

Scripting Tutorial:
**Working with
Documents**

JavaScript Version

Adobe InDesign CS



ADobe SYSTEMS INCORPORATED

Corporate Headquarters

345 Park Avenue

San Jose, CA 95110-2704

(408) 536-6000

Scripting Tutorial: Working with Documents—JavaScript Version

Copyright 2004 Adobe Systems Incorporated.

All rights reserved.

The information in this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in this document. The software described in this document is furnished under license and may only be used or copied in accordance with the terms of such license.

Acrobat, Adobe, InCopy, and InDesign are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. Macintosh and Apple are registered trademarks, and Mac OS and AppleScript are trademarks of Apple Computer, Inc. registered in the United States and other countries. Microsoft, Visual Basic, Windows, Windows 95, Windows 98, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other brand and product names are trademarks or registered trademarks of their respective holders.

Working with Documents

The work you do in InDesign revolves around documents—creating them, saving them, and populating them with page items, colors, styles, and text. Almost every document-related task can be automated using scripting.

This tutorial will show you how to:

- Create a new document
- Close a document
- Save a document
- Set document page size
- Define bleed and slug areas
- Specify page columns and margins
- Change the size of the pasteboard
- Create guides
- Use InDesign’s baseline and document grids
- Change measurement units and ruler origin
- Define and apply document presets
- Set up master pages
- Set text defaults
- Add XMP metadata (“file info”)
- Create a document template
- Print a document
- Export a document as PDF
- Export the pages of a document as EPS

If you have not already worked through the chapter, “Getting Started with InDesign Scripting” in the *Adobe InDesign CS Scripting Guide*, you might want to do so before continuing with this tutorial. This tutorial assumes that you have already read that chapter, and know how to create a script.

For more information on InDesign scripting, go to <http://www.adobe.com/products/indesign/scripting.html>, or visit the InDesign Scripting User to User forum at <http://www.adobeforums.com>.

To use the scripts in this document, copy the scripts out of the PDF and paste them into a text editor (such as Notepad in Windows or BBEdit on the Mac OS). Save the scripts as plain text with the file extension “.js” and place the saved file (or an alias/shortcut to it) in the Scripts folder inside the Presets folder in your InDesign folder (create the Scripts folder if it does not already exist). Once you’ve done this, you can run the script by double-clicking the script name in the Scripts palette (choose Window>Scripting>Scripts to display the Scripts palette).

Due to the layout of this document and the nature of the copy/paste process, some long lines of code may include unwanted line breaks when you paste them into your text editor. In some cases, these line breaks will cause the script to fail. To fix the script, you'll need to remove the unnecessary line breaks. In this document, all non-comment lines (lines that start with “//” are comments) will end with either “{”, “}”, or “;”. The following is an example of a long line:

```
guides.add(myDocument.layers.item("GuideLayer"), {orientation:HorizontalOrVertical.  
horizontal, location:marginPreferences.top, fitToPage:false});
```

When you paste this line into a text editor, remove the line break—the line should end with the semicolon (“;”). If you don't remove the line break, the script will generate an error.

Creating a New Document

Before you can do anything with a document, you've got to make one. The following script shows how to do that.

```
//MakeDocument.js  
//An InDesign CS JavaScript  
//Creates a new document.  
var myDocument = app.documents.add();
```

The `document.add` method can take two optional parameters, as shown in the following script:

```
//MakeDocumentWithParameters.js  
//An InDesign CS JavaScript  
//Shows how to use the parameters of the document.add method.  
//The first parameter (showingWindow) controls the visibility of the document. Hidden documents  
//are not minimized, and will not appear until you add a new window to the document.  
//The second parameter (documentPreset) specifies the document preset to use.  
//The following line assumes that you have a document preset named "Flyer" on your system.  
var myDocument = app.documents.add(true, app.documentPresets.item("Flyer"));
```

Opening a Document

The following example script shows how to open an existing document using scripting.

```
//OpenDocument.js  
//An InDesign CS JavaScript  
//Opens an existing document. You'll have to fill in your own file path.  
app.open(File("/c/myTestDocument.indd"));
```

You can choose to hide the document after you open it by setting the `showingWindow` parameter of the `open` method to `false` (the default value is `true`). You might want to do this to improve performance of a script. To show a hidden document, create a new window, as shown in the following script:

```
//OpenDocumentInBackground.js
//An InDesign CS JavaScript
//Opens an existing document in the background, then shows the document.
//You'll have to fill in your own file path.
var myDocument = app.open(File("/c/myTestDocument.indd"), false);
//At this point, you can do things with the document without showing the document window.
//In some cases, scripts will run faster when the document window is not visible.
//When you want to show the hidden document, create a new window.
var myLayoutWindow = myDocument.windows.add();
```

Closing a Document

The close method closes a document:

```
//CloseDocument.js
//Closes the active document.
app.activeDocument.close();
//Note that you could also use:
//app.documents.item(0).close();
```

The close method can take up to two optional parameters:

```
//CloseWithParameters.js
//Use SaveOptions.yes to save the document, SaveOptions.no to close the document
//without saving, or SaveOptions.ask to display a prompt. If you use SaveOptions.yes,
//you'll need to provide a reference to a file to save to in the second parameter (SavingIn).
//Note that the file path is provided using the JavaScript URI form rather
//than the platform-specific form.
var myFile;
//If the file has not been saved, display a prompt.
if(app.activeDocument.saved != true){
    app.activeDocument.close(SaveOptions.ask);
    //Or, to save to a specific file name:
    //var myFile = File("/c/myTestDocument.indd");
    //app.activeDocument.close(SaveOptions.yes, myFile);
}
else{
    //If the file has already been saved, save it.
    app.activeDocument.close(SaveOptions.yes);
}
```

Here's an example that closes all open documents without saving:^{*}

```
//CloseAll.js
//Closes all open documents without saving.
for(myCounter = app.documents.length; myCounter > 0; myCounter--){
    app.documents.item(myCounter-1).close(SaveOptions.no);
}
```

Saving a Document

In InDesign's user interface, you save a file using the Save command, and you save a file to another file name using the Save As command. In InDesign scripting, the save method takes care of both operations.

```
//SaveDocument.js
//If the active document has been changed since it was last saved, save it.
if(app.activeDocument.modified == true){
    app.activeDocument.save();
}
```

The save method has two optional parameters: the first (To) specifies the file to save to; the second (Stationery) can be set to true to save the document as a template.

```
//SaveDocumentAs.js
//If the active document has not been saved (ever), save it.
if(app.activeDocument.saved == false){
    //If you do not provide a file name, InDesign will display the Save dialog box.
    app.activeDocument.save(new File("/c/myTestDocument.indd"));
}
```

The following example shows how to save a document as a template.

```
//SaveAsTemplate.js
//Save the active document as a template.
var myFileName;
if(app.activeDocument.saved == true){
    //Convert the file name to a string.
    myFileName = app.activeDocument.fullName + "";
    //If the file name contains the extension ".indd", change it to ".indt".
    if(myFileName.indexOf(".indd") != -1){
        var myRegularExpression = /.indd/gi;
        myFileName = myFileName.replace(myRegularExpression, ".indt");
    }
}
//If the document has not been saved, then give it a default file name/file path.
else{
    myFileName = "/c/myTestDocument.indt";
}
app.activeDocument.save(File(myFileName), true);
```

Defining Page Size and Document Length

When you create a new document using InDesign's user interface, you can specify the page size, number of pages, and whether the document uses facing pages or not. When you create a document using InDesign scripting, you use the `documents.add` method. After you've added a document, you can use the `documentPreferences` object to control the page size and page orientation.

```
//DocumentPreferences.js
//Use the documentPreferences object to change the
//dimensions and orientation of the document.
var myDocument = app.documents.add();
with(myDocument.documentPreferences) {
    pageHeight = "800pt";
    pageWidth = "600pt";
    pageOrientation = PageOrientation.landscape;
    pagesPerDocument = 16;
}
```

Note that the `application` object also has a `documentPreferences` object, and that you can set the application defaults for page height, page width, and other properties by changing the properties of this object.

