Apple® Events Scripting with QuarkXPress®

Quark Technical Support cannot provide assistance for Apple Events scripting.

If you need assistance writing or debugging a script, consult the documentation provided with your scripting application.

' 1986 93 by Quark, Inc. All rights reserved.

This manual may not be altered electronically.

Trademark Information

Quark and QuarkXPress are registered trademarks of Quark, Inc.

Apple and Macintosh are registered trademarks of Apple Computer, Inc. System 7 and AppleScript are trademarks of Apple Computer, Inc. UserLand Frontier and UserTalk are registered trademarks of UserLand Software, Inc. All other trademarks are the property of their respective owners.

PANTONE® Computer Video simulations used in this product may not match PANTONE-identified solid color standards. Use current pantone Color Reference Manuals for accurate color. *Pantone, Inc. s check-standard trademark for color. PANTONE Color Computer Graphics 'Pantone, Inc. 1986, 1988. Pantone, Inc. is the copyright owner of PANTONE Color Computer Graphics and Software which are licensed to Quark, Inc. to distribute for use only in combination with QuarkXPress. PANTONE Color Computer Graphics and Software shall not be copied onto another diskette or into memory unless as part of the execution of QuarkXPress.

Dainippon Ink and Chemicals Mfg. Co., Ltd. is the copyright owner of DATABASE PROCESS COLOR NOTE which is licensed to Quark, Inc. to distribute for use only in combination with QuarkXPress. DATABASE PROCESS COLOR NOTE shall not be copied onto another diskette or into memory unless as part of the execution of QuarkXPress.

Toyo Ink Mfg. Co., Ltd. is the copyright owner of TOYO INK COLOR FINDER SYSTEM AND SOFTWARE which is licensed to Quark, Inc. to distribute for use only in combination with QuarkXPress. TOYO INK COLOR FINDER SYSTEM AND SOFTWARE shall not be copied onto another diskette or into memory unless as part of the execution of QuarkXPress. TOYO INK COLOR FINDER SYSTEM AND SOFTWARE® Toyo Ink Mfg. Co., Ltd., 1991. COLOR FINDER is in the process of registration as the registered trademark of Toyo Ink Mfg. Co., Ltd.

Quark, Inc. does not warrant, guarantee, or make any representations regarding the use or the results of the use of any color system included in Quark products. Video simulations may not match published color standards. Refer to current materials of the specific color company (Pantone, Inc.; FOCOLTONE, Ltd.; TRUMATCH, Inc.; and other companies involved in the process of color reproduction) for accurate color samples.

Apple Disclaimer

The following disclaimer is required by Apple Computer, Inc. It applies only to Apple software. All other software is covered by Quark s limited warranty.

APPLE COMPUTER, INC. (APPLE) MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE APPLE SOFTWARE. APPLE DOES NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE APPLE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE APPLE SOFTWARE IS ASSUMED BY YOU. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME STATES. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

IN NO EVENT WILL APPLE, ITS DIRECTORS, OFFICERS, EMPLOYEES OR AGENTS BE LIABLE TO YOU FOR ANY CONSEQUENTIAL, INCIDENTAL OR INDIRECT DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, AND THE LIKE) ARISING OUT OF THE USE OR INABILITY TO USE THE APPLE SOFTWARE EVEN IF APPLE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE MAY NOT APPLY TO YOU. APPLE S LIABILITY TO YOU FOR ACTUAL DAMAGES FROM ANY CAUSE WHATSOEVER, AND REGARDLESS OF THE FORM OF THE ACTION (WHETHER IN CONTRACT, TORT (INCLUDING NEGLIGENCE), PRODUCT LIABILI-

Table of Contents

INTRODUCTION

Before You Begin v

CHAPTER 1

Apple Events Overview 1.01
Introduction to Apple Events 1.02
The Object Model 1.03
- Objects 1.03
- Object Hierarchy 1.03
- Object References 1.04
- Insertion Points in the Hierarchy 1.04
- Events 1.05
- Suites 1.05
- QuarkXPress Object Hierarchy 1.07

Script Writing Syntax 1.09
- Sample Syntax 1.09
- Spaces 1.09
- Quotation Marks 1.09

Optimizing the Performance of Scripts 1.10
- Process in QuarkXPress 1.10
- Specify Initial Properties when Performing a Create/Make Event 1.11
- Compile Scripts 1.12

Notes to Frontier Users 1.13
- Installing the QXP.Frontier Table 1.13
- Frontier Partition 1.13
- Sample Scripts for QuarkXPress 1.13

CHAPTER 2

Script Writing Sample 2.01
- Running the Document Construction Script 2.02
- About the Script Breakdown 2.11
- Breakdown of the Document Construction Script 2.12

CHAPTER 3

Reference Material for Apple Events 3.01
Using Chapter 3 ..... 3.02

Events Terminology 3.03
Object Reference Forms 3.03
Insertion Points in the Hierarchy 3.03
Events Supported by QuarkXPress 3.05
Required Suite 3.05
Core Suite 3.05
Miscellaneous Suite 3.08
QuarkXPress Suite 3.09
Data Coercion Chart 3.10
Using Chapter 4 ..... 3.12

CHAPTER 4
Reference Material for QuarkXPress Objects 4.01

CHAPTER 5
Glossary 5.01

Before You Begin

This document is for people who are ready to write scripts that communicate with QuarkXPress™ 3.2. To write scripts, you need the following:

- Macintosh System 7.0 or greater.
- A scripting application such as UserLand Frontier™ or AppleScript.
- The documentation (included with your scripting application) that teaches you the scripting language. You should familiarize yourself with the scripting language before attempting to write scripts for QuarkXPress.
- A basic understanding of programming (including concepts such as loops, conditional processing, if-then-else constructs, and variables) gained through writing HyperTalk scripts or macros, or working in programming languages such as C, BASIC, or Pascal.

If you are not ready to write scripts, read the Preview of Apple Events Scripting with QuarkXPress provided with QuarkXPress and experiment with the sample scripts. Then, contact UserLand Software or the Apple Programmer s and Developer s Association (APDA) for information about purchasing scripting software:

UserLand Software, Inc.	(415) 369-6600
APDA	(800) 282-2732 in the United States (800) 637-0029 in Canada (716) 871-6555 elsewhere in the world

If you are prepared to write scripts, this book provides background information on Apple Events, an analysis of a sample script, and the reference material you need to write scripts for QuarkXPress. If you are unfamiliar with Apple Events terminology, read the chapters sequentially and refer to the Glossary as necessary.

While writing scripts, you will spend most of your time referring to Chapter 4, the Reference Material for QuarkXPress Objects. It is included as a separate document and can be printed or referred to on screen.

Apple Events Overview

This chapter provides an overview of Apple Events scripting with QuarkXPress. First, it introduces the concepts and terminology involved, including: the Object Model, Objects, Object Hierarchy, Object References, Insertion Points in the Hierarchy, Events, Suites, and the QuarkXPress Object Hierarchy. You should understand these concepts and terms before you attempt to write scripts for QuarkXPress.

The second part of this chapter notes the primary syntax differences between UserLand Frontier and AppleScript. Information on how to optimize the performance of scripts you write for QuarkXPress is also provided. The end of this chapter includes notes specific to UserLand Frontier.

Introduction to Apple Events

Apple Events is a Macintosh System 7 feature that allows interapplication communication on a local system or across a network. Applications communicate through standard Apple Events messages that give instructions, respond to instructions, and send or receive data. The terminology for Apple Events messages is listed in the Apple Event Registry, which is maintained by Apple. All information in the Apple Event Registry that applies to QuarkXPress is included in this document.

Apple Events can be generated by scripts, a series of statements sent to applications asking them to do a series of tasks. The scripting language is provided by scripting software such as UserLand Frontier or AppleScript. Scripts combine the scripting language syntax with the standard Apple Events terminology defined in the Apple Event Registry.

As the sample scripts included with QuarkXPress illustrate, scripts are useful for automating repetitive tasks and for linking applications. Because scripting software is developed specifically for script writing, it is more powerful than scripting systems built into applications. It allows you to use one scripting language to write scripts for any application that supports Apple Events.

The Object Model

The Apple Events Object Model is a message protocol that allows Macintosh applications to communicate. Messages built according to the Object Model consist of events, objects and, potentially, data. Objects are distinct items in an application, such as a text box. Events are the actions objects are capable of performing.

If you re familiar with QuarkXPress, you understand that an application is composed of objects. QuarkXPress documents contain pages, pages contain text boxes, text boxes contain text, and text has various styles associated with it. Each object has specific capabilities. For example, a text box can be moved, resized, copied, and linked to other boxes. It has item specifications that can be changed (such as background color, number of columns, and text inset) and it can contain formatted text.

Objects

An object is a distinct item in an application that can be manipulated by an Apple Event. QuarkXPress users are familiar with objects such as documents, pages, text boxes, picture boxes, and lines. Objects are defined according to their class, properties, elements, and the events they can respond to.

Object Class
Objects that share specific characteristics are categorized into object classes. For example, all words belong to the word object class.

Properties
Properties are the characteristics shared by objects in the same object class. For example, the object class for words has properties such as color, font, size, and style.

Elements
Elements are the objects directly accessible from within another object. For example, a character is an element of a word.

Events
Events are the actions an object is capable of performing. Objects in the same object class respond to the same events. For example, the set event can be used to change the font of all words.

Object Hierarchy

The Apple Events Object Hierarchy is based on the simple concept of placing things inside other things. For example, a stack of nested bowls or a nested Russian doll has an object hierarchy. When you open the outer shell of a Russian doll, it contains a smaller doll. The smaller doll contains an even smaller doll. Eventually you reach the smallest doll that doesn t open. The smallest doll cannot contain anything and is at the end of the hierarchy.

An application s object hierarchy usually consists of objects such as windows, documents, boxes, and contents. A specific hierarchy in QuarkXPress might include a document that contains a page. The page contains a text box and the text box contains a story. The story contains paragraphs, and the paragraphs contain lines. The lines contain words and the words contain characters. Characters are at the end of the hierarchy because they can t contain anything.

Objects that enclose other objects are referred to as *containers*. Objects that are enclosed by other objects are referred to as *elements*. For example, a document is a container for a page; the page is an element of the document.

Object References

An Apple Event message must identify a specific object in an application to communicate. Objects are identified by a *reference*. For example, the message might reference the second text box on the first page. The reference first identifies the container (the page) enclosing the object (the text box) you're specifying. It then uses a reference form to separate a specific object (the second text box) from all possible objects in the container.

Reference Form

Objects in QuarkXPress can be referred to by five reference forms: index, name, range, relative position, or test. An example of how to use each reference form is included in Chapter 3 of this document.

Index
Used to identify an ordered element in a container with an integer number (for example, the first text box on a page).

Note: Windows, documents, text boxes, and picture boxes are numbered from front to back. The active window or document is always number [1]; the frontmost picture box or text box in the document is always number [1]. (The frontmost picture box or text box will change as users manipulate and create other boxes.) Pages are numbered according to their absolute page number.

Name
Used to identify objects that are named with a text string (for example, a document named "Ad Layout" by a user).

Range
Used to identify a range of objects (for example, text boxes three through five).

Relative Position
Used to identify objects that are before or after other objects (for example, the text box before the last text box on the page).

