WIKIPEDIA

AppleScript

AppleScript is a scripting language created by Apple Inc. that facilitates automated control over scriptable Mac applications. First introduced in System 7, it is currently included in all versions of macOS as part of a package of system automation tools.[2][3] The term "AppleScript" may refer to the language itself, to an individual script written in the language, or, informally, to the macOS Open Scripting Architecture that underlies the language.[2][3]

AppleScript

Paradigm	Natural language programming, Scripting
Developer	Apple Inc.
First appeared	1993
Stable release	2.5 / October 16, 2014[1]
Typing discipline	Weak, dynamic
OS	System 7, Mac OS 8, Mac OS 9, macOS
License	Proprietary (parts available under APSL)
Filename extensions	.scpt, .scptd, .applescript
Website	https://developer.apple.com/applescript (https://developer.apple.com/library/content/documentation/AppleScript/Conceptual/AppleScriptX/AppleScriptX.html)
Influenced by	
Natural language, HyperTalk	

Contents

Overview
History
Basic concepts
Comments
Hello, world!
Natural language metaphor
Examples of scripts
Development tools
- Script editors
  - From Apple
  - From third parties
- Script launchers
- Related scripting issues

Language essentials
- Classes (data types)
- Language structures
  - Conditionals
  - Loops
- Miscellaneous information

Open Scripting Architecture
- JavaScript for Automation

See also
References
Further reading
External links

Overview

AppleScript is primarily a scripting language developed by Apple to do inter-application communication (IAC) using Apple events.[2][3] AppleScript is related to, but different from, Apple events. Apple events are designed to exchange data between and control other applications in order to automate repetitive tasks.

AppleScript has some processing abilities of its own, in addition to sending and receiving Apple events to applications. AppleScript can do basic calculations and text processing, and is extensible, allowing the use of scripting additions that add new functions to the language. Mainly, however, AppleScript relies on the functionality of applications and processes to handle complex tasks. As a structured command language, AppleScript can be compared to Unix shells, the Microsoft Windows Script Host, or IBM REXX in its purpose, but it is distinct from all three. Essential to its functionality is the fact that Macintosh applications publish "dictionaries" of addressable objects and operations.

AppleScript has some elements of procedural programming, object-oriented programming (particularly in the construction of script objects), and natural language programming tendencies in its syntax, but does not strictly conform to any of these programming paradigms.[3]:xxvi

History

In the late 1980s Apple considered using HyperCard's HyperTalk scripting language as the standard language for end-user development across the company and within its classic Mac OS operating system, and for interprocess communication between Apple and non-Apple products.[4] HyperTalk could be used by novices to program a HyperCard stack. Apple engineers recognized that a similar, but more object-oriented scripting language could be designed to be used with any application, and the AppleScript project was born as a spin-off of a research effort to modernize the Macintosh as a whole and finally became part of System 7.[5]

AppleScript was released in October 1993 as part of System 7.1.1 (System 7 Pro, the first major upgrade to System 7).[5] QuarkXPress (ver. 3.2) was one of the first major software applications that supported AppleScript. This in turn led to AppleScript being widely adopted within the publishing and prepress world, often tying together complex workflows. This was a key factor in retaining the Macintosh's dominant position in publishing and prepress, even after QuarkXpress and other publishing applications were ported to Microsoft Windows.

After some uncertainty about the future of AppleScript on Apple's next generation OS, the move to Mac OS X (around 2002) and its Cocoa frameworks greatly increased the usefulness and flexibility of AppleScript. Cocoa applications allow application developers to implement basic scriptability for their apps with minimal effort, broadening the number of applications that are directly scriptable. At the same time, the shift to the Unix underpinnings and AppleScript's ability to run Unix commands directly, with the `do shell script` command,[6] allowed AppleScripts much greater control over the operating system itself.[3]:863 AppleScript Studio, released with Mac OS X 10.2 as part of Xcode, and later AppleScriptObjC framework, released in Mac OS X 10.6, allowed users to build Cocoa applications using AppleScript.[3]:969

In a 2006 article, *Macworld* included AppleScript among its rankings of Apple's 30 most significant products to date, placing it at #17.[7]

In a 2013 article for *Macworld*, veteran Mac software developer and commentator John Gruber concluded his reflection on "the unlikely persistence of AppleScript" by noting: "In theory, AppleScript could be much better; in practice, though, it's the best thing we have that works. It exemplifies the Mac's advantages over iOS for tinkerers and advanced users."[8]

In October 2016, longtime AppleScript product manager and automation evangelist Sal Soghoian left Apple when his position was eliminated "for business reasons".[9] Veterans in the Mac community such as John Gruber and Andy Ihnatko generally responded with concern, questioning Apple's commitment to the developer community and pro users.[10] Apple senior vice president of software engineering Craig Federighi responded in an email saying that "We have every intent to continue our support for the great automation technologies in macOS!", though Jeff Gamet at *The Mac Observer* opined that it did little to assuage his doubt about the future of Apple automation in general and AppleScript in particular.[11] For the time being, AppleScript remains one component of macOS automation technologies, along with Services, Automator, and shell scripting.

Basic concepts

AppleScript was designed to be used as an accessible end-user scripting language, offering users an intelligent mechanism to control applications, and to access and modify data and documents. AppleScript uses Apple events, a set of standardized data formats that the Macintosh operating system uses to send information to applications, roughly analogous to sending XPath queries over XML-RPC in the world of web services.[3]:xxvi Apple events allow a script to work with multiple applications simultaneously, passing data between them so that complex tasks can be accomplished without human interaction.[2] For example, an AppleScript to create a simple web gallery might do the following:

1. Open a photo in a photo-editing application (by sending that application an *Open File* Apple event).
2. Tell the photo-editing application to manipulate the image (e.g. reduce its resolution, add a border, add a photo credit)
3. Tell the photo-editing application to save the changed image in a file in some different folder (by sending that application a *Save* and/or *Close* Apple event).
4. Send the new file path (via another Apple event) to a text editor or web editor application
5. Tell that editor application to write a link for the photo into an HTML file.
6. Repeat the above steps for an entire folder of images (hundreds or even thousands of photos).
7. Upload the HTML file and folder of revised photos to a website, by sending Apple events to a graphical FTP client, by using built-in AppleScript commands, or by sending Apple events to Unix FTP utilities.

