CS193E
Lecture 19

Accessibility
Scriptability

Agenda

- Why Accessorize?
- Mac OS X Accessibility Overview
- Cocoa Accessibility
- Customizing instances
- Customizing classes

Agenda

- Final Projects
- Course Evaluations
    - http://registrar.stanford.edu/students/courses/evals.htm
    - March 10 - March 23
- Accessibility
- Scriptability

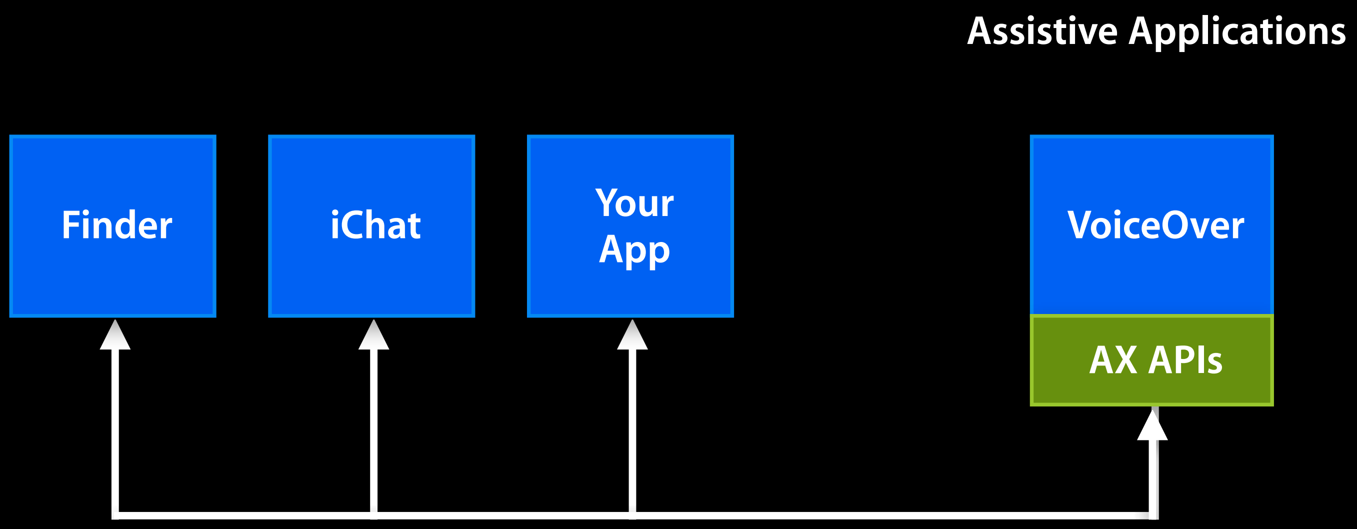
Accessibility

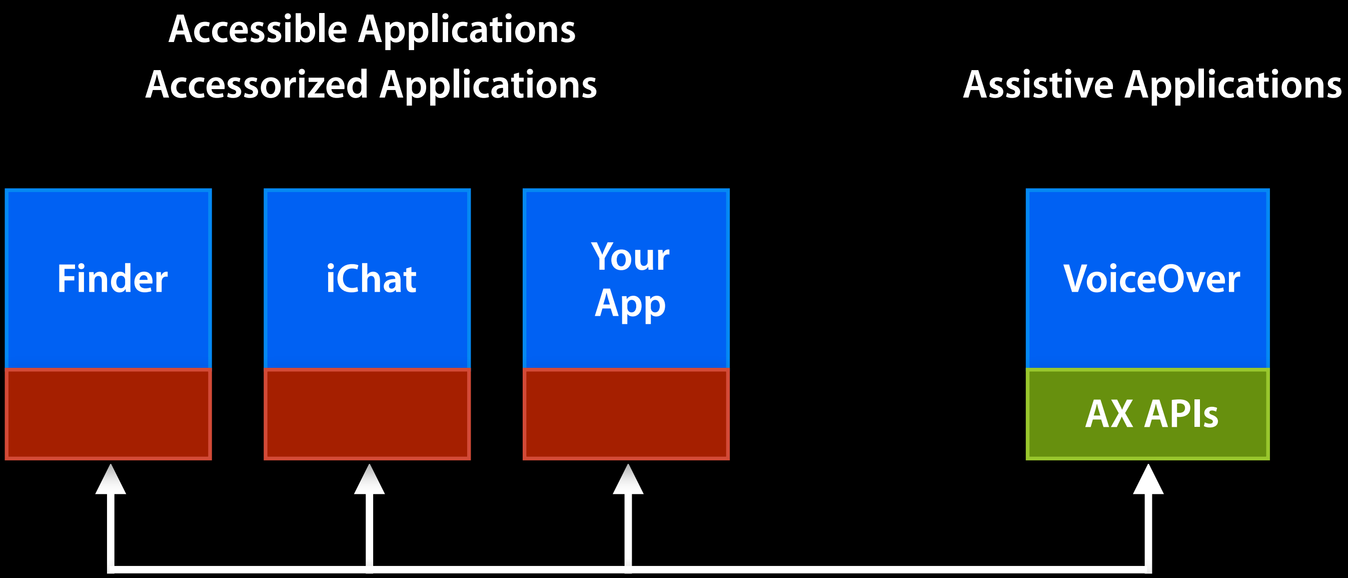
Mac OS X Accessibility

- Most system-wide accessibility features just work
  - Cocoa applications automatically take advantage
- Full keyboard navigation
  - Implement for custom views and controls
  - Provide keyboard alternatives for things like drag and drop
- Interaction with assistive applications

Why Make Your App Accessible?

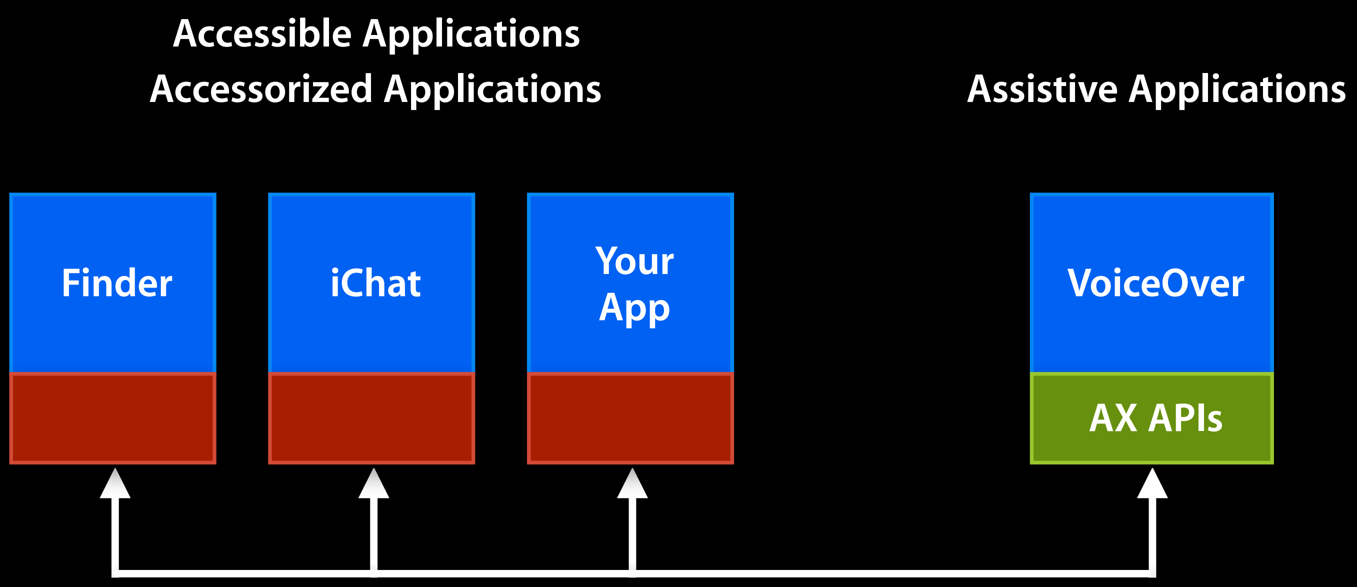
- Regulatory compliance to enable sales into government and education.
  - In the U.S. it is Section 508
- Provide enhanced user experience for customers with disabilities
- AppleScript GUI Scripting
- Automator 'Watch Me Do' feature
- Instruments record and playback feature

