returns `TRUE`. If you don't want the user to see the job setup dialog, omit the call to doJobSetup.

```
if doJobSetup(doc) then print doc
```

Finally, dispose of the document by setting its value equal to zero. This will release all the memory taken up by the PrintOMatic document.

```
set doc = 0
```

Other Routines

PrintOMatic supports a number of other routines that are not summarized in the sections above. Please consult the full message list for a complete listing of all the commands supported by the PrintOMatic Xtra.

PrintOMatic Message List

This is the full list of commands supported by the PrintOMatic Xtra:

```
-- CREATE/DESTROY/RESET A DOCUMENT
new object
forget object
reset object
--
-- DOCUMENT/JOB SETUP
doPageSetup object
doJobSetup object
--
-- DOCUMENT ATTRIBUTES
setDocumentName object, string name
setLandscapeMode object, boolean landscape
setMargins object, rect margins
setPrintableMargins object
getPageWidth object
getPageHeight object
getPaperWidth object
getPaperHeight object
--
-- CREATE/SET PAGES
newPage object -- returns page number
setPage object, int pageNumber
--
-- TEXT/GRAPHIC ATTRIBUTES
setTextFont object, string fontName
setTextSize object, int pointSize
setTextStyle object, string styleCodes
setTextJust object, string justification
setTextLineSpacing object, int spacing
setColor object, int red, int green, int blue
setGray object, int graylevel
setLineWeight object, int pointSize
--
-- GRAPHIC ELEMENTS
drawRect object, rect bounds, boolean filled
drawLine object, point start, point end
drawRoundRect object, rect bounds, int cornerRadius, boolean filled
drawOval object, rect bounds, boolean filled
drawText object, string text, point location
drawPicture object, *
drawStagePicture object, *
--
-- CREATE FRAMES AND APPEND CONTENTS
newFrame object, rect bounds, boolean linkedToPrevious
append object, * any
appendFile object, * fileName
getInsertionPoint object
--
-- CUSTOMIZE THE PROGRESS BOX
setProgressMsg object, string message
setProgressPict object, * pictCastMember
```

```
   setProgressLoc object, point location
   --
   -- PRINT OR PREVIEW
   * printPreview *
   * print *
   --
   -- MISCELLANEOUS
   hideMessages object, boolean hide
   setPageNumSymbol object, string symbol
   + register object, string serialNumber
   + setLowMemLimits object, globalHeap, localHeap
   --
   -- MACINTOSH-ONLY ROUTINES
   * printToPictFiles *
   draw1bitStagePicture object, *
   loadPageSetup object, string fileName, int resourceID
   savePageSetup object, string fileName, string fileType, string
fileCreator, int resourceID
```

Creating, Destroying and Resetting Documents

The following commands are used to create and reset PrintOMatic documents:

new	creates a new document
forget	**never** call this directly
reset	resets a document to defaults

To destroy a document, **never** call the forget method of an Xtra explicitly. Instead, use the following syntax to explicitly get rid of a PrintOMatic document when you are done using it:

```
set doc = 0
```

new

Syntax: `set doc = new(xtra "PrintOMatic")`

The `new` command is used to create a new instance of the PrintOMatic Xtra. The newly created instance is called a **document object**, since it represents a printable "document": a collection of items that are printed together in a single print job by PrintOMatic.

Once the document has been created, its settings can be modified, items can be appended to the document, and it can be printed or displayed in a print preview window.

Important Note:

`new` will return an **error code** instead of a document object if there is no currently selected printer, or a printing error occurs. Always check the result of new with the `objectP()` function to make sure you have a valid Xtra instance before continuing!

Example:

The following code example creates a new document, sets the page orientation to landscape mode, creates a new page and "frame" on the page, appends an image with a caption to the document, and prints the document:

```
set doc = new(xtra "PrintOMatic")
if not objectP(doc) then exit
setLandscapeMode doc, TRUE
newPage doc
newFrame doc, Rect(0,0,getPageWidth(doc),getPageHeight(doc))
append doc, member "picture", TRUE
append doc, RETURN & "Image printed by the PrintOMatic Xtra.", TRUE
if doJobSetup (doc) = TRUE then print doc
set doc = 0
```

forget

You should **never** call the `forget` command directly for an instance of a Lingo Xtra. Director automatically calls `forget` when an instance of an Xtra needs to be disposed; calling `forget` yourself can lead to memory leaks or crashing.