Test
Used to identify objects that meet certain conditions, (for example, the first text box with the background color red).

Insertion Points in the Hierarchy

An insertion point specifies where to place an object within the container hierarchy. An example of how to use each insertion point is included in Chapter 3 of this document.

Beginning
Used to insert or create an object at the beginning of the specified container (for example, to create a text box at the beginning of page one).

Ending

Used to insert or create an object at the end of the specified container (for example, to create a page at the end of a document).

After
Used to insert or create an object after a specified object (for example, to move the first page of a document after the fourth page).

Before
Used to insert or create an object before the specified object (for example, to move the last page of a document before the first page).

Replace
Used to replace the specified object with a new object (for example, to replace one text box with another text box).

Note: As you create and insert objects in the hierarchy, the index reference form for existing objects may change. For example, when you insert a text box before text box 1, text box 1 becomes text box 2.

Events

Events are the actions an object is capable of performing. In an English sentence, an event is comparable to a verb and an object is comparable to a noun. Events are used to tell objects what to do. QuarkXPress uses most of the standard events defined by Apple.

Suites

Groups of events and objects that relate to a similar purpose are arranged into Suites. For example, Apple has defined all standard text-related events and objects in the Text Suite. The Required Suite, Core Suite, and Miscellaneous Suite include the events and objects that most Macintosh applications support. In addition, events and objects specific to QuarkXPress are defined in the QuarkXPress Suite.

QuarkXPress supports the events and objects in the Required, Core, Miscellaneous, Text, and QuarkXPress Suites. An object can respond to events from a variety of suites, and events can apply to objects from a variety of suites. For example, objects in the QuarkXPress Suite are generally manipulated using events in the Core Suite.

Definitions and sample syntax for each event QuarkXPress supports are included in Chapter 3 of this document. Each object that QuarkXPress supports has three tables in Chapter 4: Events and Examples; Elements and Reference Forms; and Properties, Data Types, and Descriptions.

Required Suite

Events
The Required Suite events are sent to applications by the Finder: open application, open document, print document, and quit application.

Objects
The Required Suite does not define any objects.

Core Suite

Events
The Core Suite events are common to most applications: clone/duplicate, close, count, create/make, data size, delete, exists, get, get as, move, open, print, save, and set.

Objects
The Core Suite objects are common to most applications, for example: application, document, window, file, and text.

Miscellaneous Suite

Events
The Miscellaneous Suite events are related to the clipboard and other menu-related functions: copy, cut, do script, paste, redo, revert, show, undo, and uniform.

Objects
The Miscellaneous Suite does not define any objects.

Text Suite

Events
The Text Suite does not define any events.

Objects
The Text Suite objects are the text-related objects common to most applications, for example: character, line, paragraph, story, text and word.

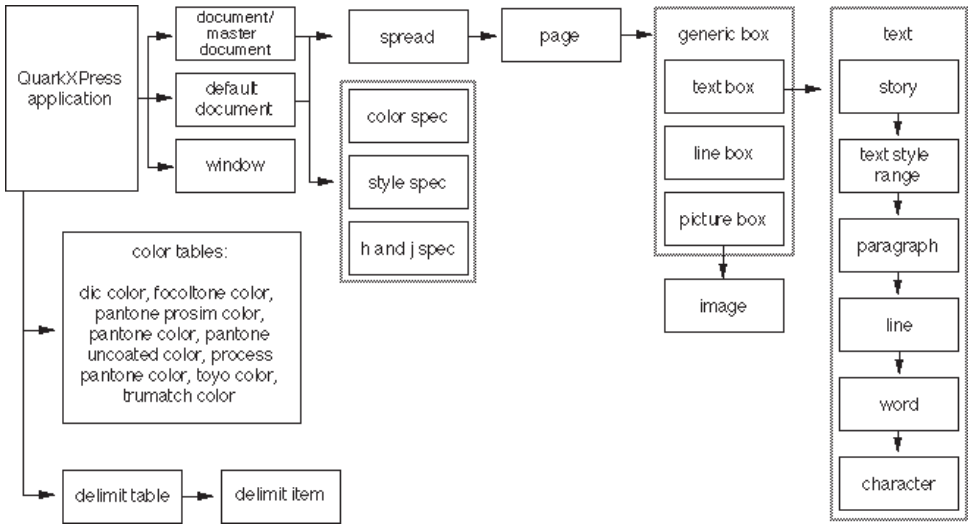
QuarkXPress Suite

Events
The QuarkXPress Suite includes one event related to redraw: do updates.

Objects
The QuarkXPress Suite objects are specific to the application, for example: horizontal guide, image, line box, master document, page, picture box, spread, style spec, text box, text style range, and vertical guide.

QuarkXPress Object Hierarchy

When you create a document in QuarkXPress, you are working within the QuarkXPress object hierarchy. From the application level, you set defaults and create documents. At the document level you create pages, spreads, style sheets, colors, and H&Js. You then add picture boxes and pictures, text boxes and text, and lines at the page level. The QuarkXPress object hierarchy is structured according to the graphic shown below.



Unfamiliar Objects

The following objects are not familiar QuarkXPress terms. Their properties are defined fully in the object tables in Chapter 4.

color tables
The color models that QuarkXPress supports.

default document
The object that contains all default document settings including colors, style sheets, H&Js, document settings specified in the New Document dialog box, and all preferences.

delimit item
Each character has an associated delimit item that QuarkXPress uses to determine where a word starts and ends.

delimit table
A container for 256 delimit items.

generic box
Any type of box on a page. Use the generic box if you want to change the properties of a box in a specific location, regardless of whether it is a picture box, text box, or line box.

master document
A container for master pages. The master document allows access to master pages and master page objects.

text style range
A range of text with a single set of styles specified. Use text style range for functions such as Find/Change.

Object Limitations

The current implementation of Apple Events in QuarkXPress does not support the following: libraries, groups, auxiliary dictionaries, hyphenation exceptions, box creation defaults, color pair trapping, blends, anchored boxes, and page sections.

Most XTensions do not currently support Apple Events. This includes the EfiColor XTension, Quark Freebies, and QuarkPrint, as well as those developed by independent XTension developers.

Future versions of QuarkXPress may support the objects listed above. Future versions of XTensions by Quark and independent XTension developers may support Apple Events. This document will be updated and distributed with new versions of QuarkXPress as necessary.

Script Writing Syntax

To write a script, first you need to learn the scripting language of the scripting application you purchase. Scripts combine events, objects, properties, and data in the order specified by the scripting language. UserTalk, UserLand Frontier s language, is similar to traditional programming languages such as C or Pascal. AppleScript is more like the English language.

Sample Syntax

UserTalk:	Event	Object	Property	Data
	Set	(textBox[1].	color.)	colorSpec["red"]
	Set	(pictureBox[5].	runaround.	manual)
AppleScript:	Event	Property	Object	Data
	Set	the color of	text box 1 to	color spec "red"
	Set	the runaround of	picture box 5 to	manual

Spaces

In UserTalk, the names of all objects and properties that contain spaces must be used without spaces. For example, you would write text box as textBox and auto text box as autoTextBox. Although UserLand Frontier is not case sensitive, it may be easier to identify objects if you capitalize the first letter of a new word.

In AppleScript, you can use the names of all objects and properties as shown in Chapter 4 of this document. AppleScript is not case sensitive.

Quotation Marks

In both UserTalk and AppleScript, you should enclose data in straight quotes and use typographer s quotation marks as foot and inch marks. For example, to specify six inches use "6 " rather than 6" or "6" ".

Optimizing the Performance of Scripts

Process in QuarkXPress

Allow QuarkXPress to do all the calculations using its own built-in functions, and minimize context switches between QuarkXPress and the scripting application.

For example, to set the color of all generic boxes to red:

Do

UserTalk:
```
set(genericBox[all].color,"Red")
```

AppleScript:
```
set the color of every generic box to "Red"
```

Don t

UserTalk:
```
numberOfBoxes = count(it,genericBox)
  for i = 1 to numberOfBoxes
    set(genericBox[i].color,"Red")
```

AppleScript:
```
set numberOfBoxes to count of generic boxes
  repeat with i from 1 to numberOfBoxes
    set color of generic box i to "Red"
```

Or, to change the color of all green generic boxes to red:

Do

UserTalk:
```
set(genericBox[color.name=="Green"].color,"Red")
```

AppleScript:
```
set every generic box whose name of color equals "Green"
to "Red"
```

Don t

UserTalk:
```
numberOfBoxes = count(it,genericBox)
  for i = 1 to numberOfBoxes
    if(get(genericBox[i].color.name)=="Green")then
    set(genericBox[i].color,"Red")
```

AppleScript:
```
set numberOfBoxes to count of generic boxes
  repeat with i from 1 to numberOfBoxes
    if name of color of generic box i equals "Green" then set color of
    generic box i to "Red"
```

Specify Initial Properties when Performing a Create/Make Event

Use the create/make event to specify initial properties rather than using subsequent set events. You can also set multiple properties. In UserTalk, you do this by making a list of properties and using it as an argument in the create statement. In AppleScript, you specify the initial properties within the create statement.

Do

UserTalk:

```
paramlist = 0
putAppleListItem(90,rotation,@paramlist)
putAppleListItem("Cyan",color,@paramlist)
putAppleListItem(80,shade,@paramlist)

create(pictureBox,0,paramlist,beginningof(page[last]))
```

AppleScript:

```
make picture box at beginning of last page with properties
{rotation:90,color:"Cyan",shade:80}
```

Or, set multiple properties:

UserTalk:

```
paramlist = 0
putAppleListItem(90,rotation,@paramlist)
putAppleListItem("Cyan",color,@paramlist)
putAppleListItem(80,shade,@paramlist)
set(pictureBox[1].properties,paramlist)
```

AppleScript:

```
tell document 1
        make picture box at beginning
        set properties of picture box 1 to
        {rotation:90;color:"Cyan";shade:80}
end tell
```

Don t

UserTalk:

```
with document[1].page[1]
        with pictureBox[1]
                set(rotation,90)
                set(color,"Cyan")
                set(shade,80)
```

AppleScript:

```
tell page 1 of document 1
        tell picture box 1
                set rotation to "90"
                set color to "Cyan"
                set shade to "80"
        end tell
end tell
```

Compile Scripts

Using the "do script" event sends a compiled script directly to QuarkXPress where it is processed completely within the application before anything appears on screen. AppleScript currently supports the "do script" event. Although UserLand Frontier 2.0 does not support the "do script" event in this manner, future versions will.

Add the following lines to AppleScripts, to have them automatically compile and execute within QuarkXPress:

```
script theRealScript
        tell application "QuarkXPress"
                --the script goes here
        end tell
end script

tell application "QuarkXPress"
        do script {theRealScript}
end tell
```

Notes to Frontier Users

Frontier Partition

To run scripts written for QuarkXPress from UserLand Frontier, you must set your UserLand Frontier minimum memory partition to 2MB.

Installing the QXP.Frontier Table

A file that installs the QuarkXPress verbs and constants into UserLand Frontier is on the Apple Events Scripting disk in the For Advanced Scripting folder. Double-click on the QXP.Frontier icon while UserLand Frontier is running. The proper verbs and constants will be installed in your root in the QuarkXPress table. (AppleScript can read the verbs and constants defined in a resource within QuarkXPress.)