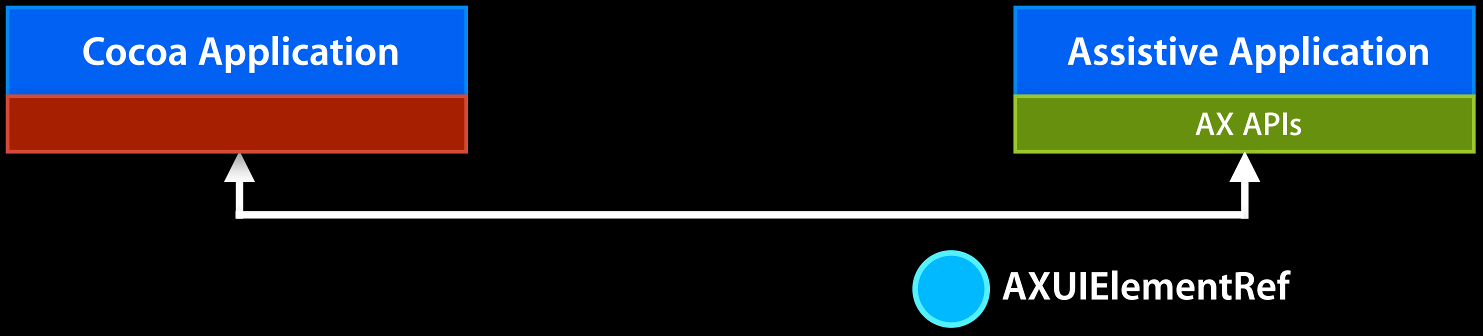

Assistive Applications
Finder
iChat
Your App
VoiceOver
AX APIs


Accessible Applications
Accessorized Applications
Assistive Applications
Finder
iChat
Your App
VoiceOver
AX APIs

Demo

VoiceOver


Accessible Applications
Accessorized Applications
Assistive Applications
Finder
iChat
Your App
VoiceOver
AX APIs

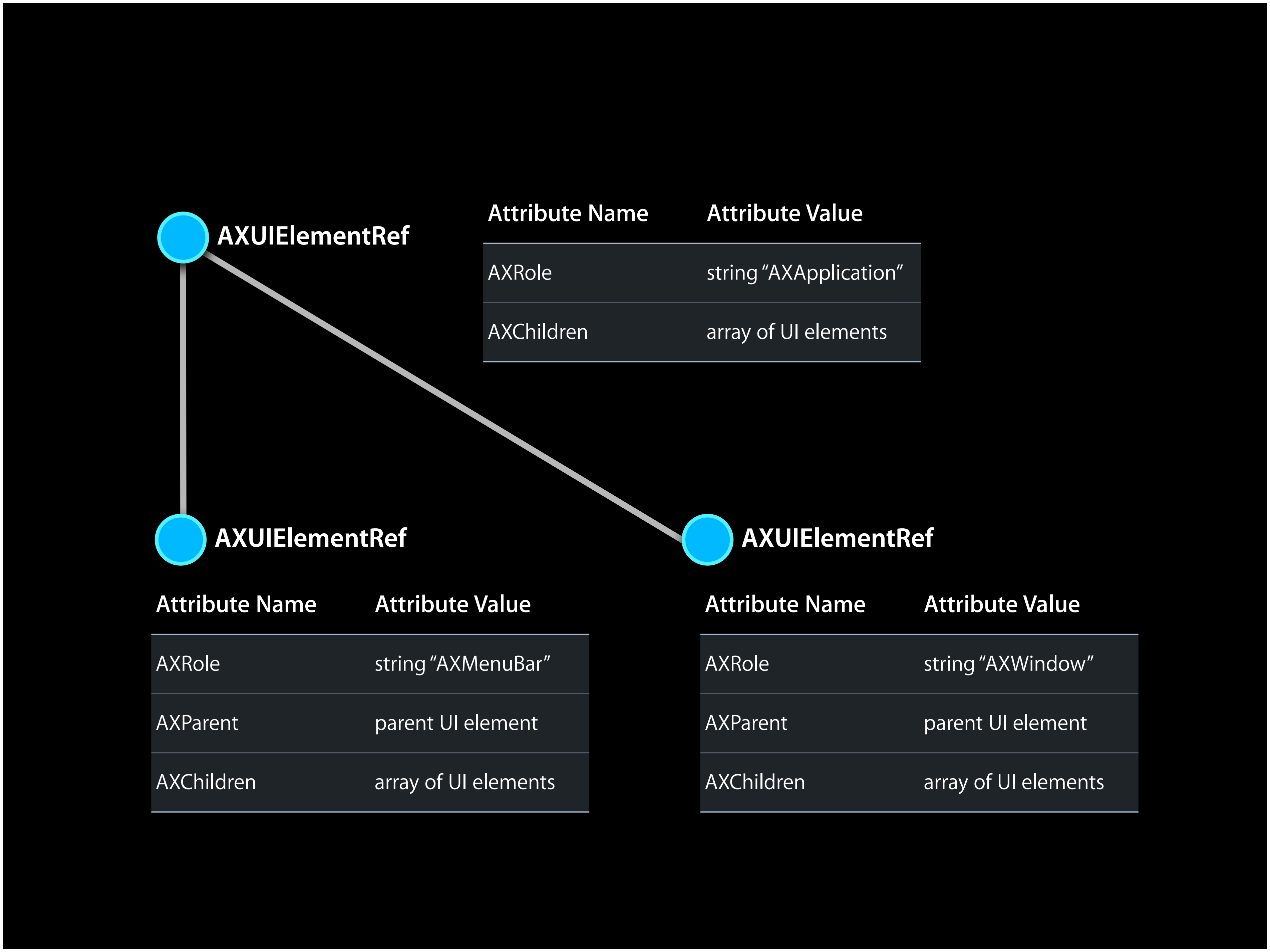


attribute list?

value for attribute?

is attribute settable?

set value for attribute!



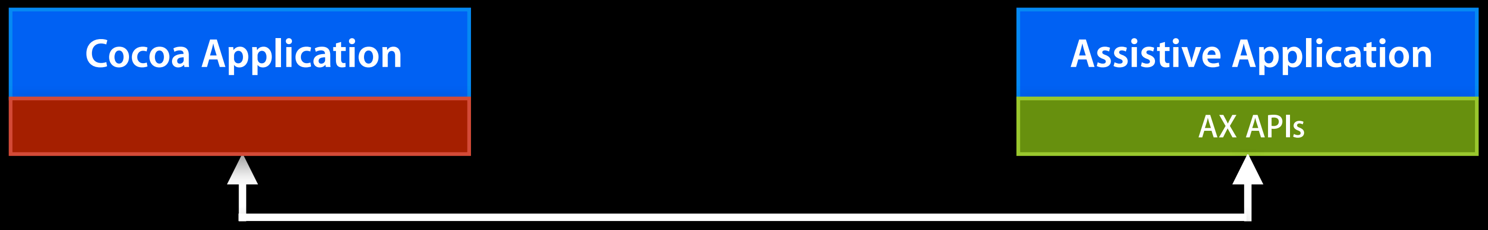
Attribute Name	Attribute Value
AXRole	string “AXApplication”
AXChildren	array of UI elements

Attribute Name	Attribute Value
AXRole	string “AXMenuBar”
AXParent	parent UI element
AXChildren	array of UI elements

Attribute Name	Attribute Value
AXRole	string “AXWindow”
AXParent	parent UI element
AXChildren	array of UI elements

Demo

Accessibility Inspector



● **NSObject (NSAccessibility)**

- NSApplication
- NSWindow
- NSView
- NSControl
- NSCell

● **AXUIElementRef**

- attribute list?
- value for attribute?
- is attribute settable?
- set value for attribute!
- action list?
- description for action?
- perform action!
- element at point?
- register for notifications

Attributes

- attribute list?

```
- (NSArray *)accessibilityAttributeNames;
```

- value for attribute?

```
- (id)accessibilityAttributeValue:(NSString *)attribute;
```

- is attribute settable?

```
- (BOOL)accessibilityIsAttributeSettable:(NSString *)attribute;
```

- set value for attribute!

```
- (void)accessibilitySetValue:(id)value
               forAttribute:(NSString *)attribute;
```

Actions

- action list?

```
- (NSArray *)accessibilityActionNames;
```

- description for action?

```
- (NSString *)accessibilityActionDescription:(NSString *)action;
```

- perform action!

```
- (void)accessibilityPerformAction:(NSString *)action;
```

Hit and focus testing

- element at point?