Bleeds and Slugs

In InDesign terms, a “bleed” or a “slug” is an area outside the page margins that can be printed or included in an exported PDF. Typically, these areas are used for objects that extend beyond the page edges (bleed) and job/document information (slug). The two areas can be printed/exported independently—you might want to omit slug information for the final printing of a document, for example. The following script sets up the bleed and slug for a new document.

```
//BleedAndSlug.js
//An InDesign CS JavaScript
//Create a new document.
myDocument = app.documents.add();
//The bleed and slug properties belong to the documentPreferences object.
with(myDocument.documentPreferences) {
    //Bleed
    documentBleedBottomOffset = "3p";
    documentBleedTopOffset = "3p";
    documentBleedInsideOrLeftOffset = "3p";
    documentBleedOutsideOrRightOffset = "3p";
    //Slug
    slugBottomOffset = "18p";
    slugTopOffset = "3p";
    slugInsideOrLeftOffset = "3p";
    slugRightOrOutsideOffset = "3p";
}
```

If all of the bleed distances are equal, as in the above example, you can use the `documentBleedUniformSize` property:

```
//UniformBleed.js
//An InDesign CS JavaScript
//Create a new document.
myDocument = app.documents.add();
//The bleed properties belong to the documentPreferences object.
with(myDocument.documentPreferences) {
    //Bleed
    documentBleedUniformSize = true;
    documentBleedBottomOffset = "3p";
}
```

If all of the slug distances are equal, you can use the `documentSlugUniformSize` property:

```
//UniformSlug.js
//An InDesign CS JavaScript
//Create a new document.
myDocument = app.documents.add();
//The slug properties belong to the documentPreferences object.
with(myDocument.documentPreferences) {
    //Slug:
    documentSlugUniformSize = true;
    slugBottomOffset = "3p";
}
```

In addition to setting the bleed and slug widths/heights, you can also control the color used to draw the guides defining the bleed and slug. This property is not found in the `documentPreferences` object--instead, it's in the `pasteboardPreferences` object.

```
//BleedSlugGuideColors.js
//Set the colors used to display the bleed and slug guides.
with(app.activeDocument.pasteboardPreferences) {
    //Any of InDesign's guides can use the UIColors constants...
    bleedGuideColor = UIColors.cuteTeal;
    slugGuideColor = UIColors.charcoal;
    //...or you can specify an array of RGB values (with values from 0 to 255)
    //bleedGuideColor = [0, 198, 192];
    //slugGuideColor = [192, 192, 192];
}
```

Page Margins and Columns

Each page in a document can have its own margin and column settings. In InDesign scripting, these properties are part of the marginPreferences object for each page. Here's an example script that creates a new document, then sets the margins and columns for the first page.

```
//MarginsAndColumns.js
//Sets up the margins and columns for the first page of an example document.
myDocument = app.documents.add();
with (myDocument.pages.item(0).marginPreferences) {
    columnCount = 3;
    //columnGutter can be a number or a measurement string.
    columnGutter = "1p";
    bottom = "6p"
    //When document.documentPreferences.facingPages == true,
    // "left" means inside; "right" means outside.
    left = "6p"
    right = "4p"
    top = "4p"
}
```

If you're creating very small pages (for individual newspaper advertisements, for example), you may have found that InDesign will not allow you to create a page that is smaller than the sum of the relevant margins (i.e., the width of the page must be greater than the sum of the left and right default page margins, the height of the page must be greater than the sum of the top and bottom margins). In the user interface, you can easily fix this by entering new values in the document default page margin fields in the New Document dialog box as you create the document.

From scripting, however, the solution is not as clear—the document is created using the application default margin preferences. These margins are applied to the pages of the document, including master pages. Setting the document margin preferences affects only new pages, and has no effect on existing pages. If you try to set the page height and page width to values smaller than the sum of the corresponding margins on any of the existing pages, InDesign will not change the page size.

There are really two solutions to this problem. The first is to set the margins of the existing pages before you try to change the page size, as shown in the following example:

```
//PageMargins.js
//Creates a new document and sets up page margins.
var myDocument = app.documents.add();
//The following assumes that your default document contains a single page.
myDocument.pages.item(0).marginPreferences.top = 0;
myDocument.pages.item(0).marginPreferences.left = 0;
myDocument.pages.item(0).marginPreferences.bottom = 0;
myDocument.pages.item(0).marginPreferences.right = 0;
//The following assumes that your default master spread contains two pages.
myDocument.masterSpreads.item(0).pages.item(0).marginPreferences.top = 0;
myDocument.masterSpreads.item(0).pages.item(0).marginPreferences.left = 0;
myDocument.masterSpreads.item(0).pages.item(0).marginPreferences.bottom = 0;
myDocument.masterSpreads.item(0).pages.item(0).marginPreferences.right = 0;
myDocument.masterSpreads.item(0).pages.item(1).marginPreferences.top = 0;
myDocument.masterSpreads.item(0).pages.item(1).marginPreferences.left = 0;
myDocument.masterSpreads.item(0).pages.item(1).marginPreferences.bottom = 0;
myDocument.masterSpreads.item(0).pages.item(1).marginPreferences.right = 0;
```

```
//Create a very small page size to demonstrate that this works:  
myDocument.documentPreferences.pageHeight = "2cm";  
myDocument.documentPreferences.pageWidth = "2cm";
```

Alternatively, you can change the application default margin preferences before you create the document, as shown in the following example:

```
//ApplicationPageMargins.js  
//Sets the application default page margins. All new documents will be created  
//using these settings; existing documents will be unaffected.  
with (app.marginPreferences){  
    //Save the current application default margin preferences.  
    var myY1 = top;  
    var myX1 = left;  
    var myY2 = bottom;  
    var myX2 = right;  
    //Set the application default margin preferences.  
    top = 0;  
    left = 0;  
    bottom = 0;  
    right = 0;  
}  
//Create a new example document to demonstrate the change.  
var myDocument = app.documents.add();  
myDocument.documentPreferences.pageHeight = "2cm";  
myDocument.documentPreferences.pageWidth = "2cm";  
//Reset the application default margin preferences to their former state.  
with (app.marginPreferences){  
    top = myY1;  
    left = myX1 ;  
    bottom = myY2;  
    right = myX2;  
}
```

Pasteboard

The pasteboard is an area that surrounds InDesign pages. It can be used for temporary storage of page items. You can change the size of the pasteboard and its color using scripting. The `pasteboardColor` property controls the color of the pasteboard in Normal mode; the `previewBackgroundColor` property sets the color of the pasteboard in Preview mode.

```
//PasteboardPreferences.js
//Create a new document and change the size of the pasteboard.
myDocument = app.documents.add();
with(myDocument.pasteboardPreferences) {
    //You can use either a number or a measurement string to set the space above/below.
    minimumSpaceAboveAndBelow = "12p";
    //You can set the pasteboard color to any of the predefined UIColor constants...
    pasteboardColor = UIColors.white;
    previewBackgroundColor = UIColors.gray;
    //...or you can specify an array of RGB values (with values from 0 to 255)
    //pasteboardColor = [0, 0, 0];
    //previewBackgroundColor = [192, 192, 192];
}
```

Guides

InDesign's guides give you an easy way to position objects on the pages of your document. Here's an example use of guides.

```
//Guides.js
//Create a new document, add guides, and set guide properties.
var myDocument = app.documents.add();
var myPageWidth = myDocument.documentPreferences.pageWidth;
var myPageHeight = myDocument.documentPreferences.pageHeight;
with(myDocument.pages.item(0)) {
    //Place guides at the margins of the page.
    guides.add(undefined, {orientation:HorizontalOrVertical.vertical, location:marginPreferences.left});
    guides.add(undefined, {orientation:HorizontalOrVertical.vertical, location:(myPageWidth -
marginPreferences.right)});
    guides.add(undefined, {orientation:HorizontalOrVertical.horizontal, location:marginPreferences.top});
    guides.add(undefined, {orientation:HorizontalOrVertical.horizontal, location:(myPageHeight -
marginPreferences.bottom)});
    //Place a guide at the vertical center of the page.
    guides.add(undefined, {orientation:HorizontalOrVertical.vertical, location:(myPageWidth/2)});
    //Place a guide at the horizontal center of the page.
    guides.add(undefined, {orientation:HorizontalOrVertical.horizontal, location:(myPageHeight/2)});
}
```

Horizontal guides can be limited to a given page, or can extend across all of the pages in a spread. In InDesign scripting, you can control this using the `fitToPage` property. (This property is ignored by vertical guides.)

```
//SpreadAndPageGuides.js
//Demonstrate the difference between spread guides and page guides.
var myDocument = app.documents.add();
myDocument.documentPreferences.facingPages = true;
myDocument.documentPreferences.pagesPerDocument = 3;
with(myDocument.spreads.item(1)){
    //Note the difference between these two guides on pages 2 and 3.
    guides.add(undefined, {orientation:HorizontalOrVertical.horizontal, location:"6p", fitToPage:true});
    guides.add(undefined, {orientation:HorizontalOrVertical.horizontal, location:"9p", fitToPage:false});
}
```

You can use scripting to change the layer, color, and visibility of guides, just as you can from the user interface.

```
//GuideOptions.js
//Shows how to set guide options.
var myGuide;
var myDocument = app.documents.add();
//Create a layer named "guide layer".
var myLayer = myDocument.layers.add({name:"guide layer"});
//Add a series of guides to page 1.
with(myDocument.pages.item(0)){
    //Create a guide on the layer we created above.
    myGuide = guides.add(myLayer, {orientation:HorizontalOrVertical.horizontal, location:"12p"});
    //Another way to make a guide on a specific layer.
    myGuide = guides.add(undefined, {itemLayer:myLayer, orientation:HorizontalOrVertical.horizontal,
location:"14p"});
    //Make a locked guide.
    myGuide = guides.add(myLayer,{locked:true, orientation:HorizontalOrVertical.horizontal, location:"16p"});
    //Set the view threshold of a guide.
    myGuide = guides.add(myLayer,{viewThreshold:100, orientation:HorizontalOrVertical.horizontal,
location:"18p"});
    //Set the guide color of a guide using a UIColors constant.
    myGuide = guides.add(myLayer,{guideColor:UIColors.gray, orientation:HorizontalOrVertical.horizontal,
location:"20p"});
    //Set the guide color of a guide using an RGB array.
    myGuide = guides.add(myLayer,{guideColor:[192, 192, 192], orientation:HorizontalOrVertical.horizontal,
location:"22p"});
}
```

