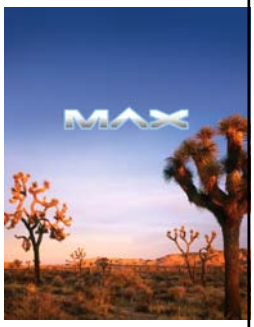
MAX 2006

Increasing Efficiency and Quality with Flex Automation

Alistair McLeod

Adobe Consulting



Who Am I?

- Alistair McLeod
  - Technical Leader at Adobe Consulting
  - Rich Internet Applications and LiveCycle, EMEA
  - Based in Edinburgh, Scotland
- Background
  - Enterprise Software Development
  - Rich Internet Application Development
  - Co-founder of iteration::two
  - Co-author "Developing Rich Clients with Macromedia Flex"
  - Core contributor to Cairngorm and FlexUnit

Introduction

- What is Automation?
- Automated Build Systems
- Unit Testing
- Functional Testing

What is Automation?

- Automatic control of software
  - Automated build systems
  - Automated test execution
- Removes need for manual intervention
  - Humans only involved if something is wrong
- Saves time
- Reduces errors
  - Computers don't make mistkaes

Automated Build Systems

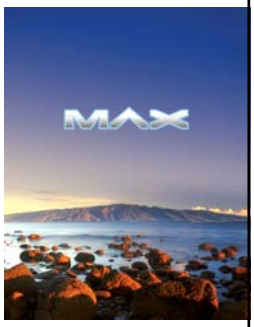
Automated Build Systems

- Monitors source control systems
- Identifies when changes have been made
- Builds the systems
- Verifies that the build is successful
- Notifies registered parties if the build fails

Cruise Control

- Framework for a continuous build process
- Open source
- Plugin-based
  - Source control integration
  - Ant integration
  - Email integration
- Enables continuous integration
  - Identifies integration issues early
- Available in Java and .NET flavours

Unit Testing



Unit Testing

- What is Unit Testing?
- Motivation for Unit Testing
- Test Driven Development
- Unit Testing Flex Applications
- Demonstration

What is Unit Testing?

- Unit Tests are Programmer Tests
- Code written to test application code
- Tests functionality of software components
  - Specifies what our code should do
- Tests at a fine level of granularity
  - At the class function level
- Tests everything that could possibly break
- Supports Test Driven Development

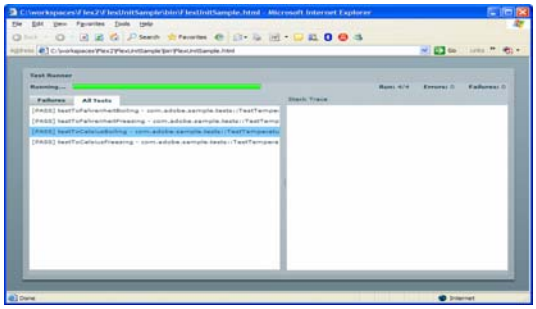
Motivation for Unit Testing

- Improves quality of code
  - Code does what it is supposed to do
  - Builds up a regression test suite
  - Simplifies maintenance
- Reduces development time
  - Bugs caught early in development process
  - Better debugging workflow, no need to trace()
  - Less debugging required
- Enables Agile development practices
  - Supports test-driven-development
  - Supports refactoring
  - Supports continuous integration
  - Supports collective code ownership

Unit Testing Flex Applications

- FlexUnit Testing Framework for Flex
  - ActionScript 3.0 classes
  - MXML <mx:Script> code
- Based on JUnit 3.2
- Now with added asynchronous testing

FlexUnit Demonstration



Automating Unit Testing with Ant

- Ant can be used to run your tests
- Can be run manually or automated
  - Manually from IDE (eg, FlexBuilder)
  - Automatically (eg, Cruise Control)
- Requires no human intervention
  - Notify registered parties only if tests fail

Collecting Automated FlexUnit Results

- FlexUnit Ant Task
  - Launches Flash Player and Opens a socket between JUnit and it
- JUnitTestRunner
  - Formats the test results for JUnitReport
  - Sends Results to Server
  - Collate with server side test results
- Controlled by Cruise Control
- Created by Adobe Consulting EMEA
  - See http://weblogs.macromedia.com/pmartin

Functional Testing

Functional Testing

- What is Functional Testing?
- Testing Flex Applications
- Functional Testing Requirements
- Demonstration
- Functional Testing and Flex

What is Functional Testing

- Tests that applications work as expected
- Tests user gestures
- Tests business logic
- Acts as a regression test
- Can also automate functional testing
  - Ability to automate the user level interactions on applications
  - Ability to verify that the application state is as expected before/after each interaction

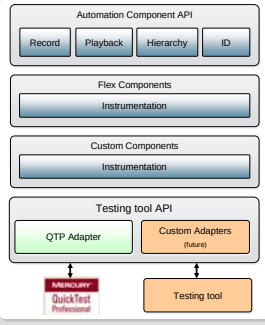
Testing Flex Applications

- Built in integration with Mercury Quick Test Pro.
  - Record and play back interactions with Flex applications
  - Check points to verify the application state
- Flex Framework has Automation built in
  - Automatable components in the Flex SDK
    - Instrumentation adds automation code
  - An automation API for Flex component developers
    - Add automation via instrumentation or "hard-coded"
- Workflow
  - Component developers develop "automatable" components
  - Application developers assemble "automatable" applications
  - QA engineers use a testing tool to automate the testing of the application
- Currently available in beta