For the user, hundreds or thousands of steps in multiple applications have been reduced to the single act of running the script, and the task is accomplished in much less time and with no possibility of random human error. A large complex script could be developed to run only once, while other scripts are used again and again.

An application's scriptable elements are visible in the application's Scripting Dictionary (distributed as part of the application), which can be viewed in any script editor. Elements are generally grouped into *suites*, according to loose functional relationships between them. There are two basic kinds of elements present in any suite: classes and commands.

- *Classes* are scriptable objects—for example, a text editing application will almost certainly have classes for windows, documents, and texts—and these classes will have properties that can be changed (window size, document background color, text font size, etc.), and may contain other classes (a window will contain one or more documents, a document will contain text, a text object will contain paragraphs and words and characters).
- *Commands*, by contrast, are instructions that can be given to scriptable objects. The general format for a block of AppleScript is to *tell* a scriptable object to run a command.

All scriptable applications share a few basic commands and objects, usually called the Standard Suite—commands to open, close or save a file, to print something, to quit, to set data to variables—as well as a basic *application* object that gives the scriptable properties of the application itself. Many applications have numerous suites capable of performing any task the application itself can perform. In exceptional cases, applications may support plugins which include their own scripting dictionaries.

AppleScript was designed with the ability to build scripts intuitively by recording user actions. Such AppleScript recordability has to be engineered into the app—the app must support Apple events and AppleScript recording;[12] as Finder supports AppleScript recording, it can be useful for reference. When AppleScript Editor (Script Editor) is open and the Record button clicked, user actions for recordable apps are converted to their equivalent AppleScript commands and output to the Script Editor window. The resulting script can be saved and re-run to duplicate the original actions, or modified to be more generally useful.

Comments

Comments can be made multiple ways. A one-line comment can begin with 2 hyphens (`--`). In AppleScript 2.0, first released in Mac OS X Leopard, it may also begin with a number sign (#). This permits a self-contained AppleScript script to be stored as an executable text file beginning with the shebang line `#!/usr/bin/osascript` Example:

```
--This is a one line comment
# So is this! (in Mac OS X Leopard or later)
```

For comments that take up multiple lines, AppleScript uses parentheses with asterisks inside. Example:

```
(* This is a
multiple
line
comment *)
```

Hello, world!

In AppleScript, the traditional "Hello, World!" program could be written in many different forms:

```
display dialog "Hello, world!" -- a modal window with "OK" and "Cancel" buttons
-- or
display alert "Hello, world!" -- a modal window with a single "OK" button and an icon
representing the app displaying the alert
-- or
say "Hello, world!" -- an audio message using a synthesized computer voice
```

AppleScript has several user interface options, including dialogs, alerts, and list of choices. (The character ¬, produced by typing ⌥ Option+return in the Script Editor, denotes continuation of a single statement across multiple lines.)

```
-- Dialog
set dialogReply to display dialog "Dialog Text" ¬
    default answer "Text Answer" ¬
    hidden answer false ¬
    buttons {"Skip", "Okay", "Cancel"} ¬
    default button "Okay" ¬
    cancel button "Skip" ¬
    with title "Dialog Window Title" ¬
    with icon note ¬
    giving up after 15
```

```
-- Choose from list
set chosenListItem to choose from list {"A", "B", "3"} ¬
    with title "List Title" ¬
    with prompt "Prompt Text" ¬
    default items "B" ¬
    OK button name "Looks Good!" ¬
    cancel button name "Nope, try again" ¬
    multiple selections allowed false ¬
    with empty selection allowed
```

```
-- Alert
set resultAlertReply to display alert "Alert Text" ¬
    as warning ¬
    buttons {"Skip", "Okay", "Cancel"} ¬
    default button 2 ¬
    cancel button 1 ¬
    giving up after 2
```

Each user interaction method can return the values of buttons clicked, items chosen or text entered for further processing. For example:

```
display alert "Hello, world!" buttons {"Rudely decline", "Happily accept"}
set theAnswer to button returned of the result
if theAnswer is "Happily accept" then
    beep 5
else
    say "Piffle!"
end if
```

Natural language metaphor

Whereas Apple events are a way to send messages into applications, AppleScript is a particular language designed to send Apple events. In keeping with the objective of ease-of-use for beginners, the AppleScript language is designed on the natural language metaphor, just as the graphical user interface is designed on the desktop metaphor. A well-written AppleScript should be clear enough to be read and understood by anyone, and easily edited. The language is based largely on HyperCard's HyperTalk language, extended to refer not only to the HyperCard world of cards and stacks, but also theoretically to any document. To this end, the AppleScript team introduced the AppleEvent Object Model (AEOM), which specifies the objects any particular application "knows".

The heart of the AppleScript language is the use of terms that act as nouns and verbs that can be combined. For example, rather than a different verb to print a page, document or range of pages (such as printPage, printDocument, printRange), AppleScript uses a single "print" verb which can be combined with an object, such as a page, a document or a range of pages.

```
print page 1

print document 2

print pages 1 thru 5 of document 2
```

Generally, AEOM defines a number of objects—like "document" or "paragraph"—and corresponding actions—like "cut" and "close". The system also defines ways to refer to properties of objects, so one can refer to the "third paragraph of the document 'Good Day'", or the "color of the last word of the front window". AEOM uses an application *dictionary* to associate the Apple events with human-readable terms, allowing the translation back and forth between human-readable AppleScript and bytecode Apple events. To discover what elements of a program are scriptable, dictionaries for supported applications may be viewed. (In the Xcode and Script Editor applications, this is under *File → Open Dictionary*.)