You can also create guides using the `createGuides` method of spreads and master spreads.

```
//CreateGuides.js
//Add a series of guides using the createGuides method.
var myDocument = app.documents.add();
with (myDocument.spreads.item(0)){
    //Parameters (all optional): row count, column count, row gutter, column gutter,
    //guide color, fit margins, remove existing, layer.
    //Note that the createGuides method does not take an RGB array for the guide color parameter.
    createGuides(4, 4, "1p", "1p", UIColors.gray, true, true, myDocument.layers.item(0));
}
```

10 Working with Documents

Setting Grid Preferences

To control the properties of the document and baseline grid, you set the properties of the gridPreferences object, as shown in the following script.

```
//DocumentAndBaselineGrids.js
//Creates a document, then sets preferences for the document grid
//and baseline grid.
var myDocument = app.documents.add();
//Set the document measurement units to points.
myDocument.viewPreferences.horizontalMeasurementUnits = MeasurementUnits.points;
myDocument.viewPreferences.verticalMeasurementUnits = MeasurementUnits.points;
//Set up grid preferences.
with(myDocument.gridPreferences){
    baselineStart = 56;
    baselineDivision = 14;
    baselineShown = false;
    horizontalGridlineDivision = 14;
    horizontalGridSubdivision = 5
    verticalGridlineDivision = 14;
    verticalGridSubdivision = 5
    documentGridShown = false;
}
}
```

Snapping to Guides and Grids

All of the “snap” settings for the grids and guides of a document are found in the properties of the guidePreferences and gridPreferences objects. Here’s an example:

```
//GuideAndGridPreferences.js
//Sets preferences for guides and grids.
//Assumes you have a document open.
var myDocument = app.activeDocument;
with(myDocument.guidePreferences){
    guidesInBack = true;
    guidesLocked = false;
    guidesShown = true;
    guidesSnapTo = true;
}
with(myDocument.gridPreferences){
    documentGridShown = false;
    documentGridSnapTo = true;
    //Objects "snap" to the baseline grid when guidePreferences.guideSnapTo is set to true.
    baselineGridShown = true;
}
```

Measurement Units

In the example scripts we've presented, you'll see that we've used "measurement strings" (strings that force InDesign to use a specific measurement unit, "8.5i", for example, for 8.5 inches). We do this because we don't know what default measurement system you might be using when you run the script.

To specify the measurement system used in a script, you need to use the document's `viewPreferences` object.

```
//ViewPreferences.js
//Changes the measurement units used by the active document.
//Assumes you have a document open.
var myDocument = app.activeDocument;
with(myDocument.viewPreferences) {
    //Measurement unit choices are:
    /* MeasurementUnits.picas
    /* MeasurementUnits.points
    /* MeasurementUnits.inches
    /* MeasurementUnits.inchesDecimal
    /* MeasurementUnits.millimeters
    /* MeasurementUnits.centimeters
    /* MeasurementUnits.ciceros
    //Set horizontal and vertical measurement units to points.
    horizontalMeasurementUnits = MeasurementUnits.points;
    verticalMeasurementUnits = MeasurementUnits.points;
}
```

If you're writing a script that needs to work with a specific measurement system, you can change the measurement units at the beginning of the script and then restore the original measurement units at the end of the script, as shown in the following example:

```
//ResetMeasurementUnits.js
//Changes, then resets the active document's measurement units.
//Assumes you have a document open.
var myDocument = app.activeDocument
with (myDocument.viewPreferences){
    var myOldXUnits = horizontalMeasurementUnits;
    var myOldYUnits = verticalMeasurementUnits;
    horizontalMeasurementUnits = MeasurementUnits.points;
    verticalMeasurementUnits = MeasurementUnits.points;
}
//At this point, you can perform any series of script actions that depend on the measurement
//units you've set. At the end of the script, reset the units to their original state.
with (myDocument.viewPreferences){
    try{
        horizontalMeasurementUnits = myOldXUnits;
        verticalMeasurementUnits = myOldYUnits;
    }
    catch(myError){
        alert("Could not reset custom measurement units.");
    }
}
```

Note that InDesign scripting does not support setting the horizontal or vertical measurement units to a custom value, as InDesign's user interface does.

Document Presets

InDesign's document presets give you a way to store and apply commonly-used document setup information (page size, page margins, columns, and bleed and slug areas). When you create a new document, you can base the document on a document preset.

To create a document preset "by example," open a document that has the document setup properties you want defined in the document preset, then run the following script:

```
//DocumentPresetByExample.js
//Creates a document preset based on the current document settings.
//Assumes you have a document open.

var myDocumentPreset;
if(app.documents.length > 0){
    var myDocument = app.activeDocument;
    //If the document preset "myDocumentPreset" does not already exist, create it.
    myDocumentPreset = app.documentPresets.item("myDocumentPreset");
    try {
        var myPresetName = myDocumentPreset.name;
    }
    catch (myError){
        myDocumentPreset = app.documentPresets.add({name:"myDocumentPreset"});
    }
    //Fill in the properties of the document preset with the corresponding
    //properties of the active document.
    with(myDocumentPreset){
        //Note that the following gets the page margins from the margin preferences
        //of the document; to get the margin preferences from the active page,
        //replace "app.activeDocument" with "app.activeWindow.activePage" in the
        //following six lines (assuming the active window is a layout window).
        left = app.activeDocument.marginPreferences.left;
        right = app.activeDocument.marginPreferences.right;
        top = app.activeDocument.marginPreferences.top;
        bottom = app.activeDocument.marginPreferences.bottom;
        columnCount = app.activeDocument.marginPreferences.columnCount;
        columnGutter = app.activeDocument.marginPreferences.columnGutter;
        documentBleedBottom = app.activeDocument.documentPreferences.documentBleedBottomOffset;
        documentBleedTop = app.activeDocument.documentPreferences.documentBleedTopOffset;
        documentBleedLeft = app.activeDocument.documentPreferences.documentBleedInsideOrLeftOffset;
        documentBleedRight = app.activeDocument.documentPreferences.documentBleedOutsideOrRightOffset;
        facingPages = app.activeDocument.documentPreferences.facingPages;
        pageHeight = app.activeDocument.documentPreferences.pageHeight;
        pageWidth = app.activeDocument.documentPreferences.pageWidth;
        pageOrientation = app.activeDocument.documentPreferences.pageOrientation;
        pagesPerDocument = app.activeDocument.documentPreferences.pagesPerDocument;
        slugBottomOffset = app.activeDocument.documentPreferences.slugBottomOffset;
        slugTopOffset = app.activeDocument.documentPreferences.slugTopOffset;
        slugInsideOrLeftOffset = app.activeDocument.documentPreferences.slugInsideOrLeftOffset;
        slugRightOrOutsideOffset = app.activeDocument.documentPreferences.slugRightOrOutsideOffset;
    }
}
```

The following example script shows how to create a document preset “from scratch”:

```
//DocumentPreset.js
//Creates a new document preset.
var myDocumentPreset;
//If the document preset "myDocumentPreset" does not already exist, create it.
myDocumentPreset = app.documentPresets.item("myDocumentPreset");
try {
    var myPresetName = myDocumentPreset.name;
}
catch (myError) {
    myDocumentPreset = app.documentPresets.add({name:"myDocumentPreset"});
}
//Fill in the properties of the document preset.
with(myDocumentPreset){
    pageHeight = "9i";
    pageWidth = "7i";
    left = "4p";
    right = "6p";
    top = "4p";
    bottom = "9p";
    columnCount = 1;
    documentBleedBottom = "3p";
    documentBleedTop = "3p";
    documentBleedLeft = "3p";
    documentBleedRight = "3p";
    facingPages = true;
    pageOrientation = PageOrientation.portrait;
    pagesPerDocument = 1;
    slugBottomOffset = "18p";
    slugTopOffset = "3p";
    slugInsideOrLeftOffset = "3p";
    slugRightOrOutsideOffset = "3p";
}
```

To create a new document using a document preset, use the document preset parameter of the add method:

```
//MakeDocumentWithPreset.js
//Creates a new document using a specified document preset.
//Replace "myDocumentPreset" in the following line with the name
//of the document preset you want to use.
var myDocument = app.documents.add(true, app.documentPresets.item("myDocumentPreset"));
```

