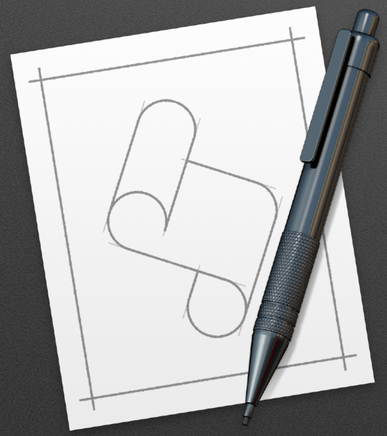
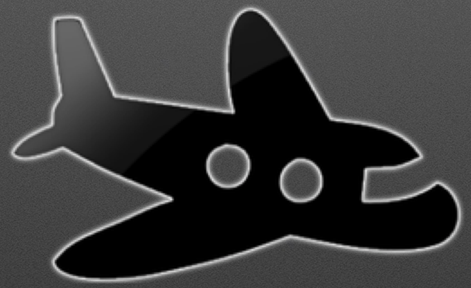
# Automating Your Mac

Ryan Eisworth and John Gaver
Houston Area Apple Users Group
August 2016



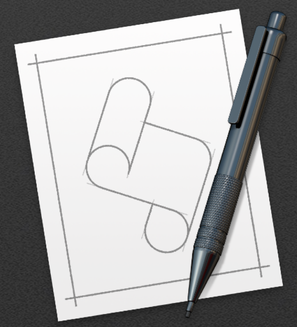


# Introduction

- We all probably perform some form of repetitive task on our Mac.
- Mac OS has some powerful features built-in that enable scripting and automation: *AppleScript, Automator, UNIX shell scripts,* and *Services.*
- There are also some third-party applications that can help automate our computers. We’re going to look at one: *ControlPlane.*

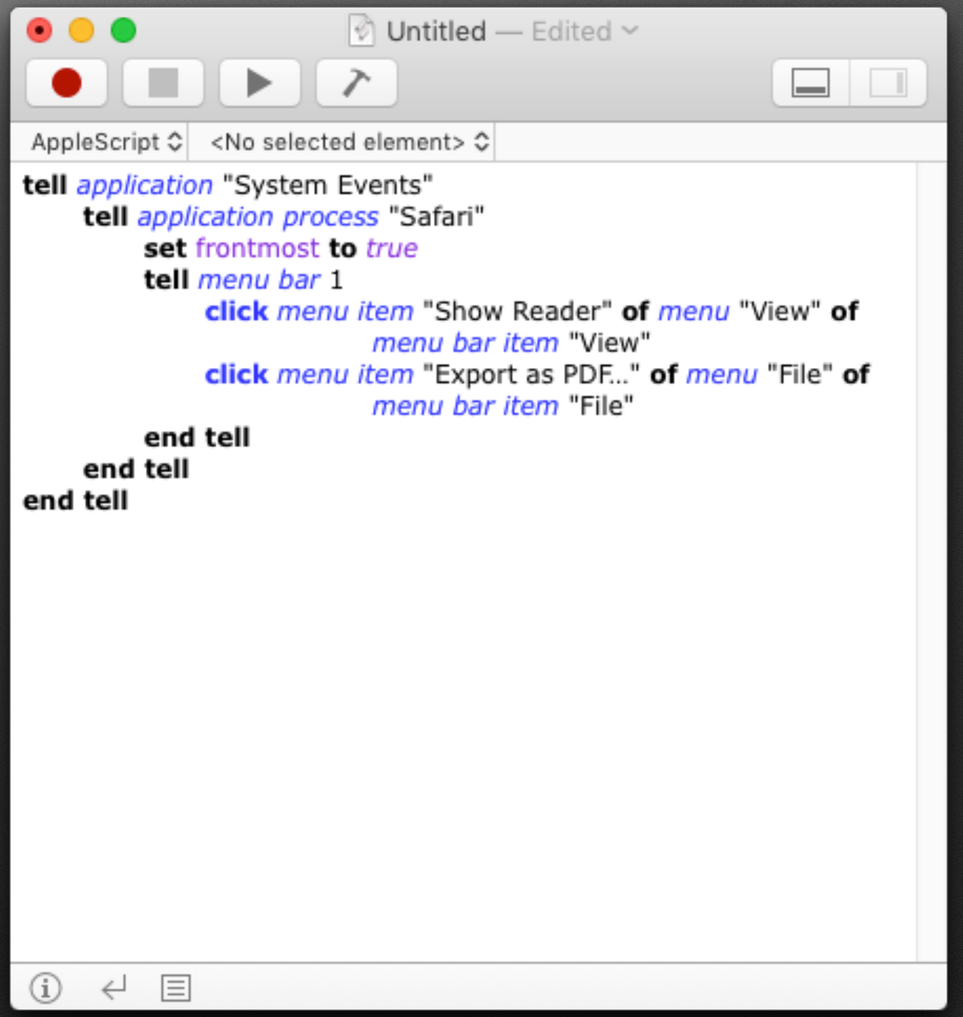
# AppleScript

- AppleScript is a scripting language that enables powerful automation of tasks in both the Mac OS and scriptable applications.
- It has been a part of the Macintosh OS since the System 7 days.
- Scripts send AppleEvents to the operating system or applications.
- Much more powerful than macros!



# AppleScript

- AppleScript is an English-like, human-readable language, but it can still be frustrating for new users.
- Many example scripts online make great starting points for new scripts.
- You need to know a few basic keywords to write scripts.
- AppleScript Dictionaries are essential to discover scriptable parts of an application.



