

Issue 3 • Number 4

X-RAY

MAGAZINE

QUARK SOFTWARE WORKFLOW SOLUTIONS AND IMPLEMENTATION



tell application "QuarkXPress"

make new document

end tell

tell application "QuarkXPress"

properties of text box 1 of page 1 of document 1

end tell

tell application "QuarkXPress"

Using AppleScript to Interact with Text

– Part 1

BY **BENJAMIN S. WALDIE**

Learning how to automate QuarkXPress with AppleScript can be simple or complex, depending upon the tasks that you plan to automate.

First, you need an understanding of the core AppleScript language. You can do this by familiarizing yourself with the AppleScript Language Guide, which can be found on Apple's AppleScript web site at <http://www.apple.com/applescript>. A comprehensive AppleScript book, such as Danny Goodman's *AppleScript Handbook*, available from SpiderWorks, LLC, (<http://www.spiderworks.com>) is also a good way to get started. Next, you will need to be familiar with QuarkXPress' AppleScript support. This can be done by opening and reviewing QuarkXPress' AppleScript dictionary in Script Editor, and by exploring QuarkXPress' Apple Events Scripting documentation, DOCUMENTS ▾ APPLE EVENTS SCRIPTING FOLDER within your QuarkXPress folder.

You will learn quickly that QuarkXPress' AppleScript support is fairly robust, and many of the repetitive and time-consuming tasks that you perform within QuarkXPress on a daily basis can be automated with AppleScript. In future articles, we will discuss using AppleScript to automate many of these tasks, including working with picture boxes, printing, and more. This month, however, we will begin with discussing how AppleScript can be used to automate interaction with something that affects every QuarkXPress user — text.

There are a large number of text-related tasks that can be automated in QuarkXPress with AppleScript, much more than can fit into a single article. Therefore, for now, we will discuss a small subset of these tasks, including creating a text box, adding text to a text box, and retrieving text from a text box. We will continue our exploration of AppleScript-based text interaction further in future articles, but, you are encouraged to continue exploring on your own as well.

Getting Started

In order to begin using AppleScript to perform text-related tasks, you will first need an opened QuarkXPress document with which your script can interact. Rather than create a document manually, why not create a document with AppleScript?

Begin by launching Script Editor,

APPLICATIONS (folder) ▾ APPLESCRIPT, and creating a new script document. To create a new QuarkXPress document using default document settings, type the following code into your Script Editor document, and click RUN [COMMAND ⌘ R].

```
tell application "QuarkXPress"
    make new document
end tell
```

While this code may suffice in many cases, I usually prefer to be a little more cautious. For example, I may want an 8.5" X 11" document, but do not always know how QuarkXPress' default document settings are configured. To ensure that my document is the proper size, I can adjust the page height and page width, which are modifiable properties when I create the document. The following code will create a new 8.5" X 11" document, regardless of how QuarkXPress' default document settings are configured.