Sample Scripts for QuarkXPress

To locate the sample Frontier scripts included with QuarkXPress, click on the Object DB button in UserLand Frontier. This opens the Frontier.root which contains tables, scripts, and other items. Double-click on the triangles next to a table name to open it. Open the system table. From the system table, open the verbs table. Continue in this fashion for the apps, table, then the QXP table. Open the examples table from the QXP table. The sample scripts are located in the examples table.

The path through the tables is: root:system:verbs:apps:QXP:examples. To look at the table of QuarkXPress objects and events follow this path: root:system:verbs:apps:QXP:defs.

Script Writing Sample

To illustrate how scripts interact with QuarkXPress objects, we have dissected a script that uses a representative sample of the objects that QuarkXPress supports. Creating objects, specifying initial properties, and changing properties are discussed. In addition, the various aspects of the scripting environment, including suites, events, the object model, the object hierarchy, and object references are discussed in the context of the objects in this script. You can apply similar syntactical constructs to other objects in other scripts.

The sample script, Document Construction, illustrates how Apple Events control objects within QuarkXPress. The script sets guides, creates text and picture boxes, builds a color, imports text and images, then specifies the properties of these objects and their elements to produce a final layout.

The sample script was written in UserLand Frontier s scripting language, UserTalk. Although this discussion occasionally refers to specific commands in UserLand Frontier, the concepts outlined apply to AppleScript as well. The syntax for the Document Construction script is shown in both UserTalk and AppleScript.

This section assumes you have purchased a scripting application. You may want to read this in front of your computer with QuarkXPress and your scripting application active. The Reference Material you need to write scripts for QuarkXPress follows in Chapter 3 and Chapter 4 of this document.

Running the Document Construction Script

Before reading this section, run the Document Construction script to familiarize yourself with its actions. The script requires text and image files in the Sample Documents folder (Apple Events Scripting disk: Apple Events Preview folder: Sample Documents folder).

To locate the script in UserLand Frontier, click on the Object DB button. This opens the Frontier.root, which contains tables, scripts, and other items. Double-click on the triangles next to a table name to open it. Open the tables following this path: root.system.verbs.apps.
QXP.examples. The Document Construction script is located in the examples tables. The UserTalk syntax for this script is shown below.

The AppleScript version of the Document Construction script is included in the For Advanced Scripting folder. You can open it in AppleScript and run it. The AppleScript syntax for this script starts on page 2.07.

Frontier Syntax: Document Construction Script

```
on ConstructDocument()
   with QXP,QXP.defs,objectModel
      «
      local(oldHeight,oldWidth,oldAutoTextBox,oldGuidesShowing,
      xDocMeasure,yDocMeasure)
      local(thepath,howmany,lastFieldNumber,filepath,count)
      local(frect,bkgrndclr)
      thepath = "Hard Drive:Apple Events Preview:Sample Documents"
      «
      sys.bringAppToFront(file.fileFromPath(QXP.appInfo.path))
      with defaultdocument[1]
         oldHeight = get(pageHeight)
         oldWidth = get(pageWidth)
         oldAutoTextBox = get(autoTextBox)
         oldGuidesShowing = get(guidesShowing)
         xDocMeasure = get(horizontalMeasure)
         yDocMeasure = get(verticalMeasure)
         set(pageHeight,"30 cm")
         set(pageWidth,"34 cm")
         set(autoTextBox,false)
         set(guidesShowing,true)
         set(guidesInFront,true)
         set(horizontalMeasure,centimeters)
         set(verticalMeasure,centimeters)
      create(document,0,0,beginningof(it))
      «
      «SET THE VIEW SCALE TO FIT IN WINDOW
      with document[1]
         set(viewscale,fitpageinwindow)
      «
      «CREATE MOUNTAIN PURPLE COLOR.
      with document[1]
         local(theColor,newColorSpec)
         newColorSpec = create(colorspec,0,0,beginningof(it))
         thecolor = textCMYKcolor(45874.5,45874.5,0,0)
```

```
    set(newColorSpec.colortype,cmyktype)
    set(newColorSpec.cmykcolor,thecolor)
    try
       set(newColorSpec.name,"Mountain Purple")
    else
       delete(newColorSpec)
«
«CREATE GUIDES TO LAYOUT ELEMENTS ON THE PAGE
with document[1]
    paramlist = 0
    putAppleListItem("4.218 cm",position,@paramlist)
    create(horizontalGuide,0,paramlist,beginningof(page[1]))
    paramlist = 0
    putAppleListItem("8.477 cm",position,@paramlist)
    create(horizontalGuide,0,paramlist,endof(page[1]))
    paramlist = 0
    putAppleListItem("27.152 cm",position,@paramlist)
    create(horizontalGuide,0,paramlist,endof(page[1]))
    paramlist = 0
    putAppleListItem("2 cm",position,@paramlist)
    create(verticalGuide,0,paramlist,endof(page[1]))
    paramlist = 0
    putAppleListItem("4.962 cm",position,@paramlist)
    create(verticalGuide,0,paramlist,endof(page[1]))
    paramlist = 0
    putAppleListItem("18.742 cm",position,@paramlist)
    create(verticalGuide,0,paramlist,endof(page[1]))
    paramlist = 0
    putAppleListItem("32 cm",position,@paramlist)
    create(verticalGuide,0,paramlist,endof(page[1]))
«
«CREATE FIRST TEXT BOX.
with document[1].page[1]
    paramlist = 0
    frect = textrect("2 cm","5 cm","8 cm","19 cm")
    putAppleListItem(frect,bounds,@paramlist)
    create(textbox,0,paramlist,beginningof(it))
    with textbox[1]
       set(VerticalJustification,bottomJustified)
       set(color,"none")
    with textbox[1].story[1]
       set(it,"Biking Gear")
       set(font,"Times")
       set(word[1].size,30)
       set(word[1].textstyle,allcaps)
       set(word[1].baseshift,60)
       set(word[1].track,50)
       set(word[1].character[last].kern,-100)
       set(word[2].size,120)
       set(word[2].color,"Mountain Purple")
       set(word[2].textstyle,Italic)
       set(word[2].character[1].kern,-5)
       set(word[2].character[2].kern,-5)
       «
«CREATE SECOND TEXT BOX.
```

```
with document[1].page[1]
   paramlist = 0
   frect = textrect("8.5 cm","5 cm","29.959 cm","18.472 cm")
   putAppleListItem(frect,bounds,@paramlist)
   create(textbox,0,paramlist,endof(it))
   with textbox[2]
      try
         set(story[1],alias(thepath+"ASB Text"))
      else
         if (file.getFileDialog("Open the file named 'ASB Text'.",@filepath,'TEXT'))
            set(story[1],alias(filepath))
            «
            «COUNT THE NUMBER OF SEPARATE STRINGS SEPARATED BY ':'.
            lastFieldNumber = string.countFields(filepath,":")
            «
            «RESET THE PATH TO THE TEXT.
            thepath = ""
            for count = 1 to lastFieldNumber-1
               «
               «RECONSTRUCT THE PATH.
               thepath =thepath+string.nthField(filepath,":",
               count)+":"
      set(story[1].size,11)
      set(story[1].leading,43)
      set(story[1].justification,full)
      set(story[1].font,"Times")
   with textbox[2].story[1].paragraph[1]
      set(dropcapchars,1)
      set(dropcaplines,3)
      set(word[1].character[1].color,"Mountain Purple")
   with textbox[2].story[1].paragraph[last]
      set(word[1].character[1].color,"Mountain Purple")
      set(word[1].character[1].size,28)
      set(ruleabove.ruleon,true)
      set(ruleabove.textlength,false)
      set(ruleabove.width,0.5)
      set(ruleabove.position,"1 cm")
      set(ruleabove.color,"Cyan")
      set(ruleabove.shade,100)
«
«CREATE FIRST PICTURE BOX.
with document[1].page[1]
   paramlist = 0
   frect = textrect("10.386 cm","20.758 cm","27.636 cm","33.508 cm")
   bkgrndclr = "None"
   putAppleListItem(frect,bounds,@paramlist)
   putAppleListItem(bkgrndclr,color,@paramlist)
   create(picturebox,0,paramlist,beginningof(it))
   with picturebox[1]
      set(rotation,-25)
      try
         set(image[1],alias(thepath+"Shirts.tiff"))
      else
         if (file.getFileDialog("Open the image named 'Shirts.tiff'.",@filepath,'TIFF'))
            set(image[1],alias(filepath))
```

```
      with image[1]
         set(scale,textpoint("115","115"))
«
«CREATE SECOND PICTURE BOX.
with document[1].page[1]
   paramlist = 0
   frect = textrect("8.471 cm","2 cm","9.971 cm","3.5 cm")
   bkgrndclr = "None"
   putAppleListItem(frect,bounds,@paramlist)
   putAppleListItem(bkgrndclr,color,@paramlist)
   create(picturebox,0,paramlist,endof(it))
   with picturebox[2]
      try
         set(image[1],alias(thepath+"Glove.tiff"))
      else
         if (file.getFileDialog("Open the image named 'Glove.tiff'.",@filepath,'TIFF'))
            set(image[1],alias(filepath))
      set(image[1].bounds,exactfit)
   clone(picturebox[2],after(picturebox[2]))
   with picturebox[3]
      set(bounds,textrect("12.471 cm","2 cm","13.971 cm","3.5 cm"))
   clone(picturebox[2],after(picturebox[3]))
   with picturebox[4]
      set(bounds,textrect("16.471 cm","2 cm","17.971 cm","3.5 cm"))
   clone(picturebox[2],after(picturebox[4]))
   with picturebox[5]
      set(bounds,textrect("20.471 cm","2 cm","21.971 cm","3.5 cm"))
«
«CREATE THIRD PICTURE BOX.
with document[1].page[1]
   create(picturebox,0,0,endof(it))
   with picturebox[6]
      set(bounds,textrect("6.875 cm","18.425 cm","12.729 cm","26.4 cm"))
      set(color,"None")
      try
         set(image[1],alias(thepath+"Helmet.tiff"))
      else
         if (file.getFileDialog("Open the image named
         'Helmet.tiff'.",@filepath,'EPSF'))
            set(image[1],alias(filepath))
      set(image[1].scale,textpoint("70","70"))
      set(image[1].offset,textpoint("0.557 cm","1.254 cm"))
«
«CREATE LINES.
with document[1].page[1]
   create(linebox,0,0,beginningof(it))
   with linebox[1]
      set(leftpoint,textpoint("0 cm","21.406 cm"))
      set(rightpoint,textpoint("8 cm","21.406 cm"))
      paramlist = 0
      putAppleListItem("Magenta",color,@paramlist)
      putAppleListItem(3,LineWidth,@paramlist)
      putAppleListItem(dottedline,LineStyle,@paramlist)
      set(Properties,paramlist)
with document[1].page[1]
```

```
        create(linebox,0,0,endof(it))
        with linebox[2]
            paramlist = 0
            set(leftpoint,textpoint("8 cm","2 cm"))
            set(rightpoint,textpoint("8 cm","32 cm"))
            set(linewidth,0.5 )
    «
    set(document[1].guidesShowing,false)
    saveDocument(document[1],thePath+"Constructed Document")
    with defaultdocument[1]
        set(pageHeight,oldHeight)
        set(pageWidth,oldWidth)
        set(autoTextBox,oldAutoTextBox)
        set(guidesShowing,oldGuidesShowing)
        set(horizontalMeasure,xDocMeasure)
        set(verticalMeasure,yDocMeasure)
ConstructDocument()
```