Use the following syntax to explicitly get rid of a PrintOMatic document when you are done using it:

```
set doc = 0
```

Where `doc` is the PrintOMatic document you want to dispose of. Explicitly disposing of documents is optional, since Director will automatically get rid of the object when it's no longer referenced anyway.

reset

Syntax: `reset` *document*

The `reset` command is used to reset an instance of a PrintOMatic document. All the contents of the document and any existing pages are deleted. The page settings, such as margins and page orientation, are reset to their default values.

Page Setup and Job Setup Dialogs

The following commands are used to display the Page Setup and the print Job Setup dialog boxes for the user:

doPageSetup	Presents the Page Setup dialog
doJobSetup	Presents the Print Job Setup dialog

Calling these routines is optional; if they are not called, the document or print job will be set up with default values.

doPageSetup

Syntax: `doPageSetup(`*document*`)`

Returns: `TRUE` if the user clicks "OK" in the page setup dialog
`FALSE` if the user clicks "Cancel" in the page setup dialog

The `doPageSetup` function displays the Page Setup dialog for a document. This function must be called on an empty document, before any elements are added to it. `doPageSetup` returns `TRUE` if the user clicks the "OK" button in the dialog box, or `FALSE` if the user clicks "Cancel".

doJobSetup

Syntax: `doJobSetup(`*document*`)`

Returns: `TRUE` if the user clicks "Print" in the job setup dialog
`FALSE` if the user clicks "Cancel" in the job setup dialog

The `doJobSetup` function displays the job setup dialog for a PrintOMatic document. This function should be called right before printing. If `doJobSetup` returns `TRUE`, the user clicked the "Print" button, and printing should proceed. If `doJobSetup` returns `FALSE`, the user clicked "Cancel", and you should not print the document. This function cannot be called on an empty document.

Example:

This is the recommended way of calling `doJobSetup` right before printing a PrintOMatic document:

```
if doJobSetup (doc) = TRUE then print doc
```

Getting and Setting Document Attributes

The following commands are used get and set the attributes of a PrintOMatic document:

setDocumentName	Sets the name of the document
setLandscapeMode	Sets landscape or portrait orientation
setMargins	Sets the margins of the document
setPrintableMargins	Sets the document margins to the maximum area the printer can print
getPageWidth	Returns the width between left and right margins
getPageHeight	Returns the height between the top and bottom margins
getPaperWidth	Returns the width of the physical paper
getPaperHeight	Returns the height of the physical paper

Routines that alter the size or orientation of a document, such as setLandscapeMode, setMargins, and setPrintableMargins, can only be called when your document is empty.

setDocumentName

Syntax: `setDocumentName` *document, name*

This command sets the name of a PrintOMatic document, which is displayed in the print progress dialog as the document prints. If background printing is enabled, this document name is also displayed by PrintMonitor (Mac) or Print Manager (Windows) as your document prints in the background.

setLandscapeMode

Syntax: `setLandscapeMode` *document, trueOrFalse*

This command switches the page orientation of a PrintOMatic document between landscape and portrait orientation. Since this method changes the whole coordinate system of the document, your document must be empty when you call `setLandscapeMode`. You can call reset on your document beforehand just to make sure.

In Windows, this method works in a very straightforward manner: call it, and the landscape mode changes.

Unfortunately, it's an entirely different story on the Macintosh. The *only* safe way to change the page orientation on the Mac is by showing the Page Setup dialog and letting the user manually select landscape mode. While showing a Page Setup dialog may be fine for normal software applications such as Word, it's often unacceptable in the context of a multimedia production.

To get around this, PrintOMatic for the Macintosh relies on a "printer database" to store default landscape and portrait page setups for the most common Macintosh printers. This printer database consists of a set of 'PHDL' resources located in the same file as the PrintOMatic Xtra. This database is used by the `setLandscapeMode` method in the Macintosh version of PrintOMatic.

If the currently selected printer is not found in the printer database, the user is asked to MANUALLY create default Page Setups for landscape and portrait modes, and those settings are saved and added to the database. This is done through a series of prompt dialogs presented automatically by PrintOMatic when you call `setLandscapeMode` for an unknown printer.

These user-configured "custom entries" to the printer database are stored in a file called "PrintOMatic Preferences" in the Preferences folder on the user's hard disk. Subsequent calls to `setLandscapeMode` on the same computer, with the same printer selected, won't present any annoying dialogs.

What the presence of this "printer database" means is that when you change the landscape mode on the Macintosh, all the other Page Setup settings such as scaling, font substitution, etc., will also revert to those found in the printer database. This is important if the user has changed any of these settings (during a call to `doPageSetup`) before `setLandscapeMode` is called.

setMargins

Syntax: `setMargins` *document*`,` *marginRect*

This command sets the margins of a PrintOMatic document. The *marginRect* parameter is in the form of a Lingo `Rect`. Values are specified in the format `Rect(`*left*`,` *top*`,` *right*`,` *bottom*`)`. The measurements are in points (72 points to the inch). Since this method changes the whole coordinate system of the document, your document must be empty when you call `setMargins`. Call `reset` on your document beforehand just to make sure.

