ADOBE® INDESIGN® CS4 SERVER

ADOBE INDESIGN CS4 SERVER SCRIPTING GUIDE

Adobe

Adobe® InDesign® CS4 Server Scripting Guide

Contents

Adobe InDesign CS4 Server Scripting Guide

Introduction

Adobe® InDesign® CS4 Server is a powerful addition to your Adobe InDesign CS4 tool set. In addition to the high quality typesetting, page-layout, color, and output features of InDesign, InDesign Server provides extra capabilities for enterprise-wide document processing.

This document gives important background information for creating scripts for InDesign Server. It provides simple examples of common scripting operations.

NOTE: We strongly recommend you develop InDesign Server scripts using InDesign CS4.

Intended audience

This document is for software developers who want to extend the capabilities of Adobe InDesign CS4 by writing plug-ins or by writing applications that integrate InDesign with a publication workflow. This document assumes that:

- You are a programmer with a working understanding of one of the supported scripting languages: AppleScript, JavaScript, or Visual Basic.
- You have a basic understanding of the InDesign scripting object model.
- You have a basic understanding of TCP/IP-based, client-server environments.
- You are familiar with SOAP and know how to use it.
- You are familiar with your system's and your network server's port configurations and know how to configure them.

  You or your team are experienced at building and supporting customer solutions.

For more information

Introduction to Adobe InDesign CS4 Server provides basic information about running InDesign Server, including installation instructions and system requirements. It includes a simple demonstration of InDesign Server's capabilities.

Adobe InDesign CS4 Scripting Guide provides details on creating scripts for InDesign CS4.

For details about AppleScript, VBScript, or JavaScript, see the documentation for those languages.

InDesign online scripting resources

For more information on InDesign Server scripting, see the following Web sites:

http://partners.adobe.com/public/developer/scripting/index.html

http://www.adobeforums.com — The InDesign scripting user-to-user forum. In the forum, scripters can ask questions, post answers, and share their newest scripts. The forum contains hundreds of sample scripts.

http://www.adobe.com/products/indesign/scripting/index.html

What you need to script InDesign Server

The language you use to write scripts depends on the scripting system of your platform: AppleScript for Mac OS®, VBScript for Windows®, or JavaScript for either platform. Although the scripting languages differ, the ways in which they work with InDesign are very similar.

Each sample script in this document is shown in all three scripting languages. Translating a script from one language to another is fairly easy.

JavaScript InDesign supports JavaScript for cross-platform scripting in both Mac OS and Windows. InDesign's JavaScript support is based on an Adobe implementation of JavaScript known as ExtendScript. The ExtendScript interpreter conforms to the current, ECMA 262 standard for JavaScript. All language features of JavaScript 1.5 are supported. Adobe Illustrator®, Adobe Photoshop®, and other Adobe Creative Suite® products also use the ExtendScript JavaScript interpreter.

While you can write scripts using other versions of JavaScript, like Microsoft® JScript (on Windows) or Late Night Software's OSA JavaScript (on Mac OS), the terms you use in those languages are not the same as the terms you use in ExtendScript. ExtendScript examples do not work in other JavaScript versions.

NOTE: Because ExtendScript tools and features are used in several Adobe products, we consolidated all ExtendScript documentation. To learn more about JavaScript utilities like the ScriptUI user-interface module and the ExtendScript Toolkit (a JavaScript development environment and object-model inspector), see *JavaScript Tools Guide*.

Windows To use InDesign scripting in Windows, you can use JavaScript or some version of Microsoft Visual Basic, like VBScript.

The Visual Basic tutorial scripts in this document are written in VBScript. We chose VBScript because no added software is required to run or edit VBScripts.

Other versions of Visual Basic include Visual Basic 5 Control Creation Edition (CCE), Visual Basic 6, Visual Basic .NET, and Visual Basic 2005 Express Edition. Versions of Visual Basic before Visual Basic .NET work well with InDesign scripting. Visual Basic .NET and newer versions do not work as well, because they lack the `Variant` data type, which is used extensively in InDesign scripting.

Many applications contain Visual Basic for Applications (VBA), such as Microsoft Word, Microsoft Excel, Microsoft Visio, or AutoCAD. Although you can use VBA to create InDesign scripts, InDesign does not include VBA.

To use VBScript or Visual Basic for InDesign scripting in Windows XP, you must install InDesign from a user account that has Administrator privileges. After you complete the installation, any user can run InDesign scripts, and any user can add scripts to the InDesign Scripts panel.

Mac OS

To use InDesign scripting on Mac OS, you can use AppleScript or JavaScript. To write AppleScripts, you must have AppleScript version 1.6 or higher and an AppleScript script editor. AppleScript comes with all Apple® systems, and it can be downloaded gratis from the Apple Web site. The Apple Script Editor is included with the Mac OS. Third-party script editors also are available; for example, Script Debugger (from Late Night Software, http://www.latenightsw.com).

Using the Tutorial Scripts

All tutorial scripts shown in this document are available on your InDesign installation discs and can be downloaded from http://www.adobe.com/products/indesign/xml_scripting.html.

The files are stored in a zip archive, `InDesignServerCS4TutorialScripts.zip`. When you uncompress the archive, you can move the folder containing the scripts written in the scripting language you want to use (AppleScript, JavaScript, or VBScript) to any location on your system. Working with the script files is much easier than entering the script yourself or copying and pasting from this document.

To use any script from this document, either open the tutorial script file (the filename is given before each script) or follow these steps:

1. Copy the script from this Adobe PDF document and paste it into your script editor, such as the Apple Script Editor (for AppleScript examples), ExtendScript Toolkit (for JavaScript examples), or a text editor like Notepad (for VBScript examples).

2. Save the script as a plain text file, using the appropriate file extension:

   AppleScript: `.applescript`
   JavaScript: `.jsx`
   VBScript: `.vbs`

3. Run the script from your script editor, or send the script to InDesign Server using the Test Client. If the script is an executable file (like a VBScript), run the script.

NOTE: If you are entering the JavaScript examples, it is very important to use the same capitalization shown in the example. JavaScript is case-sensitive, and the scripts will fail if they do not use the capitalization shown. The AppleScript and VBScript examples are not case-sensitive.

NOTE: If you are copying and pasting scripts from this document, be aware that line breaks caused by the layout of the document can cause errors in your script. As it can be very difficult to find such errors, we recommend that you use the scripts in the zip archive.

InDesign CS4 and InDesign Server CS4

In this document, we present scripts that show how to create simple documents using InDesign Server CS4. The bulk of the scripting documentation for InDesign Server, however, is in the documentation for the desktop (i.e., non-server) version of InDesign.

We assume you are developing your InDesign CS4 Server scripts using InDesign CS4, and you have read *Adobe InDesign CS4 Scripting Tutorial* and *Adobe InDesign CS4 Scripting Guide* for the scripting language you want to use. (Some of the material from *Adobe InDesign CS4 Scripting Tutorial* is shown here in abbreviated form and has been changed to work with InDesign Server.)

InDesign CS4 is an essential tool for developing scripts for InDesign Server. We assume you will develop scripts using InDesign CS4 before using them with InDesign Server CS4, because it is much easier to test and debug scripts when you can see the objects being created. To see the result of your script in InDesign Server, you would have to save the document and open it using InDesign CS4, or export the document and view the exported file in another program (like Acrobat® or Adobe Reader®). If you use InDesign CS4 to develop your InDesign CS4 Server scripts, you can reduce your development time dramatically.

We also assume you are familiar with the details of your InDesign Server installation. For the purposes of this document, we present sample scripts that work with a copy of InDesign Server on the same system as your scripting development environment.

Though the two programs are very similar, their scripting object models differ slightly. The following sections discuss objects, properties, and methods in InDesign CS4 that are not in InDesign Server CS4. Keep these in mind as you convert scripts from InDesign CS4 to InDesign Server CS4.

Active document

Many InDesign CS4 scripts refer to the front-most document in the user interface using the active document (AppleScript), activeDocument (JavaScript), or ActiveDocument (VBScript) property of the application object. This property does not exist in InDesign Server CS4. Instead, you can refer to documents by their index or name.

Active script

Many InDesign CS4 scripts refer to the currently running script to locate other script files or resources. InDesign Server does not have this property. Instead, locate the assets the script will need in specific locations on your server or system, then refer to those locations using complete file paths.

Windows

InDesign CS4 scripts often refer to the active window or the active spread of the active window. These user-interface properties are not supported by InDesign server.

Selection

As you would expect, InDesign server does not have an object corresponding to the user selection. When you convert scripts from InDesign CS4, you must remove any reference to the selection and provide references to objects based on other qualities (like the object id, index, or label).

Dialogs

InDesign CS4 can create modal dialog boxes using the dialog object and populate them with common user-interface controls, like check boxes, text-entry fields, and radio buttons. InDesign server does not support the dialog object or any user-interface controls.

Copy and paste

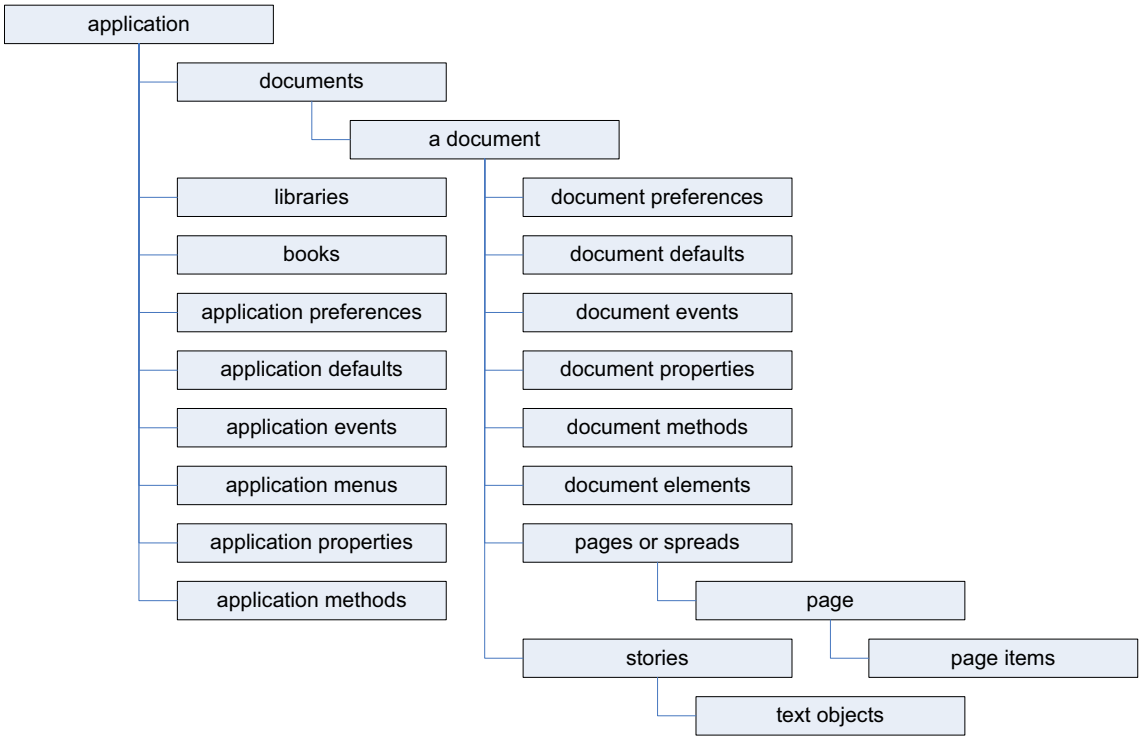
Because InDesign Server has no user interface, it also lacks the copy and paste features of InDesign CS4. Instead of using copy and paste, use duplicate and move. Both methods can create copies of objects, and can move objects from one document to another.

