

SCRIPTS

Scripts are small programs written to perform one or more tasks in InDesign in an automated or semi-automatic way. Adobe InDesign supports AppleScript (Mac), JavaScript (Mac and PC) and VisualBasic (PC).

Scripts

Adobe supplies a series of Scripts with Adobe InDesign that can be installed at leisure. These scripts are not installed by default, but can be found on the InDesign installer/resources CDs. In addition many people writing scripts share their scripts publicly on the Adobe Studio Exchange: <http://share.studio.adobe.com/>. Register an *Adobe ID* and select a password in order to download shared files.

WEB link

Installing scripts

To install a script for use in InDesign, copy the script file to the following location:

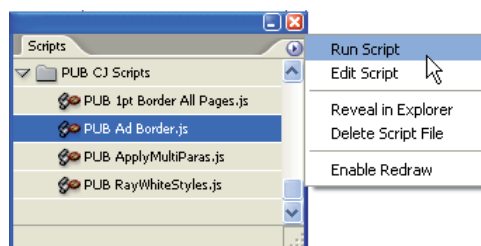
Mac Macintosh HD (or name of disk containing your applications folder)
Applications / Adobe InDesign CS2 / Presets / Scripts

PC My Computer ... (C:) drive (or name of disk containing your program files folder)
Program Files / Adobe / Adobe InDesign CS2 / Presets / Scripts

If you have a larger group of scripts it might be an idea to organise your scripts in sub-folders. The Script palette lists the folders and when the folder is expanded (click the expand triangle to the left of the folder icon) the scripts contained in the folder become accessible.

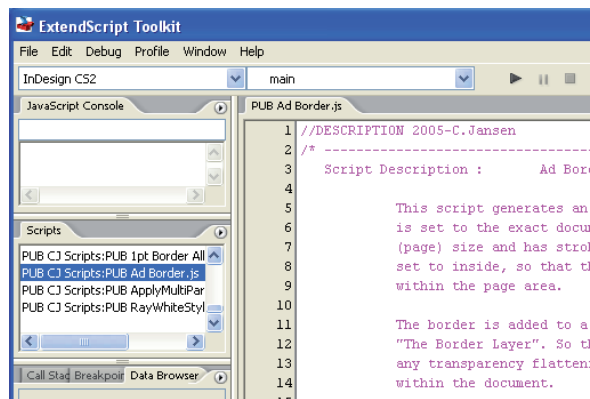
Running scripts

To have a script perform a series of actions that have been programmed, the script must be run. To run a script launch the *Scripts* palette (*Window > Automation*), then scroll to the relevant script and double click the script name or select *Run Script* from the Script palette fly-out menu.



Scripts palette (Window > Automation).

Some scripts will perform an action fully automatic, whilst others might require the entry of details in a dialog box. Most script authors will add some descriptive text at the beginning of the script text outlining how the scripts are to be used.

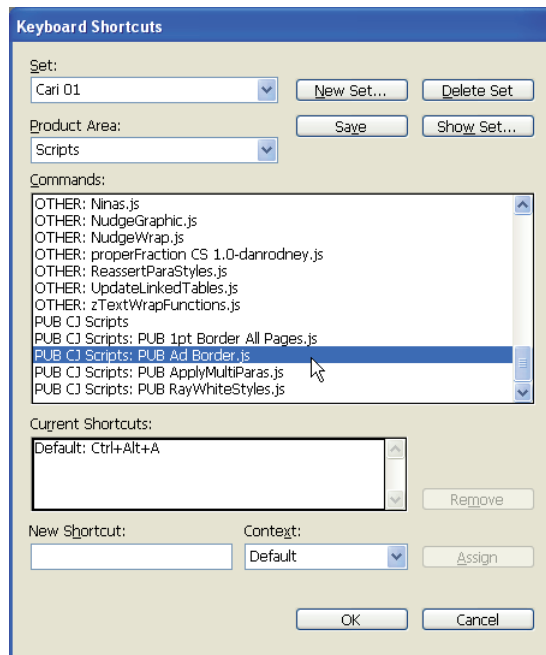


Example of script description added to script and displayed in ExtendScript Toolkit

To view the script text and reveal such notes select **Edit Script** from the Scripts palette fly-out menu. This will launch the script in the *ExtendScript Toolkit* that is installed with the Adobe Creative Suite 2. The scripts might also be launched in other script or text editors and even a web browser.

Applying shortcuts to scripts

Each script listed in the Scripts palette can have a shortcut allocated to it. When pressed, these shortcuts will cause the script to run. Select the **Keyboard Shortcuts** command from the *Edit menu*, and select **Scripts** from the **Product Area** section in the dialog box. Scroll down to the script for which you want to create a shortcut and enter a shortcut (by typing the actual shortcut on your keyboard) in the **New Shortcut** box.



Setting shortcuts for Scripts (*Edit > Keyboard shortcuts*).

If the shortcut you have entered is already assigned to another command a warning is listed below the New Shortcut box. Click **Assign** to apply the shortcut to the script, then click **OK** to close the Keyboard Shortcuts dialog box.

SCRIPTS SUPPLIED WITH ADOBE CREATIVE SUITE 2

During the installation of the Adobe Creative Suite no scripts are installed in the Presets/Scripts folder located in the Adobe InDesign application folder. However, 21 scripts are provided on the Resources and Extras Disk 1 that is part of your CreativeSuite installer pack. These scripts are located in the Technical Information>InDesign CS2>Scripting>Adobe Sample Scripts folder and can be added to InDesign separately.