# Keywords

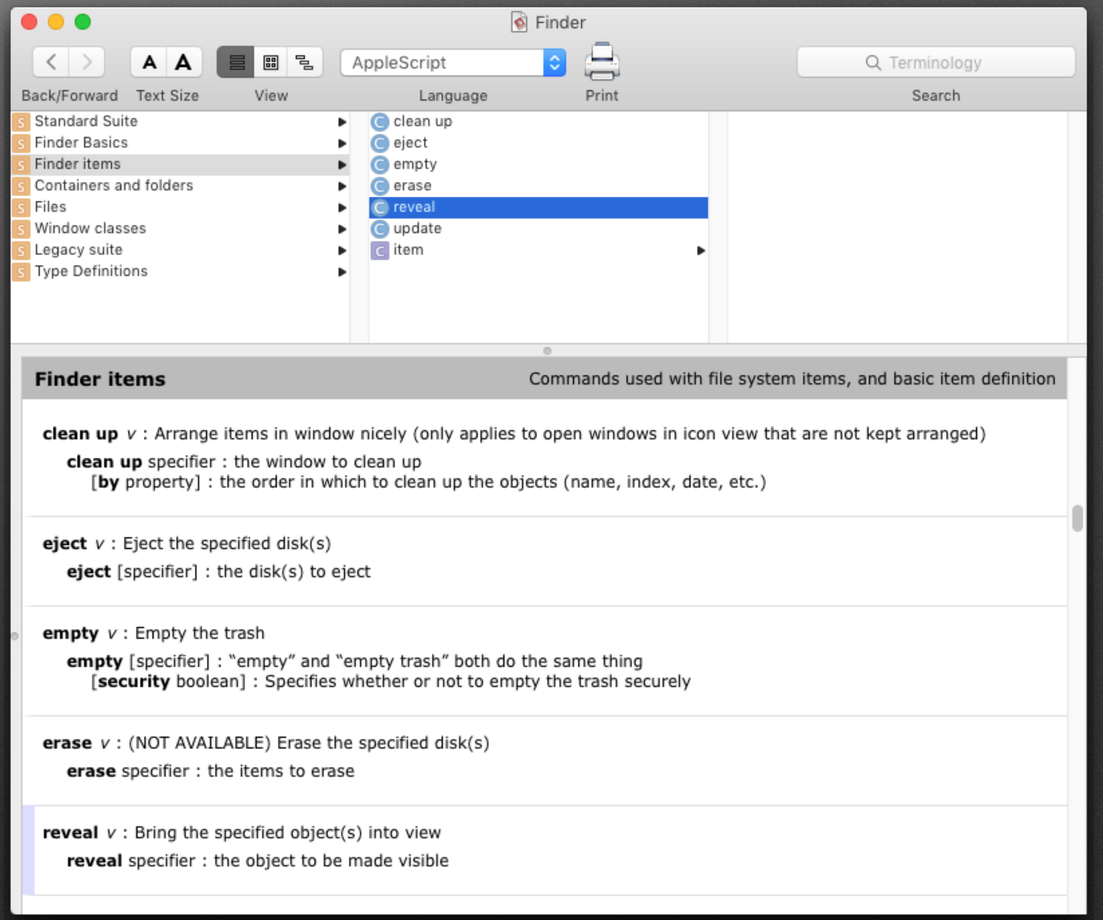
- You can think of keywords as the most basic “commands” in AppleScript.
- These special keywords are used to tell the AppleScript engine what to do and to control script flow.
- Keywords are reserved words in AppleScript.
- Basic and often used: *tell, end, set, try, repeat, while,* and many more. A complete list is on Apple Developer Connection.

# Controlling Script Flow

- Some of the keywords, like *try, while,* and *repeat* are used to control script flow.
- These keywords will come in handy as you develop more complex scripts that need to loop or have conditional branches.

# AppleScript Dictionaries

- Every scriptable application has an AppleScript Dictionary which gives details on what parts of the application are scriptable and how to script them.
- View dictionaries from Script Editor (File > Open Dictionary…)
- The Dictionary for an applications will show suites, commands, classes, properties, and elements.



# What’s in the dictionary?

- Suites are a collection of commands, classes, elements, and properties. There will be several suites displayed in each AppleScript dictionary. Some suites are system-wide and others are applications-specific.

- Commands are directives you can give an application.

- Elements and properties are used to interact information between an application and a script. Some properties are read-only, but many can be set via a script.