Understanding the InDesign Server object model

When you think about InDesign and InDesign Server documents, you probably organize each program and its components in your mind. You know paragraphs are contained by text frames which, in turn, appear on a page. A page is a part of a spread, and one or more spreads make up a document. Documents contain colors, styles, layers, and master spreads. As you think about the layouts you create, you intuitively understand there is an order to them.

InDesign Server "thinks" about the contents of a document in the same way. A document contains pages, which contain page items (text frames, rectangles, ellipses, and so on). Text frames contain characters, words, paragraphs, and anchored frames; graphics frames contain images, EPS files, or PDF files; and groups contain other page items. The things we mention here are the *objects* that make up an InDesign Server publication, and they are what we work with when we write InDesign Server scripts.

The following figure is an overview of the InDesign Server object model. The diagram is not a comprehensive list of the objects available to InDesign Server scripting; instead, it is a conceptual framework for understanding the relationships between the types of objects.



The objects in the diagram are explained in the following table:

Term	What it represents
Application	An instance of InDesign Server.
Application defaults	Application default settings, like colors, paragraph styles, and object styles. Application defaults affect all new documents for this instance of the application.
Application events	Events are generated by opening, closing, or saving a document or importing file. Scripts can be triggered by events.

Term	What it represents
Application methods	The actions the application can take; for example, finding and changing text, duplicating an object, and creating new documents.
Application preferences	Examples are text preferences, PDF-export preferences, and document preferences. Many preferences objects also exist at the document level. Application preferences are applied to new documents; document preferences change the settings of a specific document.
Application properties	The properties of the application; for example, the full path to the application, the locale of the application, and the user name.
Books	A collection of open books.
Document	An InDesign Server document.
Document defaults	Document default settings, like colors, paragraph styles, and text-formatting defaults.
Document elements	Examples are the stories, imported graphics, and pages of a document. The figure that precedes this table shows pages and stories, because those objects are very important containers for other objects, but document elements also include rectangles, ovals, groups, XML elements, and any other type of object you can import or create.
Document events	Events that occur at the document level, like importing a file. See "application events" in this table.
Document methods	The actions the document can take; for example, closing a document, printing a document, and exporting a document.
Document preferences	The preferences of a document, like guide preferences, view preferences, and document preferences.
Document properties	For example, the document filename, number of pages, and zero point location.
Documents	A collection of open documents.
Libraries	A collection of open libraries.
Page	A single page in an InDesign Server document.
Page items	Any object you can create or place on a page. There are many types of page items, such as text frames, rectangles, graphic lines, or groups.
Pages or spreads	The pages or spreads in an InDesign Server document.
Stories	The text in an InDesign Server document.
Text objects	Characters, words, lines, paragraphs, and text columns are examples of text objects in an InDesign story.

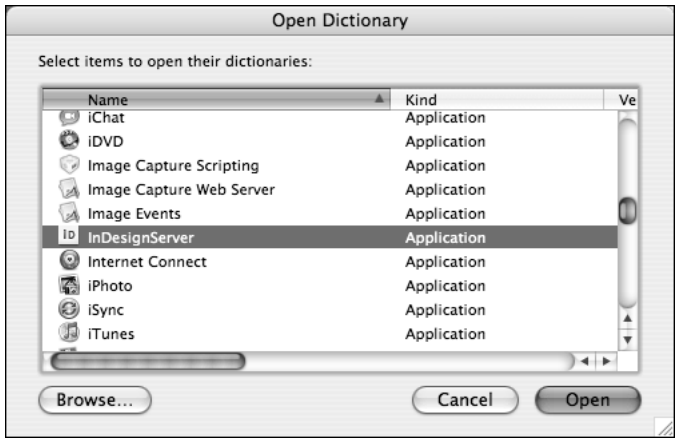
Looking at the InDesign Server object model

You can view the InDesign Server object model from inside your script-editing application. All reference information on objects and their properties and methods is stored in the model and can be viewed.

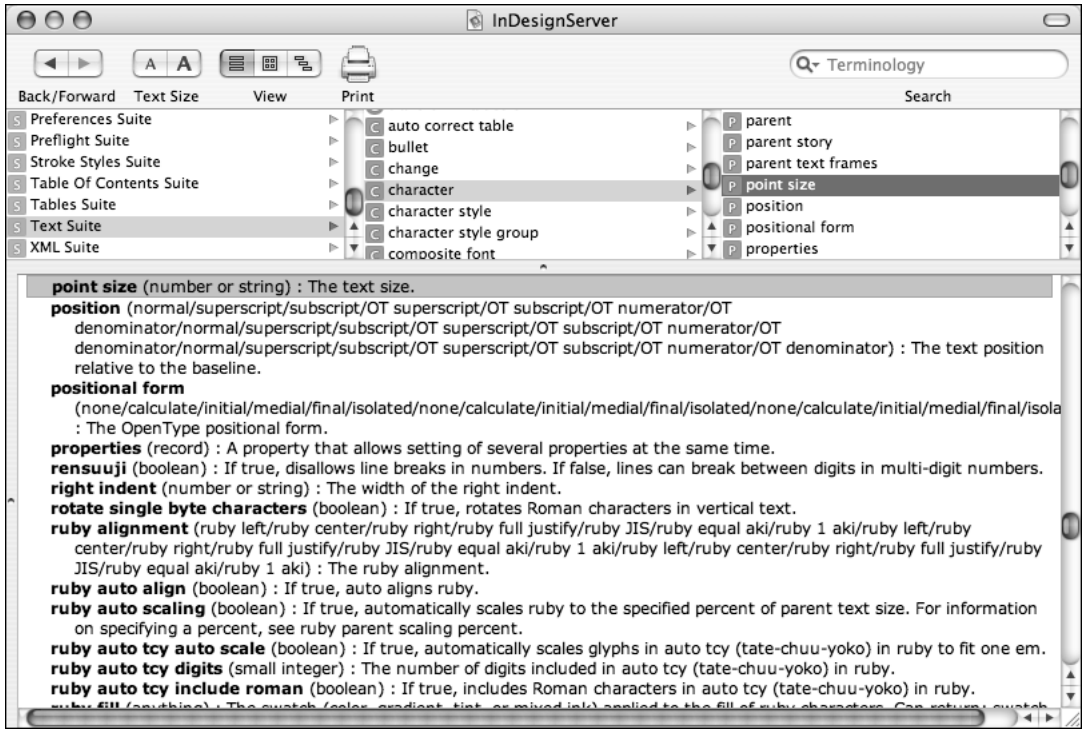
Apple-Script

To view the InDesign Server AppleScript dictionary, follow these steps:

1. Start InDesign Server.
2. Start the Apple Script Editor.
3. In the Script Editor, choose File > Open Dictionary. The Script Editor displays a list of scriptable applications:



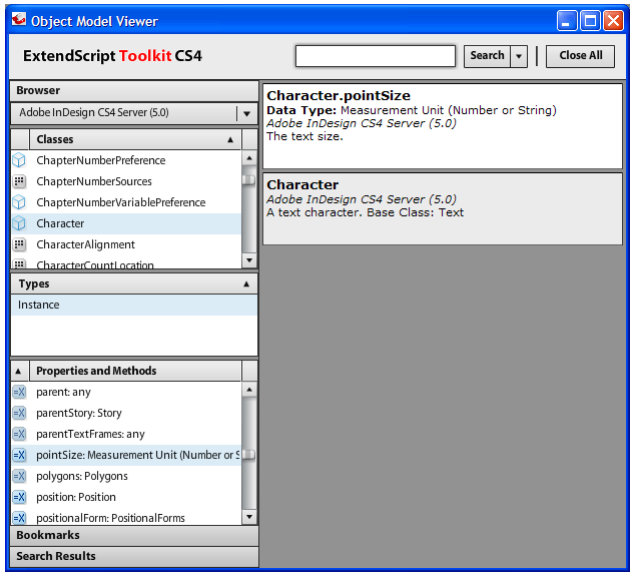
4. Select your copy of InDesign Server, and click OK. The Script Editor displays a list of the application's suites (collections of related objects):



5. Select a suite, to see the objects and methods (commands) it contains. Select an object, to see the properties associated with it.

JavaScript To view the InDesign Server object model in the ExtendScript Toolkit, follow these steps:

1. Start the ExtendScript Toolkit.
2. Choose Help > InDesign Server CS4 Main Dictionary. The ExtendScript Toolkit loads the InDesign Server object model and displays it in a separate window.
3. From the Classes list, select the object you want to view, then click the property or method you want to view in more detail in the Properties and Methods list. The ExtendScript toolkit displays more information on the property or method you selected:

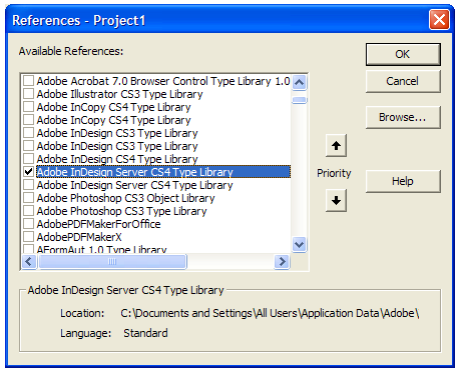


NOTE: For more on using the ExtendScript Toolkit object-model viewer, see *JavaScript Tools Guide*.

VBScript To view the InDesign Server object model, you need a VBScript editor/debugger, some version of Visual Basic, or an application that incorporates Visual Basic for Applications.