To designate which application is meant to be the target of such a message, AppleScript uses a "tell" construct:

```
tell application "Microsoft Word"
  quit
end tell
```

Alternatively, the tell may be expressed in one line by using an infinitive:

```
tell application "Microsoft Word" to quit
```

For events in the "Core Suite" (activate, open, reopen, close, print, and quit), the application may be supplied as the direct object to transitive commands:

```
quit application "Microsoft Word"
```

The concept of an object hierarchy can be expressed using nested blocks:

```
tell application "QuarkXPress"
  tell document 1
    tell page 2
      tell text box 1
        set word 5 to "Apple"
      end tell
    end tell
  end tell
end tell
```

The concept of an object hierarchy can also be expressed using nested prepositional phrases:

```
pixel 7 of row 3 of TIFF image "my bitmap"
```

which in another programming language might be expressed as sequential method calls, like in this pseudocode:

```
getTIFF("my bitmap").getRow(3).getPixel(7);
```

AppleScript includes syntax for ordinal counting, "the first paragraph", as well as cardinal, "paragraph one". Likewise, the numbers themselves can be referred to as text or numerically, "five", "fifth" and "5" are all supported; they are synonyms in AppleScript. Also, the word "the" can legally be used anywhere in the script in order to enhance readability: it has no effect on the functionality of the script.

Examples of scripts

A failsafe calculator:

```
tell application "Finder"
    -- Set variables
    set the1 to text returned of (display dialog "1st" default answer "Number here" buttons
{"Continue"} default button 1)
```

```
    set the2 to text returned of (display dialog "2nd" default answer "Number here" buttons
{"Continue"} default button 1)
    try
        set the1 to the1 as integer
        set the2 to the2 as integer
    on error
        display dialog "You may only input numbers into a calculator." with title "ERROR"
buttons {"OK"} default button 1
        return
    end try

    -- Add?
    if the button returned of (display dialog "Add?" buttons {"No", "Yes"} default button 2) is
"Yes" then
        set ans to (the1 + the2)
        display dialog ans with title "Answer" buttons {"OK"} default button 1
        say ans
    -- Subtract?
    else if the button returned of (display dialog "Subtract?" buttons {"No", "Yes"} default
button 2) is "Yes" then
        set ans to (the1 - the2)
        display dialog ans with title "Answer" buttons {"OK"} default button 1
        say ans
    -- Multiply?
    else if the button returned of (display dialog "Multiply?" buttons {"No", "Yes"} default
button 2) is "Yes" then
        set ans to (the1 * the2)
        display dialog ans with title "Answer" buttons {"OK"} default button 1
        say ans
    -- Divide?
    else if the button returned of (display dialog "Divide?" buttons {"No", "Yes"} default
button 2) is "Yes" then
        set ans to (the1 / the2)
        display dialog ans with title "Answer" buttons {"OK"} default button 1
        say ans
    else
        delay 1
        say "You haven't selected a function. The operation has cancelled."
    end if

end tell
```

A simple username and password dialog box sequence. Here, the username is John and password is app123:

```
tell application "Finder"
    set passAns to "app123"
    set userAns to "John"
    if the text returned of (display dialog "Username" default answer "") is userAns then
        display dialog "Correct" buttons {"Continue"} default button 1
        if the text returned of (display dialog "Username : John" & return & "Password" default
answer "" buttons {"Continue"} default button 1 with hidden answer) is passAns then
            display dialog "Access granted" buttons {"OK"} default button 1
        else
            display dialog "Incorrect password" buttons {"OK"} default button 1
        end if
    else
        display dialog "Incorrect username" buttons {"OK"} default button 1
    end if
end tell
```

Development tools

Script editors

Script editors provide a unified programing environment for AppleScripts, including tools for composing, validating, compiling, running, and debugging scripts. They also provide mechanisms for opening and viewing AppleScript dictionaries from scriptable applications, saving scripts in a number of formats (compiled script files, application packages, script bundles, and plain text files), and usually provide features such as syntax highlighting and prewritten code snippets.

From Apple

AppleScript Editor (Script Editor)

The editor for AppleScript packaged with macOS, called *AppleScript Editor* in Mac OS X Snow Leopard (10.6) through OS X Mavericks (10.9) and *Script Editor* in all earlier and later versions of macOS. Scripts are written in document editing windows where they can be compiled and run, and these windows contain various panes in which logged information, execution results, and other information is available for debugging purposes. Access to scripting dictionaries and prewritten code snippets is available through the application menus. Since OS X Yosemite (10.10), Script Editor includes the ability to write in both AppleScript and JavaScript.[13]

Xcode

A suite of tools for developing applications with features for editing AppleScripts or creating full-fledged applications written with AppleScript.

From third parties

Script Debugger, from Late Night Software

A third-party commercial IDE for AppleScript. Script Debugger is a more advanced AppleScript environment that allows the script writer to debug AppleScripts via single stepping, breakpoints, stepping in and out of functions/subroutines, variable tracking, etc. Script Debugger also contains an advanced dictionary browser that allows the user to see the dictionary in action in real world situations. That is, rather than just a listing of what the dictionary covers, one can open a document in Pages, for example, and see how the dictionary's terms apply to that document, making it easier to determine which parts of the dictionary to use. Script Debugger is not designed to create scripts with a GUI, other than basic alerts and dialogs, but is focused more on the coding and debugging of scripts.