Master Spreads

After you've set up the basic document page size, slug, and bleed, you'll probably want to define the document's master pages.

```
//MasterSpread.js
//Creates a document, then demonstrates setting master spread properties.
//Set up the first master spread in a new document.
myDocument = app.documents.add();
//Set up the document.
with(myDocument.documentPreferences) {
    pageHeight = "11i";
    pageWidth = "8.5i";
    facingPages = true;
    pageOrientation = PageOrientation.portrait;
}
//Set the document's ruler origin to page origin. This is very important
//---if you don't do this, getting objects to the correct position on the
//page is much more difficult.
myDocument.viewPreferences.rulerOrigin = RulerOrigin.pageOrigin;
with(myDocument.masterSpreads.item(0)){
    //Set up the left page (verso).
    with(pages.item(0)){
        with	marginPreferences{
            columnCount = 3;
            columnGutter = "1p";
            bottom = "6p";
            //"left" means inside; "right" means outside.
            left = "6p";
            right = "4p";
            top = "4p";
        }
        //Add a simple footer with a section number and page number.
        with(textFrames.add()){
            geometricBounds = ["61p", "4p", "62p", "45p"];
            insertionPoints.item(0).contents = SpecialCharacters.sectionMarker;
            insertionPoints.item(0).contents = SpecialCharacters.emSpace;
            insertionPoints.item(0).contents = SpecialCharacters.autoPageNumber;
            paragraphs.item(0).justification = Justification.leftAlign;
        }
    }
    //Set up the right page (recto).
    with(pages.item(1)){
        with	marginPreferences{
            columnCount = 3;
            columnGutter = "1p";
            bottom = "6p";
            //"left" means inside; "right" means outside.
            left = "6p";
            right = "4p";
            top = "4p";
        }
        //Add a simple footer with a section number and page number.
    }
}
```

```

        with(textFrames.add()) {
            geometricBounds = ["61p", "6p", "62p", "47p"];
            insertionPoints.item(0).contents = SpecialCharacters.autoPageNumber;
            insertionPoints.item(0).contents = SpecialCharacters.emSpace;
            insertionPoints.item(0).contents = SpecialCharacters.sectionMarker;
            paragraphs.item(0).justification = Justification.rightAlign;
        }
    }
}

```

To apply a master spread to a document page, use the appliedMaster property of the document page.

```

//ApplyMaster.js
//Applies a master spread to a page.
//Assumes that the active document has a master page named "B-Master"
//and at least three pages--page 3 is pages.item(2) because JavaScript arrays are zero-based.
app.activeDocument.pages.item(2).appliedMaster = app.activeDocument.masterSpreads.item("B-
Master");
You use the same property to apply a master spread to a master spread page.
//ApplyMasterToMaster.js
//Applies a master spread to a master page.
//Assumes that the active document has master spread named "B-Master"
//that is not the same as the first master spread in the document.
app.activeDocument.masterSpreads.item(0).pages.item(0).appliedMaster = app.activeDocument.
masterSpreads.item("B-Master");

```

Setting Text Defaults

You can set the default text formatting attributes for the application or for individual documents. If you set the text defaults for the application, they will become the text defaults for all new documents—existing documents will remain unchanged. When you set the text defaults for a document, any new text you enter in the document will pick up those defaults, and any existing text will remain unchanged.

```

//ApplicationTextDefaults.js
//Sets the application text defaults, which will become the text defaults for all
//new documents. Existing documents will remain unchanged.
with(app.textDefaults){
    alignToBaseline = true;
    try{
        appliedFont = app.fonts.item("Minion Pro");
    }
    catch(e){}
    fontStyle = "Normal";
    try{
        appliedLanguage = "English: USA";
    }
    catch(e){}
    autoLeading = 100;
    balanceRaggedLines = false;
    baselineShift = 0;
    capitalization = Capitalization.normal;
}

```

```

composer = "Adobe Paragraph Composer";
desiredGlyphScaling = 100;
desiredLetterSpacing = 0;
desiredWordSpacing = 100;
dropCapCharacters = 0;
if(dropCapCharacters != 0){
    dropCapLines = 3;
    //Assumes that the application has a default character style named "myDropCap"
    dropCapStyle = app.characterStyles.item("myDropCap");
}
fillColor = app.colors.item("Black");
fillTint = 100;
firstLineIndent = "14pt";
gradientFillAngle
gradientFillLength
gridAlignFirstLineOnly = false;
horizontalScale = 100;
hyphenateAfterFirst = 3;
hyphenateBeforeLast = 4;
hyphenateCapitalizedWords = false;
hyphenateLadderLimit = 1;
hyphenateWordsLongerThan = 5;
hyphenation = true;
hyphenationZone = "3p";
hyphenWeight = 9;
justification = Justification.leftAlign;
keepAllLinesTogether = false;
keepLinesTogether = true;
keepFirstLines = 2;
keepLastLines = 2;
keepWithNext = 0;
kerningMethod = "Optical";
kerningValue = 0;
leading = 14;
leftIndent = 0;
ligatures = true;
maximumGlyphScaling = 100;
maximumLetterSpacing = 0;
maximumWordSpacing = 160;
minimumGlyphScaling = 100;
minimumLetterSpacing = 0;
minimumWordSpacing = 80;
noBreak = false;
otfContextualAlternate = true;
otfDiscretionaryLigature = true;
otfFigureStyle = OTFFigureStyle.proportionalOldstyle;
otfFraction = true;
otfOrdinal = false;
otfSwash = false;
otfTitling = false;
overprintFill = false;
overprintStroke = false;
pointSize = 11;
position = Position.normal;

```

```

rightIndent = 0;
ruleAbove = false;
if(ruleAbove == true){
    ruleAboveColor = app.colors.item("Black");
    ruleAboveGapColor = app.swatches.item("None");
    ruleAboveGapOverprint = false;
    ruleAboveGapTint = 100;
    ruleAboveLeftIndent = 0;
    ruleAboveLineWeight = .25;
    ruleAboveOffset = 14;
    ruleAboveOverprint = false;
    ruleAboveRightIndent = 0;
    ruleAboveTint = 100;
    ruleAboveType = app.strokeStyles.item("Solid");
    ruleAboveWidth = RuleWidth.columnWidth;
}
ruleBelow = false;
if(ruleBelow == true){
    ruleBelowColor = app.colors.item("Black");
    ruleBelowGapColor = app.swatches.item("None");
    ruleBelowGapOverprint = false;
    ruleBelowGapTint = 100;
    ruleBelowLeftIndent = 0;
    ruleBelowLineWeight = .25;
    ruleBelowOffset = 0;
    ruleBelowOverprint = false;
    ruleBelowRightIndent = 0;
    ruleBelowTint = 100;
    ruleBelowType = app.strokeStyles.item("Solid");
    ruleBelowWidth = RuleWidth.columnWidth;
}
singleWordJustification = SingleWordJustification.leftAlign;
skew = 0;
spaceAfter = 0;
spaceBefore = 0;
startParagraph = StartParagraph.anywhere;
strikeThru = false;
if(strikeThru == true){
    strikeThroughColor = app.colors.item("Black");
    strikeThroughGapColor = app.swatches.item("None");
    strikeThroughGapOverprint = false;
    strikeThroughGapTint = 100;
    strikeThroughOffset = 3;
    strikeThroughOverprint = false;
    strikeThroughTint = 100;
    strikeThroughType = app.strokeStyles.item("Solid");
    strikeThroughWeight = .25;
}
strokeColor = app.swatches.item("None");
strokeTint = 100;
strokeWeight = 0;
tracking = 0;
underline = false;
if(underline == true){

```

18 Working with Documents

```
underlineColor = app.colors.item("Black");
underlineGapColor = app.swatches.item("None");
underlineGapOverprint = false;
underlineGapTint = 100;
underlineOffset = 3;
underlineOverprint = false;
underlineTint = 100;
underlineType = app.strokeStyles.item("Solid");
underlineWeight = .25;
}
verticalScale = 100;
}
```

Change the line:

```
with(app.textDefaults) {
```

to:

```
with(app.activeDocument.textDefaults) {
```

to set the text defaults for the active document.

To set the text to a default character style or paragraph style, use the following script:

```
//SetTextDefaultToStyle.js
//Assumes that the active document contains a paragraph style "BodyText"
with(app.activeDocument.textDefaults) {
    appliedParagraphStyle = app.activeDocument.paragraphStyles.item("BodyText");
}
```

XMP Metadata

Metadata is information that describes the content, origin, or other attributes of a file. In InDesign, you enter, edit, and view metadata using the File Info dialog box (File>File Info). This metadata includes the creation and modification dates of the document, the author of the document, the copyright status of the document, and other information. All of this information is stored using XMP (Adobe Extensible Metadata Platform)—an open standard for embedding metadata in a document.

You can add XMP information to a document using InDesign scripting. All of the XMP properties for a document can be found in the document's metadataPreferences object. Here's an example that fills in the standard XMP data for a document. To learn more about XMP, see the XMP specification at <http://partners.adobe.com/asn/developer/pdf/MetadataFramework.pdf>.

```
//MetadataExample.js
//Adds metadata to an example document.
var myDocument = app.documents.add();
with (myDocument.metadataPreferences) {
    author = "Olav Martin Kvern";
    copyrightInfoURL = "http://www.adobe.com";
    copyrightNotice = "This document is copyrighted.";
    copyrightStatus = CopyrightStatus.yes;
    description = "Example of xmp metadata scripting in InDesign CS";
    documentTitle = "XMP Example";
    jobName = "XMP_Example_2004";
    keywords = ["animal", "mineral", "vegetable"];
    //The metadata preferences object also includes the read-only
    //creator, format, creationDate, modificationDate, and serverURL properties that are
    //automatically entered and maintained by InDesign.
    //Create a custom XMP container, "email"
    createContainerItem("http://ns.adobe.com/xap/1.0/", "email");
    setProperty("http://ns.adobe.com/xap/1.0/", "email/*[1]", "okvern@adobe.com");
}
```

As you can see from the example above, XMP information is extensible—if you need to attach metadata to a document that does not fall into one of the categories provided by the metadata preferences object, you can create your own metadata container (“email,” in this example).