TextCleanup.jsx

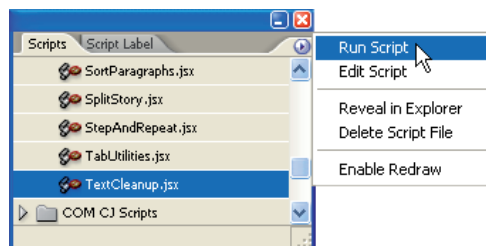
The text cleanup script is one of the free scripts provided on the Creative Suite 2 Resources CD and enables running of multiple find and change commands on a document that contains badly formatted text in a single action, by simply running the script.

Who hasn't run through a Find/Change command multiple times to change double spaces with single spaces over and over, patiently waiting for the "Search is complete. 'Ø' replacement(s) made" message, before clicking Done. Then started another Find/Change command to replace double End of Paragraph characters with single End of Paragraph Characters?

Think of replacing all those double spaces double, double tabs, double paragraphs in one hit. Rather than running through the regular Edit > Find/Change command numerous times. You can give instructions to change the word 'Street' to 'St' and at the same time 'Avenue' to 'Ave' and 'Drive' to 'Dr'.

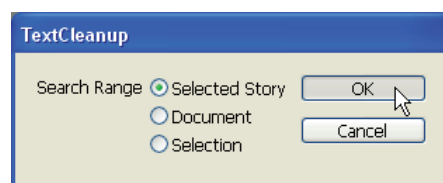
Using the script

Install the 'TextCleanup.jsx' file in the *Scripts folder*. The text cleanup script works a bit differently to other scripts. Apart from running (double clicking) the script from the Scripts palette (*Window > Automation*) it will require you to point to an external text file that stores all the individual Find/Change commands list that are to be performed by the TextCleanUp.jsx script.



Running a selected script: double click script or select Run Script from fly-out menu.

The use of an external text file enables you to run this script for different jobs each with their own find/change requirements. All that is required are individual text files that store the various find/change commands.

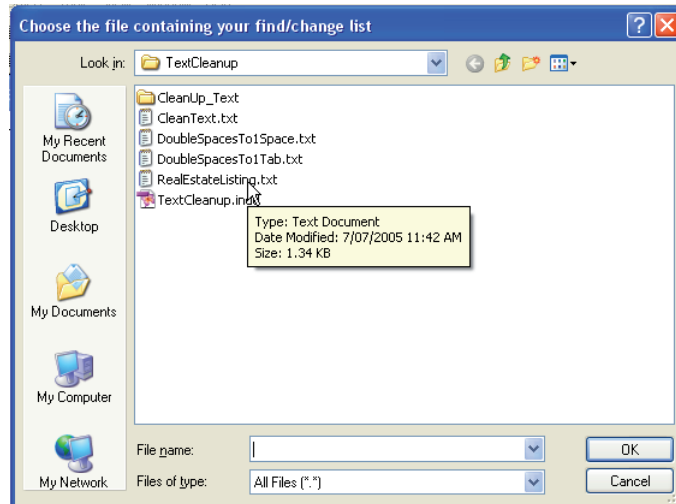


TextCleanup dialog. Confirm the text that has to be searched.

To run the script ensure that you have a text frame selected or text highlighted and have access to the external find/change commands file. As the script is executed you will be prompted

with the TextCleanup dialog box that asks you to verify the content to which you want to perform the TextCleanup.jsx. script (option1) selected story (selected textframe, including text contained in threaded frames), (option 2) document, or (option 3) selection (highlighted text).

Click **OK** to acknowledge the content selection. You will now be prompted to choose the file containing the find/change list.



Select the text file containing the Find/Change commands.

Locate the text file that contains your single or multiple find/change commands and click **OK**. The TextCleanup Script now performs all your requested actions.

About the Find/Change text file

The text file that stores your find/changes settings must be formatted in a very specific way. Create a new file in a simple text editing program such as WordPad (PC), NotePad (PC) or TextEdit (Mac).

Each individual Find/Change command (i.e. find 2 spaces and change to a single tab character), must be listed as a separate line of text in the text file containing the find/change commands. This line includes three command components and a short description of the individual find/change action to be performed. Each individual component is separated from the next one by a single <tab> keystroke on the keyboard.

```
{findText:" "} {changeText:"^t"} all Find all double spaces and replace with a tab.
{findText:"^t^t"} {changeText:"^t"} all Find all double tabs and replace with a tab.
{findText:"^p^p"} {changeText:"^p"} all Find all double end of paragraphs and replace with single carriage return.
```

*Sample find/change text file that replaces spaces by tabs, then double tabs by a single tab and double carriage returns by single carriage returns.
The result is that a each series of multiple spaces is replaced by a single tab character and no 'blank' lines between paragraphs.*

These components are (1) the text that must be found (findText), entered between double quotes, (2) the text that must replace the found text (changeText) entered between double quotes, (3) the number of times the find/change instruction must be run (either 'all' or 'once') and (4) a short description of the change/find line.

Creating a single command line in the text file

Enter the text to be replaced as {findText:"old text here"}, between double quotes. Ensure that the quotes you are using are straight quotes (not smart quotes). Next **Press <tab>**, then enter the text that replaces the found text as {changeText:"new text here"} between double

quotes, *press <tab>* and type **all** to run the script till 'all occurrences of the old text are found and replaced' or **once** to run the find/change action once, press <tab> again and enter a short description. To add another find/change line to your list of commands, press <enter> (pc) or <return> (mac) and repeat the preceding instructions.

Once or All?

As part of the single command line you must indicate how many times the script must run through your list of find/change commands.