Smile and SmileLab

A third-party freeware/commercial IDE for AppleScript, itself written entirely in AppleScript.[14] Smile is free, and primarily designed for AppleScript development. SmileLab is commercial software with extensive additions for numerical analysis, graphing, machine automation and web production. Smile and SmileLab use an assortment of different windows—AppleScript windows for running and saving full scripts, AppleScript terminals for testing code line-by-line, unicode windows for working with text and XML. Users can create complex interfaces—called dialogs—for situations where the built-in dialogs in AppleScript are insufficient.

ASObjC Explorer 4, from Shane Stanley

A discontinued third-party commercial IDE for AppleScript, especially for AppleScriptObjC.[15] The main feature is Cocoa-object/event logging, debugging and code-completion. Users can read Cocoa events and objects like other scriptable applications. This tool was originally built for AppleScript Libraries (available in OS X Mavericks). AppleScript Libraries aims for re-usable AppleScript components and supports built-in AppleScript dictionary (sdef). ASObjC Explorer 4 can be an external Xcode script editor, too.

FaceSpan, from Late Night Software

A discontinued third-party commercial IDE for creating AppleScript applications with graphic user interfaces.[16]

Script launchers

AppleScripts can be run from a script editor, but it is usually more convenient to run scripts directly, without opening a script editor application. There are a number of options for doing so:

Applets

AppleScripts can be saved from a script editor as applications (called *applets*, or *droplets* when they accept input via drag and drop).[3]:69 Applets can be run from the Dock, from the toolbar of Finder windows, from Spotlight, from third-party application launchers, or from any other place where applications can be run.

Folder actions

Using AppleScript folder actions, scripts can be launched when specific changes occur in folders (such as adding or removing files).[17] Folder actions can be assigned by clicking on a folder and choosing *Folder Actions Setup...* from the contextual menu; the location of this command differs slightly in Mac OS X 10.6.x from earlier versions. This same action can be achieved with third-party utilities such as Hazel.[18]

Hotkey launchers

Keyboard shortcuts can be assigned to AppleScripts in the script menu using the *Keyboard & Mouse Settings* Preference Pane in System Preferences. In addition, various third-party utilities are available—Alfred,[19] FastScripts,[20] Keyboard Maestro,[21] QuicKeys,[22] Quicksilver,[23] TextExpander[24]—which can run AppleScripts on demand using key combinations.

Script menu

This system-wide menu provides access to AppleScripts from the macOS menu bar, visible no matter what application is running.[25] (In addition, many Apple applications, some third party applications, and some add-ons provide their own script menus. These may be activated in different ways, but all function in essentially the same manner.) Selecting a script in the script menu launches it. Since Mac OS X 10.6.x, the system-wide script menu can be enabled from the preferences of Script Editor; in prior versions of Mac OS X, it could be enabled from the AppleScript Utility application. When first enabled, the script menu displays a default library of fairly generic, functional AppleScripts, which can also be opened in Script Editor and used as examples for learning AppleScript. Scripts can be organized so that they only appear in the menu when particular applications are in the foreground.

Unix command line and launchd

AppleScripts can be run from the Unix command line, or from launchd for scheduled tasks,[3]:716 by using the osascript command line tool.[26] The osascript tool can run compiled scripts (.scpt files) and plain text files (.applescript files—these are compiled by the tool at runtime). Script applications can be run using the Unix open command.

Related scripting issues

AppleScript Libraries

Re-usable AppleScript modules (available since OS X Mavericks), written in AppleScript or AppleScriptObjC and saved as script files or bundles in certain locations,[27] that can be called from other scripts. When saved as a bundle, a library can include an AppleScript dictionary (sdef) file,[28] thus functioning like a scripting addition but written in AppleScript or AppleScriptObjC.

AppleScript Studio

A framework for attaching Cocoa interfaces to AppleScript applications, part of the Xcode package in Mac OS X 10.4 and 10.5, now deprecated in favor of AppleScriptObjC.[29]:438

AppleScriptObjC

A Cocoa development software framework, also called AppleScript/Objective-C or ASOC,[30] part of the Xcode package since Mac OS X Snow Leopard.[31] AppleScriptObjC allows AppleScripts to use Cocoa classes and methods directly.[32] The following table shows the availability of AppleScriptObjC in various versions of macOS:[33]

Where AppleScriptObjC can be used in each macOS version

	In Xcode	In applets	In AppleScript Libraries	In Script Editor
10.6	✓			
10.7	✓	✓		
10.8	✓	✓		
10.9	✓	✓	✓	
10.10	✓	✓	✓	✓

Automator

A graphical, modular editing environment in which *workflows* are built up from *actions.* It is intended to duplicate many of the functions of AppleScript without the necessity for programming knowledge. Automator has an action specifically designed to contain and run AppleScripts, for tasks that are too complex for Automator's simplified framework.[34]

Scriptable core system applications

These background-only applications, packaged with macOS, are used to allow AppleScript to access features that would not normally be scriptable. As of Mac OS X 10.6.3 they include the scriptable applications for VoiceOver (scriptable auditory and braille screen reader package), System Events (control of non-scriptable applications and access to certain system functions and basic file operations), Printer Setup Utility (scriptable utility for handling print jobs), Image Events (core image manipulation), HelpViewer (scriptable utility for showing help displays), Database Events (minimal SQLite3 database interface), and AppleScript Utility (for scripting a few AppleScript related preferences), as well as a few utility applications used by the system.

Scripting Additions (OSAX)

Plug-ins for AppleScript developed by Apple or third parties.[35] They are designed to extend the built-in command set, expanding AppleScript's features and making it somewhat less dependent on functionality provided by applications. macOS includes a collection of scripting additions referred to as Standard Additions (*StandardAdditions.osax*) that adds a set of commands and classes that are not part of AppleScript's core features, including user interaction dialogs, reading and writing files, file system commands, date functions, and text and mathematical operations; without this OSAX, AppleScript would have no capacity to perform many basic actions not directly provided by an application.