Example:

The following example creates a new document and sets the margins to two inches (144 points) on the left, and one inch (72 points) on all other sides.

```
set doc = new(xtra "PrintOMatic")
if not objectP(doc) then exit
setMargins doc, Rect(144,72,72,72)
```

setPrintableMargins

Syntax: `setPrintableMargins` *document*

This command sets the margins of a PrintOMatic document equal to the maximum printable area supported by the current print settings. Since this command changes the whole coordinate system of the document, your document must be empty when you call `setMargins`. Call `reset` on your document beforehand just to make sure.

getPageWidth

Syntax: `getPageWidth(`*document*`)`

Returns: the distance between the left and right margins, in points

This function returns the width of a document's "live area", the distance between the left and right margins of the document.

Example:

It is often convenient to retrieve the dimensions of the live area of a document and store them in Lingo variables for use in subsequent calculations. The following example creates two linked frames on the page, for formatting printed output into two columns.

```
-- create two new frames on the page
-- with 1/2 inch (36 points) in between
set w = getPageWidth(doc)
set h = getPageHeight(doc)
newFrame doc, Rect(0,0,(w/2)-18,h), TRUE
newFrame doc, Rect(0,0,(w/2)+18,h), TRUE
```

getPageHeight

Syntax: `getPageHeight(`*document*`)`

Returns: the distance between the top and bottom margins, in points

This function returns the height of a document's "live area", the distance between the top and bottom margins of the document.

Example:

It is often convenient to retrieve the dimensions of the live area of a document and store them in Lingo variables for use in subsequent calculations. The following example creates two linked frames on the page, for formatting printed output into two columns.

```
-- create two new frames on the page
-- with 1/2 inch (36 points) in between
set w = getPageWidth(doc)
set h = getPageHeight(doc)
newFrame doc, Rect(0,0,(w/2)-18,h), TRUE
newFrame doc, Rect(0,0,(w/2)+18,h), TRUE
```

getPaperWidth

Syntax: `getPaperWidth(`*document*`)`

Returns: the width of the physical paper, in points

This function returns the width of the physical piece of paper that the current document is configured to print on.

getPaperHeight

Syntax: `getPaperHeight(`*document*`)`

Returns: the height of the physical paper, in points

This function returns the height of the physical piece of paper that the current document is configured to print on.

Adding Pages and Setting the Current Page

The following routines are used to add new pages to your document, or set the "current" page, where subsequent text and graphic elements will be placed:

`newPage` object	Create a new page
`setPage` object	Set the "current" page

About the Master Page

Page 0 of each document is the document's **master page**. The master page is the default "current" page of a newly created document, or one that has been cleared by a call to `reset`. Any text or graphic element added to the master page will be drawn on every page of the document, beneath the contents of each individual page. Calling `setPage` with a value of 0 will set the master page as the current page of the document.

newPage

Syntax: `newPage(`*document*`)`

Returns: the page number of the newly created page

The `newPage` function adds a page to the PrintOMatic document and makes it the current page. It returns the page number of the newly created page.

NOTE: If you want your document to print, it must have at least one page. A common mistake is inadvertently add a number of elements to the *master page* — instead of page 1 — of a document by forgetting to call `newPage` beforehand. The resulting document will not print because it contains no body pages.

setPage

Syntax: `setPage` *document*, *pageNumber*

The `setPage` command sets the "current" page of the document, where subsequent text and graphic elements added to the document will be placed. If the *pageNumber* value is greater than the number of pages currently in the document, new pages will be added, and the requested page will still become the "current" page.

Calling `setPage` with a value of 0 will set the **master page** to be the current page. Any text or graphic element added to the master page will be drawn on every page of the document, beneath the contents of each individual page.

Setting Text and Graphic Attributes

The following routines are used to set the default text and graphic attributes of a document. These attributes determine the color, font, etc. of subsequently added document elements:

setTextFont	Sets the default text font for non-styled text data
setTextSize	Sets the default text size for non-styled text data
setTextStyle	Sets the default text style for non-styled text data
setTextJust	Sets the line justification of text elements
setTextLineSpacing	Sets the line spacing of text elements
setColor	Sets the default color of text and graphic elements
setGray	Sets the default gray value of text and graphic elements
setLineWeight	Sets the line weight of stroked graphic elements

setTextFont

Syntax: `setTextFont` *document, fontName*

Returns: `TRUE` if the requested font was available
`FALSE` if the requested font could not be found

This command sets the text font that will be applied to non-styled text data (such as strings) that are subsequently appended to the PrintOMatic document. If the requested font is not available, the default font (Geneva on the Macintosh, Arial on Windows) is used.