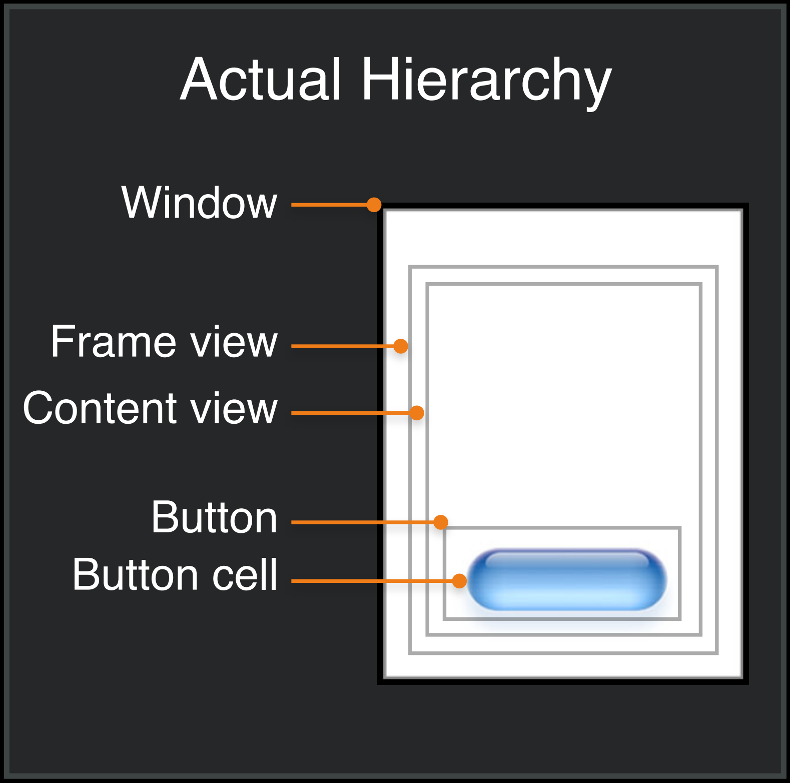
```
- (id)accessibilityHitTest:(NSPoint)point;
```

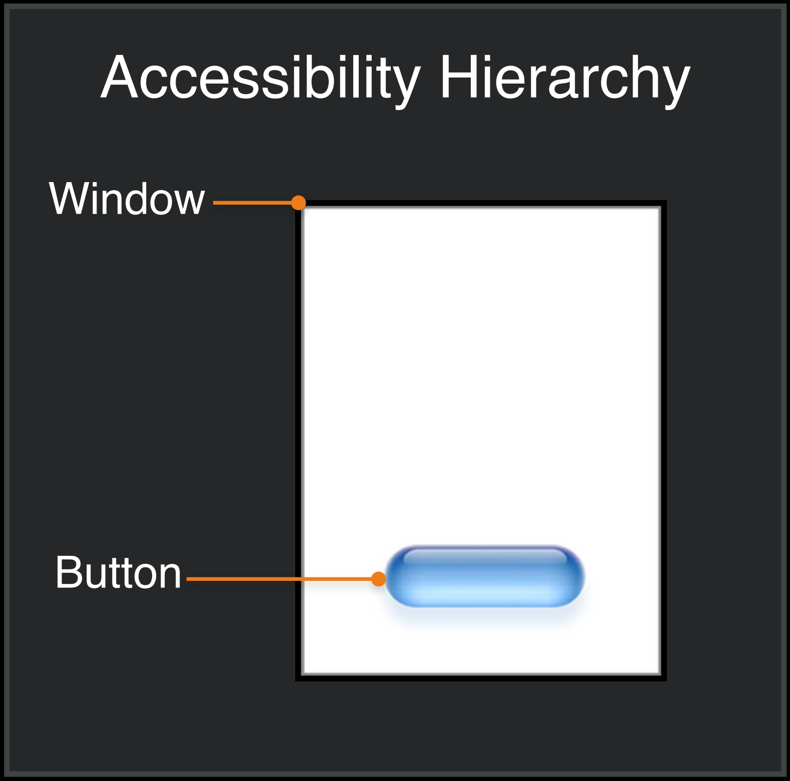
- focus testing

```
- (id)accessibilityFocusedUIElement
```

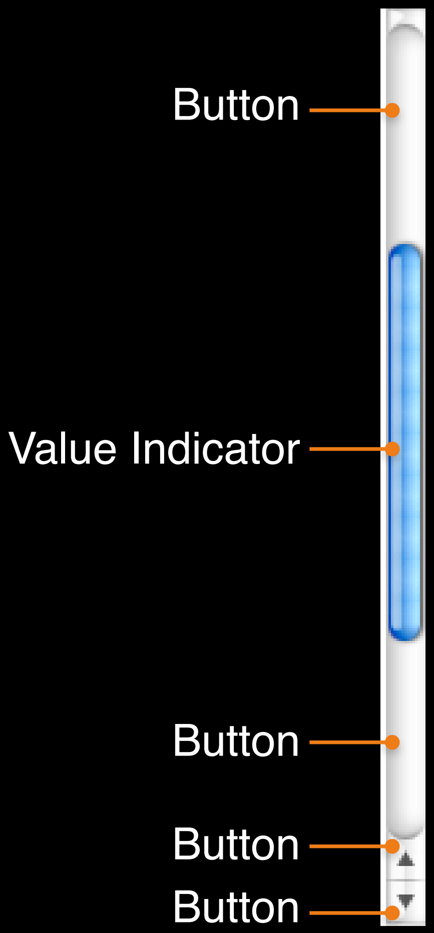
Hidden Elements

```
- (BOOL)accessibilityIsIgnored;
```