AppleScript Syntax: Document Construction Script

```
tell application "QXP AE test"
   activate
   set thepath to "Hard Drive:Apple Events Preview:Sample Documents"

   tell default document 1
      set oldHeight to page height
      set oldWidth to page width
      set oldAutoTextBox to automatic text box
      set oldGuidesShowing to guides showing
      set xDocMeasure to horizontal measure
      set yDocMeasure to vertical measure
      set page height to "30 cm"
      set page width to "34 cm"
      set automatic text box to false
      set guides showing to true
      set horizontal measure to centimeters
      set vertical measure to centimeters
   end tell
   make document at beginning

   tell document 1
      set view scale to fit page in window
   end tell

   --CREATE MOUNTAIN PURPLE COLOR.
   tell document 1
      set newcolorspec to (make color spec at beginning)
      set color type of newcolorspec to CMYK type
      set CMYK color of newcolorspec to {4.58745E+4, 4.58745E+4, 0, 0}
      set name of newcolorspec to "Mountain Purple"
   end tell

   --CREATE GUIDES TO LAYOUT ELEMENTS ON THE PAGE
   tell page 1 of document 1
      make horizontal guide at beginning with properties {position:"4.218 cm"}
      make horizontal guide at end with properties {position:"8.447 cm"}
      make horizontal guide at beginning with properties {position:
      "27.152 cm"}
      make vertical guide at end with properties {position:"2 cm"}
      make vertical guide at end with properties {position:"4.962 cm"}
      make vertical guide at end with properties {position:"18.742 cm"}
      make vertical guide at end with properties {position:"32 cm"}
   end tell

   --CREATE FIRST TEXT BOX.
   tell page 1 of document 1
      make text box at beginning with properties {bounds:{"2 cm", "5 cm", "8 cm", "19
      cm"}}
      tell text box 1
         set vertical justification to bottom justified
         set color to "none"
      end tell
   end tell
```

```
tell story 1 of text box 1 of page 1 of document 1
   set contents of it to "Biking Gear"
   set font to "Times"
   set size of word 1 to 30
   set style of word 1 to all caps
   set base shift of word 1 to 60
   set track of word 1 to 50
   set kern of last character of word 1 to -100
   set size of word 2 to 120
   set color of word 2 to "Mountain Purple"
   set style of word 2 to italic
   set kern of character 1 of word 2 to -5
   set kern of character 2 of word 2 to -5
end tell

--CREATE SECOND TEXT BOX.
tell page 1 of document 1
   make text box at end with properties {bounds:{"8.5 cm", "5 cm", "29.959 cm",
   "18.472 cm"}}
   tell text box 2
      try
         set story 1 to alias (thepath & "ASB Text")
      on error
         set story 1 to (choose file with prompt "Please select the file \"" & "ASB Text"
         & "\"")
      end try
      set size of story 1 to 11
      set leading of story 1 to 43
      set justification of story 1 to fully justified
      set font of story 1 to "Times"
   end tell
   tell paragraph 1 of story 1 of text box 2
      set drop cap characters to 1
      set drop cap lines to 3
      set color of character 1 of word 1 to "Mountain Purple"
   end tell
   tell last paragraph of story 1 of text box 2
      set rule on of rule above to true
      set text length of rule above to true
      set width of rule above to 0.5
      set position of rule above to "1 cm"
      set color of rule above to "Cyan"
      set shade of rule above to 100
   end tell
end tell

--CREATE FIRST PICTURE BOX.
tell page 1 of document 1
   make picture box at beginning with properties {bounds:{"10.386 cm", "20.758 cm",
   "27.636 cm", "33.508 cm"}, color:"none"}
   tell picture box 1
      set rotation to -25
      try
         set image 1 to alias (thepath & "Shirts.TIFF")
```

```
      on error
         set image 1 to (choose file with prompt "Please select the file \"" &
         "Shirts.TIFF" & "\"")
      end try
      tell image 1
         set scale to {"115", "115"}
      end tell
   end tell
end tell

--CREATE SECOND PICTURE BOX.
tell page 1 of document 1
   make picture box at end with properties {bounds:{"8.471 cm", "2 cm", "9.971 cm",
   "3.5 cm"}, color:"none"}
   tell picture box 2
      try
         set image 1 to alias (thepath & "Glove.TIFF")
      on error
         set image 1 to (choose file with prompt "Please select the file \"" &
         "Glove.TIFF" & "\"")
      end try
      set bounds of image 1 to exact fit
   end tell
   duplicate picture box 2 to after picture box 2
   tell picture box 3
      set bounds to {"12.471 cm", "2 cm", "13.971 cm", "3.5 cm"}
   end tell
   duplicate picture box 2 to after picture box 3
   tell picture box 4
      set bounds to {"16.471 cm", "2 cm", "17.971 cm", "3.5 cm"}
   end tell
   duplicate picture box 2 to after picture box 4
   tell picture box 5
      set bounds to {"20.471 cm", "2 cm", "21.971 cm", "3.5 cm"}
   end tell
end tell

--CREATE THIRD PICTURE BOX.
tell page 1 of document 1
   make picture box at end with properties {bounds:{"6.875 cm", "18.425 cm", "12.729
   cm", "26.4 cm"}, color:"none"}
   tell picture box 6
      try
         set image 1 to alias (thepath & "Helmet.TIFF")
      on error
         set image 1 to (choose file with prompt "Please select the file \"" &
         "Helmet.TIFF" & "\"")
      end try
      tell image 1
         set scale to {"70", "70"}
         set offset to {"0.557 cm", "1.254 cm"}
      end tell
   end tell
end tell
```

```
    --CREATE LINES
    tell page 1 of document 1
       make line box at beginning with properties {left point:{"0 cm", "21.406 cm"}, right
       point:{"8 cm", "21.406"}}
       tell line box 1
          set color to "Magenta"
          set line width to 3
          set line style to dotted line
       end tell
       make line box at end
       tell line box 2
          set left point to {"8 cm", "2 cm"}
          set right point to {"8 cm", "32 cm"}
          set line width to 0.5
       end tell
    end tell

    set guides showing of document 1 to false
    save document 1 in (thepath & "Constructed Document")

    tell default document 1
       set page height to oldHeight
       set page width to oldWidth
       set automatic text box to oldAutoTextBox
       set guides showing to oldGuidesShowing
       set horizontal measure to xDocMeasure
       set vertical measure to yDocMeasure
    end tell
end tell
```

About the Script Breakdown

This section first discusses how to direct a Frontier script to QuarkXPress. The script is then divided into the steps a user would perform when constructing a document. The steps include creating a new document, creating a text box, importing text, formatting the text, etc. The UserTalk syntax is then displayed in Courier font. Following the syntax is a concept line that translates the scripting language into actions in QuarkXPress. The events, objects, and properties set in the script are then analyzed line by line. The script breakdown follows this format:

A step in the document construction process

```
UserTalk code
```

Concepts Illustrated: The code above is described in terms of actions in QuarkXPress.

Each event, object, and/or property is discussed line by line.

Breakdown of the Document Construction Script

Locate the terminology for QuarkXPress objects and events

```
with QXP,QXP.defs,objectModel
```

Concepts Illustrated: This statement specifies the location of QuarkXPress terminology in UserLand Frontier.

Always put this with statement around all Frontier code written for QuarkXPress. In UserLand Frontier, QXP is a table and QXP.defs is a subtable of QXP. The objectModel indicates that the script uses Object Model syntax. To look at the table, click on the Object DB button in UserLand Frontier. This opens the Frontier.root, which contains tables, scripts, and other items. Double-click on the triangles next to a table name to open it. Open the system table. From the system table, open the verbs table. Continue in this fashion for the apps, table, then the QXP table. Open the defs table from the QXP table.

In the remainder of this section, the following format will be used to reference the location of items in UserLand Frontier: root.system.verbs.apps.QXP.examples.

Declare the variables

```
local(oldHeight,oldWidth,oldAutoTextBox,oldGuidesShowing,
xDocMeasure,yDocMeasure)
local(thepath,howmany,lastFieldNumber,filepath,count)
local(frect,bkgrndclr)
```

Concepts Illustrated: This statement declares local variables for the script.

Although it is not essential to declare local variables, it makes scripts much safer. Making variables local ensures that UserLand Frontier or QuarkXPress table entries will not be altered inadvertently if they have the same name as a variable used in a script.

Establish the path

```
thepath = "Hard Drive:Apple Events Preview:Sample Documents"
```

Concepts Illustrated: This statement establishes a path for sample text and image files.

This statement gives the variable thepath a string value that is the path to the location of the text and image files.

In this example, Hard Drive is the name of the local hard drive. The Apple Events Preview folder contains the Sample Documents folder, which contains the files that will be imported into the new document. If you want to access something on the desktop, reference Desktop Folder in the path name after the name of the local hard drive.

Activate QuarkXPress

```
sys.bringAppToFront(file.fileFromPath(QXP.appInfo.path))
```

Concepts Illustrated: This statement is similar to choosing QuarkXPress from the Finder menu.

This statement ensures that QuarkXPress is the active application. Rather than including the actual code, the script calls another Frontier script, bringAppToFront.

The bringAppToFront script is located at: root.system.verbs.builtins.sys. Another script, fileFromPath, is located at root.system.verbs.builtins.file. The path provides the information for UserLand Frontier to locate QuarkXPress when executing the script. Open these scripts and look at them if you wish.

Save current document default specifications

```
with defaultDocument[1]
    oldHeight = get(pageHeight)
    oldWidth = get(pageWidth)
    oldAutoTextBox = get(autoTextBox)
    oldGuidesShowing = get(guidesShowing)
    xDocMeasure = get(horizontalMeasure)
    yDocMeasure = get(verticalMeasure)
```

Concepts Illustrated: The get statements above determine your current document default specifications. When the document construction is complete, the script replaces the new default specifications with your original specifications.

The with statement references the current defaultDocument by index [1]. (The default
Document is the object that contains all default document settings including colors, style sheets, H&Js, document settings specified in the New Document dialog box, and all
preferences.)

The results of the get statements are assigned to local variables in the following six lines. The get statements determine the current page size, automatic text box setting, whether the guides are showing, and the default measurement system.

Set default specifications for a new document

```
set(pageHeight,"30 cm")
set(pageWidth,"34 cm")
set(autoTextBox,false)
set(guidesShowing,true)
set(guidesInFront,true)
set(horizontalMeasure,centimeters)
set(verticalMeasure,centimeters)
```

Concepts Illustrated: The first three set statements are similar to setting default specifications in the New Document dialog box. The next set statement is similar to choosing Show Guides from the View menu. The last three statements are settings in the General Preferences dialog box.

The first two set events specify the pageHeight and pageWidth properties.

The third set event determines whether the document will have an autoTextBox depending on the boolean operator. If the boolean operator is false, the document will not have an automatic text box. If the boolean operator is true, it will.

The fourth set event determines whether the document will have guidesShowing depending on the boolean operator. If the boolean operator is true, all guides will show. If the boolean operator is false, all guides will be hidden.