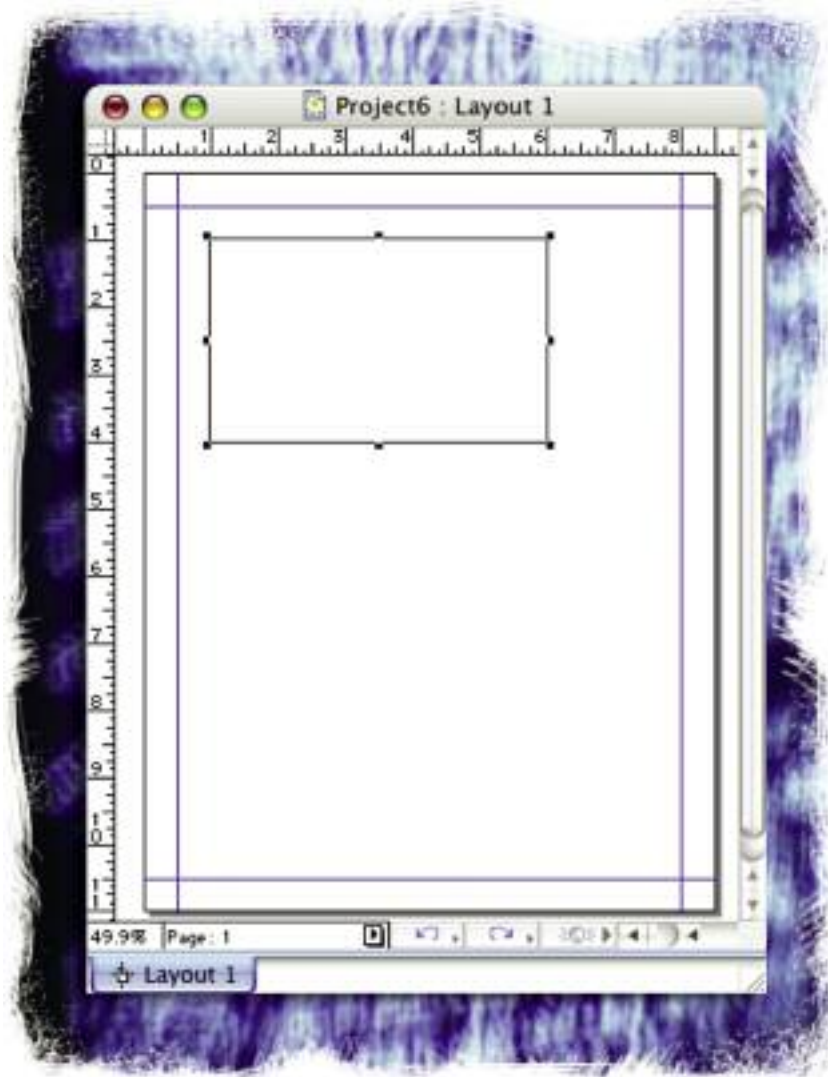
```
tell application "QuarkXPress"
    make new document with properties
    {page width:"8.5 \'", page height:"11 \'"
}
end tell
```

Note in the code above, that the page height and width are AppleScript strings, and that each of these strings contains the text \". This text symbolizes inches. The \ character tells AppleScript that the \" character is not the end of the string, but that it is actually part of the string.

There are a large number of other document properties, in addition to page width and page height, which are also accessible through scripting. A listing of document properties can be found in QuarkXPress' AppleScript dictionary. You can view this here: [QUARKXPRESS ▾ DOCUMENTS ▾ APPLE EVENTS SCRIPTING ▾ APPLE EVENTS SCRIPTING.PDF ▾ LAYOUT PROPERTIES, DATA TYPES, AND DESCRIPTIONS](#) or drag and drop the QuarkXPress icon onto the Script Editor icon to view the dictionary, as shown in figure 2 on page 14.

Building a Text Box

Now that you have a new document, you need a text box. This could be created manually, or it can be created using AppleScript. When creating a text box with AppleScript, you will first need to determine where



