Extreme Admin:

Mastering the details of OS X and Automating your Administration

MacWorld San Francisco 2006

Scott M. Neal
Senseption

Mary Norbury
IT Director, University of Colorado
Health Sciences Center

Goal of Session

- Empower System Administrators (who tend to not be programmers) to utilize developer tools to solve problems
  - Develop a clear understanding of OS X's primary configuration file format: Property Lists
  - Utilize existing command-line tools with a GUI that YOU develop
  - Automate repeated tasks that would normally be boring and prone to error
  - More spare time for you!
- This session is designed with walk-through examples for you to follow
  - Hopefully you've already installed the developer tools!

Developer Tools will Improve Our Lives

- Problems we will solve today with Developer Tools
  - Property Lists and User Defaults editing
  - Creating repeated tasks using `launchd`
  - Prettying-up shell scripts and commands by adding a GUI
- Hopefully you have already installed the Developer Tools... (this session is hands-on!)

De loping on S X

- coa
  - nterface Builder
- arbon
  - nterface Builder
- ebObjects
  - OModel
  - ebO uilder
-
- BS mpliant UNIX code
  - co line compilers
  - X11

Developing on OS X: Tools in this Session

- Property Lists: Preferences and `launchd`
  - Property List Editor
- Applescript
  - Script Editor
- Applescript Studio
  - Interface Builder
  - Xcode

Xcode

- Development environment provided with OS X Developer Tools
- Provides templates for projects
  - Initial configuration files
  - Organized structure that is project type-specific
  - VERY configurable

Property Lists & Defaults

- Configuration information for processes needs to be stored somewhere
- Apple developed the Property List (aka plist) Format to provide a consistent way to store information
  - Not ALL configuration information is stored in plist format
- Property Lists provide the backbone for the Defaults Preference infrastructure

Property Lists: Key-Value

- Property Lists are Key-Value storage mechanisms
  - The Key is the identifier for a particular property
  - The Value is the storage container referenced by its Key
- Example:

```
name = “AppleScript”
```

  - the Key is `name`
  - the Value of the Key `name` is `AppleScript`

Property Lists: Value Classes

- Values can belong to different Value Classes
  - String
    - A collection of characters
    - Keys themselves are strings
  - Number
    - A numeric value that can participate in calculations
  - Boolean
    - True/False, Yes/No, 1/0
  - Date
    - A specific numeric type used to store dates
  - Data
    - Raw data stored in hexadecimal

Property Lists: Value Classes (cont.)

- There are two types of Value Classes that are used for organizing other Value Classes
  - Array
    - An ordered list of objects, numbered from item 0 (the first item) to n (the last item)
  - Dictionary
    - A subgrouped Key-Value list, where the Key is a string (as normal), and the Value can be ANY Value Class type

Property List Formats

- There are currently 3 Property List file formats:
  - XML
  - ASCII (NeXT-style)
  - Binary
- Property Lists are stored with the file extension .plist
- The Property List infrastructure can read/write all 3 formats
  - Humans can read/write the first 2 formats

Property Lists & Defaults

- Keys are specific to each process and define configuration options for that process, such as
  - Default new window location and size
  - File location/path
  - *the sky is the limit...*
- Similar to Windows Registry

Property Lists & Defaults (cont.)

- Modifying Property Lists and the Defaults system gives YOU control over the operation of processes
- Great for troubleshooting/debugging
  - “Trashing the preferences” (which really should be “Renaming the preferences”)
- Your own scripts can utilize the Defaults system to store their own information

Property List Formats: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/
DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>AppendAMPM</key>
        <true/>
        <key>ClockDigital</key>
        <integer>1</integer>
        <key>ClockEnabled</key>
        <true/>
        <key>ClockLocation</key>
        <integer>0</integer>
        <key>DisplaySeconds</key>
        <false/>
        <key>FlashSeparators</key>
        <false/>
        <key>LastSavedGlobalTimeString</key>
        <string>h:mm:ss a</string>
        <key>PreferencesVersion</key>
        <integer>2</integer>
        <key>ShowDay</key>
        <true/>
        <key>Transparency</key>
        <real>0.80000001192092896</real>
        <key>Use24HourClock</key>
        <false/>
</dict>
</plist>
```

Property List Formats: ASCII/NeXT

```
{
    AppendAMPM = 1;
    ClockDigital = 1;
    ClockEnabled = 1;
    ClockLocation = 0;
    DisplaySeconds = 0;
    FlashSeparators = 0;
    LastSavedGlobalTimeString = "h:mm:ss a";
    PreferencesVersion = 2;
    ShowDay = 1;
    Transparency = 0.800000011920929;
    Use24HourClock = 0;
}
```

Property List Formats: Binary

```
bplist00?