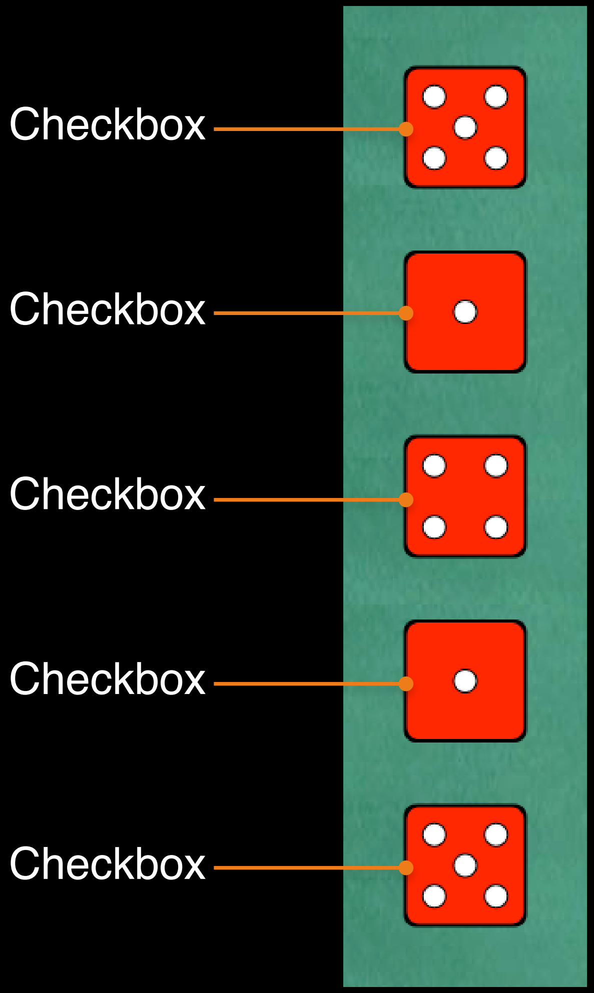




“Fictional” Subelements



Custom Subelements



Customizing Instances

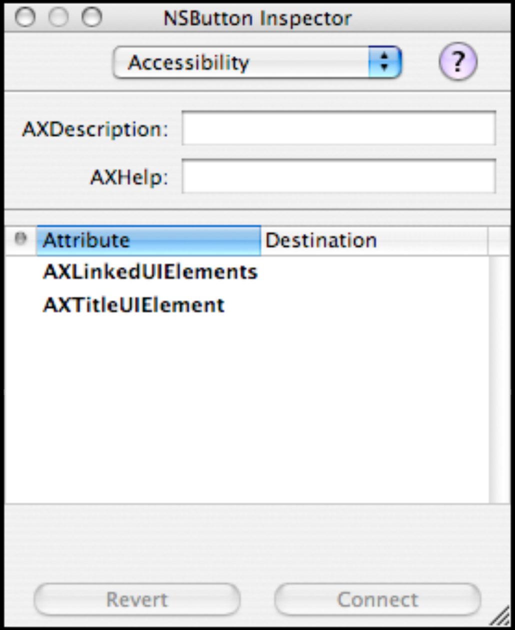
- Sometimes you just need to add or change an attribute
- No need to subclass, just override an attribute value for a particular instance

Customizing Instances

- Can set up in code using

```
-(void)accessibilitySetOverrideValue:(id)value
                        forAttribute:(NSString *)attribute
```

- Can set up many of these in Interface Builder



AXDescription:

AXHelp:

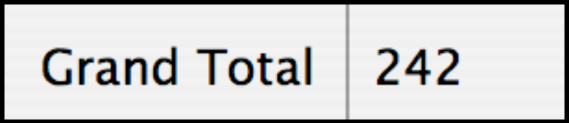
Attribute	Destination
AXLinkedUIElements	
AXTitleUIElement	

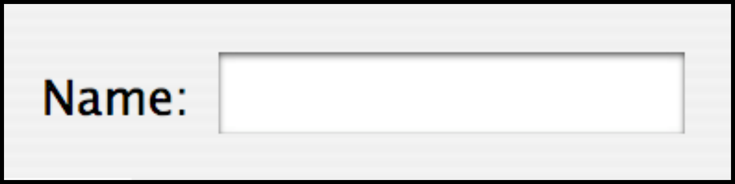
Instance Attributes—Description

- NSAccessibilityDescriptionAttribute
  - string
  - Do not include the role
    - “left align” not “left align button”
  - Often matches title, but all lowercase
  - Not to be confused with role description

Instance Attributes—Titles

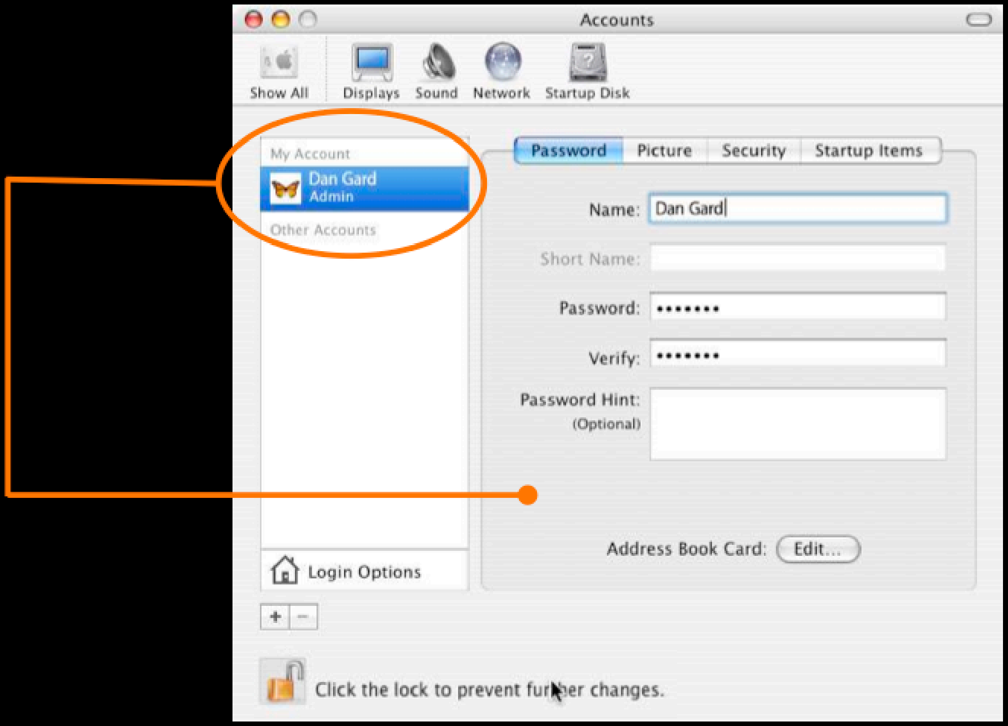
- NSAccessibilityTitleUIElementAttribute
    - UIElement





Instance Attributes—Linked UIElements

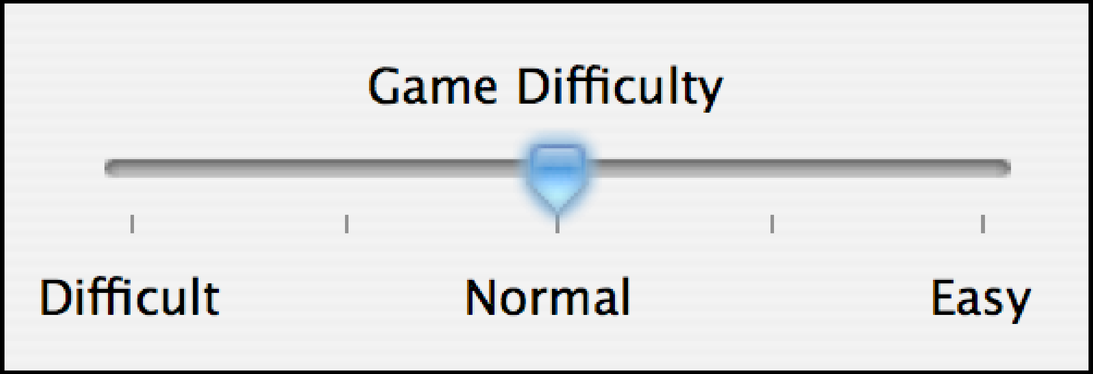
- NSAccessibilityLinkedUIElementsAttributes
  - Array of related UIElements



Name: Dan Gard
Short Name:
Password: •••••••
Verify: •••••••
Password Hint: (Optional)

Instance Attributes—Labels

- NSAccessibilityLabelUIElementsAttribute
- NSAccessibilityLabelValueAttribute
  - e.g. tick mark labels



Custom interface elements

- For custom views, you need to implement the NSAccessibility protocol and ensure proper notifications
    - Four attribute methods
    - Three action methods
    - -(BOOL)accessibilityIsIgnored
    - -(id)accessibilityHitTest:(NSPoint)point
    - -(id)accessibilityFocusTest
- If your custom view has non-view, non-cell subelements you will need to expose them as children of your view
    - Implement NSAccessibility protocol on sub-element classes
    - Create ‘faux’ UI element class the implements the protocol if there is no internal class