# Other Ways to Interact…

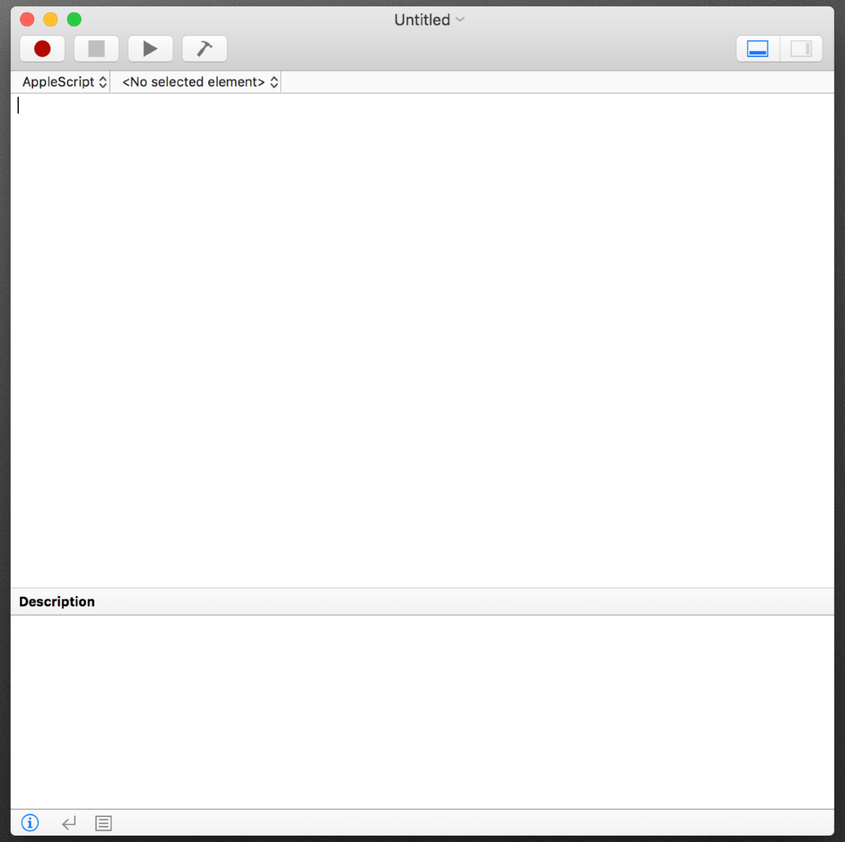
- Not everything you'll want to script will have a directly scriptable command in the application's dictionary.
- There are other ways to get around that.
  - System Events – We'll use this one a lot!
  - MouseTools, XTools – Third party scripting extensions.

# System Events

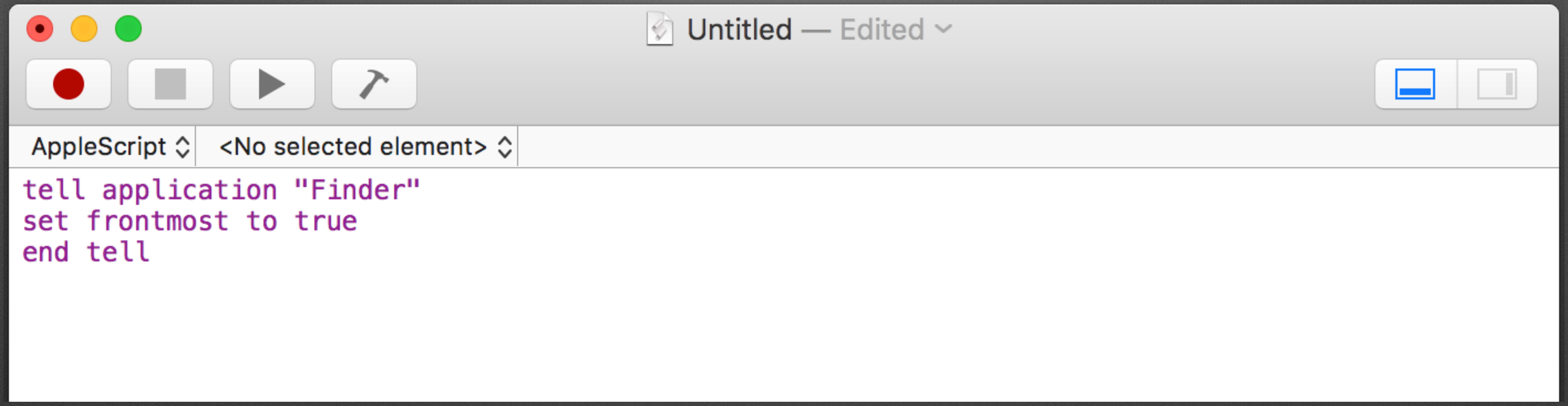
- System Events.app is an actual application (it lives in System:Library:CoreServices) that is only interacted with via AppleScript.
- System Events exposes a ton of functionality to AppleScript. It can…
  - …bring applications to the foreground, hide them, and manage their windows.
  - …invoke menu commands.
  - …directly issue keystrokes.
  - …and much more! It provides twenty-one AppleScript suites in all!
- Every script you write will probably invoke System Events at least once.

# Our First AppleScript

- Let’s write a simple script!
- First open Script Editor. It’s in Appliations:Utilities.
- Create a new script document.



# Our First AppleScript




…what does all *that* mean?

# Line-by-Line

- *tell application “Finder”* – This line starts a *tell* block, which is a basic control statement in AppleScript. Tell is possibly the most-used keyword in AppleScript. We’re letting AppleScript know where we want the following lines to be directed.

- *set frontmost to true* – *set* is an AppleScript keyword. *frontmost* is a property exposed by the “Standard Suite” which we can see if we examine Finder’s AppleScript Dictionary. *to* is another keyword interpreted by the AppleScript engine. *true* is the value we’re assigning *frontmost*.

- *end tell* – *end* is another keyword, which lets AppleScript know that what follows (the *tell* block) is over with.

# Compile and Run

- If you click the Run icon in the toolbar, you'll notice two things…
- First, Finder will become the foreground application on your computer (our script worked!)
- Second, the appearance of the script's text will change. When you run the script, Script Editor first compiles it. In the process of doing so, it checks it for errors and renders it with fancy syntax-formatting.

AppleScript ⇕ <No selected element> ⇕

```
tell application "Finder"
	set frontmost to true
end tell
```

# Other Ways to Write Scripts

- You may note noticed an intriguing Record button in the Script Editor toolbar.
- Many applications are *scriptable* but some applications are also *recordable.*
- Unfortunately, just about everything you think you'd like to do in AppleScript won't be recordable, but sometimes you get lucky, so it's worth a shot.