Language essentials

Classes (data types)

While applications can define specialized classes (or data types), AppleScript also has a number of built-in classes. These basic data classes are directly supported by the language and tend to be universally recognized by scriptable applications. The most common ones are as follows:

- Basic objects
  - **application**: an application object, used mostly as a specifier for tell statements (`tell application "Finder" …`).
  - **script**: a script object. Script objects are containers for scripts. Every AppleScript creates a script object when run, and script objects may be created within AppleScripts.
  - **class**: a meta-object that specifies the type of other objects.
  - **reference**: an object that encapsulates an unevaluated object specifier that may or may not point to a valid object. Can be evaluated on-demand by accessing its `contents` property.
- Standard data objects
  - **constant**: a constant value. There are a number of language-defined constants, such as `pi`, `tab`, and `linefeed`.
  - **boolean**: a Boolean true/false value. Actually a subclass of `constant`.
  - **number**: a rarely used abstract superclass of `integer` and `real`.
  - **integer**: an integer. Can be manipulated with built-in mathematical operators.
  - **real**: a floating-point (real) number. Can be manipulated with built-in mathematical operators.
  - **date**: a date and time.
  - **text**: text. In versions of AppleScript before 2.0 (Mac OS X 10.4 and below) the `text` class was distinct from `string` and `Unicode text`, and the three behaved somewhat differently; in 2.0 (10.5) and later, they are all synonyms and all text is handled as being UTF-16 ("Unicode")-encoded.[36]
- Containers
  - **list**: an ordered list of objects. Can contain any class, including other lists and classes defined by applications.
  - **record**: a keyed list of objects. Like a list, except structured as key-value pairs. Runtime keyed access is unsupported; all keys must be compile-time constant identifiers.
- File system
  - **alias**: a reference to a file system object (file or folder). The alias will maintain its link to the object if the object is moved or renamed.
  - **file**: a reference to a file system object (file or folder). This is a static reference, and can point to an object that does not currently exist.
  - **POSIX file**: a reference to a file system object (file or folder), in plain text, using Unix (POSIX)-style slash (/) notation. Not a true data type, as AppleScript automatically converts POSIX files to ordinary files whenever they are used.[37]
- Miscellaneous
  - **RGB color**: specifies an RGB triplet (in 16-bit high color format), for use in commands and objects that work with colors.
  - **unit types**: class that converts between standard units. For instance, a value can be defined as `square yards`, then converted to `square feet` by casting between unit types (using the `as` operator).

Language structures

Many AppleScript processes are managed by blocks of code, where a block begins with a command *command* and ends with an *end command* statement. The most important structures are described below.

Conditionals

AppleScript offers two kinds of conditionals.

```
-- Simple conditional
if x < 1000 then set x to x + 1

-- Compound conditional
if x is greater than 3 then
     -- commands
else
     -- other commands
end if
```

Loops

The repeat loop of AppleScript comes in several slightly different flavors. They all execute the block between **repeat** and **end repeat** lines a number of times. The looping can be prematurely stopped with command **exit repeat**.

Repeat forever.

```
repeat
     -- commands to be repeated
end repeat
```

Repeat a given number of times.

```
repeat 10 times
     -- commands to be repeated
end repeat
```

Conditional loops. The block inside **repeat while** loop executes as long as the condition evaluates to true. The condition is re-evaluated after each execution of the block. The **repeat until** loop is otherwise identical, but the block is executed as long as the condition evaluates to false.

```
set x to 5
repeat while x > 0
     set x to x - 1
end repeat

set x to 5
repeat until x ≤ 0
     set x to x - 1
end repeat
```

Loop with a variable. When starting the loop, the variable is assigned to the start value. After each execution of the block, the optional step value is added to the variable. Step value defaults to 1.

```
-- repeat the block 2000 times, i gets all values from 1 to 2000
repeat with i from 1 to 2000
     -- commands to be repeated
end repeat

-- repeat the block 4 times, i gets values 100, 75, 50 and 25
repeat with i from 100 to 25 by -25
    -- commands to be repeated
end repeat
```

Enumerate a list. On each iteration set the loopVariable to a new item in the given list

```
set total to 0
repeat with x in {1, 2, 3, 4, 5}
    set total to total + x
end repeat
```

Application targeting

```
-- Simple form
tell application "Safari" to activate

-- Compound
tell application "MyApp"
     -- commands for app
end tell
```

Error handling

```
try
     -- commands to be tested
on error
     -- error commands
end try
```

One important variation on this block structure is in the form of **on —end ...** blocks that are used to define *handlers* (function-like subroutines). Handlers begin with *on functionName()* and ending with *end functionName*, and are not executed as part of the normal script flow unless called from somewhere in the script.

Function handler

```
on
myFunction(parameters...)
     -- subroutine
commands
end myFunction
```

Folder actions block

```
on adding folder items to thisFolder after
receiving theseItems
     -- commands to apply to the folder or
items
end adding folder items to
```

Run handler

```
on run
     --
commands
end run
```

Handlers can also be defined using "to" in place of "on" and can be written to accept labeled parameters, not enclosed in parens.

Handler with labeled parameters

```
on rock around the clock
    display dialog (clock as
string)
end rock
-- called with:
rock around the current date
```

Handler using "to" and labeled parameters

```
to check for yourNumber from bottom thru top
    if bottom ≤ yourNumber and yourNumber ≤ top then
        display dialog "Congratulations! You
scored."
    end if
end check
--called with:
check for 8 from 7 thru 10
```

There are four types of predefined handlers in AppleScript—run, open, idle, and quit—each of which is created in the same way as the run handler shown above.