Scriptability

Overview

- Scriptability
  - Programmatic user interface
  - Controllable by various agents
  - Provides application interoperability
  - Empowers the end user

Overview

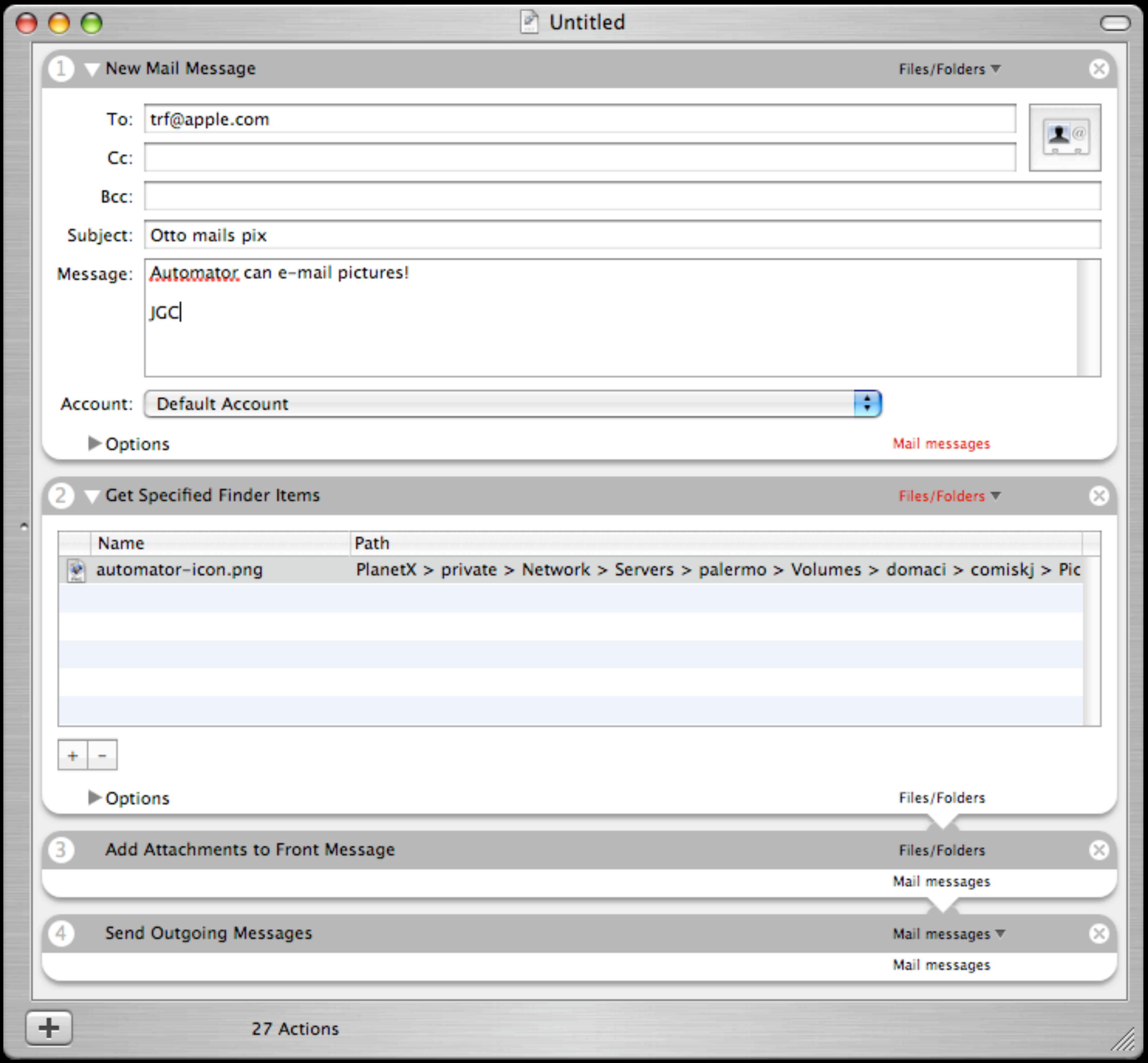
- Scriptability
  - Easy to implement
  - Benefits the developer directly
    - increasing quality
    - decreasing test time

Scriptability

- What is Scriptability?
- Why be Scriptable?
- Dictionary design

What is Scriptability?

```
tell application "Mail"
   make new message
    --   this will be much bigger
   send
end tell
```



Untitled

1 New Mail Message — Files/Folders

To: trf@apple.com

Cc:

Bcc:

Subject: Otto mails pix

Message: Automator can e-mail pictures!

JGC

Account: Default Account

Options — Mail messages

2 Get Specified Finder Items — Files/Folders

Name	Path
automator-icon.png	PlanetX > private > Network > Servers > palermo > Volumes > domaci > comiskj > Pic

Options — Files/Folders

3 Add Attachments to Front Message — Files/Folders — Mail messages

4 Send Outgoing Messages — Mail messages — Mail messages

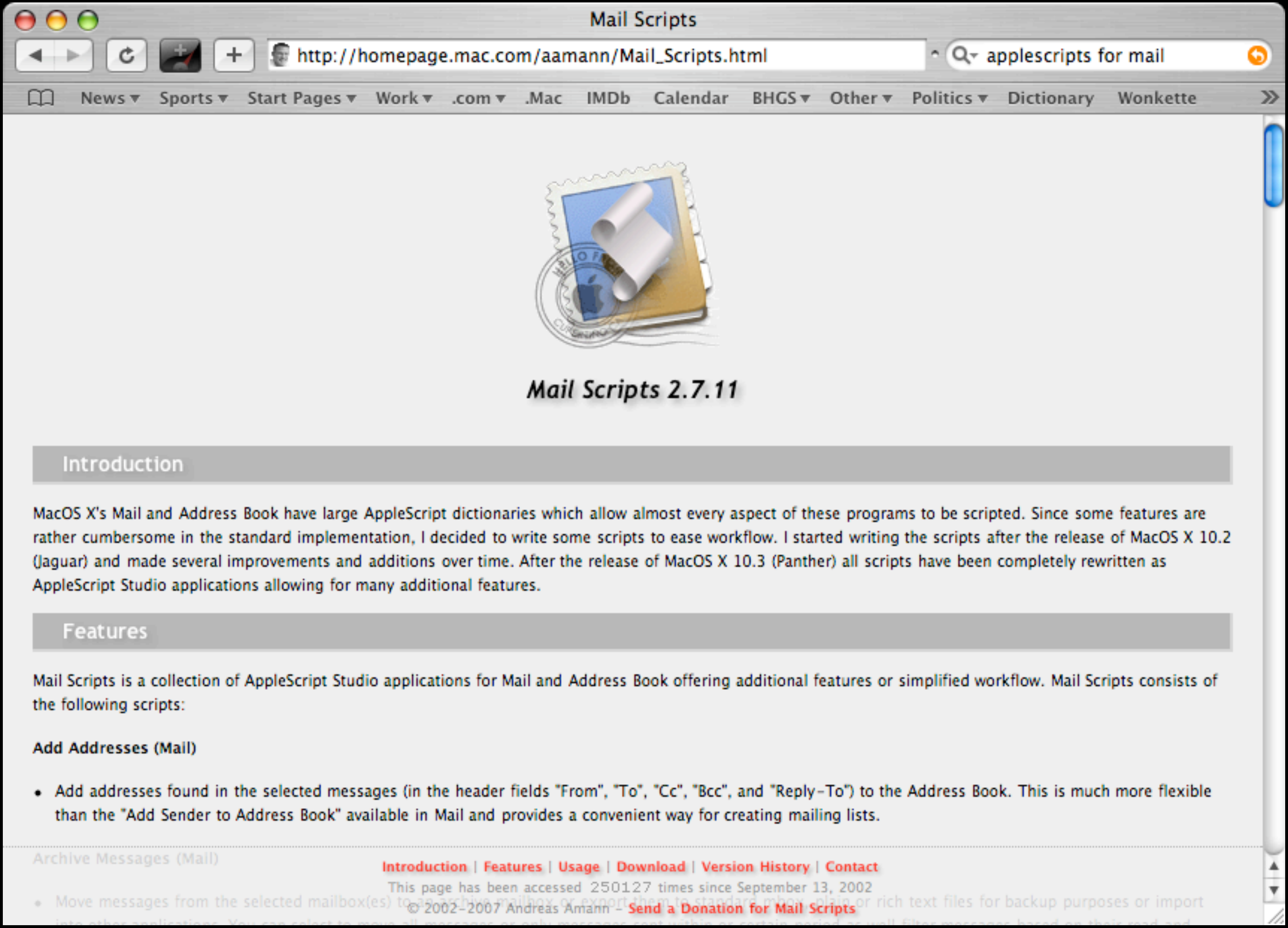
27 Actions

Why be Scriptable?