Creating a Document Template

In this example, we'll create a new document, define slug and bleed areas, add information to the document's XMP metadata, set up master pages, add page footers, and add job information to a table in the slug area.

```
//DocumentTemplate.js
//Creates a document template, including master pages, layers, a color, paragraph and character
styles, guides, and XMP information.
//Set the application default margin preferences.
with (app.marginPreferences){
    //Save the current application default margin preferences.
    var myY1 = top;
    var myX1 = left;
    var myY2 = bottom;
    var myX2 = right;
    //Set the application default margin preferences.
    //Document baseline grid will be based on 14 points, and
    //all margins are set in increments of 14 points.
    top = 14 * 4 + "pt";
    left = 14 * 4 + "pt";
    bottom = "74pt";
    right = 14 * 5 + "pt";
}
//Make a new document.
var myDocument = app.documents.add();
myDocument.documentPreferences.pageWidth = "7i";
myDocument.documentPreferences.pageHeight = "9i";
myDocument.documentPreferences.pageOrientation = PageOrientation.portrait;
//At this point, we can reset the application default margins to their original state.
with (app.marginPreferences){
    top = myY1;
    left = myX1;
    bottom = myY2;
    right = myX2;
}
//Set up the bleed and slug areas.
with (myDocument.documentPreferences) {
    //Bleed
    documentBleedBottomOffset = "3p";
    documentBleedTopOffset = "3p";
    documentBleedInsideOrLeftOffset = "3p";
    documentBleedOutsideOrRightOffset = "3p";
    //Slug
    slugBottomOffset = "18p";
    slugTopOffset = "3p";
    slugInsideOrLeftOffset = "3p";
    slugRightOrOutsideOffset = "3p";
}
//Create a color.
try{
    myDocument.colors.item("PageNumberRed").name;
}
catch (myError) {
```

```

myDocument.colors.add({name:"PageNumberRed", colorModel:ColorModel.cmyk, colorValue:[20, 100,
80, 10]});
}
//Next, set up some default styles.
//Create up a character style for the page numbers.
try{
    myDocument.characterStyles.item("page_number").name;
}
catch (myError){
    myDocument.characterStyles.add({name:"page_number"});
}
myDocument.characterStyles.item("page_number").fillColor = myDocument.colors.
item("PageNumberRed");
//Create up a pair of paragraph styles for the page footer text.
//These styles have only basic formatting.
try{
    myDocument.paragraphStyles.item("footer_left").name;
}
catch (myError){
    myDocument.paragraphStyles.add({name:"footer_left", pointSize:11, leading:14});
}
//Create up a pair of paragraph styles for the page footer text.
try{
    myDocument.paragraphStyles.item("footer_right").name;
}
catch (myError){
    myDocument.paragraphStyles.add({name:"footer_right", basedOn:myDocument.paragraphStyles.
item("footer_left"), justification:Justification.rightAlign, pointSize:11, leading:14});
}
//Create a layer for guides.
try{
    myDocument.layers.item("GuideLayer").name;
}
catch (myError){
    myDocument.layers.add({name:"GuideLayer"});
}
//Create a layer for the footer items.
try{
    myDocument.layers.item("Footer").name;
}
catch (myError){
    myDocument.layers.add({name:"Footer"});
}
//Create a layer for the slug items.
try{
    myDocument.layers.item("Slug").name;
}
catch (myError){
    myDocument.layers.add({name:"Slug"});
}
//Create a layer for the body text.
try{
    myDocument.layers.item("BodyText").name;
}

```

22 Working with Documents

```

catch (myError) {
    myDocument.layers.add({name:"BodyText"});
}
with(myDocument.viewPreferences){
    rulerOrigin = RulerOrigin.pageOrigin;
    horizontalMeasurementUnits = MeasurementUnits.points;
    verticalMeasurementUnits = MeasurementUnits.points;
}
//Document baseline grid and document grid
with(myDocument.gridPreferences){
    baselineStart = 56;
    baselineDivision = 14;
    baselineShown = false;
    horizontalGridlineDivision = 14;
    horizontalGridSubdivision = 5;
    verticalGridlineDivision = 14;
    verticalGridSubdivision = 5;
    documentGridShown = false;
}

//Document XMP information.
with (myDocument.metadataPreferences){
    author = "Olav Martin Kvern";
    copyrightInfoURL = "http://www.adobe.com";
    copyrightNotice = "This document is not copyrighted.";
    copyrightStatus = CopyrightStatus.no;
    description = "Example 7 x 9 book layout";
    documentTitle = "Example";
    jobName = "7 x 9 book layout template";
    keywords = ["7 x 9", "book", "template"];
    createContainerItem("http://ns.adobe.com/xap/1.0/", "email");
    setProperty("http://ns.adobe.com/xap/1.0/", "email/*[1]", "okvern@adobe.com");
}
//Set up the master spread.
with(myDocument.masterSpreads.item(0)){
    with(pages.item(0)){
        //Left and right are reversed for left-hand pages (becoming "inside" and "outside"-- 
        //this is also true in the InDesign user interface).
        var myBottomMargin = myDocument.documentPreferences.pageHeight - marginPreferences.bottom;
        var myRightMargin = myDocument.documentPreferences.pageWidth - marginPreferences.left;
        guides.add(myDocument.layers.item("GuideLayer"), {orientation:HorizontalOrVertical.
vertical,location:marginPreferences.right});
        guides.add(myDocument.layers.item("GuideLayer"), {orientation:HorizontalOrVertical.vertical,
location:myRightMargin});
        guides.add(myDocument.layers.item("GuideLayer"), {orientation:HorizontalOrVertical.
horizontal, location:marginPreferences.top, fitToPage:false});
        guides.add(myDocument.layers.item("GuideLayer"), {orientation:HorizontalOrVertical.
horizontal, location:myBottomMargin, fitToPage:false});
        guides.add(myDocument.layers.item("GuideLayer"), {orientation:HorizontalOrVertical.
horizontal, location:myBottomMargin + 14, fitToPage:false});
        guides.add(myDocument.layers.item("GuideLayer"), {orientation:HorizontalOrVertical.
horizontal, location:myBottomMargin + 28, fitToPage:false});
        var myLeftFooter = textFrames.add(myDocument.layers.item("Footer"), undefined, undefined,
{geometricBounds:[myBottomMargin+14, marginPreferences.right, myBottomMargin+28, myRightMargin]});
    }
}

```

```

myLeftFooter.parentStory.insertionPoints.item(0).contents = SpecialCharacters.sectionMarker;
myLeftFooter.parentStory.insertionPoints.item(0).contents = SpecialCharacters.emSpace;
myLeftFooter.parentStory.insertionPoints.item(0).contents = SpecialCharacters.
autoPageNumber;
myLeftFooter.parentStory.characters.item(0).appliedCharacterStyle = myDocument.
characterStyles.item("page_number");
myLeftFooter.parentStory.paragraphs.item(0).applyStyle(myDocument.paragraphStyles.
item("footer_left", false));
//Slug information.
with(myDocument.metadataPreferences){
    var myString = "Author:\t" + author + "\tDescription:\t" + description + "\rCreation
Date:\t" + new Date +
    "\tEmail Contact\t" + getProperty("http://ns.adobe.com/xap/1.0/", "email/*[1]");
}
var myLeftSlug = textFrames.add(myDocument.layers.item("Slug"), undefined, undefined, {geom
etricBounds:[myDocument.documentPreferences.pageHeight+36, marginPreferences.right, myDocument.
documentPreferences.pageHeight + 144, myRightMargin], contents:myString});
myLeftSlug.parentStory.tables.add();
//Body text master text frame.
var myLeftFrame = textFrames.add(myDocument.layers.item("BodyText"), undefined,
undefined, {geometricBounds:[marginPreferences.top, marginPreferences.right, myBottomMargin,
myRightMargin]} );
}
with(pages.item(1)){
    var myBottomMargin = myDocument.documentPreferences.pageHeight - marginPreferences.bottom;
    var myRightMargin = myDocument.documentPreferences.pageWidth - marginPreferences.right;
    guides.add(myDocument.layers.item("GuideLayer"), {orientation:HorizontalOrVertical.
vertical, location:marginPreferences.left});
    guides.add(myDocument.layers.item("GuideLayer"), {orientation:HorizontalOrVertical.vertical,
location:myRightMargin});
    var myRightFooter = textFrames.add(myDocument.layers.item("Footer"), undefined, undefined,
{geometricBounds:[myBottomMargin+14, marginPreferences.left, myBottomMargin+28, myRightMargin]});
    myRightFooter.parentStory.insertionPoints.item(0).contents = SpecialCharacters.
autoPageNumber;
    myRightFooter.parentStory.insertionPoints.item(0).contents = SpecialCharacters.emSpace;
    myRightFooter.parentStory.insertionPoints.item(0).contents = SpecialCharacters.
sectionMarker;
    myRightFooter.parentStory.characters.item(-1).appliedCharacterStyle = myDocument.
characterStyles.item("page_number");
    myRightFooter.parentStory.paragraphs.item(0).applyStyle(myDocument.paragraphStyles.
item("footer_right", false));
    //Slug information.
    var myRightSlug = textFrames.add(myDocument.layers.item("Slug"), undefined, undefined, {geo
metricBounds:[myDocument.documentPreferences.pageHeight+36, marginPreferences.left, myDocument.
documentPreferences.pageHeight + 144, myRightMargin], contents:myString});
    myRightSlug.parentStory.tables.add();
    //Body text master text frame.
    var myRightFrame = textFrames.add(myDocument.layers.item("BodyText"), undefined, undefined,
{geometricBounds:[marginPreferences.top, marginPreferences.left, myBottomMargin, myRightMargin],
previousTextFrame:myLeftFrame});
}
}
//Add section marker text--this text will appear in the footer.
myDocument.sections.item(0).marker = "Section 1";