Selecting 'all' option causes the Find/Change commands to be repeated until the last occurrence of the find has been found and replaced. An example of a repeated find/change action requiring the 'all' option would be a story or document containing tons of double, triple, quadruple spaces. When using the Edit > Find/Change command to change all instances of multiple spaces to a single space you could start by entering two spaces in the 'Find what' box and a single space in the 'Change to' box. Next click the 'Change All' button for a first run through (which might state that 160 occurrences were found). This first run however would still leave triple and double space instances, therefore the 'Change All' button is clicked again and again until '0' occurrences were found. The 'all' option simulates this behaviour.

Using the 'all' option when finding a word (eg. 'snippet') that appears throughout your document 50 times and requires small Caps formatting through application of a character style would result in what in programming terms is called an 'endless loop'. As the word itself is not changed, the script will continue to find occurrences and keep on repeating. Fortunately a safety net has been built into the script that warns you when the script has been rerun 100 times and enables you to stop the execution of the Find/Change list. In this case you would choose the 'once' option.

Special characters

As indicated in the Find/Change text file example listed previously a find/change of a tab character requires the entry of the text **^t** between the double quotes. The following table provides an overview of the characters you need to use for finding and changing of special characters.

SPECIAL TEXT CHARACTER	ENTER FOLLOWING CHARACTER
Paragraph	^p
Tab	^t
Right Indent Tab	^y
Space	<enter a space character>
Any tab or space	^w
Any Digit (use only for findText)	^9
Any Letter (use only for findText)	^\$
Any Character (use only for findText)	^?
Em Space	^m
En Space	^n
" (double quote)	\ "

Advanced find/change options

The TextCleanup.jsx script can be used extensively for any Find/Change command you would normally perform manually. This could include application of formatting through use of styles to the found text.

The use of appliedCharacterStyle for special formatting (all caps, small caps, baseline shift etc.) is very effective as it is much easier to globally change the appearance generated by this style at a later time. Remember that a character style can record a single text formatting feature

NOTE:

Not all available special characters are listed in the table. For a full overview please refer to the **Help Center>Text Section>Finding and Changing Text To Find, Change Formatted text help.**

(or even no features at all). For single line paragraphs you might consider applying a (nested) paragraph style.

CHANGE OPTIONS	DESCRIPTION
<code>{appliedCharacterStyle:"character stylename here"}</code>	formats the found text with the referenced character style.
<code>{appliedParagraphStyle:"paragraph stylename here"}</code>	formats the found text (and the paragraph in which it resides) with the referenced paragraph style. Great when working with price tags in ads.
<code>{changeText:"new text here"}</code>	replace the found text with inserted characters or text-string.

Examples of usage

Apart from general text cleanup (removing double spaces etc.), the TextCleanup.jsx script is quite handy when used in combination with ‘appliedCharacterStyle’ and ‘appliedParagraphStyle’ options. The following provide a few examples of other productivity improvers.

Use ‘appliedCharacterStyle’ when you want to apply a character style that only:

- ▶ sets the colour of found text
- ▶ sets the ‘capitalisation’ of found text (such as acronyms) to all Caps, small Caps
- ▶ sets the underline or strikethrough formatting (great when setting forms and you are using the tabs to line up text components, and want to underline your tabs).
- ▶ sets the size, leading, kerning or tracking values of the found text
- ▶ sets the font or font style of the found text etc.

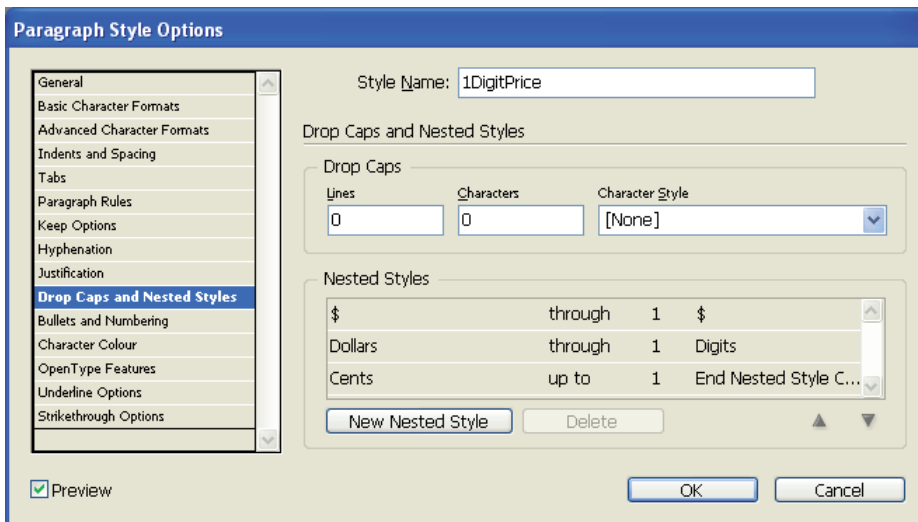
Use ‘appliedParagraphStyle’ when applying a paragraph style to the paragraph that contains the found text. This can be useful when:

- ▶ Applies a nested style to your text (as in the price tags example)
- ▶ Sets paragraph styles to all text frames in the document

Example formatting price tags in ads

Let's get the script perform a laborious task for us. The following example automatically formats price tags in a document by finding all prices and formatting them using three different paragraph styles.

In your InDesign document create a **new paragraph style** that contains nested styles (character styles for '\$', 'dollars' and 'cents').



Paragraph style for 1DigitPrice (change number of digits for other styles).