- Leverage existing technology
  - AppleScript
  - Script Menu
  - AppleScript Studio
  - Automator

Why be Scriptable?

- Customers set the agenda
  - Customer generated solutions are features
  - Spend your time where it does the most good


Mail Scripts
http://homepage.mac.com/aamann/Mail_Scripts.html
applescripts for mail
News ▾ Sports ▾ Start Pages ▾ Work ▾ .com ▾ .Mac IMDb Calendar BHGS ▾ Other ▾ Politics ▾ Dictionary Wonkette

Mail Scripts 2.7.11

Introduction

MacOS X's Mail and Address Book have large AppleScript dictionaries which allow almost every aspect of these programs to be scripted. Since some features are rather cumbersome in the standard implementation, I decided to write some scripts to ease workflow. I started writing the scripts after the release of MacOS X 10.2 (Jaguar) and made several improvements and additions over time. After the release of MacOS X 10.3 (Panther) all scripts have been completely rewritten as AppleScript Studio applications allowing for many additional features.

Features

Mail Scripts is a collection of AppleScript Studio applications for Mail and Address Book offering additional features or simplified workflow. Mail Scripts consists of the following scripts:

Add Addresses (Mail)

- Add addresses found in the selected messages (in the header fields "From", "To", "Cc", "Bcc", and "Reply-To") to the Address Book. This is much more flexible than the "Add Sender to Address Book" available in Mail and provides a convenient way for creating mailing lists.

Archive Messages (Mail)

- Move messages from the selected mailbox(es) to an archive mailbox or export them to standard mbox, plain or rich text files for backup purposes or import

Introduction | Features | Usage | Download | Version History | Contact

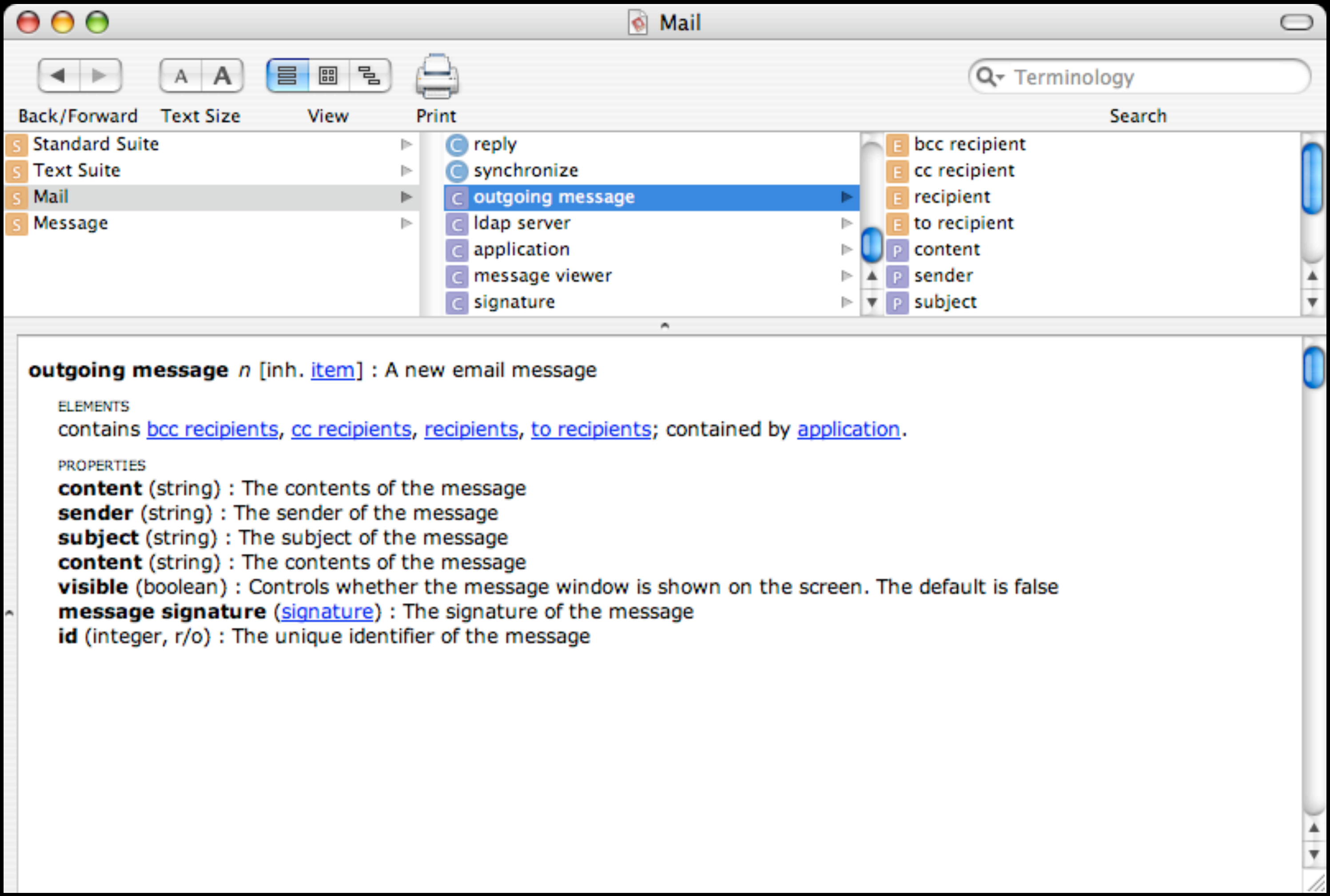
This page has been accessed 250127 times since September 13, 2002

© 2002-2007 Andreas Amann - Send a Donation for Mail Scripts

Why be Scriptable?

- Testing
  - Automated testing takes less time and effort
  - Repeatable testing catches regressions sooner

Dictionary design



Standard Suite
Text Suite
Mail
Message

reply
synchronize
outgoing message
ldap server
application
message viewer
signature

bcc recipient
cc recipient
recipient
to recipient
content
sender
subject

outgoing message *n* [inh. item] : A new email message

ELEMENTS
contains bcc recipients, cc recipients, recipients, to recipients; contained by application.