The fifth set event determines whether the guides will be displayed inFront of the page elements. The true boolean operator indicates that the guides will display in front.

The last two set events specify the default horizontal and vertical measurement system as centimeters.

Create a new document with default specifications

```
create(document,0,0,beginningOf(it))
```

Concepts Illustrated: This create event is similar to clicking OK in the New Document dialog box.

The create event is used to make the object specified by the four parameters.

The first parameter, document, refers to the object that will be created.

The second parameter 0 specifies the initial data or initial contents of the object to be created. The 0 means no initial contents. Specifying initial contents or data usually only applies to text or image containers.

The third parameter, 0, specifies the initial properties of the created object. The 0 means the document will have no initial properties set and will be created according to information in the defaultDocument object. To set properties using the third parameter, you must put the desired properties in a list and pass this list as the third parameter (see Create Guides page 2.16).

The fourth parameter, beginningOf(it) specifies where you want the object to be created. (The it refers to the last object referenced in the with statement.) You can create an object at any insertion point: beginning, ending, after, before, or replace.

Set the view scale

```
with document[1]
    set(viewScale,fitPageInWindow)
```

Concepts Illustrated: The lines above are similar to choosing Fit in Window from the View menu for the active document.

The with statement references the active document by index [1].

The set event changes the viewScale property to the data fitPageInWindow. The viewScale property can be a percentage or specific view. For example, to specify 100% view, use 100 for the second parameter. To specify thumbnails, use thumbnails for the second parameter.

Create a new color

```
with document[1]
    local(theColor,newColorSpec)
    newColorSpec = create(colorSpec,0,0,beginningOf(it))
    theColor = textCMYKcolor(45874.5,45874.5,0,0)
```

Concepts Illustrated: This statement is similar to clicking New in the Colors dialog box and then specifying CMYK values in the Edit Color dialog box.

The with statement references the active document by index [1].

The second statement declares local variables.

The create event makes a new color at the beginningOf the document s color list and assigns the result of the create event (the newly created color) to newColorSpec.

The following line finds the values specified in the parameters by calling the textCMYKcolor script located at root:system.verbs.apps.QXP.defs in UserLand Frontier. The script passes the numerical representations of the four process colors (cyan, magenta, yellow, and black), respectively. All CMYK, RGB, and HSB colors are based on the number 65535.

The four parameters reflect the CMYK build of the color Mountain Purple. The color consists of 70% cyan, 70% magenta, 0% yellow, and 0% black. The numbers that are passed as parameters reflect this: 45874.5 is 70% of 65535. If an RGB or HSB color is created, only three parameters are passed using textRGBcolor and textHSBcolor respectively.

Specify the color model and assign the CMYK values

```
set (newColorSpec.colorType,cmykType)
set (newColorSpec.cmykColor,theColor)
```

Concepts Illustrated: This statement is similar to selecting the CMYK color model from the Model pop-up menu in the Edit Color dialog box.

The first set event changes the colorType property of the newColorSpec to cmykType. Use hsbType for the HSB color model and rgbType for the RGB color model.

The second set event assigns the numerical values generated by the textCMYKcolor script to theColor created above.

Name the new color

```
try
   set(newColorSpec.name,"Mountain Purple")
else
   delete(colorSpec[newColorSpec])
```

Concepts Illustrated: The lines above are similar to typing a name for a new color in the Edit Color dialog box.

The try statement, another form of a conditional loop, checks for a color already named Mountain Purple.

If Mountain Purple exists, the newly created color is deleted (two colors cannot have the same name in QuarkXPress). The script will use the Mountain Purple color that already exists.

The set event assigns the name Mountain Purple to the newColorSpec.

Create guides

```
with document[1]
   paramlist = 0
   putAppleListItem("4.218 cm",position,@paramlist)
   create(horizontalGuide,0,paramlist,beginningof(page[1]))
   paramlist = 0
   putAppleListItem("8.477 cm",position,@paramlist)
   create(horizontalGuide,0,paramlist,endof(page[1]))
   paramlist = 0
   putAppleListItem("27.152 cm",position,@paramlist)
   create(horizontalGuide,0,paramlist,endof(page[1]))
   paramlist = 0
   putAppleListItem("2 cm",position,@paramlist)
   create(verticalGuide,0,paramlist,endof(page[1]))
   paramlist = 0
   putAppleListItem("4.962 cm",position,@paramlist)
   create(verticalGuide,0,paramlist,endof(page[1]))
   paramlist = 0
   putAppleListItem("18.742 cm",position,@paramlist)
   create(verticalGuide,0,paramlist,endof(page[1]))
   paramlist = 0
   putAppleListItem("32 cm",position,@paramlist)
   create(verticalGuide,0,paramlist,endof(page[1]))
```

Concepts Illustrated: The create events above simulate clicking on the horizontal and vertical rulers to create guides, then dragging the guides into position.

The with statement references the active document by index [1]. The index value [1] refers to the frontmost document.

The paramlist is a list containing multiple properties that can be passed as the third parameter in a create event. The paramlist = 0 statement clears any information currently in the paramlist.

The putAppleListItem statements add a property and associated data value to the given paramlist (in this case, the position property is specified in centimeters). The paramlist = 0 statement is repeated prior to each putAppleListItem statement to clear the list.

Each create event makes a horizontalGuide or verticalGuide. The guides are created with the properties specified in the paramlist (position in centimeters).

The first guide is created at the beginningOf(page[1]) in the object hierarchy according to the fourth parameter. Subsequent guides are created at endOf(page[1]).

Create the first text box

```
with document[1].page[1]
  paramlist = 0
  frect = textRect("2 cm","5 cm","8 cm","19 cm")
  putAppleListItem(frect,bounds,@paramlist)
  create(textBox,0,paramlist,beginningOf(it))
```

Concepts Illustrated: The lines above are similar to creating a text box with the text box tool, then sizing and positioning it from the Measurements palette.

The with statement references the first page of the active document; both are referenced by index [1].

The paramlist is a list containing multiple properties that can be passed as the third parameter in a create event. The paramlist = 0 statement clears any information currently in the paramlist.

A local variable, frect, is assigned a value. This value is the result of calling the textRect script (located in Frontier's QXP.defs table), which sets numerical values for the text box boundaries. The four parameters indicate how far from the top left corner of the document page the top, left, bottom, and right sides of the box will be positioned.

The putAppleListItem statement adds the frect values to the given paramlist. The bounds property is assigned the value of the variable frect.

The create event makes a textBox with default specifications (indicated by the 0 in the second parameter) and the specifications determined by the paramlist (indicated in the third parameter). The text box is placed according to the fourth parameter, beginningOf(it).

If you want to see an object after it is created while the script is running, add the line `show(it)`. This places the current object in the upper left corner of the document window.

Enter the headline into a text box

```
with textBox[1]
   set(verticalJustification,bottomJustified)
   set(color,"None")
with textBox[1].story[1]
   set(it,"Biking Gear")
```

Concepts Illustrated: The statements above are similar to specifying a Vertical Alignment and Background Color in the Text Box Specifications dialog box, and then typing Biking Gear into the text box.

The with statement references the first textBox by index [1].

The next two set events change the verticalJustification to bottom and the background color to None.

The with statement references the story in the active textBox; both are referenced by index [1]. (Only one story is possible per text box or chain of linked text boxes.)

The set event specifies it (it refers to the story, the last object referenced in the with statement). The text Biking Gear is entered into the text box. It is then formatted with properties defined in the Normal style sheet for the defaultDocument.

Format the headline

```
set(font,"Times")
set(word[1].size,30)
set(word[1].textStyle,allcaps)
set(word[1].baseShift,60)
set(word[1].track,50)
set(word[1].character[last].kern,-100)
set(word[2].size,120)
set(word[2].color,"Mountain Purple")
set(word[2].textStyle,Italic)
set(word[2].character[1].kern,-5)
set(word[2].character[2].kern,-5)
```

Concepts Illustrated: The set statements above are comparable to the Font, Size, Type Style, Color, Baseline Shift, Track, and Kern commands in the Style menu.

The first set event changes the font for the story to Times.

The next four set events reference the first word by index [1]. The size, textStyle, baseShift, and track properties of the word Biking are changed.

The next set event references the last character of the first word; the character is referenced by relative position. The kern property is changed to -100. (To kern the space between two words, reference the last character of the first word.)

The next three set events reference the second word by index [2]. The size, color, and type style properties of the word Gear are changed.

The last two set events reference the first and second character of the second word; all are referenced by index. The kern property of each character is changed to -5. (To kern a pair of characters, you only need to reference the first character.)

Create the second text box

```
with document[1].page[1]
  paramlist = 0
  frect = textRect("8.5 cm","5 cm","29.959 cm","18.472 cm")
  putAppleListItem(frect,bounds,@paramlist)
  create(textBox,0,paramlist,endOf(it))
```

Concepts Illustrated: The lines above are similar to creating a text box with the text box tool, then sizing and positioning it from the Measurements palette.

The with statement references the first page of the active document; both are referenced by index [1].

The paramlist is a list containing multiple properties that can be passed as the third parameter in a create event. The paramlist = 0 statement clears any information currently in the paramlist.

A local variable, frect, is assigned a value. This value is the result of calling the textRect script (located in Frontier's QXP.defs table), which sets numerical values for the text box boundaries. The four parameters indicate how far from the top left corner of the document page the top, left, bottom, and right sides of the box will be positioned.

The putAppleListItem statement adds the frect values to the given paramlist. The bounds property is assigned the value of the variable frect.

The create event makes a textBox with default specifications (indicated by the 0 in the second parameter) and the specifications determined by the paramlist (indicated in the third parameter). The text box is placed according to the fourth parameter, endOf(it).

Locate and import a text file

```
with textbox[2]
   try
      set(story[1],alias(thepath+"ASB Text"))
      else
         if (file.getFileDialog("Open the file named 'ASB
         Text'.",@filepath,'TEXT'))
            set(story[1],alias(filepath))
```

Concepts Illustrated: The statements above are similar to locating and import-

ing a text file in the Get Text dialog box.

The try statement looks for the ASB Text file in the location previously defined by thepath (see page 2.12). If the file exists in this location, the set event imports the ASB Text file, replacing the story in the text box.

If ASB Text does not exist in the location defined by thepath, the script will continue with the if statement. (The file will only be located via thepath if your hard drive and folders are named the same as those defined in thepath.)

The statement in the if construct calls a Frontier script, getFileDialog, located at root.system.verbs.builtins.file. The getFileDialog script brings up an Open dialog box.

The first parameter is a message to the user shown at the bottom of the dialog box, Open the file named ASB Text. The second parameter stores the path to the text file in an address this path is used to import the image files. The third parameter is the signature for a text file. Once the user locates the text file and clicks OK, the set event imports the text.

If you want to open a QuarkXPress document using the getFileDialog script, the signature would be XDOC.

Establish the new path

```
lastFieldNumber = string.countFields(filepath,":")
thepath = ""
for count = 1 to lastFieldNumber-1
   thepath =thepath+string.nthField(filepath,":",count)+":"
```

Concepts Illustrated: The path to the text file is saved and used when the script attempts to locate the image files.

In the statements above, the previous path (established on page 2.12) is defined as thepath; the new path is defined as filepath.