Note that the current font set using `setTextFont` will be overridden by the fonts, sizes and styles within styled text field cast members that you subsequently append to your document. The default font used for non-styled text data after appending a styled text field will be the last font contained within the text field.

setTextSize

Syntax: `setTextSize` *document*, *fontSize*

This command sets the text size that will be applied to non-styled text data (such as strings) that are subsequently appended to the PrintOMatic document.

setTextStyle

Syntax: `setTextStyle` *document, styleString*

This command sets the text style that will be applied to non-styled text data (such as strings) that are subsequently appended to the PrintOMatic document. The names of the style values correspond exactly to the Director "textStyle" property. Possible values for text styles are:

```
normal
plain
bold
italic
underline
```

On the Macintosh, the following additional styles are available:

```
outline
condense
extend
shadow
```

All values except `normal` or `plain` are added together in a call to `setTextStyle`, so you can combine a number of styles together in a single call.

Example:

The following example creates a PrintOMatic document, sets the default font to bold italicized 10 point Helvetica, and prints a short text string:

```
set doc = new(xtra "PrintOMatic")
if not objectP(doc) then exit
setTextStyle doc, "bold, italic"
setTextFont doc, "Helvetica"
setTextSize doc, 10
append doc, "Some sample text to print out."
if doJobSetup(doc) then print doc
set doc = 0
```

setTextJust

Syntax: `setTextJust` *document, justCode(s)*

The `setTextJust` command sets the line justification applied to all subsequently added text elements. Possible values are:

```
left
right
centered
```

Justification applies to an entire text block (or the set of linked text blocks), so if you set justification to "right" and call `append` for a block that was previously centered, the justification of the entire block will be changed to right-justified as the new text is added.

setTextLineSpacing

Syntax: `setTextLineSpacing` *document, spacing*

Sets the line spacing of text elements

setColor

Syntax: `setColor` *document*, *red*, *green*, *blue*

Sets the default color of text and graphic elements

setGray

Syntax: `setGray` *document*, *grayLevel*

Sets the default gray value of text and graphic elements

setLineWeight

Syntax: `setLineWeight` *document*, *lineWeight*

Sets the line weight of stroked graphic elements

Drawing Graphic Elements

The following routines are used to draw graphic elements on the pages of documents:

drawRect	draws a filled or stroked rectangle
drawLine	draws a line
drawRoundRect	draws a filled or storked round rect
drawOval	draws a filled or stroked oval
drawText	draws a line of text
drawPicture	draws a picture from the cast or from disk
drawStagePicture	draws a picture of the Stage contents

drawRect

Syntax: `drawRect` *document*`,` *rect*`,` *filled*

Draws a rectangle on the current page. If the *filled* parameter is `TRUE`, the rectangle is filled using the current color. Otherwise, the rectangle is stroked using the current line weight and color.

Example:

```
drawRect doc, Rect(0,0,500,100), TRUE
```

drawLine

Syntax: `drawLine` *document*, *startPoint*, *endPoint*

Draws a line on the current page from *startPoint* to *endPoint*. The line is stroked using the current line weight and color.

Example:

```
drawLine doc, Point(0,100), Point(getPageWidth(doc),100), TRUE
```

drawRoundRect

Syntax: `drawRoundRect` *document*, *rect*, *cornerRadius*, *filled*

Draws a rounded-corner rectangle on the current page, using the corner radius specified in *cornerRadius*. If the *filled* parameter is `TRUE`, the rounded rectangle is filled using the current color. Otherwise, the rounded rectangle is stroked using the current line weight and color.

Example:

```
drawRoundRect doc, Rect(0,0,500,100), 25, FALSE
```

drawOval

Syntax: `drawOval` *document*, *bounds*, *filled*

Draws an oval on the current page, bounded by the rectangle specified in *bounds*. If the *filled* parameter is `TRUE`, the oval is filled using the current color. Otherwise, the oval is stroked using the current line weight and color.