Visual Basic 6 To view the object model using Visual Basic 6, follow these steps:

1. Create a new Visual Basic project, then choose Project > References. Visual Basic displays the References dialog box:

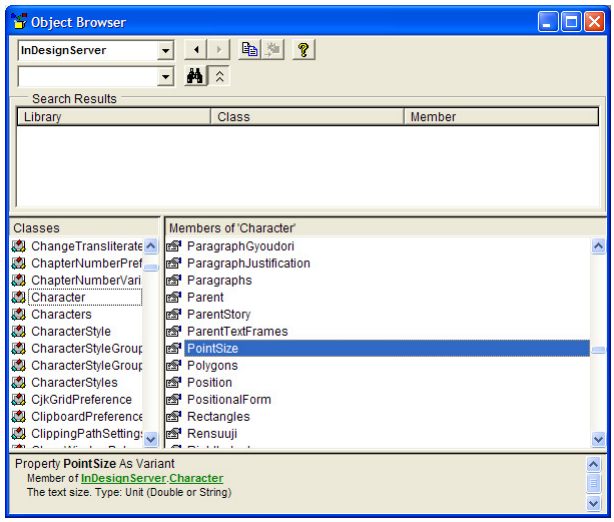


2. From the list of available references, select Adobe InDesign Server CS4 Type Library, and click OK. If the library does not appear in the list of available references, click Browse and locate and select the file Resources for Visual Basic.tlb, which usually is inside `~:\Documents and Settings\`*`user_name`*`\Application Data\Adobe\InDesign Server\Version 6.0\Scripting Support\` (where *`user_name`* is your user name). If necessary, search for the file. Once you locate the file, click Open to add the reference to your project.

3. Choose View > Object Browser. Visual Basic displays the Object Browser dialog box.

4. From the list of open libraries shown in the Project/Library menu, choose InDesign. Visual Basic displays the objects that make up the InDesign Server object model.

5. Click an object class. Visual Basic displays the properties and methods of the object. For more information on a property or method, select the item; Visual Basic displays the definition of the item at the bottom of the Object Browser window:



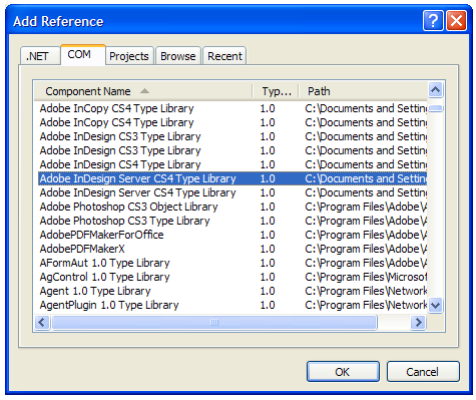
Visual Basic.NET

To view the object model using Visual Basic.NET, follow these steps:

1. Create a new Visual Basic project, then choose Project > Add Reference. Visual Basic displays the Add Reference dialog box.

2. Select the COM tab.

3. From the list of available references, select Adobe InDesign Server CS4 Type Library, and click Select. Visual Basic.NET adds the reference to the Selected Components list. If the library does not appear in the list of available references, click Browse and locate and select the `Resources for Visual .tlb` file, which usually is in `~:\Documents and Settings\`*`user_name`*`\Application Data\Adobe\InDesign Server\Version 6.0\Scripting Support\` (where *`user_name`* is your user name). Once you find the file, click Open to add the reference to your project:

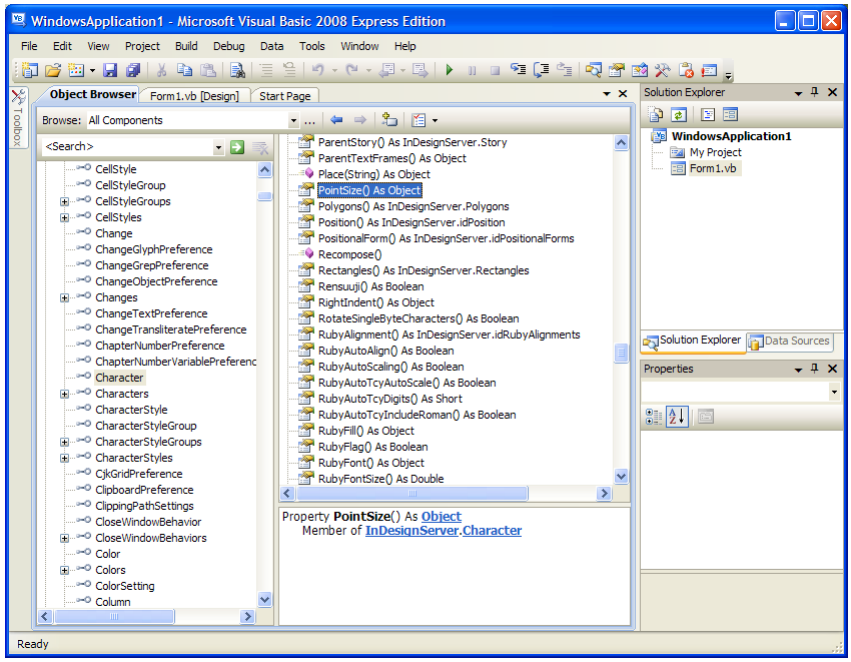


4. Click OK.

5. Choose View > Object Browser. Visual Basic displays the Object Browser tab.

6. From the list of open libraries in the Objects window, choose interop.indesign. Visual Basic.NET displays the objects that make up the InDesign Server object model.

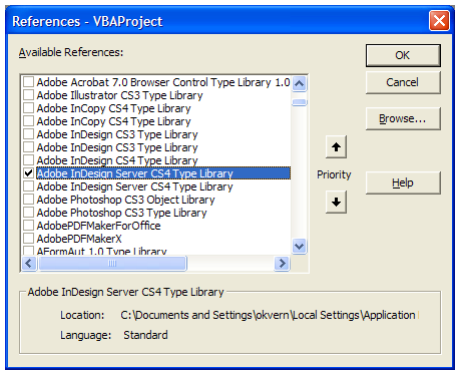
7. Click an object class. Visual Basic.NET displays the properties and methods of the object. For more information on a property or method, select the item; Visual Basic.NET displays the definition of the item at the bottom of the Object Browser window:



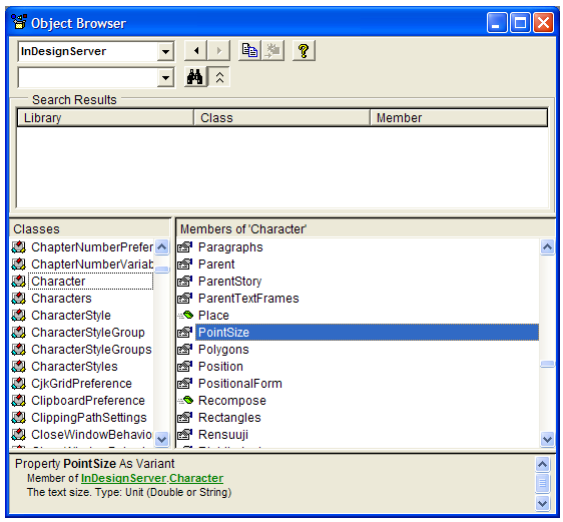
Visual Basic for Applications

To view the object model using Visual Basic for Applications from Microsoft Excel, follow these steps:

1. Start Excel.
2. Choose Tools > Macros > Visual Basic Editor. Excel displays the Visual Basic Editor window.
3. Choose Tools > References. The Visual Basic Editor displays the Add References dialog box:



4. From the list of available references, select Adobe InDesign Server CS4 Type Library option, and click Select. Visual Basic adds the reference to the Selected Components list. If the library does not appear in the list of available references, click Browse and locate and select the `Resources for Visual Basic.tlb` file, which usually is in `~:\Documents and Settings\`*`user_name`*`\Application Data\Adobe\InDesign Server\Version 6.0\Scripting Support\` (where *`user_name`* is your user name). Once you find the file, click OK to add the reference to your project.
5. Choose View > Object Browser. The Visual Basic editor display the Object Browser window.
6. From the Libraries pop-up menu, choose InDesign Server. The Visual Basic editor displays a list of the objects in the InDesign Server object library.
7. Click an object name. The Visual Basic Editor displays the properties and methods of the object. For more information on a property or method, select the item; Visual Basic displays the definition of the item at the bottom of the Object Browser window:



Measurements and positioning

All items and objects in InDesign Server are positioned on the page according to measurements you choose. It is useful to know how the InDesign coordinate system works and what measurement units it uses.

Coordinates

Like every other page-layout and drawing program, InDesign uses simple, two-dimensional geometry to set the position of objects on a page or spread. The horizontal component of a coordinate pair is referred to as *x*; the vertical component, *y*. You can see these coordinates in the Transform panel or Control panel (in InDesign CS4) when you select an object using the Selection tool. As in the InDesign user interface, coordinates are measured relative to the current location of the ruler's zero point.

There is one difference between the coordinates used in InDesign and the coordinate system used in a geometry textbook: on InDesign's vertical (or y) axis, coordinates *below* the zero point are positive numbers, and coordinates *above* the zero point are negative numbers.

NOTE: When you ask InDesign for the location of a path point, the coordinates are returned in *x, y* order. When you set the location of a path point, InDesign expects you to provide the coordinates in the same order. InDesign returns some coordinates in a different order, however, and it expects you to supply them in that order. Geometric bounds and visible bounds are arrays containing four coordinates, which define (in order) the top, left, bottom, and right edges of the object's bounding box (or `y1`, `x1`, `y2`, `x2`).

Working with measurement units

When you send measurement values to InDesign, you can send numbers (for example, 14.65) or measurement strings (for example, "1p7.1"). If you send numbers, InDesign uses the publication's current units of measurement. If you send measurement strings (see the table below), InDesign uses the units of measurement specified in the string.

InDesign returns coordinates and other measurement values using the publication's current measurement units. In some cases, these units do not resemble the measurement values shown in the InDesign Transform panel. For example, if the current measurement system is picas, InDesign returns fractional values as decimals, rather than using the picas-and-points notation used by the Transform panel. For example, "1p6" is returned as "1.5." InDesign does this because your scripting system would have trouble trying to perform arithmetic operations using measurement strings; for instance, trying to add "0p3.5" to "13p4" produces a script error, while adding .2916 to 13.333 (the converted pica measurements) does not.