Run handler

Defines the main code of the script, which is called when the script is run. Run handler blocks are optional, unless arguments are being passed to the script. If an explicit run handler block is omitted, then all code that is not contained inside handler blocks is executed as though it were in an implicit run handler.

Open handler

Defined using "on open theItems".

```
on open theItems
     repeat with thisItem in theItems
         tell application "Finder" to update thisItem
     end repeat
end open
```

When a script containing an "open handler' is saved as an applet, the applet becomes a droplet. A droplet can be identified in the Finder by its icon, which includes an arrow, indicating items can be dropped onto the icon. The droplet's open handler is executed when files or folders are dropped onto droplet's icon. References to the items dropped on the droplet's icon are passed to the droplet's script as the parameter of the open handler. A droplet can also be launched the same way as an ordinary applet, executing its run handler.

Idle handler

A subroutine that is run periodically by the system when the application is idle.

```
on idle
     --code to execute when the script's execution has completed
  return 60 -- number of seconds to pause before executing idle handler again
end idle
```

An idle handler can be used in applets or droplets saved as stay-open applets, and is useful for scripts that watch for particular data or events. The length of the idle time is 30 seconds by default,[38] but can be changed by including a 'return x' statement at the end of the subroutine, where x is the number of seconds the system should wait before running the handler again.

Quit handler

A handler that is run when the applet receives a Quit request. This can be used to save data or do other ending tasks before quitting.

```
on quit
     --commands to execute before the script quits
  continue quit -- required for the script to actually quit
end quit
```

Script objects

Script objects may be defined explicitly using the syntax:

```
script scriptName
     -- commands and handlers specific to the script
end script
```

Script objects can use the same 'tell' structures that are used for application objects, and can be loaded from and saved to files. Runtime execution time can be reduced in some cases by using script objects.

Miscellaneous information

- Variables are not strictly typed, and do not need to be declared. Variables can take any data type (including scripts and functions). The following commands are examples of the creation of variables:

```
set variable1 to 1 -- create an integer variable called variable1
set variable2 to "Hello" -- create a text variable called variable2
copy {17, "doubleday"} to variable3 -- create a list variable called variable3
set {variable4, variable5} to variable3 -- copy the list items of variable3 into separate
variables variable4 and variable5
set variable6 to script myScript -- set a variable to an instance of a script
```

- Script objects are full objects—they can encapsulate methods and data and inherit data and behavior from a parent script.
- Subroutines cannot be called directly from application tell blocks. Use the 'my' or 'of me' keywords to do so.

```
tell application "Finder"
    set x to my myHandler()
    -- or
    set x to myHandler() of me
end tell

on myHandler()
    --commands
end myHandler
```

Using the same technique for scripting addition commands can reduce errors and improve performance.

```
tell application "Finder"
    set anyNumber to my (random number from 5 to 50)
end tell
```

Open Scripting Architecture

An important aspect of the AppleScript implementation is the **Open Scripting Architecture** (**OSA**).[39] Apple provides OSA for other scripting languages and third-party scripting/automation products such as QuicKeys and UserLand Frontier, to function on an equal status with AppleScript. AppleScript was implemented as a scripting component, and the basic specs for interfacing such components to the OSA were public, allowing other developers to add their own scripting components to the system. Public client APIs for loading, saving and compiling scripts would work the same for all such components, which also meant that applets and droplets could hold scripts in any of those scripting languages.

One feature of the OSA is scripting additions, or OSAX for *Open Scripting Architecture eXtension*,[35] which were inspired by HyperCard's External Commands. Scripting additions are libraries that allow programmers to extend the function of AppleScript. Commands included as scripting additions are available system-wide, and are not dependent on an application (see also § AppleScript Libraries). The AppleScript Editor is also able to directly edit and run some of the OSA languages.

JavaScript for Automation

Under OS X Yosemite and later versions of macOS, the **JavaScript for Automation** (**JXA**) component remains the only serious OSA language alternative to AppleScript,[13] though the Macintosh versions of Perl, Python, Ruby, and Tcl all support native means of working with Apple events without being OSA components.[29]:516

JXA also provides an Objective-C (and C language) foreign language interface.[13] Being an environment based on WebKit's JavaScriptCore engine, the JavaScript feature set is in sync with the system Safari browser engine. JXA provides a JavaScript module system and it is also possible to use CommonJS modules via browserify.[40]