Example:

```
drawOval doc, Rect(0,0,100,100), FALSE
```

drawText

Syntax: `drawText` *document*, *text*, *location*

Draws a line of text on the current page, using the current text font, size, style, and justification. The justification specified using `setTextJust` determines how the text is aligned relative to the point specified in *location.*

Example:

```
drawText doc, "A little bit of text.", Point(100,50)
```

drawPicture

Syntax: `drawPicture` *document*`,` `[` *fileName* `|` `member` *castMem* `],` `[` *rect* `|` *topLeftPoint* `]`

Draws a bitmapped or PICT cast member, EPS, PICT or BMP file from disk on the current page. If a destination *rect* is specified, the picture will be sized to fit within the rectangle without distortion. If a *topLeftPoint* is specified, the image will be drawn at 100% size from the specified point.

Examples:

```
drawPicture doc, member "image", Point(100,50)
drawPicture doc, the pathName&"image.eps", Rect(0,0,100,100)
```

drawStagePicture

Syntax: `drawStagePicture` *document*`,` `[` *rect* `|` *topLeftPoint* `],` `[` *clipRect*`],` *grabOffscreen*

Places a screen shot of the stage contents on the current page (or MIAW contents if `drawStagePicture` is called from a movie-in-a-window). If a *clipRect* is specified, only that portion of the stage or MIAW will be grabbed. Note that `drawStagePicture` takes a "faithful" screen grab of the stage contents, including the cursor, sprite "trails", and any other windows that might overlap the stage. However, if *grabOffscreen* is `TRUE`, the stage picture will be grabbed from Director's off-screen buffer instead, and these extraneous elements will not be included in the resulting picture.

Examples:

```
drawStagePicture doc, Point(0,0)
drawStagePicture doc, Point(0,0), Rect(0,0,100,300), TRUE
```

Creating Frames and Appending their Contents

The PrintOMatic Xtra uses a "frames" metaphor for creating areas on the page where text or graphics can be flowed. The basic procedure is to create one or more "linked" frames on a page of a document, then append items to the newly created frames. The following commands are used to create frames and append their contents.

`newFrame`	creates a new frame on the current page
`append`	appends contents to the current frame
`appendFile`	appends the contents of a text or graphic file to the current frame
`getInsertionPoint`	gets the location of the insertion point within the current frame

newFrame

Syntax: `newFrame` *document, rect, linkedToPrevious*

The `newFrame` command adds a new "frame" to the current page of the document, and makes it the "current" frame. Text and graphic items can be flowed into the current frame using the `append` and `appendFile` commands.

The *linkedToPrevious* parameter determines whether the contents of the previous frame will flow into the new frame once the previous frame is filled. When you create a new frame that is not *linkedToPrevious,* you can no longer append items to the previously "current" frame. Also note that you cannot link frames between the master page and a body page.

append

Syntax:
```
append document, member whichCastmember [, ... , autoAppend ]
append document, member whichCastmember of castLib whichLib [, ... , autoAppend ]
append document, castLib whichCast [, ... , autoAppend ]
append document, sprite whichSprite [, ... , autoAppend ]
append document, string [, ... , autoAppend ]
append document, list [, ... , autoAppend ]
```

The `append` command appends one or more items to the current "frame" of a PrintOMatic document. If no current frame exists in your document, the PrintOMatic Xtra will attempt to create a "default" frame for you, which is the width and height of the page, minus the current margins.

The *autoAppend* parameter, which is always the last parameter in a call to `append,` controls whether or not new pages will be created "on the fly" if the PrintOMatic Xtra runs out of space on the last "frame" in your document. If this parameter is set to `TRUE`, new pages will be created with the same "frame" layout as the last page of your document. PrintOMatic will create as many new pages as are necessary to flow all the specified elements into your document . If this parameter is not specified when you call `append,` its value defaults to `TRUE`.

AutoAppending is not allowed when flowing items into a frame on the master page of a document.

type of object	what gets appended
text field cast member	the text of the field, using the specified fonts and styles
rich text cast member	the bitmap image of the cast member, including anti-aliasing
bitmap cast member	the cast member graphic
PICT cast member	the cast member graphic
cast library	all printable cast members in the library, in cast sequence
sprite	the cast member of the sprite
text string	the text string, in the default font (Geneva10pt on Macintosh, Arial 10pt on Windows)
list (linear or property)	the elements in the list

Example:

The following example creates a PrintOMatic document, sets the document name, creates the first page, adds a frame to it, and appends a number of items to the document, and prints it:

```
set doc = new(xtra "PrintOMatic")
if not objectP(doc) then exit
setDocumentName doc, "My Example Document"
newPage doc
newFrame doc, Rect(0,0,getPageWidth(doc),getPageHeight(doc)
append doc, sprite 1, TRUE
append doc, [member "image", member "caption", sprite 5], TRUE
append doc, castLib "printout", TRUE
if doJobSetup(doc) then print doc
set doc = 0
```

appendFile

Syntax: `appendFile` *document*`,` *fileName* `[,` *...* `,` *autoAppend* `]`

The `appendFile` command appends one or more text or graphics files to the current "frame" of a PrintOMatic document. If no current frame exists in your document, the PrintOMatic Xtra will attempt to create a "default" frame for you, which is the width and height of the page, minus the current margins.