# Okay, let's write a *real* script now.

- We're going to write a script for Apple Mail.
- This script will find all messages sent by the sender of the currently selected message.
- We're going to need a few more keywords than we've learned so far, but we'll cover those along the way.
- We'll also need to dig into another "fun" tool: Accessibility Inspector
- This is going to move kind of fast; remember, we're looking at the process here. You're not expected to be able to replicate this on your own after seeing it once!

# AppleScript Demo

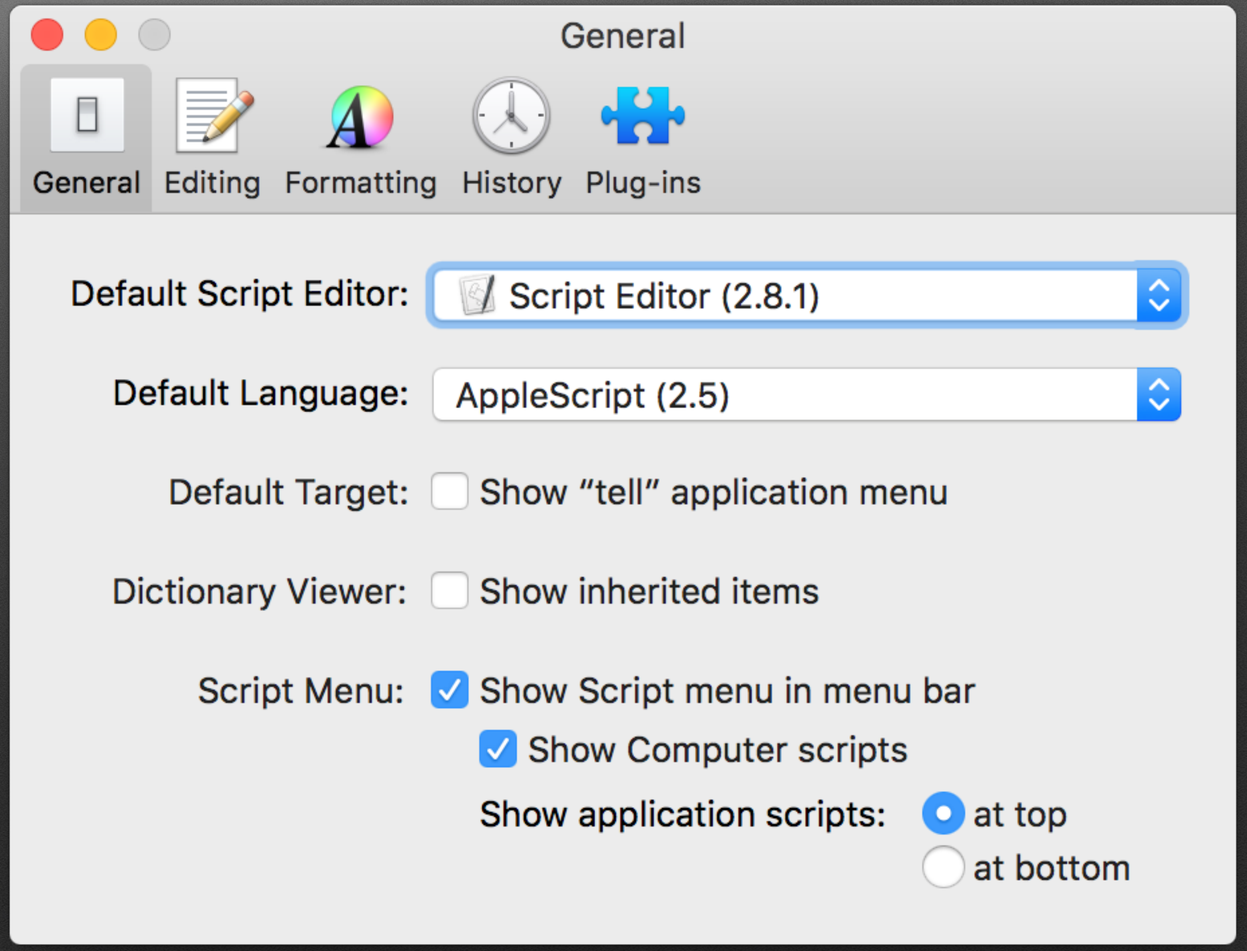
# We wrote a script. Now what?

- AppleScripts are supposed to save us time, but it certainly takes longer to go find and open our AppleScript and run it than it would to manually search for an email's sender.

- This is where it gets *really exciting!*

# Making Scripts Accessible

- We need to enable the Script menu!
- Go to Script Editor's preferences and check "Show script menu in menu bar".
- You may also want to change "Show application scripts" to "at top".