The first statement assigns the variable lastFieldNumber to the results of another Frontier script, string.countFields. The string.countFields counts the number of strings in the path separated by colons.

The second statement clears thepath by setting it to a null string.

The third statement starts a loop that counts the total number of separate strings separated by a colon in the filepath.

The fourth statement reconstructs thepath from the location established in the filepath. The location of the text file is now defined as thepath.

Format the body copy

```
set(story[1].font,"Times")
```

```
set(story[1].size,11)
set(story[1].leading,43)
set(story[1].justification,full)
```

Concepts Illustrated: The set statements above are comparable to choosing a Font, Size, Leading, and Alignment from the Style menu.

The four set events reference the entire story by index [1]. The font, size, leading, and justification properties of the story are set.

Create a colored drop cap

```
with textBox[2].story[1].paragraph[1]
  set(dropCapChars,1)
  set(dropCapLines,3)
  set(word[1].character[1].color,"Mountain Purple")
```

Concepts Illustrated: The statements above are similar to specifying a Drop Cap in the Paragraph Formats dialog box. The color Mountain Purple is then applied to the drop cap character.

The with statement references the first paragraph of the story in the second textBox ; the objects are all referenced by index.

The first set event specifies that the first character will be a drop cap. The second set event specifies that it will be a three-line drop cap.

The third set event references the drop cap, which is the first character of the first word ; both are referenced by index [1]. The color property is changed to Mountain Purple.

Create an initial cap

```
with textBox[2].story[1].paragraph[last]
   set(word[1].character[1].color,"Mountain Purple")
   set(word[1].character[1].size,28)
```

Concepts Illustrated: The lines above are similar to creating a decorative initial cap with local formatting.

The with statement references the last paragraph of story in the second textBox. The story and textBox are referenced by index, the paragraph is referenced by relative position.

The two set statements references the first character of the first word ; they are referenced by index. The color property is changed to Mountain Purple and the size property is changed to 28.

Specify a Rule Above

```
set(ruleAbove.ruleOn,true)
set(ruleAbove.textLength,false)
```

```
set(ruleAbove.width,0.5)
set(ruleAbove.position,"1 cm")
set(ruleAbove.color,"Cyan")
set(ruleAbove.shade,100)
```

Concepts Illustrated: The set events above are comparable to settings in the expanded Rules dialog box.

The first set event uses a boolean operator to determine if the paragraph's ruleAbove will be turned on (ruleOn). The true boolean operator indicates that the paragraph will have a rule above it.

The second set event uses a boolean operator to determine if the rule will match the textLength. The false boolean operator indicates that it will not match the length of the text. The rule will extend the width of the text box (minus any defined text inset).

The last four set events specify the width, position, color, and shade properties of the ruleAbove.

Create the first picture box

```
with document[1].page[1]
   paramlist = 0
   frect = textrect("10.386 cm","20.758 cm","27.636 cm","33.508 cm")
   bkgrndclr = "None"
   putAppleListItem(frect,bounds,@paramlist)
   putAppleListItem(bkgrndclr,color,@paramlist)
   create(picturebox,0,paramlist,beginningof(it))
```

Concepts Illustrated: The lines above are similar to creating a picture box, sizing and positioning it, and then specifying a Background Color as you would in the Picture Box Specifications dialog box.

The with statement references the first page of the active document; both are referenced by index [1].

The paramlist is a list containing multiple properties that can be passed as the third parameter in a create event. The paramlist = 0 statement clears any information currently in the paramlist.

A local variable, frect, is assigned a value. This value is the result of calling the textRect script (located in Frontier's QXP.defs table), which sets numerical values for the picture box boundaries. The four parameters indicate how far from the top left corner of the document page the top, left, bottom, and right sides of the box will be positioned.

A local variable, bkgrndclr is assigned a value. In this case, the background color of the picture box will be None.

The putAppleListItem statements add the frect and bkgrndclr values to the given paramlist. The bounds property is assigned the value of the variable frect and the color property is assigned the value of the variable bkgrndclr.

The create event makes a pictureBox with default specifications (indicated by the 0 in the second parameter) and the specifications determined by the paramlist (indicated in the third parameter). The picture box is placed according to the fourth parameter, beginningOf(it).

Import the first picture

```
with pictureBox[1]
   set(rotation,-25)
   try
      set(image[1],alias(thepath+"Shirts.tiff"))
   else
      if (file.getFileDialog("Open the image named
      'Shirts.tiff'.",@filepath,'TIFF'))
         set(image[1],alias(filepath))
    with image[1]
      set(scale,textPoint("115","115"))
```

Concepts Illustrated: The statement above are similar to locating and importing an image file in the Get Picture dialog box (File menu).

The with statement references the first pictureBox by index [1].

The first set event specifies the rotation property of the pictureBox.

The try statement looks for the Shirts.tiff file in the location previously defined by thepath. If the file exists in this location, the set event specifies Shirts.tiff as the image in the picture box. (A picture box can only have one image.)

If Shirts.tiff does not exist in the location defined by thepath, the script will continue with the if statement. The statement in the if construct calls a Frontier script, getFileDialog, located at root.system.verbs.builtins.file. The getFileDialog script brings up an Open dialog box.

The first parameter is a message to the user shown at the bottom of the dialog box, Open the image named Shirts.tiff. Once the user locates the image file and clicks OK, the set event imports the image.

The second with statement references the image by index [1].

The set event specifies the scale property of the image by calling the textPoint script (located in Frontier's QXP.defs table). The textPoint script passes two values to be coerced into a property-specific data type by QuarkXPress. In this case, the values for scale are coerced into percentage types.

Create the second picture box and import a picture

```
with document[1].page[1]
   paramlist = 0
   frect = textRect("8.471 cm","2 cm","9.971 cm","3.5 cm")
   bkgrndclr = "None"
   putAppleListItem(frect,bounds,@paramlist)
   putAppleListItem(bkgrndclr,color,@paramlist)
   create(pictureBox,0,paramlist,endOf(it))
   with pictureBox[2]
      try
         set(image[1],alias(thepath+"Glove.tiff"))
      else
         if (file.getFileDialog("Open the image named
         'Glove.tiff'.",@filepath,'TIFF'))
            set(image[1],alias(filepath))
      set(image[1].bounds,exactFit)
```

Concepts Illustrated: The first seven lines above are similar to creating a picture box, sizing and positioning it, and then specifying a Background Color as you would in the Picture Box Specifications dialog box. The last six statements are similar to locating and importing an image file in the Get Picture dialog box.

The with statement references the first page of the active document ; both are referenced by index [1].

The paramlist is a list containing multiple properties that can be passed as the third parameter in a create event. The paramlist = 0 statement clears any information currently in the paramlist.

A local variable, frect, is assigned a value. This value is the result of calling the textRect script (located in Frontier s QXP.defs table), which sets numerical values for the picture box boundaries. The four parameters indicate how far from the top left corner of the document page the top, left, bottom, and right sides of the box will be positioned.

A local variable, bkgrndclr is assigned a value. In this case, the background color of the picture box will be none.

The putAppleListItem statements add the frect and bkgrndclr values to the given paramlist. The bounds property is assigned the value of the variable frect and the color property is assigned the value of the variable bkgrndclr.

The create event makes a pictureBox with default specifications (indicated by the 0 in the second parameter) and the specifications determined by the paramlist (indicated in the third parameter). The picture box is placed according to the fourth parameter, beginningOf(it).

The with statement references the second pictureBox by index [2].

The try statement looks for the Glove.tiff file in the location previously defined by thepath. If the file exists in this location, the set event specifies Glove.tiff as the image in the second picture box.

If Glove.tiff does not exist in the location defined by thepath, the script will continue with the if statement. The statement in the if construct calls a Frontier script, getFileDialog, located at root.system.verbs.builtins.file. The getFileDialog script brings up an Open dialog box.

The first parameter is a message to the user shown at the bottom of the dialog box, Open the image named Glove.tiff. Once the user locates the image file and clicks OK, the set event imports the image.

The last set event references the image by index [1]. The bounds property of the image is set to exactFit.

Create and position copies of the picture box

```
clone(pictureBox[2],after(pictureBox[2]))
with pictureBox[3]
   set(bounds,textrect("12.471 cm","2 cm","13.971 cm","3.5 cm"))
clone(pictureBox[2],after(pictureBox[3]))
with pictureBox[4]
   set(bounds,textrect("16.471 cm","2 cm","17.971 cm","3.5 cm"))
clone(pictureBox[2],after(pictureBox[4]))
with pictureBox[5]
   set(bounds,textrect("20.471 cm","2 cm","21.971 cm","3.5 cm"))
```

Concepts Illustrated: The clone and set events above are similar to using the Step and Repeat feature.

The first clone event references the second pictureBox by index [2]. A copy of the pictureBox is placed after the second pictureBox.

The first with statement references the new pictureBox by index [3].

The first set statement calls the textRect script, which sets numerical values for the picture box boundaries. The four parameters indicate how far from the top left corner of the document page the top, left, bottom, and right sides of the box will be positioned.

The following lines use the object hierarchy to clone pictureBox[2] after pictureBox[3], then after pictureBox[4]. Each new pictureBox is referenced by index and positioned.

Create the third picture box and import a picture

```
with document[1].page[1]
   create(pictureBox,0,0,endOf(it))
   with pictureBox[6]
      set(bounds,textRect("6.875 cm","18.425 cm","12.729 cm",
      "26.4 cm"))
      set(color,"None")
      try
         set(image[1],alias(thepath+"Helmet.tiff"))
      else
         if (file.getFileDialog("Open the image named
         'Helmit.tiff'.",@filepath,'TIFF'))
            set(image[1],alias(filepath))
      set(image[1].scale,textpoint("70","70"))
      set(image[1].offset,textpoint("0.557 cm","1.254 cm"))
```

Concepts Illustrated: The statements above are similar to creating a picture box and importing a picture. The properties are specified with set events rather than by using a paramlist as the third parameter in a create event.

The with statement references the first page of the active document ; both are referenced by index [1].

The create event makes a pictureBox with default specifications (indicated by the 0 in the second and third parameters). The picture box is placed according to the fourth parameter endOf(it).

The second with statement references the third pictureBox by index [3].

The first set event calls the textRect script (located in Frontier's QXP.defs table) to set numerical values for the picture box boundaries.

The four parameters indicate how far from the top left corner of the document page the top, left, bottom, and right sides of the box will be positioned.

The second set events sets the background color property of the pictureBox to None.

The try statement looks for the Helmut.tiff file in the location previously defined by thepath. If the file exists in this location, the set event specifies Helmut.tiff as the image in the second picture box.

If Glove.tiff does not exist in the location defined by thepath, the script will continue with the if statement. The statement in the if construct calls a Frontier script, getFile
Dialog, located at root.system.verbs.builtins.file. The getFileDialog script brings up an Open dialog box.

The first parameter is a message to the user shown at the bottom of the dialog box, Open the image named Helmut.tiff. Once the user locates the image file and clicks OK, the set event imports the image.

The fourth set event sets the scale property of the image to 70%. The last set event sets the offset property of the image in centimeters. In both cases, the textPoint script passes two values to be coerced into a property-specific data type by QuarkXPress.