▲ fig 1.
A newly created text box

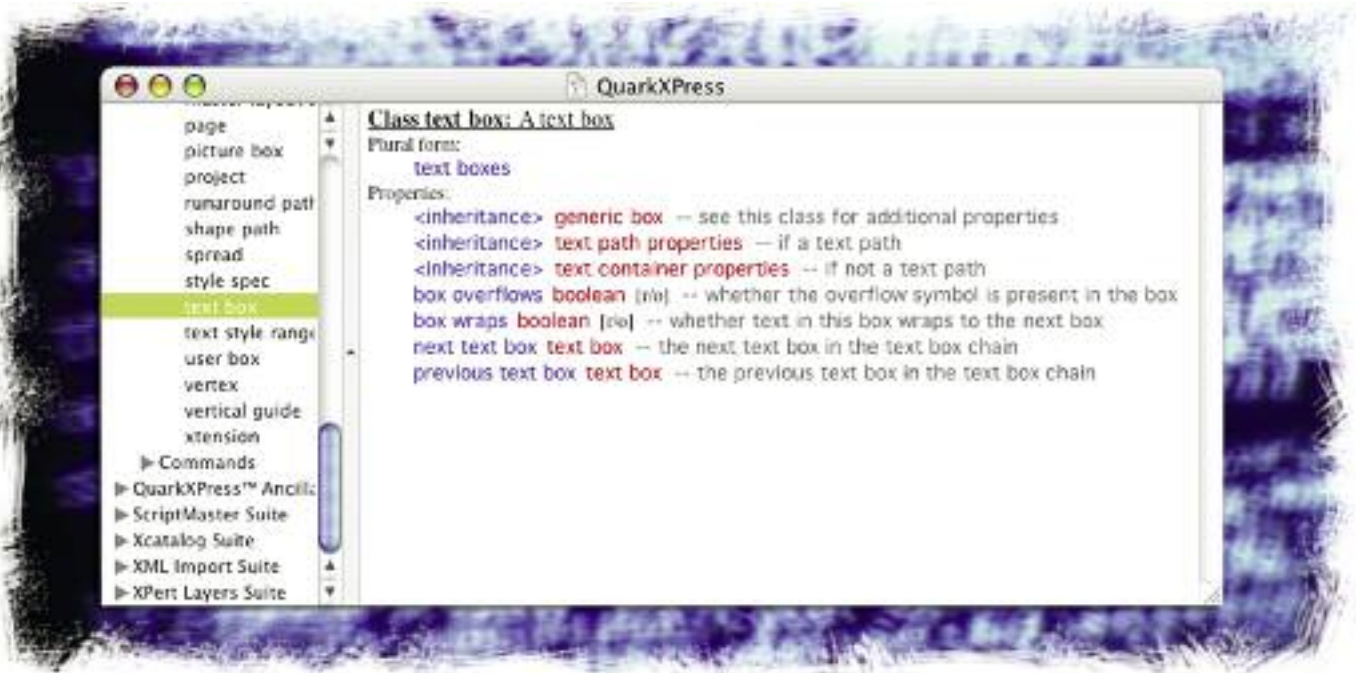
you would like the text box to appear on the page, and how large it should be. This information will then need to be passed to QuarkXPress by your AppleScript code, so that the text box appears as expected.

The size and position of a text box is indicated by specifying a value for the box's bounds property, when the text box is created. The bounds of a box must appear as a list of four unit values, which should appear in the following order enclosed in brackets as shown here:

```
{top of box, left of box, bottom of box,
right of box}
```

The following code will create a new text box in a specified position on page one of the front document (see figure 1).

```
tell application "QuarkXPress"
    tell page 1 of document 1
        make new text box at beginning with
        properties {bounds:{ "1 \'", "1 \'", "4 \'"
        \'", "6 \'"}}
    end tell
end tell
```



▲ fig. 2
QuarkXPress' text box
class

Like documents, text boxes possess a number of additional properties, which may be accessed and/or modified through scripting. Options can be found in the dictionary. Additional properties are inherited from other related classes, as indicated in figure 2. You may locate these other classes in the dictionary to view lists of any inherited properties.

Referring to a Text Box

When using AppleScript to refer to elements within QuarkXPress, including text boxes, picture boxes, pages, and documents, you need to be as specific as possible. The reason for this is because there may be multiple instances of the same type of element present with the page.

For example, you may have multiple documents opened, each containing multiple text boxes. To ensure that your script interacts with the proper elements, you need to indicate which document, page, text box, and so on should be addressed.

One way to address an element in QuarkXPress is by its index, or front-to-back order. For example, the following code will retrieve all of the properties of the front text box on the first page of the front document.

```
tell application "QuarkXPress"
  properties of text box 1 of page 1 of
  document 1
end tell
```

```
-> {class:text box, object
reference:text box 1 of page 1 of
document "Project12" of application
"QuarkXPress", layername:"Default",
anchored:false, background trap:default...
```

While referring to an element by its index may seem straightforward enough, it does have a drawback, especially when dealing with documents, text, and picture boxes. The main problem is that, if a script refers to an element by its index, if new elements are added, or if the front-to-back order changes, the script may behave in an undesirable manner. To make scripts more reliable, it is advisable to refer to documents and boxes by name, whenever possible. This will allow a script to accurately interact with a specific element, regardless of its position.