If your script depends on adding, subtracting, multiplying, or dividing specific measurement values, you might want to set the corresponding measurement units at the beginning of the script. At the end of the script, you can set the measurement units back to whatever they were before you ran the script. Alternately, you can use measurement overrides, like many of the sample scripts. A measurement override is a string containing a special character, as shown in the following table:

Override	Meaning	Example
ag	Agates	3.5 ag
ap	American points	18 ap
c	Ciceros (add didots after the c, if necessary)	1.4 c

Override	Meaning	Example
cm	Centimeters	.635 cm
H	Ha	25.4 H
i (or in)	Inches	.25 i
mm	Millimeters	6.35 mm
p	Picas (add points after the p, if necessary)	1p6
pt	Points	18 pt

Your first InDesign Server script

The traditional first project in any programming language is to display, or print, the message "Hello World!" In this example, we create a new InDesign Server publication, add a frame containing this message, and save the document. At that point, you can open and view the document using InDesign CS4.

NOTE: These examples assume you are entering and running the script on the system on which InDesign Server is installed. For instructions on running scripts using the Test Client or SOAP commands, see *Introduction to Adobe InDesign Server CS4*.

AppleScript

To create an AppleScript script, follow these steps:

1. Locate and open the AppleScript Editor or any text editor.
2. Enter the following script. The lines preceded by double dashes (`--`) are comments and are ignored by the scripting system. They are included to describe the operation of the program. As you look through the script, you will see how we create, then address, each object in turn. The AppleScript command `tell` specifies which object will receive the next message we send.

```
--HelloWorld.as
tell application "InDesignServer"
    --Create a new document.
    set myDocument to make document
    --Get a reference to the first page.
    tell myDocument
        --Create a new text frame on the first page.
        tell page 1
            set myTextFrame to make text frame
            --Change the size of the text frame.
            set geometric bounds of myTextFrame to {"6p", "6p", "18p", "18p"}
            --Enter text in the text frame.
            set contents of myTextFrame to "Hello World!"
        end tell
        --Save the file (fill in a valid file path).
        save to "yukino:HelloWorld.indd"
    end tell
    --Close the document.
    close document 1
end tell
```

3. Save the script as a text file with the file extension `.applescript`.

4. To run the script from the Script Editor, click the Run button.
5. Open and view the document with InDesign.

JavaScript

To create a JavaScript, follow these steps:

1. Using the ExtendScript Toolkit, enter the following script:

```
//HelloWorld.jsx
//Create a new document.
var myDocument = app.documents.add();
//Get a reference to the first page.
var myPage = myDocument.pages.item(0);
//Create a text frame.
var myTextFrame = myPage.textFrames.add();
//Specify the size and shape of the text frame.
myTextFrame.geometricBounds = ["6p0", "6p0", "18p0", "18p0"];
//Enter text in the text frame.
myTextFrame.contents = "Hello World!";
//Save the document (fill in a valid file path).
myDocument.save(new File("/c/HelloWorld.indd"));
//Close the document.
app.documents.item(0).close();
```

2. Save the text as a plain text file with the file extension `.jsx`.
3. Choose InDesign Server from the target drop-down menu.
4. Click Run.
5. Open and view the document with InDesign.

VBScript

To create a VBScript, follow these steps:

1. Using any text editor (such as Notepad), enter the following script:

```
Rem HelloWorld.vbs
Set myInDesignServer = CreateObject("InDesignServer.Application")
Rem Create a new document.
Set myDocument = myInDesignServer.Documents.Add
Rem Get a reference to the first page.
Set myPage = myDocument.Pages.Item(1)
Rem Create a text frame.
Set myTextFrame = myPage.TextFrames.Add
Rem Specify the size and shape of the text frame.
myTextFrame.GeometricBounds = Array("6p0", "6p0", "18p0", "18p0")
Rem Enter text in the text frame.
myTextFrame.Contents = "Hello World!"
Rem Save the document (fill in a valid file path).
myDocument.Save "c:\HelloWorld.indd"
Rem Close the document.
myInDesignServer.Documents.Item(1).Close
```

2. Save the file as text with the file extension `.vbs`.

3. To run the script, double-click the file in Windows Explorer.
4. Open and view the document with InDesign.

Adding features to "Hello World"

Next, we create a script that creates another "Hello World" document. Our second script will do the following:

Use a function (or *handler* in AppleScript) to get the page dimensions and page margins of a document.

Resize a text frame.

Change the formatting of the text in the text frame.

Export as PDF.

AppleScript

To create the script (see ImprovedHelloWorld.applescript), follow these steps:

1. Choose File > New in the Script Editor and enter the following script:

```
--ImprovedHello World.as
tell application "InDesignServer"
    set myDocument to make document
    set myPage to page 1 of myDocument
    tell myPage
        set myTextFrame to make text frame
    end tell
    set contents of myTextFrame to "Hello World!"
    --Get page width and page height using the function "myGetBounds".
    set myBounds to my myGetBounds(myDocument, myPage)
    --Resize the text frame to match the publication margins.
    set geometric bounds of myTextFrame to myBounds
    set myParagraph to object reference of paragraph 1 of myTextFrame
    --Change the font, size, and alignment.
    --Enter the name of a font on your system, if necessary.
    try
        set myFont to font "Arial" of application "InDesignServer"
        set applied font of myParagraph to myFont
    end try
    --Change the size of the text.
    set point size of myParagraph to 48
    --Set the justification of the paragraph to center align.
    set justification of myParagraph to center align
    --Set the first baseline offset of the text frame to ascent.
    set first baseline offset of text frame preferences of myTextFrame
    to ascent offset
    --Set the vertical justification of the text frame to center.
    set vertical justification of text frame preferences of
    myTextFrame to center align
    --Export the document as PDF (fill in a valid file path).
```

```
    tell myDocument
        --You'll need to fill in a valid file path on your system.
        export format PDF type to "yukino:ImprovedHelloWorld.pdf"
    end tell
    --Close the document.
    close document 1 saving no
end tell
on myGetBounds(myDocument, myPage)
    tell application "InDesignServer"
        set myPageHeight to page height of document preferences of myDocument
        set myPageWidth to page width of document preferences of myDocument
        tell margin preferences of myPage
            if side of myPage is left hand then
                set myX2 to left
                set myX1 to right
            else
                set myX1 to left
                set myX2 to right
            end if
            set myY1 to top
            set myY2 to bottom
        end tell
        set myX2 to myPageWidth - myX2
        set myY2 to myPageHeight - myY2
        return {myTop, myLeft, myBottom, myRight}
    end tell
end myGetBounds
```

2. Save the script as a text file with the file extension `.applescript`.
3. To run the script, click the Run button in the Script Editor.
4. Open and view the exported PDF with Acrobat or Adobe Reader.

JavaScript

To create the script (see ImprovedHelloWorld.jsx), follow these steps:

1. Using the ExtendScript Toolkit, enter the following script:

```
//ImprovedHelloWorld.jsx
var myDocument = app.documents.add();
var myPage = myDocument.pages.item(0);
var myTextFrame = myPage.textFrames.add();
myTextFrame.contents = "Hello World!";
//Get page width and page height using the function "myGetBounds".
myBounds = myGetBounds(myDocument, myPage);
//Resize the text frame to match the publication margins.
myTextFrame.geometricBounds = myBounds;
var myParagraph = myTextFrame.paragraphs.item(0);
//Change the font, size, and alignment.
//Enter the name of a font on your system, if necessary.
try {
    var myFont = app.fonts.item("Arial");
    myParagraph.appliedFont = myFont;
}
catch (e){}
//Change the size of the text.
myParagraph.pointSize = 48;
```

```
//Set the justification of the paragraph to center align.
myParagraph.justification = Justification.centerAlign
//Set the first baseline offset of the text frame to ascent.
myTextFrame.textFramePreferences.firstBaselineOffset = FirstBaseline.ascentOffset;
//Set the vertical justification of the text frame to center.
myTextFrame.textFramePreferences.verticalJustification =
VerticalJustification.centerAlign;
//Export the document as PDF (fill in a valid file path).
myDocument.exportFile(ExportFormat.pdfType, new
File("/c/ImprovedHelloWorld.pdf"));
//Close the document.
app.documents.item(0).close();
function myGetBounds(myDocument, myPage){
    var myPageWidth = myDocument.documentPreferences.pageWidth;
    var myPageHeight = myDocument.documentPreferences.pageHeight
    if(myPage.side == PageSideOptions.leftHand){
        var myX2 = myPage.marginPreferences.left;
        var myX1 = myPage.marginPreferences.right;
    }
    else{
        var myX1 = myPage.marginPreferences.left;
        var myX2 = myPage.marginPreferences.right;
    }

    var myY1 = myPage.marginPreferences.top;
    var myX2 = myPageWidth - myX2;
    var myY2 = myPageHeight - myPage.marginPreferences.bottom;
    return [myY1, myX1, myY2, myX2];
}
```

2. Save the text as a plain text file with the file extension `.jsx`.
3. Choose InDesign Server from the target drop-down menu.
4. Click the Run button.
5. Open and view the exported PDF with Acrobat or Adobe Reader.

VBScript

To create the script (see ImprovedHelloWorld.vbs), follow these steps:

1. Start any text editor (such as Notepad) and enter the following script:

```
Rem ImprovedHelloWorld.vbs
Set myInDesignServer = CreateObject("InDesignServer.Application")
Set myDocument = myInDesignServer.Documents.Add
Set myPage = myDocument.Pages.Item(1)
Set myTextFrame = myPage.TextFrames.Add
myTextFrame.Contents = "Hello World!"
Rem Get page width and page height using the function "myGetBounds".
myBounds = myGetBounds(myDocument, myPage)
Rem Resize the text frame to match the publication margins.
myTextFrame.GeometricBounds = myBounds
Set myParagraph = myTextFrame.Paragraphs.Item(1)
```

```
Rem Change the font, size, and alignment.
Rem Enter the name of a font on your system, if necessary.
If TypeName(myInDesignServer.Fonts.Item("Arial"))<>"Nothing" Then
    Set myFont = myInDesignServer.Fonts.Item("Arial")
    myParagraph.AppliedFont = myFont
End If
myParagraph.PointSize = 48
Rem If you are running the script using the Application.DoScript
Rem method, you can use the full name of the justification enumeration:
Rem myParagraph.Justification = idJustification.idCenterAlign
Rem If you are running the script from Windows Explorer, use
Rem the decimal version of the enumeration:
myParagraph.Justification = 1667591796
Rem Set the first baseline offset of the text frame to ascent.
Rem If you are running the script using the Application.DoScript
Rem method, you can use the full name of the justification enumeration:
Rem myTextFrame.TextFramePreferences.FirstBaselineOffset =
idFirstBaseline.idAscentOffset
Rem If you are running the script from Windows Explorer, use
Rem the decimal version of the enumeration:
myTextFrame.TextFramePreferences.FirstBaselineOffset = 1296135023
Rem Set the vertical justification of the text frame to center.
Rem If you are running the script using the Application.DoScript
Rem method, you can use the full name of the justification enumeration:
Rem myTextFrame.TextFramePreferences.VerticalJustification =
idVerticalJustification.idCenterAlign
Rem If you are running the script from Windows Explorer, use
Rem the decimal version of the enumeration:
myTextFrame.TextFramePreferences.VerticalJustification = 1667591796
Rem Export the document as PDF (fill in a valid file path).
Rem myDocument.Export idExportFormat.idPDFType "c:\ImprovedHelloWorld.indd"
myDocument.Export 1952403524, "c:\ImprovedHelloWorld.pdf"
Rem Close the document.
myInDesignServer.Documents.Item(1).Close
Function myGetBounds(myDocument, myPage)
    myPageWidth = myDocument.documentPreferences.pageWidth
    myPageHeight = myDocument.documentPreferences.pageHeight
    If myPage.Side = idPageSideOptions.idLeftHand Then
        myX2 = myPage.marginPreferences.Left
        myX1 = myPage.marginPreferences.Right
    Else
        myX1 = myPage.marginPreferences.Left
        myX2 = myPage.marginPreferences.Right
    End If
    myX2 = myPageWidth - myX2
    myY1 = myPage.marginPreferences.Top
    myY2 = myPageHeight - myPage.marginPreferences.bottom
    myGetBounds = Array(myY1, myX1, myY2, myX2)
End Function
```

2. Save the text as a plain text file with the file extension `.vbs`.
3. To run the script, double-click the file in Windows Explorer.
4. Open and view the exported PDF with Acrobat or Adobe Reader.