For details on how to use the *autoAppend* parameter, please see the for the `append` command.

The following file formats are supported by `appendFile`. The actual format of the file will be auto-detected by the `appendFile` command.

file type	notes
plain text	Normal ASCII format text, with or without DOS line-feed characters
styled text	(Macintosh only) Macintosh ASCII text file with a 'styl' resource, such as files created by SimpleText
EPS	Encapsulated PostScript file, with or without a preview image
PICT	Macintosh PICT format file (only raster PICT files 8-bits or less supported on Windows)
BMP	(Windows only) BMP files of any bit depth

Notes on Printing EPS Files

You should avoid using EPS files if you want your printing code to work reliably with all types of printers. Many, many popular printers attached to Macintosh and Windows PC's DO NOT support PostScript printing. The output that PrintOMatic generates on these types of printers can vary from low-resolution bitmaps to placeholder boxes to nothing at all.

Assuming you decide not to heed this warning, here are some tips that may improve your success.

Many applications that generate EPS files allow you to create files in "ASCII" or "binary" format. PrintOMatic prints ASCII format PostScript files MUCH more reliably than binary files. Under some conditions, PrintOMatic will print binary PostScript files just fine. However, certain types of printer connections work very poorly with binary PostScript. Specifically, serial printers that use XON/XOFF flow control often mistake binary data for flow control codes, and will seriously garble or crash your print job.

Some types of PostScript files, notably those generated by using the "print to disk" feature of the LaserWriter driver, don't contain "bounding box" information. PrintOMatic needs bounding box information to determine the size of a PostScript image for placement on the page, and will generate an error if this information can't be found.

You can manually add bounding box information to PostScript files using a text editing program. Insert the following line of text into the file somewhere between `the !%PS-Adobe-3.0` and `%%EndComments` lines at the beginning of the file, substituting the width and height of the page (in points) for the 'x' and 'y' values:

```
%%BoundingBox: 0 0 x y
```

Finally, keep in mind that PostScript is a programming language with a broad set of features, some of which are supported differently by different printers. This makes EPS files MUCH more prone to printing errors and incompatibilities than other file formats, such as PICT or BMP. This is another good reason to avoid using EPS files if at all possible.

getInsertionPoint

Syntax: `getInsertionPoint(`*document*`)`

Returns: A string in the format `"page, x, y"`, or `VOID` if there is no insertion point.

The `getInsertionPoint` function returns the page number and coordinates at which the next `append` or `appendFile` command will insert new content into the document. This can be useful for deciding when to manually break pages, or allow you to place graphics in the margins of a document next to accompanying text.

Example:

The following code checks for an insertion point, and if there is one, places a graphic in the margin next to the insertion point:

```
set the itemDelimiter = ","
set insPt = getInsertionPoint(doc)
if stringP(insPt) then
   drawPicture member "dingbat", Point(-10, integer(item 3 of insPt))
end if
```

Customizing the Print Progress Window

The following routines are used to customize the contents of PrintOMatic's Print Progress window. The setProgressMsg and setProgressPict routines are available to registered users of PrintOMatic only. Attempting to use these routines on an unregistered copy will result in an error.

setProgressMsg	puts a customized text message in the progress window
setProgressPict	puts a customized bitmap image in the progress window
setProgressLoc	sets the location of the progress window

setProgressMsg

Syntax: `setProgressMsg` *document*, *message*

The `setProgressMsg` command puts a customized text message into the Print Progress window. This command is available to registered users of PrintOMatic only. Attempting to use this command on an unregistered copy will result in an error.

The default message in the print progress window is:

```
Printing document "<document name>"
```

You can change the document's name using the `setDocumentName` command.

setProgressPict

Syntax: `setProgressPict` *document*`,` `member` *pictCastMember*

The `setProgressPict` command puts a customized image into the Print Progress window. This command is available to registered users of PrintOMatic only. Attempting to use this command on an unregistered copy will result in an error.

setProgressLoc

Syntax: `setProgressLoc` *document*, *topLeftPoint*

The `setProgressLoc` command sets the location of the top left corner of the print progress dialog displayed during printing. If you want to hide the print progress dialog, position it off the screen using this method. Note that `setProgressLoc` uses global (screen) coordinates for positioning, not stage coordinates.