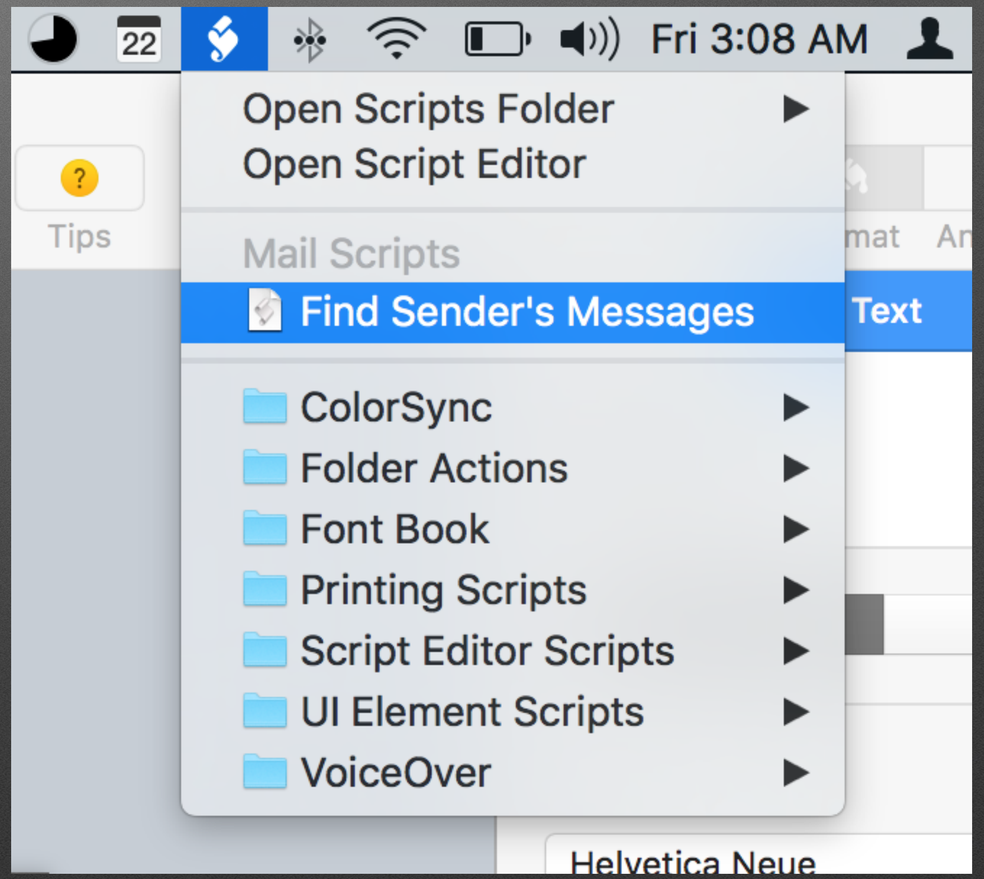


# Making Scripts Accessible

- We now have a small AppleScript icon in your menu bar (on the right side somewhere, near the clock).
- Open Mail, click the Script menu, hover over “Open Script Folder” and select “Open Mail Scripts Folder”.
- Mac OS creates a folder just for our Mail scripts and opens it in Finder. We’ll place our new script here.

# Using Our Script

- Now we can run our script directly from the Script menu in Mail!



# Let’s look at a few more scripts…

Mail Rules, Calling AppleScripts with John Gaver

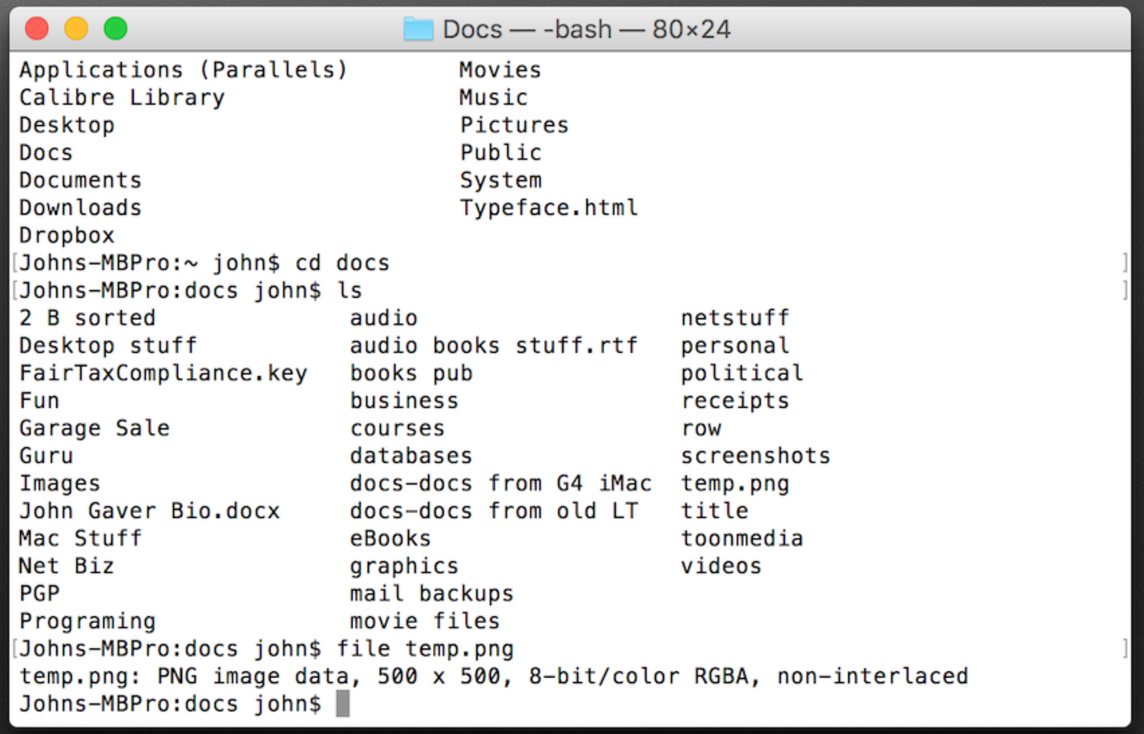
# AppleScript in Conclusion

- AppleScript is extremely powerful. It’s possible to automate just about anything on a Mac using AppleScript, although depending on how scriptable your applications are, it may not be easy!
- You can use AppleScript to add simple features to applications you already use.
- Google and example scripts are your friends, along with patience and an allowance for trail and error.

# UNIX Shell Scripts

# UNIX Shell Script

- The Mac interface is built on top of UNIX, which has a very powerful, albeit cryptic scripting capability, and is **NOT** for the faint of heart.
- There are many examples of *UNIX Shell Scripts* on the Internet, as a starting point.
- You will probably only use individual UNIX commands, instead of a full script.
- There are several members of HAAUG, who are highly qualified to help you craft a *UNIX Shell Script*, should it become necessary.