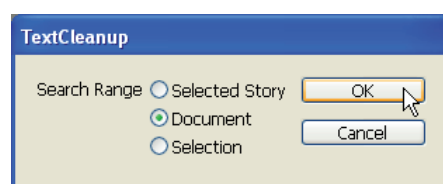
I've started with my **1DigitPrice** paragraph style and then duplicated this style twice and renamed it **2DigitPrice** and **3DigitPrice**, and edited the style options for the second and third paragraph styles by increasing the number of digits in the nested style (Dollars through 1 Digits) to two and three.

Create a **new text file** in a simple text editor (TextEdit / NotePad) and type your find/change command lines. For the **findText** option enter the number of digits your price tag has, type **^9** (any digit) for each digit in your price tag (eg. \$1⁰⁰ would be ^9^9^9) between straight quotes.

```
{findText:"$^9^9^9"} {appliedParagraphStyle:"1DigitPrice"} once set paragraph style
{findText:"$^9^9^9^9"} {appliedParagraphStyle:"2DigitPrice"} once set paragraph style
{findText:"$^9^9^9^9^9"} {appliedParagraphStyle:"3DigitPrice"} once set paragraph style
```

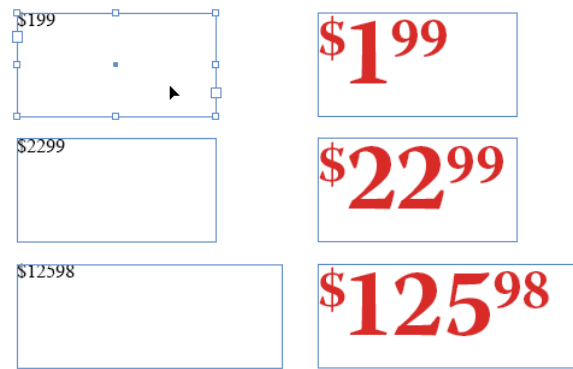
Use **appliedParagraphStyle** for the change section of your command line and enter your paragraph style name (eg. 1DigitPrice) between straight quotes. Set the number of find/changes to be performed to **once**. Repeat this same format for each command line, starting with the lowest number of 'dollars'-digits (i.e. \$1 before \$10).

Save the Find/Change list file you've just created as **myPrices.txt** (text only format) and select one of the unformatted price tag frames in your document and run the TextCleanup.jsx script. Enter Document for Search Range when prompted and select the myPrices.txt file when prompted to choose the file containing your find/change list.



Select Document to perform the changefind on all price tag frames.

The script is executed... and tadaaaaa... all is formatted.



Example of formatting price tags: before (left) and after (right).

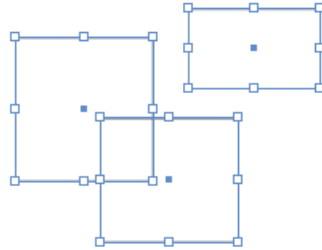
In conclusion

If you find that you are performing numerous find/change tasks repeatedly, then the installation of the TextCleanup.jsx script is a must. I'm convinced that half way through reading this article you would have jumped off your chair and found your installer CDs in order to locate and install the scripts. Launch a simple text editor and start writing those find/change command lines. Happy Find/Changing!

OTHER SCRIPTS

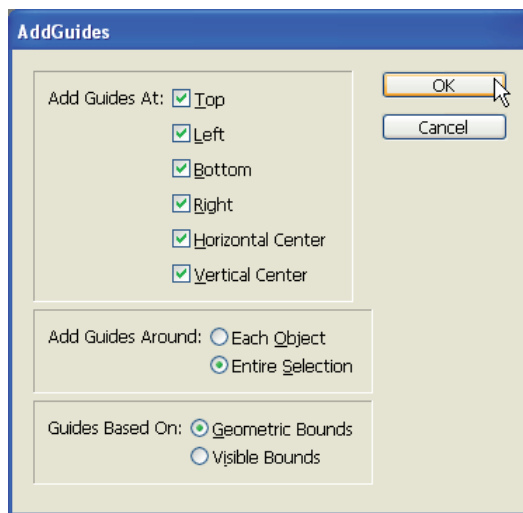
AddGuides.jsx

The AddGuides.jsx script automatically draws ruler guides based on objects that are selected on the page.



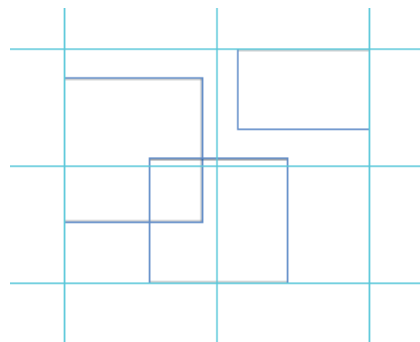
Selected objects on the page.

To use it start by selecting one or more objects on the page, then run the script. A dialog appears providing you with guide setting options.



Add guides dialog box.

Guides can be placed individually (*Each Object*) around each of the selected objects or around all objects, as if they were part of a group (*Entire Selection*).

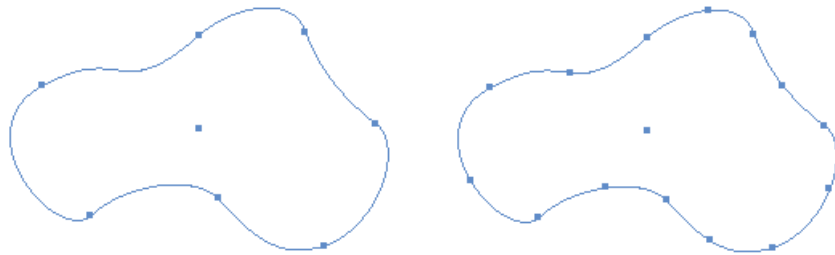


Guides added around entire selection.