Printing and Print Preview

The following commands are used to print or display a print preview of a PrintOMatic document, or just about anything else you want to print out:

printPreview	displays an on-screen print preview of the requested item(s)
print	prints the requested item(s) to the currently selected printer

printPreview

Syntax: `printPreview(`*document*`)`
`printPreview(`*any* `[, ... ])`

Returns: `TRUE` if the user previews all the pages in the document
`FALSE` if the user cancels the print preview

This function displays an on-screen facsimile of the output that the `print` command will generate when it is passed the same set of parameters. `printPreview` will display a preview of a document as well as sets of strings, sprites, cast members, lists, etc. See the `print` command for a complete list of the elements that can be previewed.

Typing any key or clicking in the preview window advances to the next page; typing command-period (Escape on Windows) cancels the preview without displaying all the pages. When the document can't be displayed on the main screen at 100% size (which is most of the time, unless you have a big monitor), the page is scaled to fit.

print

Syntax:
```
print member whichCastmember [,... ]
print member whichCastmember of castLib whichLib [, ... ]
print castLib whichCast  [, ... ]
print sprite whichSprite  [, ... ]
print string [, ... ]
print list [, ... ]
print document
```

The `print` command is an extension to Lingo provided by the PrintOMatic Xtra. This command is used to print one or more items to the currently selected printer. The following Director objects are printable:

type of object	what gets printed
text field cast member	the text of the field, using the specified fonts and styles
rich text cast member	the bitmap image of the cast member, including anti-aliasing
bitmap cast member	the cast member graphic
PICT cast member	the cast member graphic
cast library	all printable cast members in the library, in cast sequence
sprite	the cast member of the sprite
text string	the text string, in the default font (Geneva10pt on Macintosh, Arial 10pt on Windows)
list (linear or property)	the elements in the list
document	the contents of the PrintOMatic document

Examples:

You can print a number of objects together with a single call to the `print` command:

```
print "Printed Output" & RETURN , member "image" , member "someText"
```

One of the most powerful uses of the `print` command is to assemble all the elements of a document into a single cast library, and print the entire cast library with one line of Lingo:

```
print castLib "document"
```

Miscellaneous Routines

The following commands are odds and ends that didn't fit any other category:

hideMessages	hides the PrintOMatic Xtra's error alerts
setPageNumSymbol	sets the page numbering substitution symbol
register	registers your copy of the PrintOMatic Xtra
setLowMemLimits	sets the Xtra's low-memory limits

hideMessages

Syntax: `hideMessages` *document, trueOrFalse*

The `hideMessages` command will prevent PrintOMatic from displaying alert dialogs when something goes wrong. You should strive to make your code solid enough so PrintOMatic will never have to display an error alert (without invoking this command). But if you need to suppress the messages, you can do it here.

setPageNumSymbol

Syntax: `setPageNumSymbol` *document, symbol*

The `setPageNumSymbol` command selects a text symbol whose every occurrence in the document will be replaced with the current page number at printing time. This lets you number your pages by including a text element on the master page containing the symbol you define for page numbering. Call `setPageNumSymbol` with a value of "" (an empty string) to turn off this feature.

register

Syntax: `register(xtra "PrintOMatic", serialNumber)`

Returns: `TRUE` if the serial number is valid

The `register` command will register the PrintOMatic Xtra, once you license PrintOMatic and obtain a valid serial number. This will enable the methods for fully customizing your print job, such as `setProgressMsg` and `setProgressPict`. It will also remove those annoying "unregistered copy" messages from your print progress dialog and printed output.

The registration process only registers your copy of PrintOMatic **temporarily**. Your serial number is stored in Director's registry and might not be stored for any longer than the current Director session. If you re-install Director, move your movie to another computer, or use PrintOMatic from a Projector, the serial number will likely be unavailable the next time you use PrintOMatic.

Therefore, you should add the registration code below to the `startMovie` handler of **every** movie that uses the PrintOMatic Xtra. This will ensure that PrintOMatic can always find its serial number when it needs to.

Example:

The following code registers your copy of the PrintOMatic Xtra for the duration of the current Director session:

```
register(xtra "PrintOMatic", "<your serial number>")
```

setLowMemLimits

Syntax: `setLowMemLimits` *document*, *globalHeap*, *localHeap*

This feature is not yet implemented in the current version of the PrintOMatic Xtra.