# UNIX Shell Scripts

- *UNIX Shell Scripts* can be run on their own, from the Terminal app, but there is little need for that.

- By far, the easiest and most likely way you will run Shell Scripts is from Automator. But most things can be done with Automator and AppleScript alone.

- When all else fails, it's a safe bet that whatever you're unable to do with *Automator* or *AppleScript* alone, can be done with a *UNIX Shell Script*.

- Learn to use the *"man"* command in the *Terminal* app.
  To start with, just type *"man man"* at a *Terminal* prompt.
  Type *"man -k [keyword]"* to find man pages related to a *keyword*.

# UNIX Shell Script Demo

- Since you will most likely be using UNIX Shell Scripts from within Automator, the UNIX Shell Script Demo will be a part of some Automator Demos.

# Automator

# Automator

- Automator is another way of automating tasks on the Mac.
- Automator was introduced with Mac OS 10.4 “Tiger”.
- The interface for creating automator actions is much more user friendly than AppleScript. More drag and drop, less coding!
- Sometimes Automator can be limiting. The provided building blocks may not do exactly what you want. Third-party Automator “packs” are out there though and you can always invoke AppleScript or a UNIX Shell Script from an Automator workflow!

# Automator Document Types

- When you first open Automator, you’re asked what kind of document you want to create.
- Workflow - This is the default document format in Automator. Workflows are easily editable in the future and run inside the Automator application.
- Application - This lets you create a standalone app that will perform your workflow when you either double-click or drag another item onto your workflow’s icon.
- Service - This lets you create a service that integrates into Mac OS. More on that later.
- Folder Action - Folder Actions are attached to folders in Finder, and are run when items are added to that folder.

# Automator Document Types

- Print Plugin - Print plugins are workflows that install in the print dialog box. They appear in the PDF dropdown menu at the bottom-left corner of the screen. You can create your own workflows to go along with what’s already there.

- Image Capture Plugin - These workflows are available when scanning and importing images in Image Capture.

# Let’s Make a Workflow

- Let’s make a simple Application workflow.
  - We’ll take a PDF file as input,
  - make a copy of it,
  - convert the copy to black and white in order to reduce its filesize, and
  - attach it to a new email.

# Automator Demo

# That was so quick, let’s do it again!

- For this demo we’ll make an Application workflow that makes a text file listing the contents of a folder.

# Automator Demo

# Let’s look at a few more Automator Actions…

Passing control between AppleScripts and UNIX Shell Scripts from within Automator Workflow sand compiled Applications, with John Gaver.

# Automator in Conclusion

- Automator is a very quick way to script a workflow as long as you can replicate your workflow with Automator actions (building blocks).
- Additional actions are available. Some will show up in Automator just because an application is on your Mac (for instance GraphicConverter adds a ton of image-related actions). There are also Automator action packs out there on the Internet to download.
- You can always call AppleScript or a UNIX Shell Script from an Automator workflow, making Automator much more powerful.

# Services

# Services

- Services are the coolest thing you've probably never used before.
- The functionality was one of the things that came to the Mac from NeXTstep when Mac OS X was created.
- Services allows for *inter-application handoff* of selected or active data.
- You may be using Services without even realizing it.

# The Services Menu

- There’s always a menu item called Services under the application menu, however what is listed in the Services menu depends on what you currently have selected.
- Services are context-sensitive and handle one type of data. Text services will only appear when text is selected, image services will only appear when acting on an image, and so on.
- Services can also be invoked by control-clicking or right-clicking.

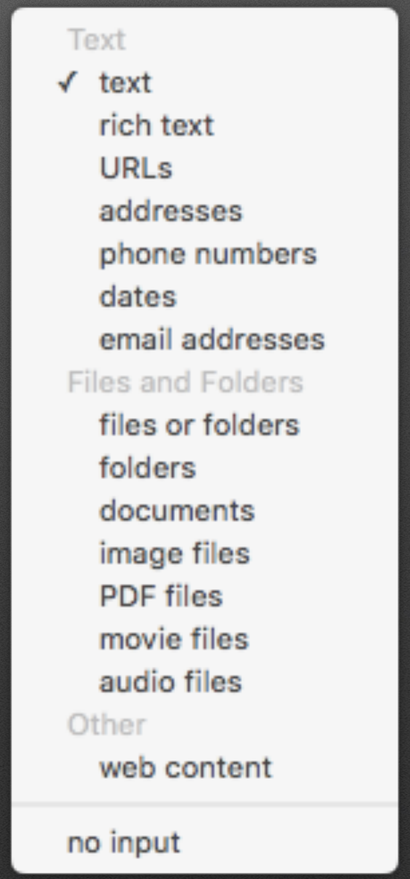
# Cleaning Up the Services Menu

- Many applications provide Services, and many of them aren’t useful for most users. Scroll through the Services list in System Preferences and uncheck what you know you’ll never use! Enable what you think will be useful.
- Services preferences live in the Keyboard preference pane (yes, I know…) on the Shortcuts page.
    - As an aside, some Services have long names that are hard to decipher in the list in System Preferences. Unfortunatly you can’t make the window bigger but you can move the divider to give the Service names more room.
- Let’s take a look.

# Services Menu Preferences Demo

# Creating Your Own Services

- You can make your own Services!
- The easiest way to create a Service is with Automator.
- Remember that Services are context-sensitive. Be sure to choose the data type correctly when making your own Service!