_FlashSeparators_PreferencesVersionWShowDay\ClockDigitalZAppendAMPM\ClockEnabled^
DisplaySeconds_LastSavedGlobalTimeString]ClockLocation\Transparency^Use24HourClock
Yh:mm:ss a#?陙1FN[fs?????????????
```

Editing Property Lists

- Text Editor (TextEdit, vi, nano, etc.)
  - Doesn't work with Binary (unless plist is converted to one of the other formats)
- Property List Editor
  - Located in /Developer/Applications/Utilities
  - Works with ALL plist formats

Converting from one plist format to another

- “Save To” option of Property List Editor
- command-line `plutil` tool

Property Lists: Summary

- YOU are now empowered to edit Property List files directly
  - Double-edged sword: you can also SCREW UP a process' configuration, making your process (or even entire machine) unusable, so PLEASE backup plist files and edit copies!
  - Comes in handy for troubleshooting, especially in single user mode
  - Can be utilized for your own scripts (we will see this later)

Property Lists Locations

- OS X has a standard hierarchical resource search policy (for Fonts, etc.):
  ~/Library/*
  /Library/*
  /Network/Library/* (if it exists)
  /System/Library/*
- Property Lists are NOT searched hierarchically and can be stored ANYWHERE, including (but not limited to):
  - /etc/*
  - Application Bundles
  - Preferences (but where?)

Preferences: Defaults Infrastructure

- OS X has a mechanism called Defaults that is a “portal” into the Preferences plist system for processes
    - It does NOT incorporate EVERY plist on the system (phew!), only the ones dedicated to preferences and stored in specific folders
    - Processes themselves don’t need to understand how to read/write or find plist files, they use the Defaults infrastructure

Defaults Infrastructure

- Location of Preference files:
  - ~/Library/Preferences
  - /Library/Preferences
  - /var/root/Library/Preferences
- Most files in these folders are in plist format
- Note that some files in */Preferences are NOT plist format
  - These will not be accessible through the Defaults system, but are read directly by the process that owns that preference

Defaults Domains

- Preferences in the Defaults system are organized by domain (and, optionally, host)
  - Typically correspond to individual applications/ processes
- Nomenclature for these domains typically (not always) follows Java reverse-FQDN syntax WITHOUT the .plist extension
  - Prevents namespace collisions
  - Examples:

    com.apple.MenuBarClock
    loginwindow

Accessing Defaults

- Defaults are accessed using the CLI tool `defaults`
  - When only a Domain is specified, `defaults` searches ~/Library/Preferences for a plist file matching the Domain argument with .plist appended
- Example:

```
defaults read com.apple.MenuBarClock
```

- Note that output from `defaults` is in ASCII/NeXT format, independent of the plist format itself (Binary, XML, or ASCII/NeXT)
- VERY useful in automation and scripting!

Reading/Writing Values for Specific Keys

- A specific Key can be read using the same `defaults` tool
  - Example:

```
defaults read com.apple.MenuBarClock
ClockEnabled
```

- A Key's Value can be modified:
  - Example:

```
defaults write com.apple.MenuBarClock
ClockEnabled 0
```

Other Defaults Domains

- `NSGlobalDomain`
  - Used for default shared Key-Value combos which are not process, host, or domain-specific
  - Can use `-g` instead of specifying `NSGlobalDomain`
  - Example:

```
defaults read NSGlobalDomain
AppleMiniaturizeOnDoubleClick
defaults read -g
AppleMiniaturizeOnDoubleClick
```

Other Defaults Domains (cont.)

- A full path to a .plist file (minus the .plist extension) can be specified as a domain
  - Example:

```
defaults read /Library/Preferences/
.GlobalPreferences
```

- Take a look at the Domains available in each of the Preference file locations:
  - ~/Library/Preferences
  - /Library/Preferences
  - /var/root/Library/Preferences

Specifying a Host with Defaults

- The `ByHost` folder seen in some of the previous Preference locations stores host-specific information
  - Useful for processes that utilize more than one host, and want to have host-specific preferences
- You may specify a hostname as either a MAC (IP) address or `-currentHost`
- Examples:

```
defaults -currentHost read
com.apple.networkConnect

defaults -host 000a95a92943 read
```

Where is the .plist for NSGlobalDomain?