Document names are obvious, however, box names are a little more complicated. Unfortunately, QuarkXPress itself does not provide a visual way to assign names to boxes, at least, not without a third-party XTensions software module. ScriptMaster XT, a commercial XTensions module from Jintek (<http://www.jintek.com>), provides a visual way to manipulate box names in QuarkXPress. Without third-party XTensions, you must use AppleScript to assign names to boxes. As you will see by consulting QuarkXPress' AppleScript documentation, text and picture boxes possess a name property, which may be used to refer to boxes, in place of using their index. The following example code demonstrates how to assign a name to a text box:

```
tell application "QuarkXPress"
  tell text box 1 of page 1 of document 1
    set name to "myBox"
  end tell
end tell
```

Once you have assigned a name to a box, you can now refer to that box in your script by its name. If your script will be interacting with pre-existing template documents, it is advisable to take the time to assign names to any boxes in those templates with which the script will interact. Doing so will ensure that, even if new boxes are added, or boxes are moved around in the template, the script will still continue to function as desired.

The following code demonstrates how to retrieve the properties of a text box by the box's name, rather than its index.

```
tell application "QuarkXPress"
    properties of text box "myBox" of
page 1 of document 1
end tell
```

Adding Text to a Text Box

Now that we have discussed ways to create and refer to text boxes, let's discuss adding content to text boxes. This will be essential, if you want to pull information from a database, or from another application, and place it into your QuarkXPress document.

A text box may contain text as an element. To set the contents of a text box, set the box's text to a specified string. Note that doing so will replace any existing text in the box. The following example code will set the contents of a text box to the string *test paragraph 1*:

```
tell application "QuarkXPress"
    tell page 1 of document 1
        set text of text box 1 to
"test paragraph 1"
    end tell
end tell
```

In some cases, you may not want to replace the existing text of a box with new text. Instead, you may want to simply add to it. The following code demonstrates adding text to an existing text box, beginning at the end of the existing text. In this case, a carriage return is added, and then the string *test paragraph 2*.

```
tell application "QuarkXPress"
    tell page 1 of document 1
        make new text at end of text box 1
with properties {contents:return &
"test paragraph 2"}
    end tell
end tell
```

Assuming you have been running the code that you have written so far, at this point your text box should contain two paragraphs. The first should contain the text *test paragraph 1*, and the second should contain the text *test paragraph 2*.

Retrieving Text from a Text Box

There may be times at which you need to retrieve text from a text box, rather than placing text. This may be useful if you need to extract text from your QuarkXPress document and bring it into another application.

To retrieve the entire contents of a text box, you can use the following code:

```
tell application "QuarkXPress"
    tell page 1 of document 1
        return text of text box 1
    end tell
end tell
```

```
-> "test paragraph 1
test paragraph 2"
```

You may also be more specific when retrieving text from a box. For example, the following code will retrieve only the second paragraph of the contents of a text box.

```
tell application "QuarkXPress"
    tell page 1 of document 1
        return paragraph 2 of text box 1
    end tell
end tell
```

```
-> "test paragraph 2"
```

The following example code will retrieve only the first character of the second paragraph of the contents of a text box.

```
tell application "QuarkXPress"
    tell page 1 of document 1
        return character 1 of paragraph 2
of text box 1
    end tell
end tell
```

```
-> "t"
```

Many of the repetitive and time-consuming tasks that you perform within QuarkXPress on a daily

In Conclusion

We have really only scratched the surface of using AppleScript to interact with text. We will continue this discussion in future articles by discussing ways to modify text attributes, such as the font and point size, applying paragraph and character style sheets, detecting text overflow, and more. In the meantime, as suggested earlier, spend some time putting the practices discussed in this month's article to work for you. As you add to your scripting knowledge, begin putting the things you have learned together. Before you know it, you will be creating fully automated QuarkXPress-based workflows.

See you in the trenches. ☒