Flex Automated Testing Concepts

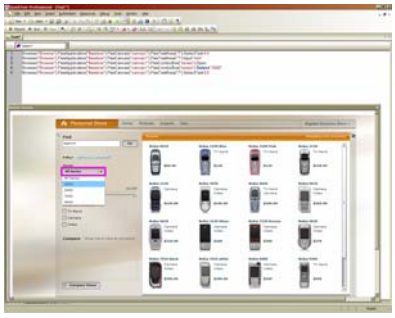
- Identity
  - Need a persistent, human readable identity for every object in the application.
- Hierarchy
  - Not all UI components are relevant, such as boxes
- Record
  - Record a single semantic event for an end-user gesture
  - Checkpoint creation
- Playback
  - Replay semantic events by simulating the original end-user gestures
  - Checkpoint verification
- Testing vendors
  - API to support multiple testing vendors

Flex Automated Testing Architecture



- Automation Component API
  - IAutomationObject
  - IAutomationObjectContainer
  - IAutomationObjectHelper
- Flex components
  - All components will be already instrumented
- Custom components
  - Leverage the existing Flex component instrumentation
  - Can add new instrumentations when needed
- Testing Tool API
  - IAutomationManager
  - QTPAdapter

Functional Testing Demonstration



Functional Testing Requirements

- The tests should not be brittle
  - Bit map comparison of screens is not an option!
  - Cosmetic changes in the application should not require recreation of automation scripts.
- QA engineers should not need to know the internals of the application
- Test scripts should be easy to read and maintain.
- Scripts should not need updating as the application changes

Component Developer Responsibilities

- Composite controls
  - Usually no changes required
- Components extending UIComponent
  - Define the contract with Quick Test Pro
    - Identity of the new component
    - Operations exposed to Quick Test Pro
    - Properties to be used in checkpoints
  - Exclude containers with no visible impact on the app
    - Reduce non interactive components such as boxes, spacers, rules
  - Add dispatch/playback of new interactions

Application Developer Responsibilities

- Human readable unique identifiers for each object in the app
  - Eg: id property of a Panel inside a TabNavigator should be submit_panel rather than p1.
  - Understand default id generation (e.g: Button label)
  - Fix automationName during objects life cycle (beta API)
- Different labels for multiple tabs in accordion, tabNavigator etc
- Compile the application with "automatable" compiler flag turned on
- HTML wrapper with object id (Flex Builder does this automatically)

Identification

- Persistent Identity
  - A collection of unchanging, persistent attributes
  - Examples: id, Alert title, image source, button text
- Display Name
  - Special attribute in the persistent id displayed to the QA engineer
  - id is used by default
  - Overridden by numerous controls with other values such as source or toolTip
  - Can be set by developer
  - index is used when no name is available
  - IAutomationObject.automationName

Verification

- Many properties & styles are exposed for verification
  - Examples: visible, enabled, fillAlphas
  - Complex values are transcoded – arrays, colors, objects, selectedItems
- Value
  - Special property created for automated testing - IAutomationObject.automationValue
  - Similar to itemToString but more general
  - An array of properties
    - Used to populate tabular data
    - Most components return an array of one - containers return an array of values
- Tabular data
  - A list of all values in a tabular-like UI component
  - Examples: List, DataGrid, Accordion & Flexstore products & checkout
  - Not the same as dataProvider - WYSIWYG
  - Not just visible values (Lists go through all values in the dataProvider and create an offscreen item renderer for each one to capture the displayed value)

Hierarchy

- Parent/child relationship can differ from display hierarchy
  - IAutomationObject.automationParent
  - IAutomationObjectContainer.getAutomationChildAt
  - Examples: Combobox, ColorPicker, Menu, & Repeater
- Hide "composited controls"
  - IAutomationObject.isCompositeChild
  - Examples: Button in ScrollBar
- Hide meaningless components
  - Too many components in the hierarchy is confusing
  - IAutomationObject.showInAutomationHierarchy
  - Reduce non interactive components such as boxes, spacers, rules
  - Some parents force children into the hierarchy, i.e. Accordion

Record & Playback

- Record
  - Components are responsible for what/when to record
  - Record events signifying high level user interactions
  - Recording information is put into an Event
- Playback
  - Components are responsible for replaying their recordings by simulating the replay with low-level events
  - Components add synchronization callbacks Eg: ComboBox open/close, DragManager drag

Summary

- Numerous tools available for testing and automation
- Ant for building and running tests
- Cruise Control for automated building and testing
- FlexUnit to unit test your Flex applications
- Quick Test Professional and Flex Plugin for Functional Testing


Questions
Questions?
2006 Adobe Systems Incorporated. All Rights Reserved.
Adobe