Use *visible bounds* to set guides to the outer edge of strokes that might be attached to the objects, use *geometric bounds* to set guides to the frame edges of the objects (ignoring any stroke weights).

AddPoints.jsx

The AddPoints.jsx script adds new anchor points to selected paths. The points are placed half way between to pre-existing anchor points.

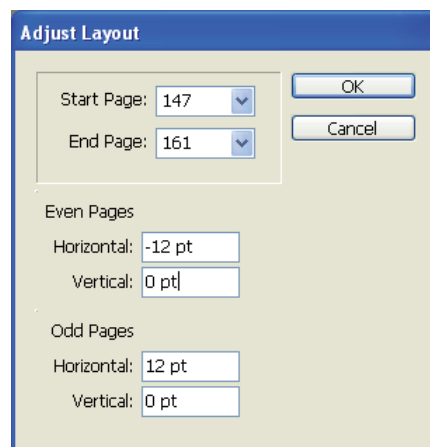


Original path (selected with group selection tool) and path with added anchor points (right).

Visually this script works well when you're adding points to circular and rectangular shapes, however when used on oddly shaped paths it may appear as if the new anchor points aren't added half way between two existing points, this is related to the curvature of the paths.

AdjustLayout.jsx

Within a particular page range, the AdjustLayout.jsx script enables movement of all objects placed on a page in a given direction.

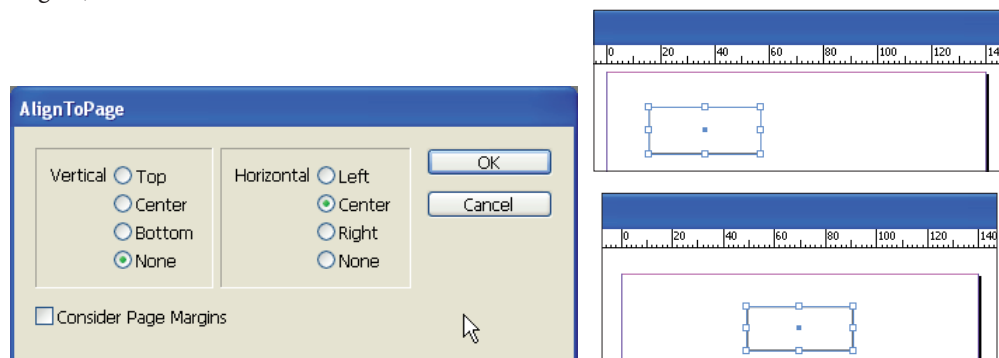


Adjust Layout dialog box.

Selected objects and locked objects will be shift. Master page objects will not be shifted.

AlignToPage.jsx

The AlignToPage.jsx script aligns selected objects in reference to the page boundaries (or the margins).

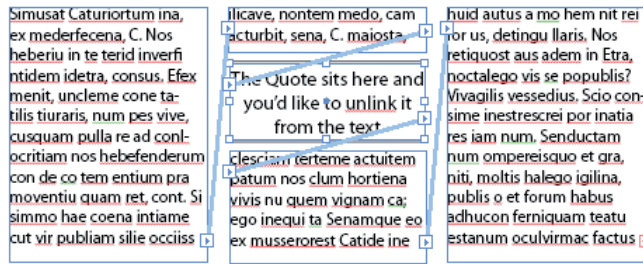


Vertical and horizontal alignment options. Object (right) horizontally aligned.

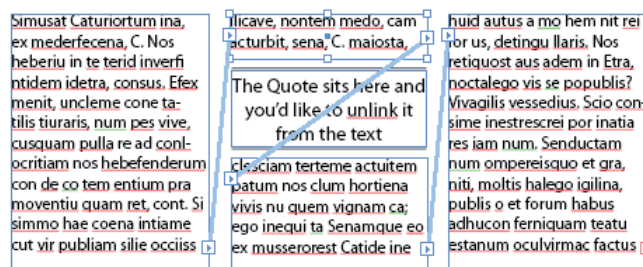
Objects can be aligned vertically and/or horizontally.

BreakFrame.jsx

The BreakFrame.jsx script “cuts” a selected text frame that is part of a story consisting of multiple threaded frames into an individual story (unthreaded text frame).