A simple script runner script

Another way to run a script in InDesign Server is to take advantage of the `do script` (in AppleScript; `doScript` in JavaScript; or `DoScript` in VBScript) method to send a script to the application. This gives you a way to send JavaScripts directly to the application, rather than running them from the ExtendScript Toolkit. Below, we provide AppleScript and VBScript examples you can run from the Finder (on Mac OS) or Explorer (on Windows). For more information about working with the `do script` method, see *Adobe InDesign CS4 Scripting Guide*.

This script is something like a miniature Test Client for use on your local system (see DoScript).

AppleScript

```
set myScript file to choose file with prompt "Select a script file:"
tell application "Finder"
    set myFileExtension to name extension of myScriptFile
end tell
tell application "Adobe InDesign Server CS4"
    if myFileExtension is "jsx" then
    do script myScriptFile language javascript
else
    do script myScriptFile language applescript language
end if
end tell
```

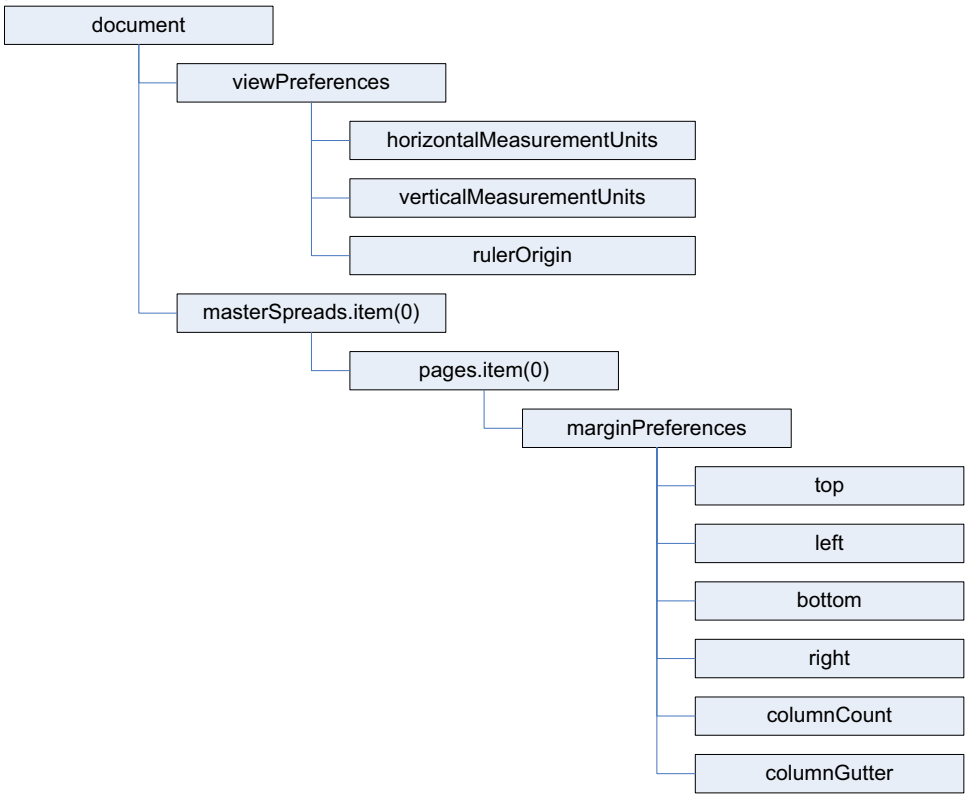
VBScript

```
Set myDialog = CreateObject("UserAccounts.CommonDialog")
myDialog.Filter = "JavaScript Files|*.jsx|VBScript Files|*.vbs|"
myDialog.FilterIndex = 1
myDialog.InitialDir = "C:\"
myResult = myDialog.ShowOpen
If myResult = True Then
    myScriptFile = myDialog.FileName
    Set myFileSystemObject = CreateObject("Scripting.FileSystemObject")
    myFileExtension = myFileSystemObject.GetExtensionName(myScriptFile)
    If myFileExtension = "jsx" Or myFileExtension = "vbs" Then
        Set myInDesign = CreateObject("InDesign.Application")
        Rem Use the decimal form of the enumerations, since this script
        Rem will be run from Explorer.
        If myFileExtension = "jsx" Then
            Rem idScriptLanguage.idJavaScript = 1246973031
            myInDesign.DoScript myScriptFile, 1246973031
        Else
            Rem idScriptLanguage.idVisualBasic = 1447185511
            myInDesign.DoScript myScriptFile, 1447185511
        End If
    End If
End If
```

Constructing a document

Obviously, our "Hello World!" script is not very useful in your daily work. While you can use an InDesign script at any point in your production process, we start by creating scripts that start at the same point you do—creating new documents, setting page margins, and creating and applying master pages. The following figure shows the objects with which we will work.

.



In this section, we create one long script by adding short blocks of scripting code. Each block demonstrates a specific area or task in InDesign Server scripting. As you enter each block, you can run the script to see what happens. If you are using AppleScript, you will need to add the text `end tell` to the end of the script before you run it, then remove the text before continuing.

NOTE: The figure above uses the JavaScript version of the scripting terms. For AppleScript, you would add spaces between words (`view preferences`, rather than `viewPreferences`). For VBScript, you would use an item index starting at 1, rather than 0 (`masterSpreads.item(1)`, rather than `masterSpreads.item(0)`).

Setting up measurement units and master spread margins

The following script shows how to create a new document and set the margins of the first master spread. In this section, we show how to build a complex script using simple building blocks of scripting code. Start your script editor and enter the following lines in the scripting language of your choice.

AppleScript

Enter the following code in the Script Editor, or open the `DocumentConstruction.applescript` tutorial script:

```
tell application "Adobe InDesign Server CS4"
    --Create a new document.
    set myDocument to make document
    --Set the measurement units and ruler origin.
    set horizontal measurement units of view preferences to points
    set vertical measurement units of view preferences to points
    set ruler origin of view preferences to page origin
    --Get a reference to the first master spread.
    set myMasterSpread to master spread 1 of myDocument
    --Get a reference to the margin preferences of
    --the first page in the master spread.
    set myMarginPreferences to margin preferences of page 1 of myMasterSpread
    --Now set up the page margins and columns.
    set left of myMarginPreferences to 84
    set top of myMarginPreferences to 70
    set right of myMarginPreferences to 70
    set bottom of myMarginPreferences to 78
    set column count of myMarginPreferences to 3
    set column gutter of myMarginPreferences to 14
    --Page margins and columns for the right-hand page.
    set myMarginPreferences to margin preferences of page 2 of myMasterSpread
    set left of myMarginPreferences to 84
    set top of myMarginPreferences to 70
    set right of myMarginPreferences to 70
    set bottom of myMarginPreferences to 78
    set column count of myMarginPreferences to 3
    set column gutter of myMarginPreferences to 14
```

JavaScript

Enter the following code in the ExtendScript Toolkit, or open the `DocumentConstruction.jsx` tutorial script:

```
//Create a new document.
var myDocument = app.documents.add();
//Set the measurement units and ruler origin.
myDocument.viewPreferences.horizontalMeasurementUnits = MeasurementUnits.points;
myDocument.viewPreferences.verticalMeasurementUnits = MeasurementUnits.points;
myDocument.viewPreferences.rulerOrigin = RulerOrigin.pageOrigin;
//Get a reference to the first master spread.
var myMasterSpread = myDocument.masterSpreads.item(0);
//Get a reference to the margin preferences of the first page in the master spread.
var myMarginPreferences = myMasterSpread.pages.item(0).marginPreferences;
//Now set up the page margins and columns.
myMarginPreferences.left = 84;
myMarginPreferences.top = 70;
myMarginPreferences.right = 70;
myMarginPreferences.bottom = 78;
myMarginPreferences.columnCount = 3;
myMarginPreferences.columnGutter = 14;
//Page margins and columns for the right-hand page.
var myMarginPreferences = myMasterSpread.pages.item(1).marginPreferences;
myMarginPreferences.left = 84;
myMarginPreferences.top = 70;
myMarginPreferences.right = 70;
myMarginPreferences.bottom = 78;
myMarginPreferences.columnCount = 3;
myMarginPreferences.columnGutter = 14;
```