# Services Demo

Create Directory Listing

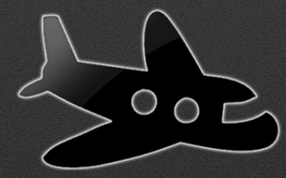
# Services in Conclusion

- A lot of useful Services are already in Mac OS and many more may be provided by the applications you already use. You just have to turn them on and know they’re there.
- You can assign keyboard shortcuts to Services in System Preferences.
- If you find yourself performing the same action to a datatype often you can create your own Services with Automator.

# Control Plane

# ControlPlane

- Who remembers the Location Manager from System 7 through Mac OS 9?
- Locations stuck around in Mac OS X but they don’t do as much as they once did. Now they only handle network settings.
- ControlPlane replaces the functionality of Location Manager and much, much more.
- It’s the best price: *Free* from <http://www.controlplaneapp.com/>.



# From the Developer…

ControlPlane allows you to build configuration profiles, (contexts in ControlPlane lingo), for your Mac based on where you are or what you are doing.  ControlPlane determines where you are or what you are doing based on a number of available evidence sources and then automatically reconfigures your Mac based on your preferences.

# How does it work?

- You configure different *contexts*. Contexts are similar to Locations from the old Location Manager, and are like configuration profiles for your Mac.
    - A few example contexts might be *Home, Office, On Battery,* or *External Monitor Connected*
- *Evidence Sources* are used to determine what context should be active. There are many evidence sources to choose from in ControlPlane. You must enable an evidence source to make it available to a rule.

# Rules

- *Rules* are the actual links between *evidence sources* and *contexts*.
- When you make a rule you can drag the confidence slider up and down to affect the weight the rule has on making a context switch. The confidence needed to make the switch happen is in the General tab of ControlPlane's settings.
- For our Home or Office context we might choose *Nearby WiFi Network* as a rule, give it our home or office network name, and set the confidence all the way to max.

# Actions

- *Actions* are the stuff that actually happens when a context change occurs. You can set rules to trigger *On Arrival, On Departure*, or *Both*.
- There are many, many Actions built into ControlPlane, including mounting and dismounting network shares, changing your network settings, changing your default printer, and more.
- Some example actions…
  - You can have your default printer change and your home NAS mount automatically when ControlPlane sees that you’re connected to your home WiFi network.
  - You can have your Time Machine Destination change depending on if you’re at home or work.

# Custom Actions

- If you want ControlPlane to do something that's not listed in its built-in actions, you can also have it trigger a shell script or an AppleScript.
- The possibilities are endless!

# ControlPlane Demo

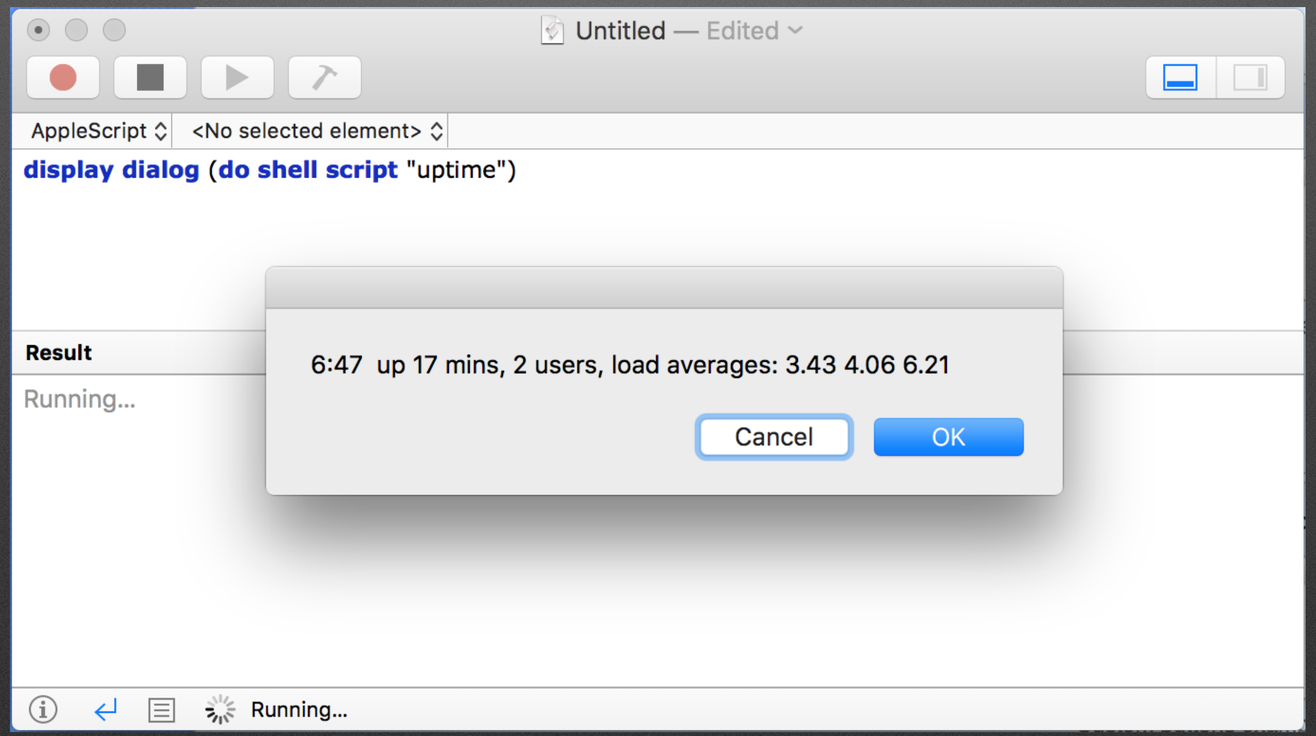
# Final Thoughts

# Script Interoperability

- We’ve talked about three scripting languages: *AppleScript, Automator* and *UNIX Shell Scripts.*
- What if you are writing an AppleScript and come across something that would be easier to do in a UNIX shell script, or you want to invoke an AppleScript from a shell script or the command line?
- You can script across languages with a few simple commands.