Quote in story still threaded with rest of story

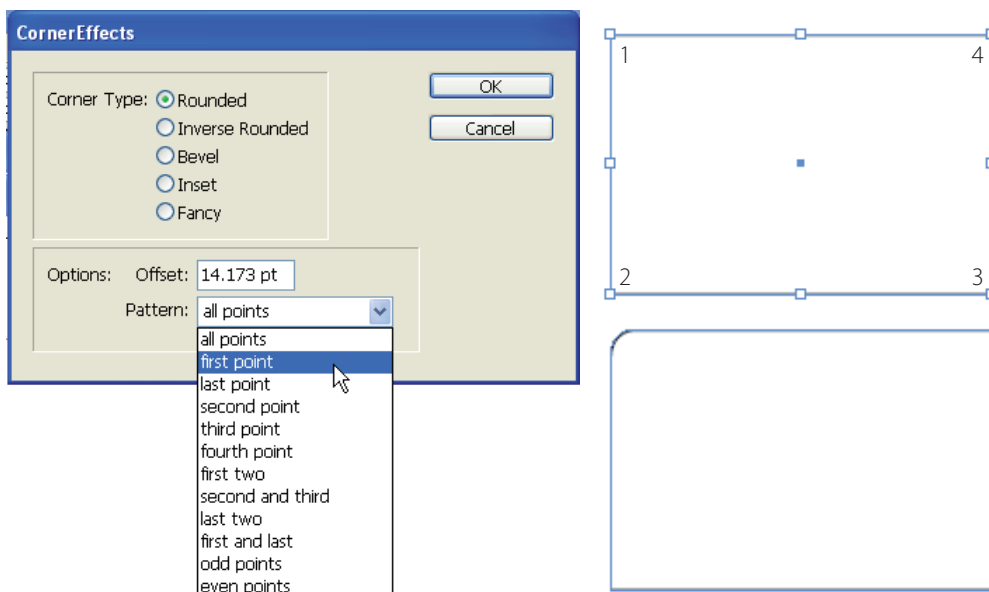


Post running script the quote is no longer part of the story

Select the frame that is part of a series of threaded frames, that needs to be ‘unthreaded’ and run the script. This script comes in handy when removing quotes from linked stories.

CornerEffects.jsx

The CornerEffects.jsx script enables application of the default corner effects available in InDesign to one or more selected corners on a rectangular shape.



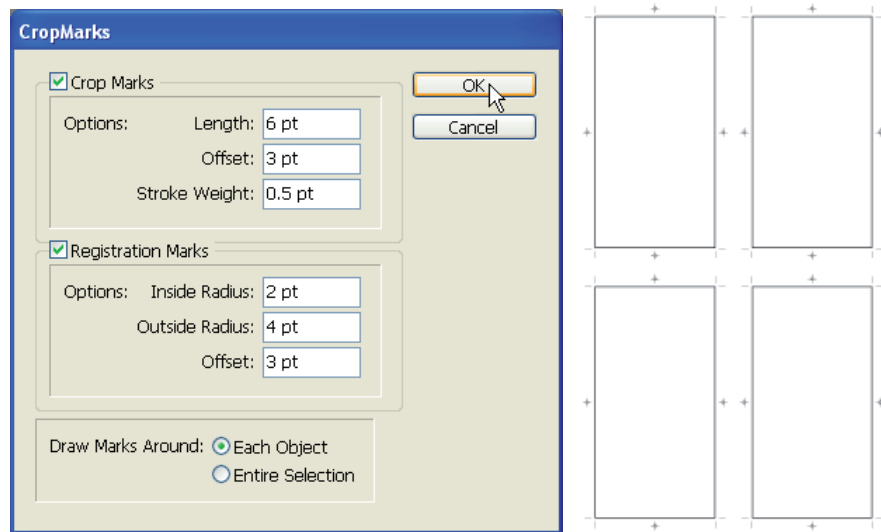
Round Corner Effect applied to the first point on the path.

CreateCharacterStyle.jsx

The CreateCharacterStyle.jsx script creates a new character style from selected text. The character style that is generated records every text formatting feature applied to the text (font, leading, size etc.), not just those features that differ from the Basic Paragraph setting or the currently applied paragraph style. The latter would only record those features that differ from the pre-edited text.

CropMarks.jsx

The CropMarks.jsx script generates crop and/or registration marks around an object that is selected, or around multiple objects. Crop marks are added to a separate layer, so they can easily hidden/removed.

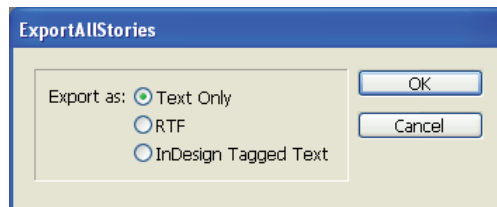


Cropmarks added around each of the four selected objects.

Crop marks can be added to individual selected objects or around the bounding box that surrounds the entire selection. Example of usage is a sheet of business cards layed out 8-up that requires addition of individual crop marks for each card.

ExportAllStories.jsx

The ExportAllStories.jsx script export all stories in an InDesign document. This script is great when a customer asks for the final text in a publication that might receive repeats in the future, but might need editing.

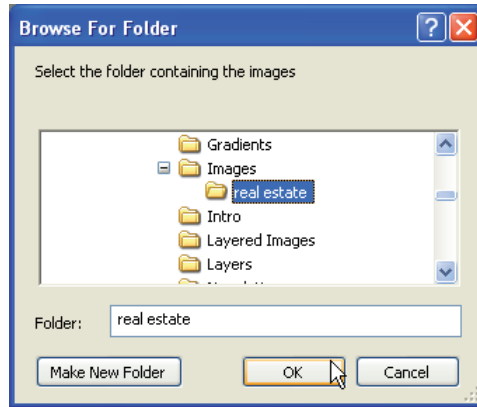


Export All Stories dialog box.

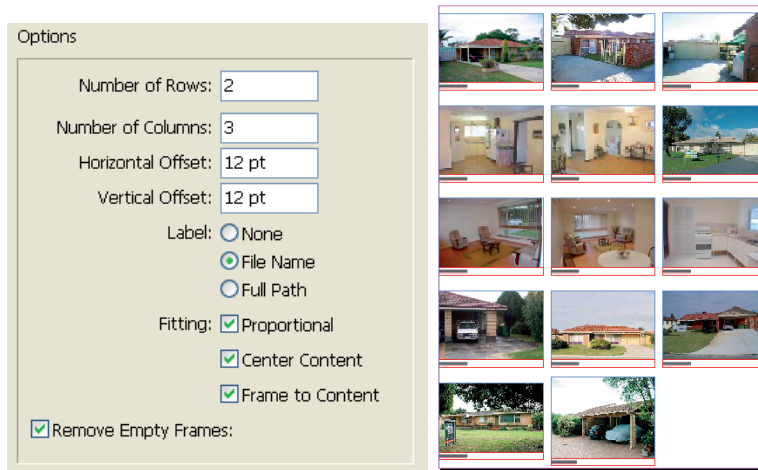
Text can be exported in Text Only format, as a Rich Text File (.rtf) or InDesign Tagged Text.