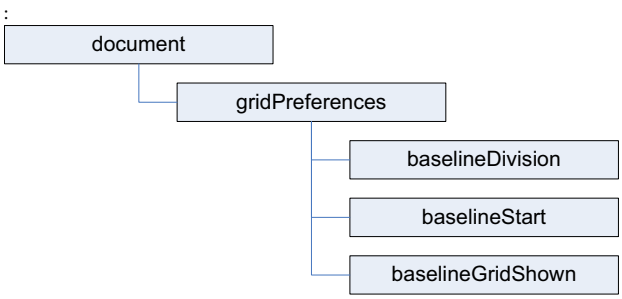
VBScript

Enter the following code in your script or text editor, or open the `DocumentConstruction.vbs` tutorial script.

```
Set myInDesign = CreateObject("InDesign.Application")
Rem Create a new document.
Set myDocument = myInDesign.Documents.Add()
Rem Set the measurement units and ruler origin.
myDocument.ViewPreferences.HorizontalMeasurementUnits =
idMeasurementUnits.idPoints
myDocument.ViewPreferences.VerticalMeasurementUnits = idMeasurementUnits.idPoints
myDocument.ViewPreferences.RulerOrigin = idRulerOrigin.idPageOrigin
Rem Get a reference to the first master spread.
Set myMasterSpread = myDocument.MasterSpreads.Item(1)
Rem Get a reference to the margin preferences of the first page in the master spread.
Set myMarginPreferences = myMasterSpread.Pages.Item(1).MarginPreferences
Rem Now set up the page margins and columns.
myMarginPreferences.Left = 84
myMarginPreferences.Top = 70
myMarginPreferences.Right = 70
myMarginPreferences.Bottom = 78
myMarginPreferences.ColumnCount = 3
myMarginPreferences.ColumnGutter = 14
Rem Page margins and columns for the right-hand page.
Set myMarginPreferences = myMasterSpread.Pages.Item(2).MarginPreferences
myMarginPreferences.Left = 84
myMarginPreferences.Top = 70
myMarginPreferences.Right = 70
myMarginPreferences.Bottom = 78
myMarginPreferences.ColumnCount = 3
myMarginPreferences.ColumnGutter = 14
```

Adding a baseline grid

Now that we have a master spread set up, we add a baseline grid. Add the following script lines (from the appropriate language) to the end of the script you created earlier. Here is a diagram (with the scripting terms shown in their JavaScript form):



AppleScript

```
set myGridPreferences to grid preferences
set baseline division of myGridPreferences to 14
set baseline start of myGridPreferences to 70
set baseline grid shown of myGridPreferences to true
```

JavaScript

```
var myGridPreferences = myDocument.gridPreferences;
myGridPreferences.baselineDivision = 14;
myGridPreferences.baselineStart = 70;
myGridPreferences.baselineGridShown = true;
```

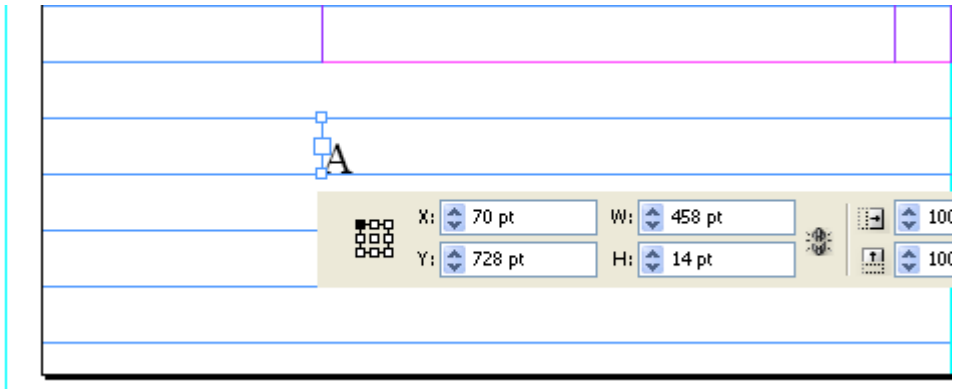
VBScript

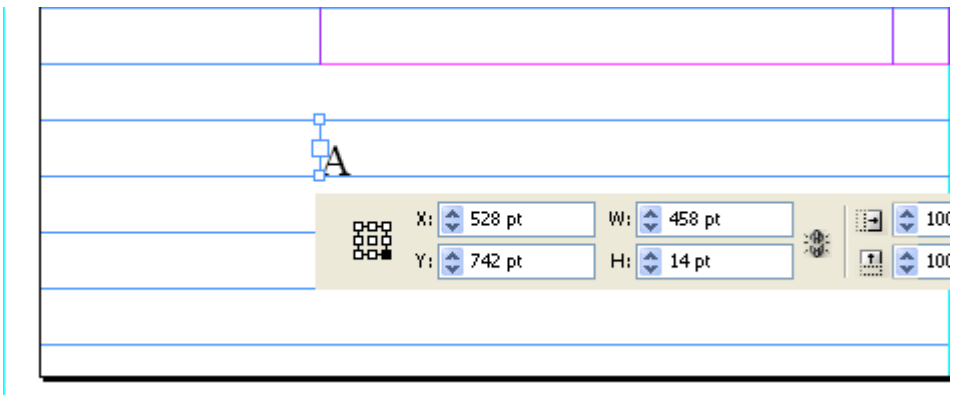
```
Set myGridPreferences = myDocument.GridPreferences
myGridPreferences.BaselineDivision = 14
myGridPreferences.BaselineStart = 70
myGridPreferences.BaselineGridShown = True
```

Adding master page items

Next, we add two text frames to the master pages. These frames will contain the auto-page-number special character and be positioned at the bottom of the page.

In the "Hello World" example, we created a text frame and specified its position and size using the geometric bounds property—an array containing the top, left, bottom, and right coordinates for the frame. The coordinates correspond to the corners of the frame, just as they would appear in the Control panel. The geometric bounds are as follows: top = 728, left = 70, bottom = 742, and right = 528, as shown in the following two figures (which show what it would look like in InDesign):





AppleScript

```
set myLeftPage to page 1 of myMasterSpread
set myRightPage to page 2 of myMasterSpread
tell myLeftPage
    set myLeftFooter to make text frame
    set geometric bounds of myLeftFooter to {728, 70, 742, 528}
    set first baseline offset of text frame preferences of myLeftFooter to leading
offset
    set contents of myLeftFooter to auto page number
    set point size of character 1 of parent story of myLeftFooter to 11
    set leading of character 1 of myLeftFooter to 14
end tell
tell myRightPage
    set myRightFooter to make text frame
    set geometric bounds of myRightFooter to {728, 84, 742, 542}
    set first baseline offset of text frame preferences of myRightFooter to leading
offset
    set contents of myRightFooter to auto page number
    set point size of character 1 of parent story of myRightFooter to 11
    set leading of character 1 of myRightFooter to 14
    set justification of character 1 of myRightFooter to right align
end tell
```

JavaScript

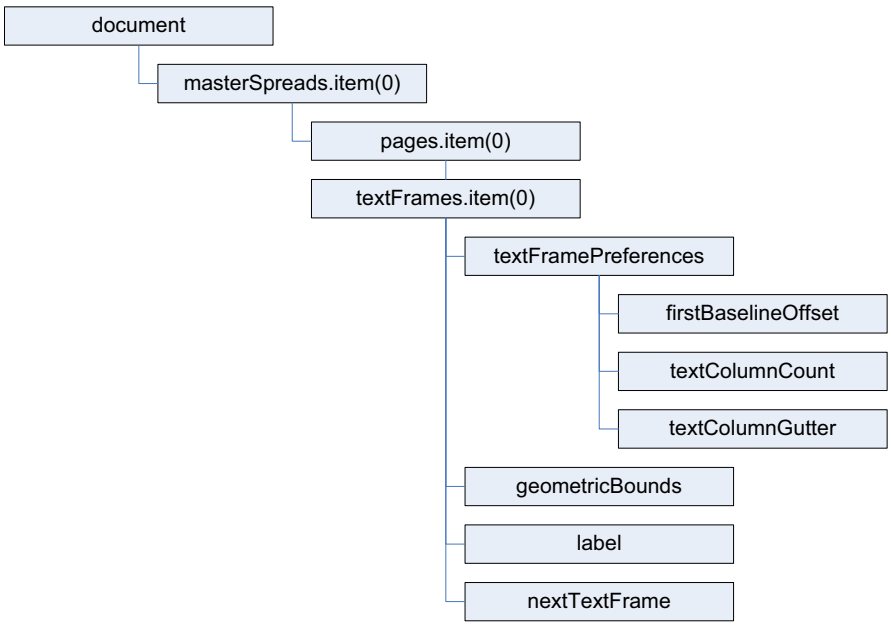
```
var myMasterSpread = myDocument.masterSpreads.item(0);
var myLeftPage = myMasterSpread.pages.item(0);
var myRightPage = myMasterSpread.pages.item(1);
var myLeftFooter = myLeftPage.textFrames.add();
myLeftFooter.geometricBounds = [728, 70, 742, 528];
myLeftFooter.textFramePreferences.firstBaselineOffset = FirstBaseline.leadingOffset;
myLeftFooter.contents = SpecialCharacters.autoPageNumber;
myLeftFooter.parentStory.characters.item(0).pointSize = 11;
myLeftFooter.parentStory.characters.item(0).leading = 14;
var myRightFooter = myRightPage.textFrames.add();
myRightFooter.geometricBounds = [728, 84, 742, 542];
myRightFooter.textFramePreferences.firstBaselineOffset = FirstBaseline.leadingOffset;
myRightFooter.contents = SpecialCharacters.autoPageNumber;
myRightFooter.parentStory.characters.item(0).pointSize = 11;
myRightFooter.parentStory.characters.item(0).leading = 14;
myRightFooter.parentStory.characters.item(0).justification =
Justification.rightAlign;
```

VBScript

```
Set myMasterSpread = myDocument.MasterSpreads.Item(1)
Set myLeftPage = myMasterSpread.Pages.Item(1)
Set myRightPage = myMasterSpread.Pages.Item(2)
Set myLeftFooter = myLeftPage.TextFrames.Add
myLeftFooter.GeometricBounds = Array(728, 70, 742, 528)
myLeftFooter.TextFramePreferences.FirstBaselineOffset =
idFirstBaseline.idLeadingOffset
myLeftFooter.Contents = idSpecialCharacters.idAutoPageNumber
myLeftFooter.ParentStory.Characters.Item(1).PointSize = 11
myLeftFooter.ParentStory.Characters.Item(1).Leading = 14
Set myRightFooter = myRightPage.TextFrames.Add()
myRightFooter.GeometricBounds = Array(728, 84, 742, 542)
myRightFooter.TextFramePreferences.FirstBaselineOffset =
idFirstBaseline.idLeadingOffset
myRightFooter.Contents = idSpecialCharacters.idAutoPageNumber
myRightFooter.ParentStory.Characters.Item(1).PointSize = 11
myRightFooter.ParentStory.Characters.Item(1).Leading = 14
myRightFooter.ParentStory.Characters.Item(1).Justification =
idJustification.idRightAlign
```

Adding master text frames

Next, we add master text frames. The following block diagram shows the objects and properties with which we will work (the diagram uses the JavaScript form of the scripting terms):