PROPERTIES
content (string) : The contents of the message
sender (string) : The sender of the message
subject (string) : The subject of the message
content (string) : The contents of the message
visible (boolean) : Controls whether the message window is shown on the screen. The default is false
message signature (signature) : The signature of the message
id (integer, r/o) : The unique identifier of the message

Making your app scriptable

1. For new applications think about scripting up front
2. Define scripting dictionary
3. Concentrate scriptability in model objects
   These objects should use Key Value Coding
4. Create an sdef file that maps AppleScript terms/codes to your object model
   The sdef is a resource in your Xcode project

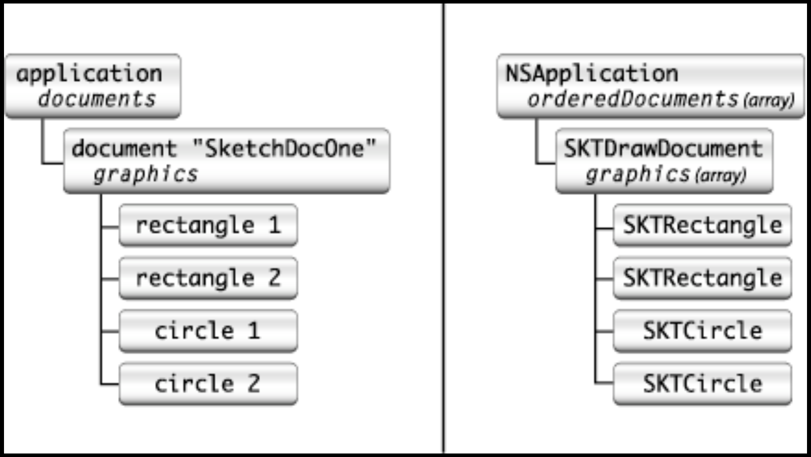
Making your app scriptable

5. Turn on scriptability in Info.plist file
6. Implement object specifier methods
7. Implement commands if needed
8. Built-in support: documents and text
9. Test

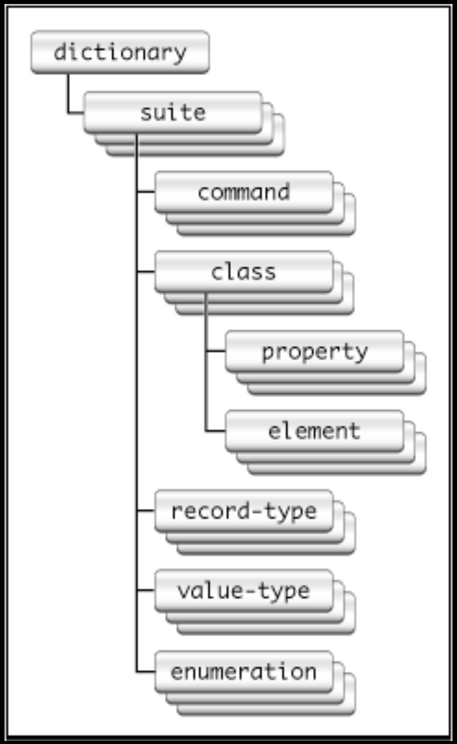
Dictionary design

- What is a dictionary?
  - How do I design one?
    - What do I design?
      - Where do I get the design?
        - What about four byte codes?

The Object Model



Dictionary structure



Scripting definition

```
... (from the Sketch suite)
<class name="graphic" code="grph"
    description="A graphic. This abstract class represents the
        individual shapes in a Sketch document.
        There are subclasses for each specific type of graphic.">
    <cocoa class="SKTGraphic"/>
    <property name="x position" code="xpos" type="real"
        description="The x coordinate of the graphic's bounding rectangle."/>
    <property name="y position" code="ypos" type="real"
        description="The y coordinate of the graphic's bounding rectangle."/>
    <property name="width" code="widt" type="real"
        description="The width of the graphic's bounding rectangle."/>
... (some properties omitted)
</class>
<class name="rectangle" code="d2rc" inherits="graphic"
        description="A rectangle graphic.">
    <cocoa class="SKTRectangle"/>
    <property name="orientation" code="orin" type="orientation"/>
    <responds-to name="rotate">
        <cocoa method="rotate:"/>
    </responds-to>
</class>
```

Scripting Interface Guidelines

- Deliver an Object Oriented solution
  - Provide tools, rather than solve problems
  - Empower, rather than anticipate
- Achieve interoperability
- Tech Note 2106

How to design

- Favor objects over commands
  - No class names in commands
  - No verbs in class or property names

What to design

- Design globally, implement locally
  - Design the entire universe
  - Phase to scale for each release

Where do I get the design?

- End user perception
- Internal structure, within reason
- Simplification
  - classic data hiding

Four Byte Codes

- Avoid conflicts
- Reuse existing terminology
  - Same term => use the same code
  - Same code => use the same term
    - e. g. “startup disk” => ‘boot’
- Search for “Apple event codes”
  - Tech Note 2106

Declaring Your App’s Scriptability

The Old Way

- .scriptSuite/.scriptTerminology files
  - Hard to work with
  - Still supported for compatibility

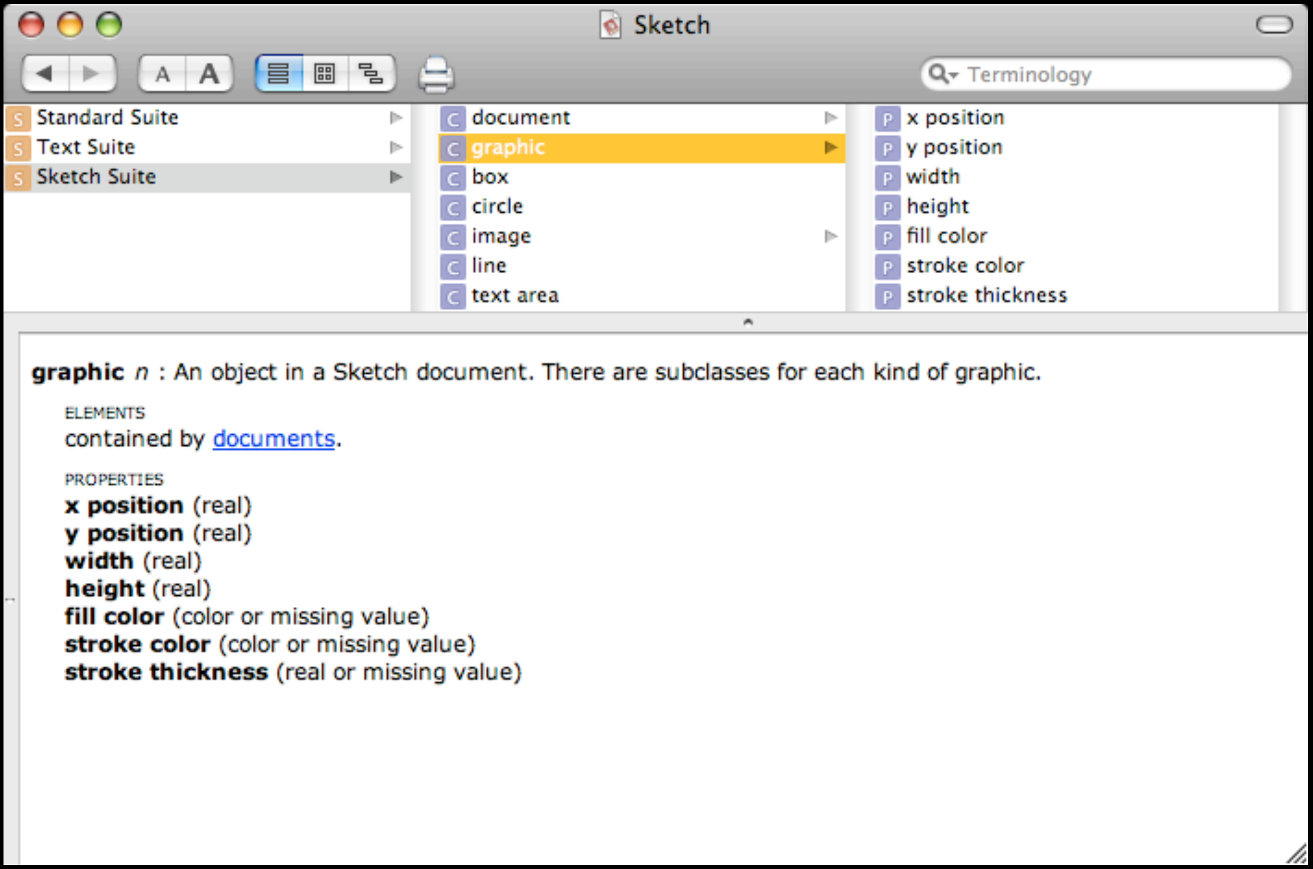
The New Way

- .sdef files
  - Much easier to work with
    - Direct mapping to AppleScript and the rest
    - More obvious what the implementation details are
  - Cocoa started supporting them in Tiger