ImageCatalog.jsx

The ImageCatalog.jsx script, places columns and rows of images located in a folder in a newly created InDesign document.



Select a folder containing images.

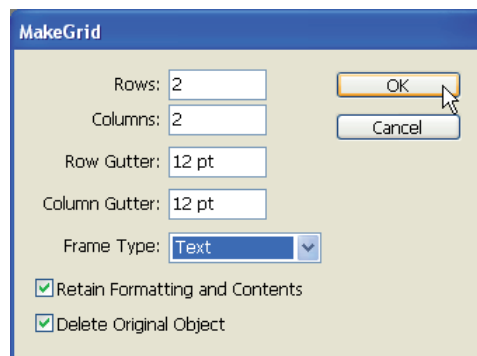


5 rows, 3 columns catalog page created (right).

Labels containing the filename can be added below the images and placed on a separate 'labels' layer. A label paragraph style is used for text formatting and can be amended if needed.

MakeGrid.jsx

The MakeGrid.jsx script can be used to generate a grid of graphic or text frames.



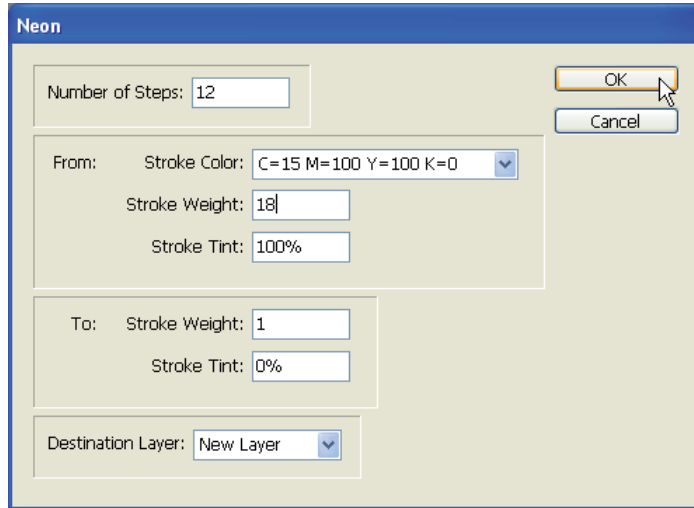
MakeGrid settings.

An frame containing a graphic can be broken into multiple frames. Choose Retain Formatting and Contents to preserve the original image location. The image itself will be duplicated across the multiple frames created. If the frame that is to be divided contains text, and Retain

Formatting and Contents is selected, the text will be duplicated across the newly created frames. Select a frame that is to be subdivided, then run the script.

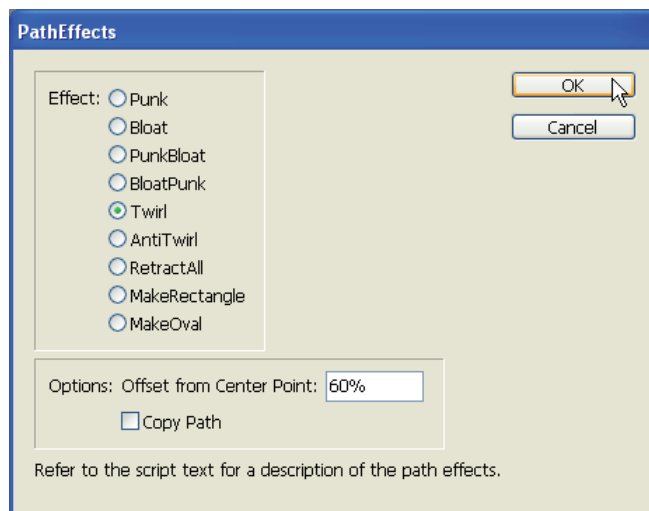
Neon.jsx

The Neon.jsx script creates an artificial blend by duplicating an object over a number of repeated steps and gradually changing the colour values. Thereby simulating a neon effect



Neon script applied to single line.

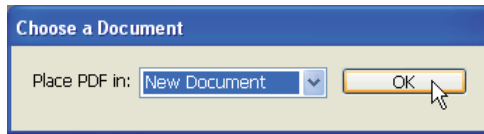
PathEffects.jsx



Path Effects applied to a circle.

PlaceMultiplepagePDF.jsx

The PlacingMultiplepagePDF.jsx script places pages (selected page range) from a PDF into a InDesign document. In a single page document, new pages are added for placement of the PDF pages.