```

24 Working with Documents

```
//When you link the master page text frames, one of the frames  
//sometimes becomes selected. Deselect it.  
app.select(NothingEnum.nothing, undefined);
```

Printing a Document

The following script shows how to print the active document using the current print preferences.

```
//PrintDocument.js  
//Prints the active document.  
app.activeDocument.print();
```

To specify a page range to print, set the `pageRange` property of the document's `printPreferences` object before printing:

```
//PrintPageRange.js  
//Prints a page range from the active document.  
//Assumes that you have a document open, that it contains a page named "22".  
//The page range can be either PageRange.allPages or a page range string.  
//A page number entered in the page range must correspond to a page  
//name in the document (i.e., not the page index). If the page name is  
//not found, InDesign will display an error message.  
app.activeDocument.printPreferences.pageRange = "22"  
app.activeDocument.print(false);
```

The `printPreferences` object contains properties corresponding to the options you see in the panels of the Print dialog box. Here's an example script that shows how to set print preferences.

```
//PrintPreferences.js  
//Sets the print preferences of the active document.  
with(app.activeDocument.printPreferences){  
    //Properties corresponding to the controls in the General panel of the Print dialog box.  
    //activePrinterPreset is ignored in this example--we'll set our own print preferences.  
    //printer can be either a string (the name of the printer) or  
    //Printer.postscriptFile.  
    printer = "AGFA-SelectSet 5000SF v2013.108";  
    //If the printer property is the name of a printer, then the ppd property  
    //is locked (and will return an error if you try to set it).  
    //ppd = "AGFA-SelectSet5000SF";  
    //If the printer property is set to Printer.postscript file, the copies  
    //property is unavailable. Attempting to set it will generate an error.  
    copies = 1;  
    //If the printer property is set to Printer.postscript file, or if the  
    //selected printer does not support collation, then the collating  
    //property is unavailable. Attempting to set it will generate an error.  
    //collating = false;  
    reverseOrder = false;  
    //pageRange can be either PageRange.allPages or a page range string.  
    pageRange = PageRange.allPages;  
    printSpreads = false;  
    printMasterPages = false;
```

```

//If the printer property is set to Printer.postScript file, then
//the printFile property contains the file path to the output file.
//printFile = "/c/test.ps";
sequence = Sequences.all;
//If trapping is on, setting the following properties will produce an error.
if(trapping == Trapping.off) {
    printBlankPages = false;
    printGuidesGrids = false;
    printNonprinting = false;
}
//-----
//Properties corresponding to the controls in the Setup panel of the Print dialog box.
//-----
paperSize = PaperSizes.custom;
//Page width and height are ignored if paperSize is not PaperSizes.custom.
//paperHeight = 1200;
//paperWidth = 1200;
printPageOrientation = PrintPageOrientation.portrait;
pagePosition = PagePositions.centered;
paperGap = 0;
paperOffset = 0;
paperTransverse = false;
scaleHeight = 100;
scaleWidth = 100;
scaleMode = ScaleModes.scaleWidthHeight;
scaleProportional = true;
//If trapping is on, attempting to set the following properties will produce an error.
if(trapping == Trapping.off) {
    textAsBlack = false;
    thumbnails = false;
    //The following properties is not needed because thumbnails is set to false.
    //thumbnailsPerPage = 4;
    tile = false;
    //The following properties are not needed because tile is set to false.
    //tilingOverlap = 12;
    //tilingType = TilingTypes.auto;
}

//-----
//Properties corresponding to the controls in the Marks and Bleed panel of the Print dialog
box.
//-----
//Set the following property to true to print all printer's marks.
//allPrinterMarks = true;
useDocumentBleedToPrint = false;
//If useDocumentBleedToPrint = false then setting any of the bleed properties
//will result in an error.
//Get the bleed amounts from the document's bleed and add a bit.
bleedBottom = app.activeDocument.documentPreferences.documentBleedBottomOffset+3;
bleedTop = app.activeDocument.documentPreferences.documentBleedTopOffset+3;
bleedInside = app.activeDocument.documentPreferences.documentBleedInsideOrLeftOffset+3;
bleedOutside = app.activeDocument.documentPreferences.documentBleedOutsideOrRightOffset+3;
//If any bleed area is greater than zero, then export the bleed marks.

```

26 Working with Documents

```

if(bleedBottom == 0 && bleedTop == 0 && bleedInside == 0 && bleedOutside == 0) {
    bleedMarks = true;
}
else{
    bleedMarks = false;
}
colorBars = true;
cropMarks = true;
includeSlugToPrint = false;
markLineWeight = MarkLineWeight.p125pt
markOffset = 6;
//markType = MarkTypes.default;
pageInformationMarks = true;
registrationMarks = true;

-----
//Properties corresponding to the controls in the Output panel of the Print dialog box.
-----
negative = true;
colorOutput = ColorOutputModes.separations;
//Note the lowercase "i" in "BuiltIn"
trapping = Trapping.applicationBuiltIn;
screening = "175 lpi/2400 dpi";
flip = Flip.none;
//If trapping is on, attempting to set the following properties will generate an error.
if(trapping == Trapping.off) {
    printBlack = true;
    printCyan = true;
    printMagenta = true;
    printYellow = true;
}
//Only change the ink angle and frequency when you want to override the
//screening set by the screening specified by the screening property.
//blackAngle = 45;
//blackFrequency = 175;
//cyanAngle = 15;
//cyanFrequency = 175;
//magentaAngle = 75;
//magentaFrequency = 175;
//yellowAngle = 0;
//yellowFrequency = 175;
//The following properties are not needed (because colorOutput is set to separations).
//compositeAngle = 45;
//compositeFrequency = 175;
//simulateOverprint = false;

-----
//Properties corresponding to the controls in the Graphics panel of the Print dialog box.
-----
sendImageData = ImageDataTypes.allImageData;
fontDownloading = FontDownloading.complete;
downloadPPDFOns = true;
try{
    dataFormat = DataFormat.binary;
}

```

```

        }
        catch(e){}
        try{
            postScriptLevel = PostScriptLevels.level3;
        }
        catch(e){}
    }

    //-----
    //Properties corresponding to the Color Management panel of the Print dialog box.
    //-----
    //If the useColorManagement property of app.colorSettings is false,
    //attempting to set the following properties will return an error.
    try{
        sourceSpace = SourceSpaces.useDocument;
        intent = RenderingIntent.useColorSettings;
        crd = ColorRenderingDictionary.useDocument;
        profile = Profile.postscriptCMS;
    }
    catch(e){}
}

//-----
//Properties corresponding to the controls in the Advanced panel of the Print dialog box.
//-----
opiImageReplacement = false;
omitBitmaps = false;
omitEPS = false;
omitPDF = false;
//The following line assumes that you have a flattener preset named "high quality flattener".
try{
    flattenerPresetName = "high quality flattener";
}
catch(e){}
ignoreSpreadOverrides = false;
}

```

To print a document using a printer preset, include the printer preset in the print method.

```

//PrintDocumentWithPreset.js
//Prints the active document using the specified printer preset.
//Assumes you have a printer preset named "myPreset" and that a document is open.
app.activeDocument.print(false, app.printerPresets.item("myPreset"));

```

To create a document preset from the print preferences of a document, use the following script.

```

//CreatePrinterPreset.js
//Creates a new printer preset.
//If the preset does not already exist, then create it;
//otherwise, fill in the properties of the existing preset.
var myPreset;
myPreset = app.printerPresets.item("myPreset");
try{
    myPreset.name;
}
catch(myError) {