# Script Interoperability

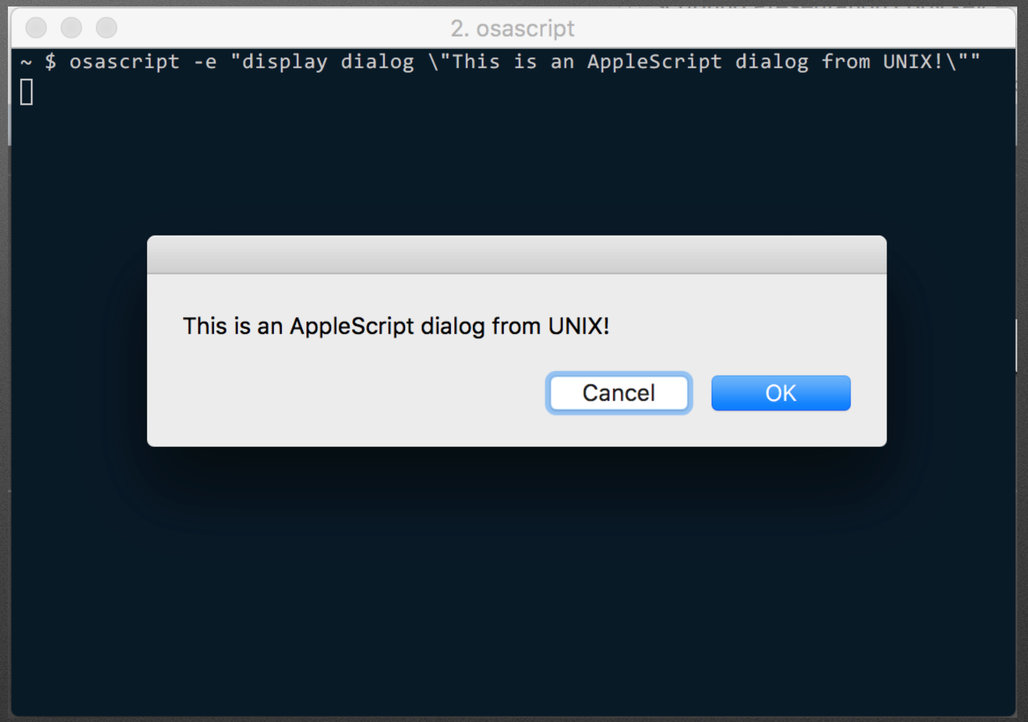
- In AppleScript, the keyword *do shell script* will run a UNIX shell script or command.



# Script Interoperability

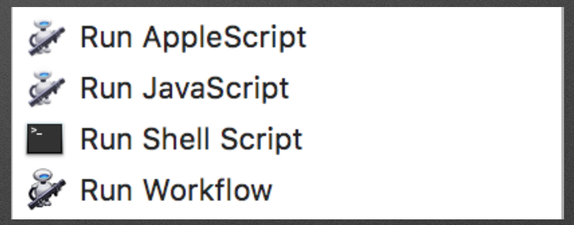
- In the UNIX command line or shell scripts, the *osascript* command will execute AppleScript. Use *osascript -e* to execute a single line of AppleScript or *osascript filename.scpt* to run a script file.

- Adding AppleScript to your shell scripts is a great way to handle user interaction (dialog boxes, requests for input, etc.)



# Script Interoperability

- Automator can call AppleScript and UNIX shell scripts, as well as other Automator Workflows.
- You can call Automator from the UNIX command line or shell scripts with the *automator* command. With the AppleScript *do shell script* command this allows you to run Automator from AppleScript as well.*




* There may be a way to do this from AppleScript directly. If you find it, let me know!

# Conclusions

- There are many, many ways to automate your Mac.
- **AppleScript** is a powerful way to script Mac OS and user applications. It can be used for complex automation that involves loops, conditional branches or user input. Because AppleScript is heavily ingrained in Mac OS, scripts can be easily triggered within applications from the script menu or through application-specific methods (like Mail Rules).
- **Automator** is ideal for quickly automating a manual process provided you can utilize the building blocks Automator provides to do so. You can also invoke AppleScripts from Automator, making it yet more powerful. Special Automator document types (like Print Plugins) let you add functionality to specific parts of Mac OS.

# Conclusions

- **Services** lets you leverage of inter-process communication by enabling a system-wide action that is context-sensitive depending on the selected datatype (text, image, and so on). Many applications on your computer have probably already added handy Services to your Mac, so check in the Keyboard preferences pane under the Shortcuts tab to see what Services are available to you. You can create new Services from Automator. Services are invoked through the Services submenu of the Application Menu or by control-clicking or right-clicking data.

- **UNIX Shell Scripts** let you leverage the power of UNIX in your scripting and automation.

# Conclusions

- **ControlPlane** enables context-sensitive automation, letting you define rules and actions that are triggered automatically depending on where you are, what peripherals are connected to your computer, and more. You can trigger AppleScripts or other applications from ControlPlane, making it even more powerful.

- You can always interoperate between scripting languages. AppleScript can be called from Automator or shell scripts, shell scripts can be called from AppleScript or Automator, etc.

- Remember that Internet resources and example scripts can be very helpful in writing AppleScripts!

# Questions and Answers?