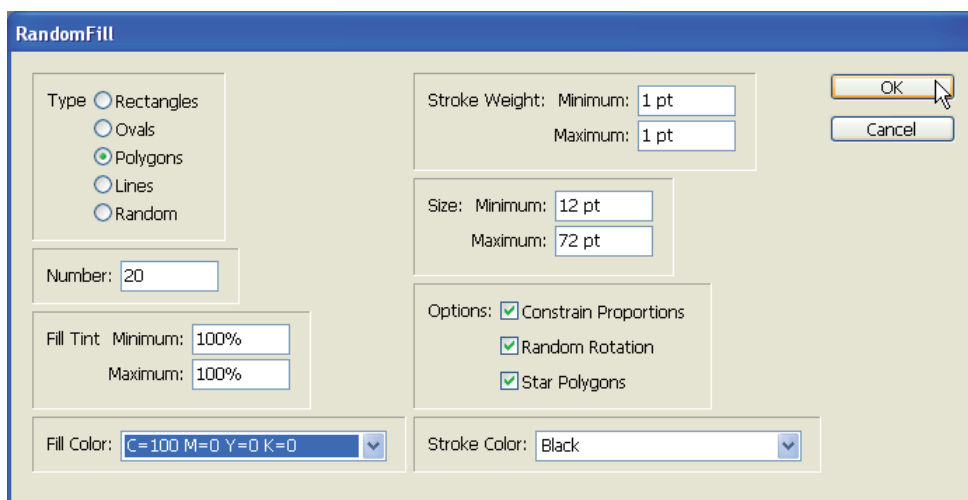


After choosing the document, select placement document for PDF.

The script prompts the user to select a PDF, and determine in which document the PDF pages are to be placed.

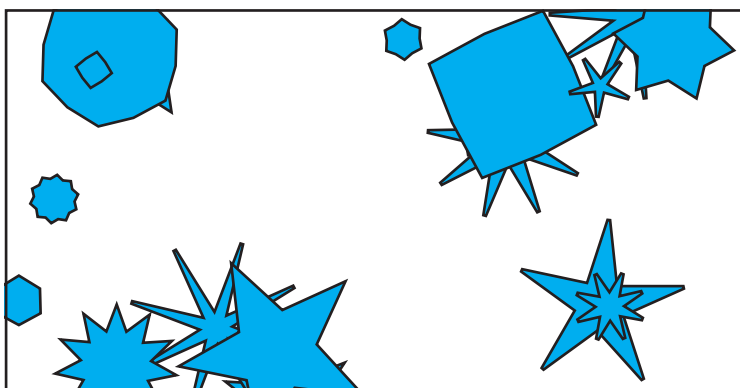
RandomFill.jsx

The RandomFill.jsx fills a selected frame with one or more randomly generated shapes.



Random Fill settings dialog box.

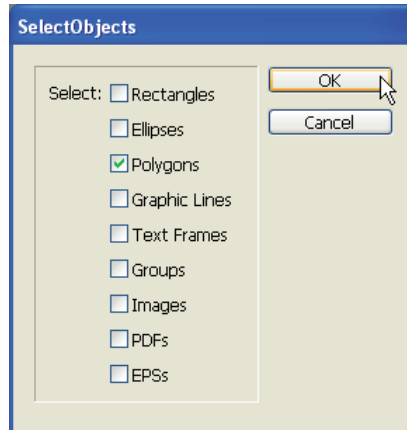
Shapes are pasted into the selected frame and can be individually selected or edited using the direct selection tool.



Randomly applied shapes.

SelectObjects.jsx

The SelectObject.jsx script prompts the user with a dialog box in which one or more object types can be marked for selection.



Select object types to select on spread.

Selection occurs on the active page spread only, not document wide and is restricted to selection of actual frames not the content of the frames.

SortParagraphs.jsx

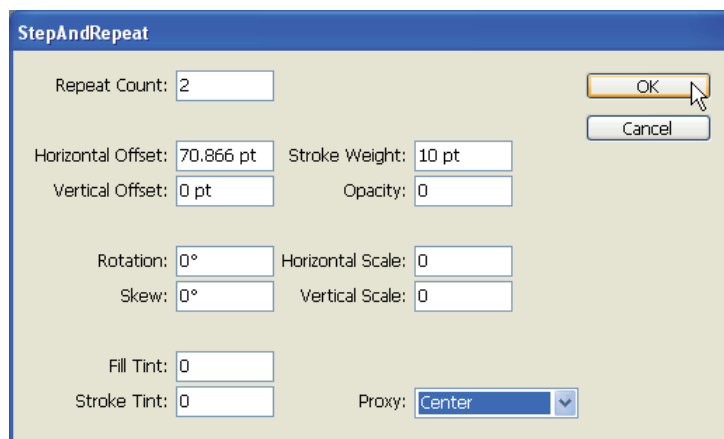
The SortParagraphs.jsx script sorts selected paragraphs in alphabetical order.

SplitStory.jsx

The SplitStory.jsx script splits a series of threaded text frames into unrelated text frames, whilst retaining each individual text frame's content. Hyphenated words will move to one frame, and pre-existing overset text will be removed.

StepAndRepeat.jsx

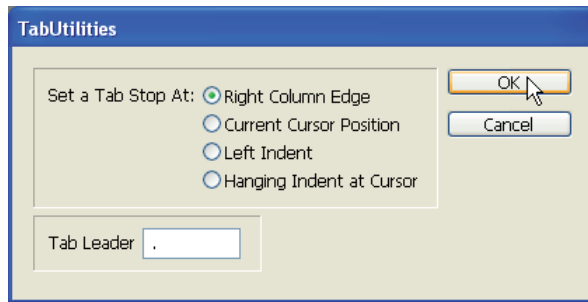
The StepAndRepeat.jsx script generates duplicates for a selected object, not only creating exact duplicates but at the same time amending stroke/fill attributes.



Step and Repeat dialog box.

TabUtilities.jsx

The TabUtilities.jsx script can set a variety of tabs in the tab-ruler. The user must still manually set the tabs in the text.



Tab Utilities dialog box.