```

28 Working with Documents

```

myPreset = app.printerPresets.add({name:"myPreset"});
}

with(app.activeDocument.printPreferences){
    //Because many printing properties are dependent on other printing properties,
    //we've surrounded each property-setting line with try...catch statements--
    //these will make it easier for you to experiment with print preset settings.
    try{
        myPreset.printer = printer;
    }
    catch(e){}
    try{
        myPreset.ppd = ppd;
    }
    catch(e){}
    try{
        myPreset.copies = copies;
    }
    catch(e){}
    try{
        myPreset.collating = collating;
    }
    catch(e){}
    try{
        myPreset.reverseOrder = reverseOrder;
    }
    catch(e){}
    try{
        myPreset.pageRange = pageRange;
    }
    catch(e){}
    try{
        myPreset.printSpreads = printSpreads;
    }
    catch(e){}
    try{
        myPreset.printMasterPages = printMasterPages;
    }
    catch(e){}
    try{
        myPreset.printFile = printFile;
    }
    catch(e){}
    try{
        myPreset.sequence = sequence;
    }
    catch(e){}
    try{
        myPreset.printBlankPages = printBlankPages;
    }
    catch(e){}
    try{
        myPreset.printGuidesGrids = printGuidesGrids;
    }
    catch(e){}
}

```

```

try{
    myPreset.printNonprinting = printNonprinting;
}
catch(e){}
try{
    myPreset.paperSize = paperSize;
}
catch(e){}
try{
    myPreset.paperHeight = paperHeight;
}
catch(e){}
try{
    myPreset.paperWidth = paperWidth;
}
catch(e){}
try{
    myPreset.printPageOrientation = printPageOrientation;
}
catch(e){}
try{
    myPreset.pagePosition = pagePosition;
}
catch(e){}
try{
    myPreset.paperGap = paperGap;
}
catch(e){}
try{
    myPreset.paperOffset = paperOffset;
}
catch(e){}
try{
    myPreset.paperTransverse = paperTransverse;
}
catch(e){}
try{
    myPreset.scaleHeight = scaleHeight;
}
catch(e){}
try{
    myPreset.scaleWidth = scaleWidth;
}
catch(e){}
try{
    myPreset.scaleMode = scaleMode;
}

}
catch(e){}
try{
    myPreset.scaleProportional = scaleProportional;
}
catch(e){}
try{

```

```

        myPreset.textAsBlack = textAsBlack;
    }
    catch(e){}
    try{
        myPreset-thumbnails = thumbnails;
    }
    catch(e){}
    try{
        myPreset-thumbnailsPerPage = thumbnailsPerPage;
    }
    catch(e){}
    try{
        myPreset.tile = tile;
    }
    catch(e){}
    try{
        myPreset.tilingType = tilingType;
    }
    catch(e){}
    try{
        myPreset.tilingOverlap = tilingOverlap;
    }
    catch(e){}
    try{
        myPreset-allPrinterMarks = allPrinterMarks;
    }
    catch(e){}
    try{
        myPreset-useDocumentBleedToPrint = useDocumentBleedToPrint;
    }
    catch(e){}
    try{
        myPreset-bleedBottom = bleedBottom;
    }
    catch(e){}
    try{
        myPreset-bleedTop = bleedTop;
    }
    catch(e){}
    try{
        myPreset-bleedInside = bleedInside;
    }
    catch(e){}
    try{
        myPreset-bleedOutside = bleedOutside;
    }
    catch(e){}
    try{
        myPreset-bleedMarks = bleedMarks;
    }
    catch(e){}
    try{
        myPreset-colorBars = colorBars;
    }
}

```

```

        catch(e){}
        try{
            myPreset.cropMarks = cropMarks;
        }
        catch(e){}
        try{
            myPreset.includeSlugToPrint = includeSlugToPrint;
        }
        catch(e){}
        try{
            myPreset.markLineWeight = markLineWeight;
        }
        catch(e){}
        try{
            myPreset.markOffset = markOffset;
        }
        catch(e){}
        try{
            myPreset.markType = markType;
        }
        catch(e){}
        try{
            myPreset.pageInformationMarks = pageInformationMarks;
        }
        catch(e){}
        try{
            myPreset.registrationMarks = registrationMarks;
        }
        catch(e){}
        try{
            myPreset.negative = negative;
        }
        catch(e){}
        try{
            myPreset.colorOutput = colorOutput ;
        }
        catch(e){}
        try{
            myPreset.trapping = trapping;
        }
        catch(e){}
        try{
            myPreset.screening = screening;
        }
        catch(e){}
        try{
            myPreset.flip = flip;
        }
        catch(e){}
        try{
            myPreset.printBlack = printBlack;
        }
        catch(e){}
        try{

```

```

    myPreset.printCyan = printCyan;
}
catch(e){}
try{
    myPreset.printMagenta = printMagenta;
}
catch(e){}
try{
    myPreset.printYellow = printYellow;
}
catch(e){}
try{
    myPreset.blackAngle = blackAngle;
}
catch(e){}
try{
    myPreset.blackFrequency = blackFrequency;
}
catch(e){}
try{
    myPreset.cyanAngle = cyanAngle;
}
catch(e){}
try{
    myPreset.cyanFrequency = cyanFrequency;
}
catch(e){}
try{
    myPreset.magentaAngle = magentaAngle;
}
catch(e){}
try{
    myPreset.magentaFrequency = magentaFrequency;
}
catch(e){}
try{
    myPreset.yellowAngle = yellowAngle;
}
catch(e){}
try{
    myPreset.yellowFrequency = yellowFrequency;
}
catch(e){}
try{
    myPreset.compositeAngle = compositeAngle;
}
catch(e){}
try{
    myPreset.compositeFrequency = compositeFrequency;
}
catch(e){}
try{
    myPreset.simulateOverprint = simulateOverprint;
}

```

```

        catch(e){}
        try{
            myPreset.sendImageData = sendImageData;
        }
        catch(e){}
        try{
            myPreset.fontDownloading = fontDownloading;
        }
        catch(e){}
        try{
            myPreset.downloadPPDFOns = downloadPPDFOns;
        }
        catch(e){}
        try{
            myPreset.dataFormat = dataFormat;
        }
        catch(e){}
        try{
            myPreset.postScriptLevel = postscriptLevel;
        }
        catch(e){}
        try{
            myPreset.sourceSpace = sourceSpace;
        }
        catch(e){}
        try{
            myPreset.intent = intent;
        }
        catch(e){}
        try{
            myPreset.crd = crd;
        }
        catch(e){}
        try{
            myPreset.profile = profile;
        }
        catch(e){}
        try{
            myPreset.opiImageReplacement = opiImageReplacement;
        }
        catch(e){}
        try{
            myPreset.omitBitmaps = omitBitmaps;
        }
        catch(e){}
        try{
            myPreset.omitEPS = omitEPS;
        }
        catch(e){}
        try{
            myPreset.omitPDF = omitPDF;
        }
        catch(e){}
        try{

```

```

        myPreset.flattenerPresetName = flattenerPresetName ;
    }
    catch(e){}
    try{
        myPreset.ignoreSpreadOverrides = ignoreSpreadOverrides;
    }
    catch(e){}
    alert("Done!");
}

```

Exporting a Document as PDF

InDesign scripting offers full control over the creation of PDF files from your page layout documents. The following scripts shows how to export the current document as PDF.

```

//ExportPDF.js
//Assumes you have a document open and that you have defined a PDF export preset
//named "prepress".
//document.export parameters are:
//Format as (use either the ExportFormat.pdfType constant or the string "Adobe PDF")
//To as string (for JavaScript, use a URI rather than a platform-specific file path)
//ShowingOptions as boolean (setting this option to true displays the PDF Export dialog box)
//Using as PDF export preset (or a string that is the name of a PDF export preset)
app.activeDocument.exportFile(ExportFormat.pdfType, "/c/myTestDocument.pdf", false, app.
PDFExportPresets.item("prepress"));

```

The above example exports the file as a PDF using the current PDF export options. The following examples shows how to set the PDF export options before exporting.

```

//ExportEachPageAsPDF.js
//Exports each page of an InDesign CS document as a separate PDF to a selected folder using the
current PDF export settings.
//Display a "choose folder" dialog box.
if(app.documents.length != 0{
    var myFolder = Folder.selectDialog ("Choose a Folder");
    if(myFolder != null){
        myExportPages(myFolder);
    }
}
else{
    alert("Please open a document and try again.");
}
function myExportPages(myFolder){
    var myDocument = app.activeDocument;
    var myDocumentName = myDocument.name;
    var myDialog = app.dialogs.add();
    with(myDialog.dialogColumns.add().dialogRows.add()){
        staticTexts.add({staticLabel:"Base name:"});
        var myBaseNameField = textEditboxes.add({editContents:myDocumentName, minWidth:160});
    }
    var myResult = myDialog.show({name:"ExportPages"});
}