AppleScript

```
tell myLeftPage
    set myLeftTextFrame to make text frame
    set geometric bounds of myLeftTextFrame to {70, 70, 714, 528}
    set first baseline offset of text frame preferences of myLeftTextFrame to leading
offset
    set text column count of text frame preferences of myLeftTextFrame to 3
    set text column gutter of text frame preferences of myLeftTextFrame to 14
    --Add a label to make the frame easier to find later on.
    set label of myLeftTextFrame to "BodyTextFrame"
end tell
tell myRightPage
    set myRightTextFrame to make text frame
    set geometric bounds of myRightTextFrame to {70, 84, 714, 542}
    set first baseline offset of text frame preferences of myRightTextFrame to leading
offset
    set text column count of text frame preferences of myRightTextFrame to 3
    set text column gutter of text frame preferences of myRightTextFrame to 14
    --Add a label to make the frame easier to find later on.
    set label of myRightTextFrame to "BodyTextFrame"
end tell
--Link the two frames using the next text frame property.
set next text frame of myLeftTextFrame to myRightTextFrame
```

JavaScript

```
var myLeftPage = myMasterSpread.pages.item(0);
var myRightPage = myMasterSpread.pages.item(1);
var myLeftTextFrame = myLeftPage.textFrames.add();
myLeftTextFrame.geometricBounds = [70, 70, 714, 528];
myLeftTextFrame.textFramePreferences.firstBaselineOffset =
FirstBaseline.leadingOffset;
myLeftTextFrame.textFramePreferences.textColumnCount = 3;
myLeftTextFrame.textFramePreferences.textColumnGutter = 14;
//Add a label to make the frame easier to find later on.
myLeftTextFrame.label = "BodyTextFrame";
var myRightTextFrame = myRightPage.textFrames.add();
myRightTextFrame.geometricBounds = [70, 84, 714, 542];
myRightTextFrame.textFramePreferences.firstBaselineOffset =
FirstBaseline.leadingOffset;
myRightTextFrame.textFramePreferences.textColumnCount = 3;
myRightTextFrame.textFramePreferences.textColumnGutter = 14;
//Add a label to make the frame easier to find later on.
myRightTextFrame.label = "BodyTextFrame";
//Link the two frames using the nextTextFrame property.
myLeftTextFrame.nextTextFrame = myRightTextFrame;
```

VBScript

```
Set myLeftTextFrame = myLeftPage.TextFrames.Add
myLeftTextFrame.GeometricBounds = Array(70, 70, 714, 528)
myLeftTextFrame.TextFramePreferences.FirstBaselineOffset =
idFirstBaseline.idLeadingOffset
myLeftTextFrame.TextFramePreferences.TextColumnCount = 3
myLeftTextFrame.TextFramePreferences.TextColumnGutter = 14
Rem Add a label to make the frame easier to find later on.
myLeftTextFrame.Label = "BodyTextFrame"
Set myRightTextFrame = myRightPage.TextFrames.Add
myRightTextFrame.GeometricBounds = Array(70, 84, 714, 542)
myRightTextFrame.TextFramePreferences.FirstBaselineOffset =
idFirstBaseline.idLeadingOffset
myRightTextFrame.TextFramePreferences.TextColumnCount = 3
myRightTextFrame.TextFramePreferences.TextColumnGutter = 14
Rem Add a label to make the frame easier to find later on.
myRightTextFrame.Label = "BodyTextFrame"
Rem Link the two frames using the nextTextFrame property.
myLeftTextFrame.NextTextFrame = myRightTextFrame
```

Overriding master page items and adding text

Next, we override one of the master text frames we created and add text to it. Again, add this script to the end of the script we have been working on.

AppleScript

```
tell text frame 1 of page 2 of master spread 1 of myDocument
    set myTextFrame to override destination page page 1 of myDocument
end tell
--Add text by setting the contents of an insertion point to a string.
--In AppleScript, "return" is a return character.
set contents of insertion point 1 of myTextFrame to "Headline!" & return
```

JavaScript

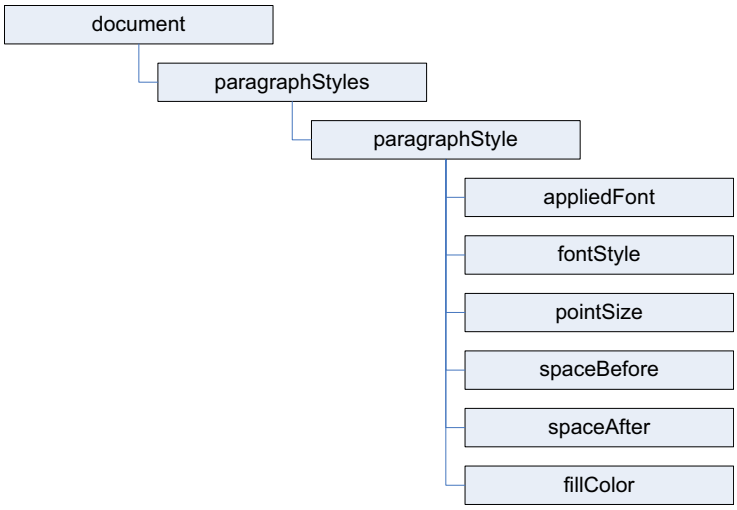
```
var myTextFrame =
myDocument.masterSpreads.item(0).pages.item(1).textFrames.item(0).override(myDocument
.pages.item(0));
//Add text by setting the contents of an insertion point to a string.
//In JavaScript, "\r" is a return character.
myTextFrame.insertionPoints.item(0).contents = "Headline!\r";
```

VBScript

```
Set myTextFrame =
myDocument.MasterSpreads.Item(1).Pages.Item(2).TextFrames.Item(1).Override(myDocument
.Pages.Item(1))
Rem Add text by setting the contents of an insertion point to a string.
Rem In VBScript, vbCr is a return character.
myTextFrame.InsertionPoints.Item(1).Contents = "Headline!" & vbCr
```

Adding and applying a paragraph style

Our headline looks plain, so we format it using a paragraph style. First, we must create the paragraph style. The following diagram shows the objects and properties we will work with (again, the text in this diagram uses the JavaScript form of the scripting terms):



AppleScript

```
--First, check to see if the paragraph style already exists.
try
    set myParagraphStyle to paragraph style "Heading 1" of myDocument
on error
    --The paragraph style did not exist, so create it.
    tell myDocument
        set myParagraphStyle to make paragraph style with properties {name:"Heading 1"}
    end tell
end try
--We'll need to create a color. Check to see if the color already exists.
try
    set myColor to color "Red"
on error
    --The color did not exist, so create it.
    set myColor to make color with properties {name:"Red", model:process, color
value:{0, 100, 100, 0}}
end try
--Now set the formatting of the paragraph style.
set applied font of myParagraphStyle to "Arial"
set font style of myParagraphStyle to "Bold"
set point size of myParagraphStyle to 24
set space after of myParagraphStyle to 24
set space before of myParagraphStyle to 24
set fill color of myParagraphStyle to color "Red" of myDocument
--Apply the style to the paragraph.
tell paragraph 1 of myTextFrame to apply paragraph style using myParagraphStyle with
clearing overrides
--You could also use:
--set applied paragraph style of paragraph 1 of myTextFrame to myParagraphStyle
```

JavaScript

```
var myParagraphStyle = myDocument.paragraphStyles.item("Heading 1");
try {
    var myName = myParagraphStyle.name;
}
catch (myError){
    //The paragraph style did not exist, so create it.
    myParagraphStyle = myDocument.paragraphStyles.add({name:"Heading 1"});
}
//We'll need to create a color. Check to see if the color already exists.
var myColor = myDocument.colors.item("Red");
try {
    myName = myColor.name;
}
catch (myError){
    //The color did not exist, so create it.
    myColor = myDocument.colors.add({name:"Red", model:ColorModel.process,
    colorValue:[0,100,100,0]});
}
//Now set the formatting of the paragraph style.
myParagraphStyle.appliedFont = "Arial";
myParagraphStyle.fontStyle = "Bold";
myParagraphStyle.pointSize = 24;
myParagraphStyle.spaceAfter = 24;
myParagraphStyle.spaceBefore = 24;
myParagraphStyle.fillColor = myDocument.colors.item("Red");
//Apply the style to the paragraph.
myDocument.pages.item(0).textFrames.item(0).paragraphs.item(0).applyParagraphStyle(
myParagraphStyle, true);
//You could also use:
//myDocument.pages.item(0).textFrames.item(0).paragraphs.item(0).appliedParagraphStyl
e = myParagraphStyle;
```

VBScript

```
Rem First, check to see if the paragraph style already exists.
Rem to do this, we disable error checking:
On Error Resume Next
Set myParagraphStyle = myDocument.ParagraphStyles.Item("Heading 1")
Rem if an error occurred on the previous line, then the paragraph
Rem style did not exist.
If Error.Number <> 0 Then
    Set myParagraphStyle = myDocument.ParagraphStyles.Add
    myParagraphStyle.Name = "Heading 1"
    Error.Clear
End If
Rem We'll need to create a color. Check to see if the color already exists.
Set myColor = myDocument.Colors.Item("Red")
If Error.Number <> 0 Then
    Set myColor = myDocument.Colors.Add
    myColor.Name = "Red"
    myColor.Model = idColorModel.idProcess
    myColor.colorValue = Array(0, 100, 100, 0)
    Error.Clear
End If
Rem Resume normal error handling.
On Error GoTo 0
```

```
Rem Now set the formatting of the paragraph style.
myParagraphStyle.AppliedFont = "Arial"
myParagraphStyle.FontStyle = "Bold"
myParagraphStyle.PointSize = 24
myParagraphStyle.SpaceAfter = 24
myParagraphStyle.SpaceBefore = 24
myParagraphStyle.FillColor = myDocument.Colors.Item("Red")
Rem Apply the style to the paragraph.
myDocument.Pages.Item(1).TextFrames.Item(1).Paragraphs.Item(1).ApplyParagraphStyle
myParagraphStyle, True
Rem You could also use:
Rem
myDocument.pages.item(1).textFrames.item(1).paragraphs.item(1).appliedParagraphStyle
= myParagraphStyle
```

Placing a text file

Next, we import a text file. We add the text after the headline in the first text frame on the first page. The script displays a dialog box you can use to select the text file you want to import. Again, add this script to the end of the script we have been working on.