See also

- BBEdit — a highly scriptable text editor

References

1. "OS X 10.10 Yosemite release date" (https://www.theverge.com/2014/10/16/6978157/mac-os-x-yosemite-release-mac-app-store-october-16th). Retrieved November 16, 2014.
2. Goldstein, Adam (2005). *AppleScript: the missing manual* (https://books.google.com/books?id=-ynfWvkwzpwC). Sebastopol, CA: O'Reilly Media. ISBN 0596008503. OCLC 56912218 (https://www.worldcat.org/oclc/56912218).
3. Sanderson, Hamish; Rosenthal, Hanaan (2009). *Learn AppleScript: the comprehensive guide to scripting and automation on Mac OS X* (https://books.google.com/books?id=_40AkCe9nOUC) (3rd ed.). Berkeley: Apress. doi:10.1007/978-1-4302-2362-7_27 (https://doi.org/10.1007%2F978-1-4302-2362-7_27). ISBN 9781430223610. OCLC 308193726 (https://www.worldcat.org/oclc/308193726).
4. Flynn, Laurie (February 27, 1989). "Apple Ponders Standardizing on HyperTalk" (https://books.google.com/books?id=lToEAAAAMBAJ&lpg=PT30&ots=Udk9JVeM1G&pg=PT30#v=onepage&q&f=true). *InfoWorld*. p. 31.
5. Cook, William (2007). "AppleScript" (http://www.cs.utexas.edu/~wcook/Drafts/2006/ashopl.pdf) (PDF). *History of Programming Languages (HOPL III)*. Proceedings of the third ACM SIGPLAN conference. Association for Computing Machinery: 1–21. doi:10.1145/1238844.1238845 (https://doi.org/10.1145%2F1238844.1238845).
6. "AppleScript Language Guide commands reference: do shell script" (https://developer.apple.com/library/archive/documentation/AppleScript/Conceptual/AppleScriptLangGuide/reference/ASLR_cmds.html#//apple_ref/doc/uid/TP40000983-CH216-SW40). *developer.apple.com*. Retrieved September 15, 2019.
7. "Apple's greatest hits: 30 significant products" (https://www.macworld.com/article/1050178/30applelist.html). *macworld.com*. Retrieved September 12, 2019.
8. Gruber, John (March 2013). "The unlikely persistence of AppleScript" (http://www.macworld.com/article/2018607/the-unlikely-persistence-of-applescript.html). *Macworld*. **30** (3): 100.
9. Lovejoy, Ben (November 17, 2016). "Sal Soghoian says 'ask Apple' about future of Mac user automation as company eliminates position" (https://9to5mac.com/2016/11/17/mac-user-automation-sal-soghoian/). *9to5mac.com*. Retrieved May 8, 2017.
10. Evans, Jonny (November 17, 2016). "Does Apple really want to kill Automator, AppleScript? Shock termination of veteran Apple developer guru sends shockwaves across the Mac community" (http://www.computerworld.com/article/3142666/apple-mac/does-apple-really-want-to-kill-automator-applescript.html). *Computerworld.com*. Retrieved May 8, 2017.
11. Gamet, Jeff (November 23, 2016). "Apple's intent isn't the same as committing to mac automation" (https://www.macobserver.com/analysis/apple-automation-commitment/). *macobserver.com*. Retrieved May 8, 2017.
12. "Scriptable Applications" (https://developer.apple.com/library/archive/documentation/AppleScript/Conceptual/AppleScriptX/Concepts/scriptable_apps.html#//apple_ref/doc/uid/TP40001569-1153888-BAJICJEG). *developer.apple.com*. Retrieved July 26, 2018.
13. Siracusa, John (October 16, 2014). "OS X 10.10 Yosemite: The Ars Technica Review: JavaScript automation" (https://arstechnica.com/apple/2014/10/os-x-10-10/24/#javascript-automation). *Ars Technica*. Retrieved May 8, 2017.
14. "Smile and SmileLab Home Page" (http://www.satimage.fr/software/en/). *satimage.fr*. Retrieved May 8, 2017.
15. "ASObjC Explorer 4 Discontinued" (https://web.archive.org/web/20170621152438/http://www.macosxautomation.com/applescript/apps/explorer.html). *macosxautomation.com*. Archived from the original (https://www.macosxautomation.com/applescript/apps/explorer.html) on June 21, 2017. Retrieved May 8, 2017.

16. "Mark Alldritt's Journal » FaceSpan" (http://blog.latenightsw.com/?cat=10). *blog.latenightsw.com*. Retrieved May 8, 2017.
17. "AppleScript Language Guide: Folder Actions Reference" (https://developer.apple.com/library/content/documentation/AppleScript/Conceptual/AppleScriptLangGuide/reference/ASLR_folder_actions.html). *developer.apple.com*. Retrieved May 8, 2017.
18. Miller, Dan (December 22, 2010). "Capsule review: Hazel 2.3" (http://www.macworld.com/article/1156565/hazel.html). *Macworld.com*. Retrieved May 8, 2017.
19. Beam, Brian (February 10, 2015). "Alfred review: This Mac app launcher continues to shine, but Alfred Remote doesn't stack up" (http://www.macworld.com/article/2881953/alfred-review-this-mac-app-launcher-continues-to-shine-but-alfred-remote-doesnt-stack-up.html). *Macworld.com*. Retrieved May 10, 2017.
20. Frakes, Dan (June 2, 2011). "Capsule review: FastScripts 2.5" (http://www.macworld.com/article/1160256/fastscripts_25.html). *Macworld.com*. Retrieved May 8, 2017.
21. Breen, Christopher (June 4, 2013). "Mac Gems: Keyboard Maestro 6 is a genius at repetitive tasks" (http://www.macworld.com/article/2040496/mac-gems-keyboard-maestro-6-is-a-genius-at-repetitive-tasks.html). *Macworld.com*. Retrieved May 10, 2017.
22. Breen, Christopher (May 7, 2010). "Capsule review: QuicKeys 4" (http://www.macworld.com/article/1150918/quickeys4_review.html). *Macworld.com*. Retrieved May 8, 2017.
23. "AppleScripts – Quicksilver Wiki" (https://qsapp.com/wiki/AppleScripts). *qsapp.com*. Retrieved May 10, 2017.
24. Fleishman, Glenn (June 12, 2015). "TextExpander 5 review" (http://www.macworld.com/article/2931533/textexpander-5-review-typing-shortcut-utility-makes-you-more-productive-by-learning-your-habits.html). *Macworld.com*. Retrieved May 8, 2017.
25. "Mac Automation Scripting Guide: Using the Systemwide Script Menu" (https://developer.apple.com/library/content/documentation/LanguagesUtilities/Conceptual/MacAutomationScriptingGuide/UsetheSystem-WideScriptMenu.html). *developer.apple.com*. Retrieved May 8, 2017.
26. "osascript(1) Mac OS X Manual Page" (https://developer.apple.com/legacy/library/documentation/Darwin/Reference/ManPages/man1/osascript.1.html). *developer.apple.com*. Retrieved May 8, 2017.
27. "AppleScript Libraries" (http://www.macosautomation.com/mavericks/libraries/). *macosautomation.com*. Retrieved May 8, 2017.
28. "WWDC 2013 Session 416: Introducing AppleScript Libraries" (https://asciiwwdc.com/2013/sessions/416). *asciiwwdc.com*. Retrieved May 8, 2017.
29. Neuburg, Matt (2006). *AppleScript: the definitive guide* (2nd ed.). Beijing: O'Reilly Media. ISBN 0596102119. OCLC 68694976 (https://www.worldcat.org/oclc/68694976).
30. Tsai, Michael (September 29, 2014). "AppleScript and Yosemite" (https://mjtsai.com/blog/2014/10/29/applescript-and-yosemite/). *mjtsai.com*. Retrieved May 9, 2017.
31. "AppleScriptObjC Release Notes" (https://developer.apple.com/library/content/releasenotes/ScriptingAutomation/RN-AppleScriptObjC/). *developer.apple.com*. Retrieved May 8, 2017.
32. Waldie, Ben (September 6, 2012). "Building a basic AppleScriptObjC (Cocoa-AppleScript) application with Xcode" (http://www.peachpit.com/articles/article.aspx?p=1942301). *peachpit.com*. Retrieved May 9, 2017.
33. Table adapted from: Naganoya, Takaaki. "ASObjCExtras Scripting Guide" (http://www.piyocast.com/download_file/ASObjCExtras_scripting_guide.pdf) (PDF). *piyocast.com*. Retrieved May 9, 2017.
34. "Introduction to Automator AppleScript Actions Tutorial" (https://developer.apple.com/library/content/documentation/AppleApplications/Conceptual/AutomatorTutorialAppleScript/Introduction/Introduction.html). *developer.apple.com*. Retrieved May 8, 2017.
35. "Technical Note TN1164: Scripting Additions for Mac OS X" (https://developer.apple.com/library/content/technotes/tn1164/_index.html). *developer.apple.com*. Retrieved May 8, 2017.