```

```

if(myResult == true) {
    //The name of the exported files will be the base name + the value of the counter + ".pdf".
    var myBaseName = myBaseNameField.editContents;
    //Remove the dialog box from memory.
    myDialog.destroy();
    for(var myCounter = 0; myCounter < myDocument.pages.length; myCounter++){
        app.pdfExportPreferences.pageRange = (myCounter+1) + "";
        //Generate a file path from the folder name, the base document name, and the counter
        value.
        var myFilePath = myFolder + "/" + myBaseName + (myCounter+1) + ".pdf";
        var myFile = new File(myFilePath);
        app.activeDocument.exportFile(ExportFormat.pdfType, myFile, false);
    }
}
else{
    myDialog.destroy();
}
}

```

The next example shows an example set of PDF export preferences (this set is not complete—no security features are changed).

```

//ExportPDFWithOptions.js
//Sets PDF export options, then exports the active document as PDF.
with(app.pdfExportPreferences) {
    //Basic PDF output options.
    pageRange = PageRange.allPages;
    acrobatCompatibility = AcrobatCompatibility.acrobat6;
    exportGuidesAndGrids = false;
    exportLayers = false;
    exportNonPrintingObjects = false;
    exportReaderSpreads = false;
    generateThumbnails = false;
    ignoreSpreadOverrides = false;
    includeBookmarks = true;
    includeHyperlinks = true;
    includeICCPProfiles = true;
    includeSlugWithPDF = false;
    includeStructure = false;
    interactiveElements = false;
    //Setting subsetFontsBelow to zero disallows font subsetting;
    //set subsetFontsBelow to some other value to use font subsetting.
    subsetFontsBelow = 0;
    //Bitmap compression/sampling/quality options.
    colorBitmapCompression = BitmapCompression.zip;
    colorBitmapQuality = CompressionQuality.eightBit;
    colorBitmapSampling = Sampling.none;
    //thresholdToCompressColor is not needed in this example.
    //colorBitmapSamplingDPI is not needed when colorBitmapSampling is set to none.
    grayscaleBitmapCompression = BitmapCompression.zip;
    grayscaleBitmapQuality = CompressionQuality.eightBit;
    grayscaleBitmapSampling = Sampling.none;
    //thresholdToCompressGray is not needed in this example.
    //grayscaleBitmapSamplingDPI is not needed when grayscaleBitmapSampling is set to none.
}

```

36 Working with Documents

```

monochromeBitmapCompression = BitmapCompression.zip;
monochromeBitmapSampling = Sampling.none;
//thresholdToCompressMonochrome is not needed in this example.
//monochromeBitmapSamplingDPI is not needed when monochromeBitmapSampling is set to none.
//Other compression options.
compressionType = PDFCompressionType.compressNone;
compressTextAndLineArt = true;
contentToEmbed = embedAll;
cropImagesToFrames = true;
optimizePDF = true;
//Printers marks and prepress options.
//Get the bleed amounts from the document's bleed.
bleedBottom = app.activeDocument.documentPreferences.documentBleedBottomOffset;
bleedTop = app.activeDocument.documentPreferences.documentBleedTopOffset;
bleedInside = app.activeDocument.documentPreferences.documentBleedInsideOrLeftOffset;
bleedOutside = app.activeDocument.documentPreferences.documentBleedOutsideOrRightOffset;
//If any bleed area is greater than zero, then export the bleed marks.
if(bleedBottom == 0 && bleedTop == 0 && bleedInside == 0 && bleedOutside == 0){
    bleedMarks = true;
}
else{
    bleedMarks = false;
}
colorBars = true;
//The colorTileSize and grayTileSize properties are only applicable when
//the corresponding bitmap compression properties are set to BitmapCompression.jpeg2000.
//colorTileSize = 256;
//grayTileSize = 256;
cropMarks = true;
omitBitmaps = false;
omitEPS = false;
omitPDF = false;
pageInformationMarks = true;
pageMarksOffset = 12;
pdfColorSpace = PDFColorSpace.unchangedColorSpace;
pdfMarkType = MarkType.default;
printerMarkWeight = pdfMarkWeight.p125pt;
registrationMarks = true;
simulateOverprint = false;
useDocumentBleedWithPDF = true;
//Set viewPDF to true to open the PDF in Acrobat or Adobe Reader.
viewPDF = false;
}
//Now export the document. You'll have to fill in your own file path.
app.activeDocument.exportFile(ExportFormat.pdfType, "/c/myTestDocument.pdf", false);

```

The following example exports each page in a document as an individual PDF.

```
//ExportEachPageAsPDF.js
//Exports each page of an InDesign CS document as a separate PDF to
//a selected folder using the current PDF export settings.
//Display a "choose folder" dialog box.
if(app.documents.length != 0){
    var myFolder = Folder.selectDialog ("Choose a Folder");
    if(myFolder != null){
        myExportPages(myFolder);
    }
}
else{
    alert("Please open a document and try again.");
}

function myExportPages(myFolder){
    var myPageName, myFilePath, myFile;
    var myDocument = app.activeDocument;
    var myDocumentName = myDocument.name;
    var myDialog = app.dialogs.add();
    with(myDialog.dialogColumns.add().dialogRows.add()){
        staticTexts.add({staticLabel:"Base name:"});
        var myBaseNameField = textEditboxes.add({editContents:myDocumentName, minWidth:160});
    }
    var myResult = myDialog.show({name:"ExportPages"});
    if(myResult == true){
        var myBaseName = myBaseNameField.editContents;
        //Remove the dialog box from memory.
        myDialog.destroy();
        for(var myCounter = 0; myCounter < myDocument.pages.length; myCounter++){
            myPageName = myDocument.pages.item(myCounter).name;
            app.pdfExportPreferences.pageRange = myPageName;
            //The name of the exported files will be the base name + the page name + ".pdf".
            //If the page name contains a colon (as it will if the document contains sections),
            //then remove the colon.
            var myRegExp = new RegExp(":", "gi");
            myPageName = myPageName.replace(myRegExp, "_");
            myFilePath = myFolder + "/" + myBaseName + "_" + myPageName + ".pdf";
            myFile = new File(myFilePath);
            myDocument.exportFile(ExportFormat.pdfType, myFile, false);
        }
    }
    else{
        myDialog.destroy();
    }
}
```

Exporting Pages as EPS

The following script exports the pages of the active document to one or more EPS files. When you export a document as EPS, InDesign will save each page of the file as a separate EPS graphic (an EPS, by definition, can contain only a single page). If you're exporting more than a single page, InDesign will append the index of the page to the file name. The index of the page in the document is not necessarily the name of the page (as defined by the section options for the section containing the page).

```
//ExportAsEPS.js
//Exports the pages of the active document as EPS.
var myFile = new File("/c/myTestFile.eps");
app.activeDocument.exportFile(ExportFormat.epsType, myFile, false);
```

To control the pages exported as EPS, set the `pageRange` property of the EPS export preferences to a page range string containing the page or pages you want to export before exporting.

```
//ExportSelectedPagesAsEPS.js
//Enter the name of the page you want to export in the following line.
//Note that the page name is not necessarily the index of the page in the
//document (e.g., the first page of a document whose page numbering starts
//with page 21 will be "21", not 1).
app.epsExportPreferences.pageRange = "1-3, 6, 9";
var myFile = new File("/d/test/myFile.eps");
app.activeDocument.exportFile(ExportFormat.epsType, myFile, false);</pre>
```

In addition to the page range, you can control other EPS export options using scripting by setting the properties of the `epsExportPreferences` object.

```
//ExportEachPageAsEPS.js
//An InDesign CS JavaScript.
//Exports each page of an InDesign CS document as a separate EPS to a
//selected folder using the current EPS export settings.
//Display a "choose folder" dialog box.
if(app.documents.length != 0){
    var myFolder = Folder.selectDialog ("Choose a Folder");
    if(myFolder != null){
        myExportPages(myFolder);
    }
}
else{
    alert("Please open a document and try again.");
}

function myExportPages(myFolder){
    var myFilePath, myPageName, myFile;
    var myDocument = app.activeDocument;
    var myDocumentName = myDocument.name;
    var myDialog = app.dialogs.add({name:"ExportPages"});
    with(myDialog.dialogColumns.add().dialogRows.add()){
        staticTexts.add({staticLabel:"Base name:"});
        var myBaseNameField = textEditboxes.add({editContents:myDocumentName, minWidth:160});
    }
    var myResult = myDialog.show();
    if(myResult == true){
```

```

//The name of the exported files will be the base name + the page name + ".eps".
var myBaseName = myBaseNameField.editContents;
//Remove the dialog box from memory.
myDialog.destroy();
//Generate a file path from the folder name, the base document name, page name.
for(var myCounter = 0; myCounter < myDocument.pages.length; myCounter++){
    myPageName = myDocument.pages.item(myCounter).name;
    app.epsExportPreferences.pageRange = myPageName;
    //The name of the exported files will be the base name + the page name + ".eps".
    //If the page name contains a colon (as it will if the document contains sections),
    //then remove the colon.
    var myRegExp = new RegExp(":", "gi");
    myPageName = myPageName.replace(myRegExp, "_");
    myFilePath = myFolder + "/" + myBaseName + "_" + myPageName + ".eps";
    myFile = new File(myFilePath);
    app.activeDocument.exportFile(ExportFormat.epsType, myFile, false);
}
}
else{
    myDialog.destroy();
}
}

```