Create a vertical line

```
with document[1].page[1]
   create(lineBox,0,0,beginningOf(it))
   with lineBox[1]
      set(leftPoint,textPoint("0 cm","21.406 cm"))
      set(rightPoint,textPoint("8 cm","21.406 cm"))
      paramlist = 0
      putAppleListItem("Magenta",color,@paramlist)
      putAppleListItem(3,lineWidth,@paramlist)
      putAppleListItem(dottedLine,LineStyle,@paramlist)
      set(Properties,paramlist)
```

Concepts Illustrated: The statements above are similar to creating and positioning a vertical line with the Line tool and choosing a Color, Width, and

LineStyle from the Style menu.

The with statement references the first page of the active document ; both are referenced by index [1].

The create event makes a lineBox with default specifications (indicated by the 0 in the second and third parameters). The line is placed according to the fourth parameter beginningOf(it).

The second with statement references the lineBox by index [1].

The next two set events specify the leftPoint and rightPoint of the line in centimeters. The textPoint script passes two values to be coerced into a property-specific data type by QuarkXPress. In this case, the values are coerced into centimeters.

The paramlist = 0 statement clears any information currently in the paramlist.

The putAppleListItem statements add specifications for the color, lineWidth, and lineStyle properties to the given paramlist.

The last set event sets the properties of the lineBox according to the paramlist.

Create a horizontal line

```
with document[1].page[1]
   create(linebox,0,0,endof(it))
   with linebox[2]
      set(leftpoint,textpoint("8 cm","2 cm"))
      set(rightpoint,textpoint("8 cm","32 cm"))
      set(linewidth,0.5 )
```

Concepts Illustrated: The statements above are similar to creating and positioning a horizontal line with the Line tool and choosing a Width from the Style menu.

The with statement references the first page of the active document ; both are referenced by index [1].

The create event makes a lineBox with default specifications (indicated by the 0 in the second and third parameters). The line is placed according to the fourth parameter endOf(it).

The second with statement references the second lineBox by index [2].

The next two set events specify the leftPoint and rightPoint of the line in centimeters. The textPoint script passes two values to be coerced into a property-specific data type by QuarkXPress. In this case, the values are coerced into centimeters.

The last set event sets the lineWidth property of the lineBox to 0.5. Line widths are set according to points, the default line measurement system in QuarkXPress.

Hide guides and save the document

```
set(document[1].guidesShowing,false)
with document[1]
   saveDocument(document[1],thepath+"Constructed Document")
```

Concepts Illustrated: The lines above simulate choosing Hide Guides from the View menu and Save As from the File menu.

The first set event determines whether the document will have guidesShowing. The boolean operator false indicates that guides will not be showing.

The with statement references the active document by index [1].

The save event saves the document in thepath with the name Constructed Document.

Reset default specifications for future documents

```
with defaultdocument[1]
   set(pageHeight,oldHeight)
   set(pageWidth,oldWidth)
   set(autoTextBox,oldAutoTextBox)
   set(guidesShowing,oldGuidesShowing)
   set(horizontalMeasure,xDocMeasure)
   set(verticalMeasure,yDocMeasure)
ConstructDocument()
```

Concepts Illustrated: The set statements above replace the document default specifications with your original specifications.

The with statement references the current defaultDocument by index [1].

Each set statement specifies a property of the defaultDocument according to the local variable. For example, the pageHeight property is specified as the variable oldHeight. The original page size, automatic text box setting, whether the guides are showing, and the default measurement system are reset.

Reference Material for Apple Events

The explanations in the Script Writing Sample provide an overview of how scripts use events to control QuarkXPress objects and their properties. However, the "Document Construction" script could not use and discuss every possible event and object. Chapters 3 and 4 provide detailed descriptions of each event and object that QuarkXPress supports.

This chapter, the "Reference Material for Apple Events," provides definitions and examples (in UserTalk and AppleScript) for object references, insertion points, and each event that Quark
XPress supports. This section covers the events in the five Suites QuarkXPress supports: the Required, Core, Miscellaneous, Text, and QuarkXPress Suites. (Apple occasionally refers to the Core Suite as the Standard Suite.)

Once you are familiar with the scripting language's syntax, you should be able to write scripts for QuarkXPress by referring to the information in Chapters 3 and 4. This chapter also includes an explanation of how to use Chapter 4, the "Reference Material for QuarkXPress Objects" on page 3.12.

Using Chapter 3

The Reference Material for Apple Events

This chapter includes definitions and examples of object reference forms, insertion points, and the events supported by QuarkXPress. Each object reference form and insertion point term is listed with a description of its usage and an example in both UserTalk and AppleScript. The examples are taken from other scripts and are shown out of context.

Each event is listed with a description of its usage, a prototype in UserTalk and AppleScript, and any applicable possible values and results. The terms and events are shown in the following format:

Term or Event

Usage:	Description of when to use this term or event.
UserTalk:	*UserTalk prototype with parameters in italics*
AppleScript:	*AppleScript prototype with parameters in italics*
Possible Values:	variable: choices in Courier font
	variable: user-defined values in Times font
Result:	result in Times font

Events Terminology

Object Reference Forms

An Apple Event message must reference a specific object in an application to communicate. The reference first identifies the container enclosing the object you're specifying. It then uses a reference form to separate a specific object from all possible objects in the container. The reference form can be defined by index, name, range, relative position, or test.

Index
Usage: To identify ordered elements in a container with an integer number.
UserTalk: `set(word[2].character[1].kern,-14)`
AppleScript: `set the kern of character 1 of word 2 to -14`

Name
Usage: To identify named objects with a text string.
UserTalk: `set(pictureBox["Ralph"].runaround,manualRunaround)`
AppleScript: `set runaround of picture box "Ralph" to manual runaround`

Range
Usage: To identify a range of objects.
UserTalk: `set(word[2]to[5]).color,"cyan")`
AppleScript: `set color of word 2 through word 5 to "cyan"`

Relative Position
Usage: To identify objects that are before or after other objects.
UserTalk: `set(textBox[2].pictureBox[next].rotation,45)`
AppleScript: `set the rotation of the picture box after text box 2 to 45`

Test
Usage: To identify objects that meet certain conditions.
UserTalk: `set (textStyleRange[color.name=="red"],"blue")`
AppleScript: `set the color of (every word whose color = "red") to "blue"`

Insertion Points in the Hierarchy

An insertion point specifies where you want to place an object within the container hierarchy. As you create and insert objects in the hierarchy, the index reference form for existing objects may change.

After
Usage: To insert or create an object after a specified object (the specified object will not be the container). For example, use after to move the first paragraph in a story to after the seventh paragraph.
UserTalk: `move(paragraph[1],after(paragraph[7]))`
AppleScript: `move paragraph 1 to after paragraph 7`

Before
Usage: To insert or create an object before the specified object (the specified object will not be the container). For example, use before to paste a copy of the fifth word before the first word in a sentence.

UserTalk: duplicate(word[5],before(word[1]))
AppleScript: duplicate word 5 to before word 1

Events Supported by QuarkXPress

Beginning

Usage: To insert or create an object at the beginning of the specified container. For example, use beginning to insert a word as the first word of a paragraph.

UserTalk: `move(word[1],beginningOf(paragraph[1]))`

AppleScript: `move word 1 to beginning of paragraph 1`

Ending

Usage: To insert or create an object at the end of the specified container. For example, use ending to create a text box that is the last text box on the document page.

UserTalk: `create(textBox,0,0,endOf(it))`

AppleScript: `make text box at end`

Replace

Usage: To replace the specified object with a new object. For example, use replace to substitute a new text box for the third text box in a document. AppleScript uses make to crete an object in the same location, rather than replace.

UserTalk: `create(textBox,0,0,replace(textBox[3]))`

AppleScript: `make text box at text box 3`

Required Suite

The Required Suite consists of four events the Finder sends to applications.

Open Application
Usage: To launch your application as normal (without opening or printing any documents).
UserTalk: openApplication()
AppleScript: run alias to application

Open Document
Usage: To open the specified document.
UserTalk: openDocument(alias to document)
AppleScript: open alias to document

Print Document
Usage: To print the specified document.
UserTalk: printDocument(alias to document)
AppleScript: print alias to document

Quit Application
Usage: To close your application as normal (including closing documents, displaying save alerts, and releasing memory).
UserTalk: quitApplication()
AppleScript: quit

Core Suite

The Core Suite consists of basic events that most applications use to communicate.

Close
Usage: To close a specified object and determine whether to save it. Close is usually used for a window or document.
UserTalk: Close(reference,save options,alias)
AppleScript: Close reference saving save options in alias
Possible Values: save options: yes, no, ask

Clone/Duplicate
Usage: To copy the data and properties of a specified object and create a new object with the same data and properties. You can specify an insertion point for the new object. (If you don t specify a new insertion point, the new object is placed in the same container as the initial object, at the end of the container s elements.) Duplicate is similar to create.
UserTalk: duplicate(reference,insertion location)
AppleScript: duplicate reference to insertion location
Possible Values: insertion location: see page 3.03 for insertion point information
Result: reference (to the new object)

Count
Usage: To determine how many objects in a specified class are contained in a specified object.
UserTalk: Count(reference,object class)
AppleScript: Count of object class in reference
Possible Values: object class: any object class
Result: integer

Create/Make
Usage: To create a new element of an object. You can specify the type of object you want to create, set properties in the new object, and specify an insertion point.
UserTalk: create(object type,data,properties,insertion location)
AppleScript: make object type at insertion location initial data data initial properties properties
Possible Values: data: object specific data
properties: record
insertion location: see page 3.03 for insertion point information
Results: reference (to the new object)

Data Size
Usage: To obtain an object s size in bytes.
UserTalk: dataSize(reference,type)
AppleScript: data size reference as type
Possible Values: type: type class
Result: integer

Delete
Usage: To remove a specified object from an object or application.
UserTalk: delete(reference)
AppleScript: delete reference

Exists
Usage: To check for the existence of a specified object.
UserTalk: exists(reference)
AppleScript: exists reference
Result: boolean

Get
Usage: To determine the data structure for an object. Get and set are usually used to read and write an object s internal data and properties, rather than the whole object.
UserTalk: Get(reference)
AppleScript: Get reference
Result: the properties of the object you reference

Get As
Usage: To determine the data structure for an object in a specific data type.
UserTalk: GetAs(reference.property,type)
AppleScript: Get property of reference as type
Possible Values: see the Coercion Chart on page 3.10
Result: the properties of the object you reference in the data type you

specify

Move

Usage: To change an object s position in an application s container hierarchy. The specified object is moved from its current location to the specified insertion point. Move is not used to change the physical location of an object. To change the location, you would use set to change its properties.

UserTalk: Move(reference,insertion location)

AppleScript: Move reference to insertion location

Possible Values: insertion location: see page 3.03 for insertion point information

Result: reference (to the object in its new location)

Save

Usage: To save a specified object to a specified file on disk.

UserTalk: Save(reference,alias, filetype,EPS Format,EPS Data,EPS OPI)

AppleScript: Save reference in alias as file type EPS format EPS format EPS data EPS data EPS OPI EPS OPI

Possible Values: EPS Format: mac in BW, mac in Color, mac DCS, mac DCS2, PC in BW, PC in Color, PC DCS, PC DCS2
EPS Data: ASCII EPS, binary EPS, EPS include images
EPS OPI: omit TIFF, omit TIFF and EPS

Set

Usage: To change an object. Get and set are usually used to read and write an object s internal data and properties; rather than the whole object.