Macintosh-Only Routines

The following commands are only available in the Macintosh version of the PrintOMatic Xtra. They are provided primarily for functional backwards-compatibility with the older, XObject version of PrintOMatic:

printToPictFiles	prints the document to a series of PICT images on disk
draw1bitStagePicture	draws the stage contents as a 1-bit dithered image
loadPageSetup	loads saved Page Setup settings from a file on disk
savePageSetup	saves the current Page Setup settings to a file on disk

If you are creating a cross-platform Director production, you should avoid using these routines, or make sure to only use them within blocks of Lingo that are executed when your production is running on a Macintosh computer:

```
if the machineType <> 256 then
    -- running on a Mac
    printToPictFiles doc, the pathName&"picts"
end if
```

printToPictFiles

Syntax: `printToPictFiles` *document*, *folderName*

NOTE: This is a Macintosh-only command. Do not use this command with the Windows version of the PrintOMatic Xtra.

Outputs your document to a series of PICT files on disk. If you do not provide a *folderName*, the user will be prompted to select a folder. The PICT files are sequentially titled "Page-#". The PICTs will over-write any files in the destination folder with the same name. If you only want to print certain pages to disk, present the user with the print job setup dialog using `doJobSetup` first. `printToPictFiles` will honor the start and end page set from the job setup dialog.

draw1bitStagePicture

Syntax: `draw1bitStagePicture` *document*`,` `[` *rect* `|` *topLeftPoint* `],` `[` *clipRect*`],` *grabOffscreen*

NOTE: This is a Macintosh-only command. Do not use this command with the Windows version of the PrintOMatic Xtra.

Places a screen shot of the stage contents, dithered to 1-bit color, on the current page (or MIAW contents if `draw1bitStagePicture` is called from a movie-in-a-window). If a *clipRect* is specified, only that portion of the stage or MIAW will be grabbed. Note that `draw1bitStagePicture` takes a "faithful" screen grab of the stage contents, including the cursor, sprite "trails", and any other windows that might overlap the stage. However, if *grabOffscreen* is `TRUE`, the stage picture will be grabbed from Director's off-screen buffer instead, and these extraneous elements will not be included in the resulting picture.

Examples:

```
draw1bitStagePicture doc, Point(0,0)
draw1bitStagePicture doc, Point(0,0), Rect(0,0,100,300), TRUE
```

loadPageSetup

Syntax: `loadPageSetup(`*document, fileName, resourceID*`)`

Returns: `TRUE` if page settings don't match the current printer, otherwise `FALSE`

NOTE: This is a Macintosh-only command. Do not use this command with the Windows version of the PrintOMatic Xtra.

Retrieves Page Setup information stored in a file on disk. If one or more Page Setup values needs to be changed to match the currently selected printer, this command returns `TRUE`. In this case, you should probably warn the user, and display the Page Setup dialog using `doPageSetup`. Otherwise, this command returns `FALSE`.

savePageSetup

Syntax: `savePageSetup` *document, fileName, fileType, fileCreator, resourceID*

NOTE: This is a Macintosh-only command. Do not use this command with the Windows version of the PrintOMatic Xtra.

Saves the Page Setup information of the current document to a 'PHDL' resource in a file on disk. If the specified file doesn't exist, PrintOMatic creates the file, using the 4-*fileCreator* and *fileType* codes you specify. If a 'PHDL' resource with the requested *resourceID* already exists, it is replaced by the current settings.

PrintOMatic Release Notes

1.5d1 First Macintosh beta release. Documentation and help file (specifically, the "Using" section) are still incomplete.

1.5d2 First Windows beta release. Improved file format checking code on both platforms. Fixed PICT file sizing problems on Windows. Updated "unregistered copy" messages with e-mail contact information.

1.5d3 Fixed a Windows bug that caused `drawStagePicture` to draw a black rectangle in 16- and 32-bit color. Updated the About Box with new artwork. Added `draw1BitStagePicture`, `printToPictFiles`, `loadPageSetup`, and `savePageSetup` routines to Macintosh version ONLY (for backwards-compatibility). Make the "new" handler fail silently when there's an error (such as no printer selected), so you can present your own dialog to users.

1.5d4 Added version numbers to the About box. Improved the Mac printer database to save the name of the print driver along with the settings.

1.5 Fixed a bug in the Macintosh version that would print the wrong pages if a specific page rage was selected in the Job Setup dialog. Added protective code to defend against a Director bug that returns a text size of 0 from text fields. Recompiled to remedy a bug that botched printing in Windows 95.

1.5.1 Fixed a bug that caused `append` to lock up in cases when a series of different-styled carriage returns spanned a page break. Fixed Macintosh bugs that caused print preview to mess up the Stage palette, and `drawText` to occasionally print text in funny colors.

1.5.2 Fixed a bug that caused the Windows version to crash when calling `setPage` 0.

1.5.3 Fixed a Windows bug that caused problems printing files from disk in Windows 95.