Terminology

- "Property"
  - Cocoa programmers call it an attribute or to-one-relationship
- "Element Class"
  - Cocoa programmers call it a to-many relationship

What a Scripting Dictionary Looks Like

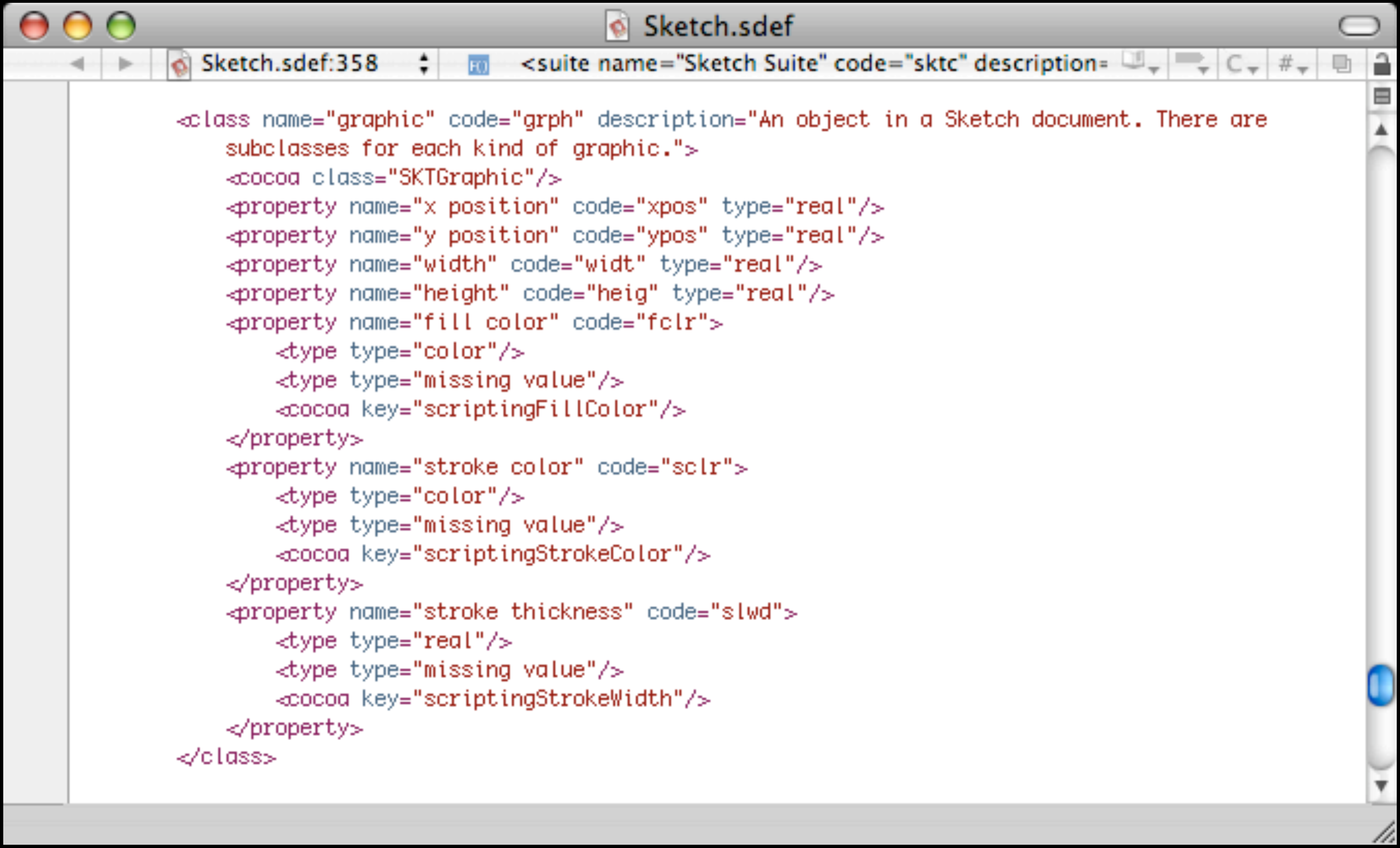


graphic *n* : An object in a Sketch document. There are subclasses for each kind of graphic.

ELEMENTS
contained by documents.

PROPERTIES
x position (real)
y position (real)
width (real)
height (real)
fill color (color or missing value)
stroke color (color or missing value)
stroke thickness (real or missing value)

What .sdef Looks Like



```
<class name="graphic" code="grph" description="An object in a Sketch document. There are
    subclasses for each kind of graphic.">
    <cocoa class="SKTGraphic"/>
    <property name="x position" code="xpos" type="real"/>
    <property name="y position" code="ypos" type="real"/>
    <property name="width" code="widt" type="real"/>
    <property name="height" code="heig" type="real"/>
    <property name="fill color" code="fclr">
        <type type="color"/>
        <type type="missing value"/>
        <cocoa key="scriptingFillColor"/>
    </property>
    <property name="stroke color" code="sclr">
        <type type="color"/>
        <type type="missing value"/>
        <cocoa key="scriptingStrokeColor"/>
    </property>
    <property name="stroke thickness" code="slwd">
        <type type="real"/>
        <type type="missing value"/>
        <cocoa key="scriptingStrokeWidth"/>
    </property>
</class>
```

The Code You Write

The Code You Write

- Make the standard commands work
  - get
  - set
  - count
  - exists
  - make
  - delete
  - duplicate
  - move

Implement Scriptable Classes

- .sdef

```
<class name="graphic" code="grph" description="…">
    <cocoa class="SKTGraphic"/>
    …
</class>
```

- Code

```
@interface SKTGraphic : NSObject<NSCopying> {
…
@end
```

Implement Scriptable Classes

- Why does Cocoa care about the classes?
  - The Make command
  - Type checking

Implement Properties

- .sdef

```
<property name="fill color" code="fclr"
     type="color">
    <cocoa key="fillColor"/>
</property>
```

- Code

```
- (NSColor *)fillColor {
  …
}
- (void)setFillColor:(NSColor *)fillColor {
  …
}
```

Implement Properties

- Why does Cocoa care about the properties?
    - Object specifier evaluation
    - The Set command
    - The “with properties” argument of the Make and Duplicate commands

Implement Properties

- .sdef

```
<property name="fill color" code="fclr"
     type="color">
    <cocoa key="fillColor"/>
</property>
```

- Code

```
- (NSColor *)fillColor {
  …
}
- (void)setFillColor:(NSColor *)fillColor {
  …
}
```

Implement Properties

- How do AppleScript types map to Objective-C classes?
  - Declared in .sdef files
  - Standard types are declared by Foundation
    - Completely documented
    - Intrinsics.sdef
  - You can add new types

Implement Properties

- .sdef

```
<property name="fill color" code="fclr"
     type="color">
    <cocoa key="fillColor"/>
</property>
```

- Code

```
- (NSColor *)fillColor {
    …
}
- (void)setFillColor:(NSColor *)fillColor {
    …
}
```

Implement Properties

- How does AppleScript property access map to Objective-C messages?
    - Key-value coding (KVC)
        - Used by Cocoa Bindings and Core Data
        - Many implementation options

Implement Elements

- .sdef

```
<class name="document" code="docu" description="A
     Sketch document.">
    <cocoa class="SKTDrawDocument"/>
    <element type="graphic"/>
    ...
</class>
```

  - <cocoa key="graphics"/> is optional

Implement Elements

- Code

```
- (NSArray *)graphics {
    …
}
- (void)insertGraphics:(NSArray *)graphics
      atIndexes:(NSIndexSet *)indexes {
    …
}
- (void)removeGraphicsAtIndexes:
      (NSIndexSet *)indexes {
    …
}
```

Implement Elements

- How does AppleScript element access map to Objective-C messages?
  - More KVC
    - Again, many implementation options
  - Scripting KVC vs. regular KVC