UserTalk: Set(reference,data,replacing)

AppleScript: Set reference to data replacing

Possible Values: data: object specific data
replacing*: ask/ignore/replace/rename
*Replacing is used for importing text with style sheets in any text file format supported by QuarkXPress.

Open

Usage: To open an object or file.

UserTalk: Open(reference,useDocPrefs,remapFonts,doAutoPictureReimport)

AppleScript: Open reference, use doc prefs use doc prefs remap fonts remap fonts do auto picture reimport do auto picture reimport)

Possible Values: use doc prefs: yes, no, ask
remap fonts: no, ask
do auto picture reimport: yes, no, ask

Print

Usage: To print a specified object.

UserTalk: print(reference,copies,OPI,cover page,paper source to alias,list of plates)

AppleScript: print reference, copies copies OPI OPI cover page cover page paper source paper source to alias plates list of plates

Possible Values: copies: short integer
OPI: include images, omit TIFF, omit TIFF and EPS
cover page: no, first page, last page

paper source: paper cassette, manual feed
plates: a list of strings

Miscellaneous Suite

The Miscellaneous Suite consists of functions related to the clipboard and other menu-driven functions.

Copy
Usage: To place a copy of the selected object on the clipboard.
UserTalk: copy
AppleScript: copy

Cut
Usage: To place the selected object on the clipboard.
UserTalk: cut
AppleScript: cut

Do Script
Usage: To execute a script.
UserTalk: doScript(data,type)
AppleScript: do script data script type type
Result: result of the script execution

Paste
Usage: To place the data on the clipboard into a document.
UserTalk: paste
AppleScript: paste

Redo
Usage: To reverse the action of the immediately preceding undo.
UserTalk: redo
AppleScript: redo

Revert
Usage: To restore an object to its last saved state.
UserTalk: revert(reference)
AppleScript: revert reference

Show
Usage: To bring an object into view.
UserTalk: show(reference)
AppleScript: show reference

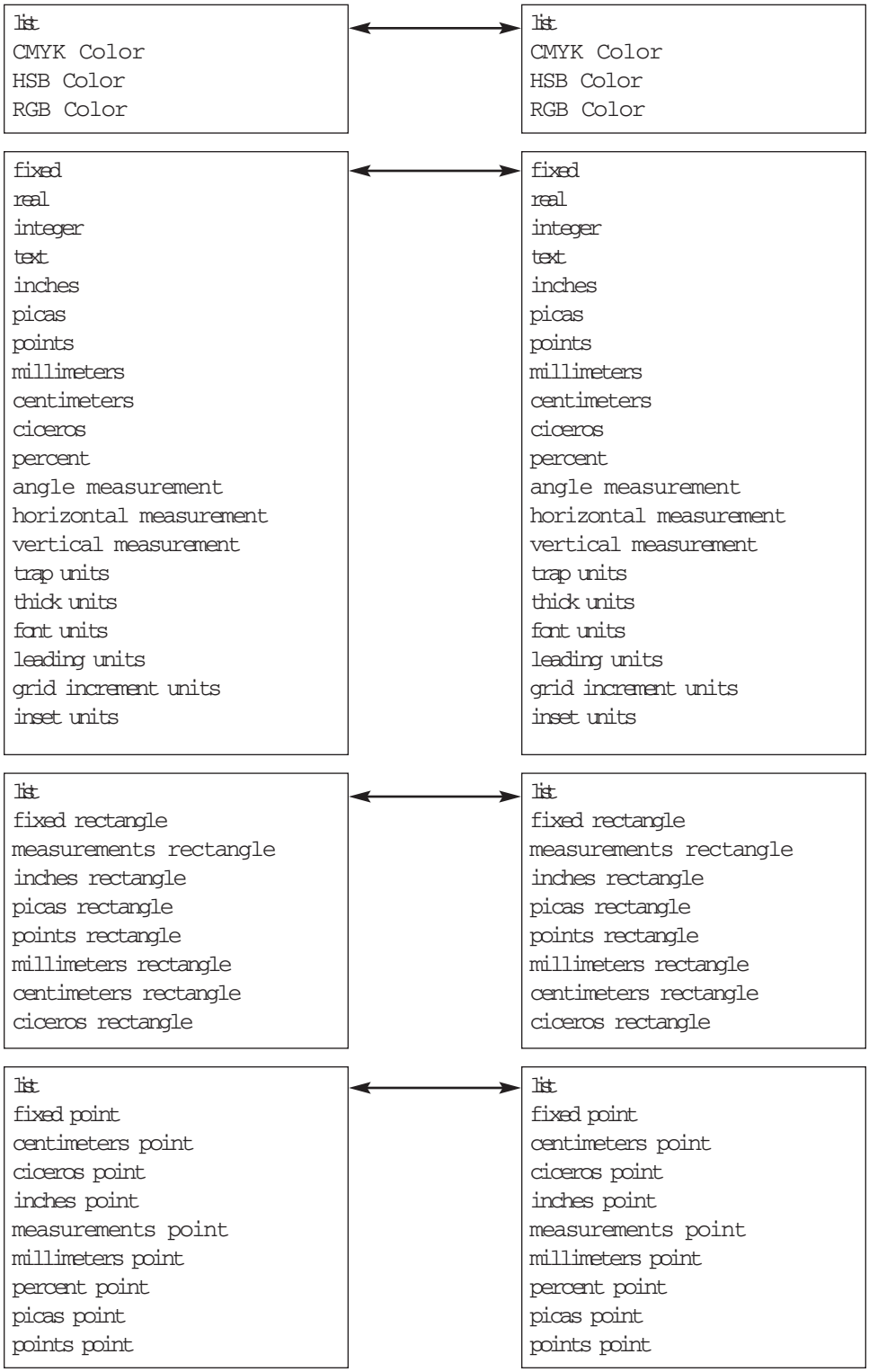
Undo
Usage: To undo the action of the immediately preceding user interaction (you cannot undo an event).
UserTalk: undo
AppleScript: undo

Uniform
Usage: To confirm that all objects in a group have the same value for a specified property.

Data Coercion Chart

UserTalk:	uniform(reference, property)
AppleScript:	uniform reference in property

The data types in each pair of boxes linked by a double arrow can be coerced into each other.



Using Chapter 4

QuarkXPress Suite

The Miscellaneous Suite consists of one event related to redraw.

Do Updates

Usage:	To redraw the screen after the execution of a script.
UserTalk:	doUpdates
AppleScript:	do updates

This chart lists the possible data structure you can request with a Get As event. The right-facing arrow % indicates that the data types on the left can be coerced into the data types at the right. The indicates that both data types can be coerced into each other. The left-facing arrow indicates that the data types on the right can be coerced into the data types on the left.

text	%	alias
integer text		boolean
color spec		string integer list RGB Color
style spec		string integer
h and j spec		string integer

(Continued on page 3.11.)

The Reference Material for QuarkXPress Objects

The Reference Material for QuarkXPress Objects is included in the For Advanced Scripting folder as a separate QuarkXPress document entitled, Chapter 4. The document is in landscape format and can be three-hole punched and placed in a binder for easy reference.

To illustrate how to use this material, we've mapped out the syntax for an event in UserTalk and AppleScript. The set event is used because it can contain information from each table in Chapter 4.

UserLand Frontier:	set(object[<reference>].property,data)
AppleScript:	set property of reference object to data

To write a set event, you need to find syntax for the object, the way it can be referenced, the properties that can be applied to it, and what type of data to use (everything shown in italics above). This information is listed in the reference material as follows:

Objects: The objects are listed alphabetically in the Object Index on the first page. Look up the specific object you want, then turn to the page containing that object's tables. Use the syntax shown in the table headings for each object.

Events and Examples (table): To find the events the object can respond to, look in the Events column. Examples of syntax for each event are listed in the UserTalk and AppleScript columns.

Elements and Reference Forms (table): Objects that the object class can contain or be contained by are in the Elements column. This table includes five columns, listing possible reference specifiers for each element: by numeric index, by name, by range, satisfying a test, and before/after another element. A dot under an individual reference form column indicates that the element can be referenced that way.

Properties, Data Types, and Description (table): The properties that apply to the object class and their syntax are listed in the Properties column. The first column, r/o indicates whether the property is read only (with a dot). The second column lists the property name. The third column lists either the data type or the possible data choices for that property. The fourth column offers a general description of the the property.

Glossary

Apple Events
Messages sent from one Macintosh application or process to another which give instructions, respond to instructions, and send or receive data. Apple events are defined by Apple Computer, Inc. or other application developers and must conform to the Apple Event Interprocess Messaging Protocol.

Apple Event Registry: Standard Suites
A compilation of standard Apple events defined by Apple Computer, Inc. or other application developers including: Apple events, object classes, primitive object classes, descriptor types. key forms, and constants. The Apple Event Registry: Standard Suites is maintained by the Apple Event Developers Association.

Apple Event Interprocess Messaging Protocol
The protocol for interapplication communication defined by Apple Computer, Inc. Interapplication messages must conform to this protocol to qualify as Apple events.

AppleScript
A system-wide scripting language developed by Apple Computer, Inc. AppleScript scripts can control the Macintosh operating system and applications that support Apple events.

Attribute
The component of an Apple event that identifies it and the tasks it can perform on the data specified in the parameters. Attributes consist of an event class and event ID.

Container
The object that contains the element specified by an Apple event.

Core Suite
The basic Apple events and objects that most applications use to communicate. The events include: get, set, create, duplicate, move, delete, count, close, save, print, open, data size, and exists. Objects include windows, documents, pages, etc.

Element
An object contained by another Apple events object. The element classes in the Apple Event Registry: Standard Suites define the types of objects each Apple events object can contain.

Event Class
The attribute of an Apple event that identifies which suite (group of related Apple events) it belongs to such as the Required Suite, Core Suite, etc.

Event ID
The attribute of an Apple event that uniquely identifies it within its event class and defines the tasks it can perform.

Events
The part of an Apple events message that tells objects what to do (similar to a verb).

Functional-area Suites
Groups of objects and events that relate to similar functional areas, including: the Text Suite, the Quickdraw Graphics Suite, the Table Sutie, and Miscellaneous Standards.

Insertion Points
A reference with a parameter that defines where in the container hierarchy an object should be placed.

Miscellaneous Suite
Basic Apple events, related to the clipboard and other menu-driven functions, that most applications supports. The events include: cut, copy, paste, undo, etc.

Object Class
A category for Apple event objects that share specific characteristics and capabilities.

Object Model
The Apple Events Object Model is a standard message structure that allows Macintosh applications to communicate. Messages built according to the Object Model consist of events, objects, and potentially data.

Object
A distinct item in an application that can respond to an Apple event. Objects are usually items a user can identify and manipulate.

Object Hierarchy
The breakdown of an application into specific objects and object classes. To support the Core and Functional-area Suites, an application must define an object hierarchy based on standard classifications in the Apple Event Registry: Standard Suites.

Parameter
A method for identifying the object an Apple event is sent to, the task it will perform, and the desired options for performing the task.

Property
Characteristics used to describe Apple events objects in the same object class.