```
~/Library/Preferences/ByHost/
.GlobalPreferences.MacAddress.plist
```

- This is an FYI, since you really should be using `defaults` to edit the global data (it's MUCH easier!)

Defaults (Summary)

- Preferences are managed through the Defaults infrastructure, and stored in plist files in specific Domains
- You can leverage the `defaults` command to read/write pre-existing domains
- You can also use `defaults` to create/read/write/delete domains on-the-fly
    - Even scripts can have saved preferences!

`launchd`

- The `launchd` daemon process is new to OS X Tiger, and replaces MANY previous processes and methods that controlled the launch of processes:
    - `rc` files
    - `init`
    - `inetd/xinetd`
    - `SystemStarter`
- `launchd` is controlled via Property Lists and commands that load/unload those plists

`launchd.plist`

- The `launchd.plist` files consist of Keys-Values that define launch properties
  - You set the right Keys to the right Values, and YOU control the launching of processes!
- There are 2 `launchd.plist` file categories, system Daemons and user-specific Agents
  - Daemons are launched at boot in the following order:
    - `/System/Library/LaunchDaemons`
    - `/Library/LaunchDaemons`
  - Agents are then launched once a user logs in as follows:
    - `/System/Library/LaunchAgents`
    - `/Library/LaunchAgents`
    - `~/Library/LaunchAgents`

`launchd.plist` Basic Keys

- `Label`
  - A unique identifier for this process
- `OnDemand`
  - Designates whether process launches immediately, or waits for demand (for instance, on an incoming port)
- `Program`
  - Absolute path to the executable file which launches the process
  - optional if `ProgramArguments` specifies not only arguments but launch path too
- `ProgramArguments`
  - An array of arguments for the process

`launchd.plist` Some Optional Keys

- `UserName`
  - The username under which the process should run
- `RootDirectory`
  - A directory that will be the root directory for the process (using `chroot`)
- `WatchPaths`
  - A list of paths that, when modified, will cause the process to launch (if not already launched)
- `StartInterval`
  - An interval in seconds specifying how often the process should be started
- `StandardCalendarInterval`
  - A specification of a repeated calendar interval

launchctl

- Used to load/unload `launchd.plist` files into the launchd system
- Example:

```
launchctl load ~/Library/LaunchAgents/
com.apple.TextEdit.plist

launchctl unload ~/Library/LaunchAgents/
com.apple.TextEdit.plist
```

Walk-Through: TextEdit

- We are going to create a simple plist file for TextEdit to ensure that it is always running

`launchd` (Summary)

- We have only scratched the surface of `launchd` here
  - Please consult online documentation and the man pages for `launchd`, `launchd.plist`, `launchctl`, etc.

AppleScript

- One of the (if not THE) easiest scripting languages to learn
- Syntax is flexible, and punctuation is not as necessary (semi-colons and braces and back ticks OH MY!)
- AppleScript can easily call ANY script, command, or program (or combination)
- ANY script or program can easily call AppleScripts through the Open Scripting Architecture (OSA)

Apple Events

- Apple Events are commands that are sent to an application
- MANY applications are AppleScript-able through Apple Events
- Apple Events are defined for each Apple Event-compatible application
  - Standard Suite
  - Application-specific
- Use Script Editor to see what Apple Events an application supports
  - /Applications/AppleScript/Script Editor
- Apple Events are NOT the same as Automator Actions

AppleScript Example

```
tell application "Finder"
  set finderWins to (every Finder window)
  repeat with w in finderWins
    set finderWindowName to name of w
    display dialog finderWindowName
  end repeat
end tell
```

ScriptEditor Record Mode

- Will record your actions and create AppleScript commands from them!
- Great for creating a template script that you can modify/addend
- Limited by what events the application provides for recording
  - It often seems like the ONE THING you need is missing

AppleScript Droplets

- AppleScripts can be converted into Droplets, which allow things to be "dropped" onto them in the Finder
- Dialogs can then be optional, and the script will operate on all files/folders dropped onto it
- Great for workflow automation

Bringing Executables into AppleScript

- So how can we incorporate programs that don’t respond to Apple Events into our flow?
- How can we make applications that only run in the Terminal be double-clickable for neophyte, Terminal-phobic users?

AppleScript providing a frontend to `rsync`

- Demo

Running AppleScripts from UNIX scripts

- Use `osascript` command
- OSA stands for Open Scripting Architecture
  - There is an OSA-compatible implementation of JavaScript
    http://www.latenightsw.com/freeware/JavaScriptOSA/
- So why is this cool...?
  - Answer: UNIX scripts can now ALSO access AppleEvent-supported applications
    - Quicktime compression
    - Adobe CS
    - etc.

AppleScript Studio: Adding a GUI to `rsync`

- We have seen how to make AppleScripts communicate with pre-existing programs
- Wouldn’t it be COOL if we could use Apple Developer Tools to put a fancy GUI in front of command-line tools?

AppleScript & AppleScript Studio Summary

- Intuitive, syntax-friendly scripting language
- Part of DNA of OS X
  - Apple Events
  - Droplets
- AppleScript Studio brings power of full-powered GUI to your scripts without ANY programming (just connect-the-dots)

Synopsis

- Even non-programmers have the power to harness developer tools
- You can use Property List Editor to
  - Edit process' properties
  - Control the launching of applications with `launchd`
- You can use the Preference Defaults system for troubleshooting and control, and for use in your own scripts
- You can create scripts/programs (or get them off the Internet!) and add fancy GUIs to them with AppleScript Studio

Resources

- The latest version of these slides and a comprehensive list of resources on this topic can be found at:
  - From Finder, select
    Go->iDisk->Other User’s Public Folder
  - Enter
    norburym
  - For the password, enter
    mw2006

Thank You!

Extreme Admin:

Mastering the details of OS X and Automating your Administration

MacWorld San Francisco 2006

Scott M. Neal
Senseption

Mary Norbury
IT Director, University of Colorado
Health Sciences Center