36. "AppleScript Release Notes: 10.5 Changes" (https://developer.apple.com/mac/library/releasenotes/AppleScript/RN-AppleScript/RN-10_5/RN-10_5.html). *developer.apple.com*. Retrieved May 8, 2017.
37. "POSIX file — Class Reference — AppleScript Language Guide" (https://developer.apple.com/library/content/documentation/AppleScript/Conceptual/AppleScriptLangGuide/reference/ASLR_classes.html#//apple_ref/doc/uid/TP40000983-CH1g-SW15). *developer.apple.com*. Retrieved January 8, 2018.
38. "AppleScript Language Guide: Handlers in Script Applications" (https://developer.apple.com/library/mac/#documentation/AppleScript/Conceptual/AppleScriptLangGuide/conceptual/ASLR_about_handlers.html#//apple_ref/doc/uid/TP40000983-CH206-SW14). *developer.apple.com*. Retrieved July 21, 2013.
39. "AppleScript Overview: Open Scripting Architecture" (https://developer.apple.com/library/content/documentation/AppleScript/Conceptual/AppleScriptX/Concepts/osa.html). *developer.apple.com*. Retrieved May 8, 2017.
40. "Importing Scripts" (https://github.com/JXA-Cookbook/JXA-Cookbook/wiki/Importing-Scripts). *GitHub*. JXA-Cookbook. December 6, 2019. Retrieved December 9, 2019.

Further reading

- Munro, Mark Conway (2010). *AppleScript*. Developer Reference. Wiley. ISBN 978-0-470-56229-1.
- Rosenthal, Hanaan; Sanderson, Hamish (2010). *Learn AppleScript: The Comprehensive Guide to Scripting and Automation on Mac OS X* (Third ed.). Apress. ISBN 978-1-4302-2361-0.
- Soghoian, Sal; Cheeseman, Bill (2009). *Apple Training Series: AppleScript 1-2-3*. Peachpit Press. ISBN 978-0-321-14931-2.
- Cook, William (2007). "AppleScript" (http://www.cs.utexas.edu/~wcook/Drafts/2006/ashopl.pdf) (PDF). *History of Programming Languages (HOPL III)*. Proceedings of the third ACM SIGPLAN conference. ACM: 1–21. doi:10.1145/1238844.1238845 (https://doi.org/10.1145%2F1238844.1238845).
- Ford Jr., Jerry Lee (2007). *AppleScript Programming for the Absolute Beginner*. Course Technology. ISBN 978-1-59863-384-9.
- Neuburg, Matt (2006). *AppleScript: The Definitive Guide*. O'Reilly Media. ISBN 0-596-10211-9.
- Goldstein, Adam (2005). *AppleScript: The Missing Manual*. O'Reilly Media. ISBN 0-596-00850-3.
- Trinko, Tom (2004). *AppleScript for Dummies*. For Dummies. ISBN 978-0-7645-7494-8.

External links

- Official website (https://developer.apple.com/applescript/)
- AppleScript (https://curlie.org/Computers/Systems/Apple/Macintosh/Development/Languages/Scripting/AppleScript/) at Curlie
- "AppleScript for Python Programmers (Comparison Chart)" (http://aurelio.net/articles/applescript-vs-python.html). *aurelio.net*. 2005. Retrieved May 9, 2017.
- "AppleScript Language Guide [html]" (https://developer.apple.com/mac/library/documentation/AppleScript/conceptual/AppleScriptlangguide/introduction/ASLR_intro.html). *developer.apple.com*. 2016. Retrieved May 9, 2017.
- *AppleScript Language Guide [pdf]*. Apple Inc. 2015. CiteSeerX 10.1.1.697.5220 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.697.5220).
- "Doug's AppleScripts for iTunes" (http://dougscripts.com/itunes/). *dougscripts.com*. Retrieved May 9, 2017.

- "Mac OS X Automation" (http://macosautomation.com/). *macosautomation.com*. Retrieved May 9, 2017.
- "MacScripter AppleScript community" (http://macscripter.net/). *macscripter.net*. Retrieved May 9, 2017.