AppleScript

```
--Display a standard open file dialog box to select a text file.
set myTextFile to choose file ("Choose a text file")
--If a text file was selected, and if you didn't press Cancel,
--place the text file at the first insertion point after the headline.
if myTextFile is not "" then
    tell insertion point -1 of myTextFrame to place myTextFile
end if
```

JavaScript

```
//Display a standard open file dialog box to select a text file.
var myTextFile = File.openDialog("Choose a text file");
//If a text file was selected, and if you didn't press Cancel,
//place the text file at the first insertion point after the headline.
if((myTextFile != "")&&(myTextFile != null)){
    myTextFrame.insertionPoints.item(-1).place(myTextFile);
}
```

VBScript

```
Rem Display a standard open file dialog box to select a text file.
Rem VBScript does not have the ability to do this, so we'll use
Rem a JavaScript to get a file name. We'll run the JavaScript using
Rem InDesign's DoScript feature.
Rem Disable normal error handling.
On Error Resume Next
Rem Create a JavaScript as a string.
myJavaScriptString = "var myTextFile = File.openDialog(""Choose a text
file"");myTextFile.fsName;"
Rem Run the JavaScript using DoScript.
myFileName = myInDesign.DoScript(myJavaScriptString, idScriptLanguage.idJavascript)
If Error.Number = 0 Then
    Rem Place the text file at the end of the text frame.
    myTextFrame.InsertionPoints.Item(-1).Place myFileName
    Error.Clear
End If
Rem Restore normal error handling.
On Error GoTo 0
```

Placing a graphic

Placing a graphic is like importing a text file. Again, the script displays a dialog box you can use to select the graphic you want to place. When we place the graphic, InDesign Server returns a reference to the graphic itself, rather than to the frame containing the graphic. To get a reference to the frame, use the `parent` property of the graphic. Once we have that reference, we can apply an object style to the frame. Again, add this script to the end of the script we have been working on.

AppleScript

```
--Display a standard open file dialog box to select a graphic file.
set myGraphicFile to choose file "Choose graphic file."
--If a graphic file was selected, and if you didn't press Cancel,
--place the graphic file on the page.
if myGraphicFile is not "" then
    set myGraphic to place myGraphicFile on page 1 of myDocument
    --Since you can place multiple graphics at once, the place method
    --returns an array. To get the graphic you placed, get the first
    --item in the array.
    set myGraphic to item 1 of myGraphic
    --Create an object style to apply to the graphic frame.
    try
        set myObjectStyle to object style "GraphicFrame" of myDocument on error
        --The object style did not exist, so create it.
        tell myDocument
            set myObjectStyle to make object style with properties
            {name:"GraphicFrame"}
        end tell
    end try
    set enable stroke of myObjectStyle to true
    set stroke weight of myObjectStyle to 3
    set stroke type of myObjectStyle to stroke style "Solid" of myDocument
    set stroke color of myObjectStyle to color "Red" of myDocument
```

```
        --The frame containing the graphic is the parent of the graphic.
        set myFrame to parent of myGraphic
        tell myFrame to apply object style using myObjectStyle
        --Resize the frame to a specific size.
        set geometric bounds of myFrame to {0, 0, 144, 144}
        --Fit the graphic to the frame proportionally.
        fit myFrame given proportionally
        --Next, fit frame to the resized graphic.
        fit myFrame given frame to content
        set myBounds to geometric bounds of myFrame
        set myGraphicWidth to (item 4 of myBounds) - (item 2 of myBounds)
        --Move the graphic frame.
        set myPageWidth to page width of document preferences of myDocument
        set myMarginPreferences to margin preferences of page 1 of myDocument
        set myTopMargin to top of myMarginPreferences
        move myFrame to {myPageWidth - myGraphicWidth, myTopMargin}
        --Apply a text wrap to the graphic frame.
        set text wrap type of text wrap preferences
        of myFrame to bounding box text wrap
        set text wrap offset of text wrap preferences of myFrame to {24, 12, 24, 12}
    end if
end tell
```

JavaScript

```
//Display a standard open file dialog box to select a graphic file.
var myGraphicFile = File.openDialog("Choose a graphic file");
//If a graphic file was selected, and if you didn't press Cancel,
//place the graphic file on the page.
if((myGraphicFile != "")&&(myGraphicFile != null)){
    var myGraphic = myDocument.pages.item(0).place(myGraphicFile);
    //Since you can place multiple graphics at once, the place method
    //returns an array. To get the graphic you placed, get the first
    //item in the array (JavaScript arrays start with item 0).
    myGraphic = myGraphic[0];
    //Create an object style to apply to the graphic frame.
    var myObjectStyle = myDocument.objectStyles.item("GraphicFrame");
    try {
        var myName = myObjectStyle.name;
    }
    catch (myError){
        //The object style did not exist, so create it.
        myObjectStyle = myDocument.objectStyles.add({name:"GraphicFrame"});
    }
    myObjectStyle.enableStroke = true;
    myObjectStyle.strokeWeight = 3;
    myObjectStyle.strokeType = myDocument.strokeStyles.item("Solid");
    myObjectStyle.strokeColor = myDocument.colors.item("Red");
    //The frame containing the graphic is the parent of the graphic.
    var myFrame = myGraphic.parent;
    myFrame.applyObjectStyle(myObjectStyle, true);
    //Resize the frame to a specific size.
    myFrame.geometricBounds = [0,0,144,144];
    //Fit the graphic to the frame proportionally.
    myFrame.fit(FitOptions.proportionally);
    //Next, fit frame to the resized graphic.
    myFrame.fit(FitOptions.frameToContent);
    var myBounds = myFrame.geometricBounds;
    var myGraphicWidth = myBounds[3]-myBounds[1];
```

```
    //Move the graphic frame.
    var myPageWidth = myDocument.documentPreferences.pageWidth;
    var myTopMargin = myDocument.pages.item(0).marginPreferences.top;
    myFrame.move([myPageWidth-myGraphicWidth, myTopMargin]);
    //Apply a text wrap to the graphic frame.
    myFrame.textWrapPreferences.textWrapType = TextWrapTypes.boundingBoxTextWrap;
    myFrame.textWrapPreferences.textWrapOffset = [24, 12, 24, 12];
}
```

VBScript

```
Rem create an object style
On Error Resume Next
Set myObjectStyle = myDocument.ObjectStyles.Item("GraphicFrame")
If Error.Number <> 0 Then
    Set myObjectStyle = myDocument.ObjectStyles.Add
    myObjectStyle.Name = "GraphicFrame"
    Error.Clear
End If
On Error GoTo 0
myObjectStyle.EnableStroke = True
myObjectStyle.StrokeWeight = 3
myObjectStyle.StrokeType = myDocument.StrokeStyles.Item("Solid")
myObjectStyle.StrokeColor = myDocument.Colors.Item("Red")
Rem Again, we'll use a JavaScript to get a file name.
Rem Disable normal error handling.
On Error Resume Next
Rem Create a JavaScript as a string.
myJavaScriptString = "var myTextFile = File.openDialog(""Choose a graphic
file"");myTextFile.fsName;"
Rem Run the JavaScript using DoScript.
myGraphicFileName = myInDesign.DoScript(myJavaScriptString,
idScriptLanguage.idJavascript)
If Error.Number = 0 Then
    On Error GoTo 0
    Set myGraphic = myDocument.Pages.Item(1).Place(myGraphicFileName)
    Rem Since you can place multiple graphics at once, the place method
    Rem returns an object collection. To get the graphic you placed, get the first
    Rem item in the collection.
    Set myGraphic = myGraphic.Item(1)
    Rem Create an object style to apply to the graphic frame.
    Rem The frame containing the graphic is the parent of the graphic.
    Set myFrame = myGraphic.Parent
    myFrame.ApplyObjectStyle myObjectStyle, True
    Rem Resize the frame to a specific size.
    myFrame.GeometricBounds = Array(0, 0, 144, 144)
    Rem Fit the graphic to the frame proportionally.
    myFrame.Fit idFitOptions.idProportionally
    Rem Next, fit frame to the resized graphic.
    myFrame.Fit idFitOptions.idFrameToContent
    myBounds = myFrame.GeometricBounds
    myGraphicWidth = myBounds(3) - myBounds(1)
    Rem Move the graphic frame.
    myPageWidth = myDocument.DocumentPreferences.PageWidth
    myTopMargin = myDocument.Pages.Item(1).MarginPreferences.Top
    myFrame.Move Array(myPageWidth - myGraphicWidth, myTopMargin)
```

```
    Rem Apply a text wrap to the graphic frame.
    myFrame.TextWrapPreferences.TextWrapType = idTextWrapTypes.idBoundingBoxTextWrap
    myFrame.TextWrapPreferences.TextWrapOffset = Array(24, 12, 24, 12)
End If
```

Converting an InDesign script to InDesign Server

InDesign CS4 comes with a variety of sample scripts, some of which might be useful in an InDesign Server production environment. The FindChangeByList script is a good example; it performs a series of find/change operations defined by a tab-delimited text file. The functions in this script could be very useful if your use of InDesign server involves many find/change operations.

The following steps show how to convert FindChangeByList to an InDesign Server script (see FindChangeByList for the complete script).

AppleScript

1. Replace the `main` handler with the following:

```
on main()
    tell application "InDesign Server"
        if (count documents) is greater than 0 then
            --Provide a story object or a document object
            my myFindChangeByList(document 1, false)
        end if
    end tell
end main
```

2. Delete the `myDisplayDialog` handler.

3. Change the line:

```
set myFindChangeFile to my myFindFile("FindChangeSupport:FindChangeList.txt")
```

   to (you will have to fill in a valid file path for your system):

```
set myFindChangeFile to "yukino:FindChangeSupport:FindChangeList.txt"
Delete the "myFindChangeFile" handler
```

4. Delete the `myGetScriptPath` handler.

JavaScript

1. Replace the `main` function with the following:

```
function main(){
    if(app.documents.length > 0){
    //Provide a story object or a document object.
    myFindChangeByList(app.documents.item(0), false);
    }
}
```

2. Delete the `myDisplayDialog` function.

3. Change the line:

   ```
   var myFindChangeFile = "/c/FindChangeSupport/FindChangeList.txt";
   ```

   to (you will have to fill in a valid file path for your system):

   ```
   set myFindChangeFile to "yukino:FindChangeSupport:FindChangeList.txt"
   ```

4. Delete the `myFindChangeFile` function.
5. Delete the `myGetScriptPath` function.

VBScript

1. Replace the `main` function with the following:

   ```
   Function main()
   Set myInDesign = CreateObject("InDesignServer.Application")
       If myInDesign.Documents.Count > 0 then
           Rem Provide a story object or a document object.
           myFindChangeByList myInDesign.Documents.Item(1), False
       End If
   End Function
   Delete the "myDisplayDialog" handler.
   ```

2. Change the line:

   ```
   myFindChangeFile = "\FindChangeSupport\FindChangeList.txt"
   ```

   to (you will have to fill in a valid file path for your system):

   ```
   myFindChangeFile = "c:\FindChangeSupport\FindChangeList.txt"
   ```

3. Delete the `myFindChangeFile` function.
4. Delete the `myGetScriptPath` function.