Introduction to Seaside

Randal L. Schwartz, merlyn@stonehenge.com
Version 3.05 on 18 July 2010

This document is copyright 2008, 2009, 2010 by Randal L. Schwartz, Stonehenge Consulting Services, Inc.
This work is licensed under Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported License
http://creativecommons.org/licenses/by-nc-sa/3.0/
Goals

- Components
- Callbacks
- HTML Generation (including forms)
- Persistence (including ORM)
- Deploying your application
- AJAX Integration
- Pier Content Management
Getting Seaside

- Squeak - one click image or packages
- VisualWorks - Store or WebVelocity
- GNU Smalltalk - ?
- VisualAge - ?
- GemStone/S - GLASS
The Counter Example

- Launch one-click image
- Ignore GUI for a moment
- Navigate to URL
- Count!
- All that excitement (yawn)
The Smalltalk GUI

- Workspace - run snippets of code
- Browser - edit and view code
- Debugger - “d” bugs
- Inspector - look at complex values
- Monticello - share and load code
The Workspace

- Evaluate code snippets
- “do it”: just run the code
- “print it”: run the code, show the result
- “debug it”: run the code in the debugger
- Operates on selection
  - If nothing selected, line cursor is on
Smalltalk in a Hurry

- Just enough to understand Seaside
Objects and classes

• Everything’s an object
• An object belongs to a class
• An object has methods
  • The class (also an object) has methods
• A class inherits from a single superclass
  • Class-side and instance-side separately
Variables

- Alphanumerics
- CamelCased with initial lowercase:
  - rate, accelerationRate
- Value belongs to a class, variables don’t care
- Assign to get a value:
  - rate := 30
- Special vars:
  - self, true, false, nil, super, thisContext
Methods

- Unary: single name, follows variable:
  - rate squared
- Binary: 1-2 punctuation chars:
  - rate * time
- Keyword: names and colons:
  - rate raisedTo: 2.5
  - rate between: 5 and: 10
- Simple precedence!
Literal data

- Strings: ‘hello world’
- Numeric data: 3 2.5 1.23e45 -2e-5
- Symbols: #size #foo:bar: +#
Classes

• Alphanumeric, initial uppercase
• Class methods are often constructors:
  • rates := Set new.
• But could also have other uses:
  • superclassOfSet := Set superclass.
  • defaultCar := Car default.
Method syntax

- Signature (like message send without self):
  - squared
  - * aNumber
  - raisedTo: aNumber
  - between: lowNumber and: highNumber
- Temporaries: | aDog aCat |
- Statements separated by periods
- Last statement can have ^ ("answer this")
- Comments are in "double quotes"
Control structures

- **Conditionals:**
  - `aBoolExpr ifTrue: [some. code. here].`
  - `aBoolExpr ifFalse: [some. other. code].`
  - `#ifTrue:ifFalse:, #ifFalse:ifTrue:`

- **Loops:**
  - `[code. code. aBoolExpr] whileTrue.
  - `[code. aBoolExpr] whileTrue: [code].`
  - `#whileFalse, #whileFalse:`
The Code Browser

- Packages - groups of classes
- Classes
- Class/instance/comment toggle
- Method categories (including “all”)
- Method names
- Lower pane views/edits selection
  - Sometimes preloaded with a template
  - Lots of coding help available in menus
The Debugger

- Debug notifier: proceed/cancel/full
- Full debugger:
  - Stack
  - Code pane (current line highlighted)
  - Instance vars
  - Temps and arguments
- Everything is live, editable, resumable
- Action buttons to step in, over, through
Hello World

- Create class for top-level component
  - Should inherit from WACComponent
- class #canBeRoot for GUI access
- Or register during class #initialize
- Components implement:
  renderContentOn: html
- In our case:
  html text: ‘hello world’.
- Stupid smart quotes:
  PreferenceBrowser open
Configure the app

- Visit configuration screen (/config)
- Create a new URL path (below /)
- Select our class as the root component
- Visit the URL!
When the Web Breaks

- Add time display to output
- Refactor it to use concatenation
- Boom! (Needs #asString)
  - Walkback in browser
  - Select debug to use Smalltalk GUI
  - “Proceed”, and browser refreshed
Halos

- Inspect components
- Edit CSS (for prototyping)
- View pretty-printed HTML source
- Edit source code (proof of concept)
Web Velocity

• Stay in web browser for:
  • Code browser
  • Debugger
  • Inspector
  • Source code management
• Everything!
• Lots of scaffolding for database views
Configuring brushes

- Each step separate:
  brush := html heading.
  brush level: 3.
  brush with: ‘my third level heading’.

- Combining first and second steps:
  brush := (html heading) level: 3.
  brush with: ‘...’.

- Any number of configurations
- But #with: has to be last!
• Cascade omits common object:
  batallion selectTank target: enemy; fire.
• Same as (without the variable):
  aTank := batallion selectTank.
  aTank target: enemy.
  aTank fire.
• So our heading looks like:
  html heading level: 3; with: ‘my head’.
Fancier Blocks

• Like methods in square brackets
• Argument list (if any)
  :arg1 :arg2 :arg3 |
• Temporaries (if any)
  | temp1 temp2 |
• Statements
  arg1 dothis. arg2 dothat. arg3 + arg1.
• ^ exits enclosing method, not block!
Example

- add3ToDoubleOf := [:x | x * 2 + 3].
- a := add3ToDoubleOf.
- b := a value: 17.
- c := a value: (17 / 2).
Callbacks

- html anchor with: ‘text’ - not useful
- Exterior links:
  html anchor url: ‘http://www.google.com’;
  with: ‘go go google!’.
- Action links!
  html anchor
  callback: [self increase];
  with: ‘text’.
Components

- Reusable chunks of HTML
- Individual instance vars for state
  - Example: current counter value
- State can be rewound if a URL is reused
  - Or not, depending on coder’s choice
- Components can be nested
  - Need to be declared with `#children`
Back to the Counter

- Simple code
- Instance var holds the counter
- Main view shows increase/decrease buttons
- Actions linked via callbacks
Collection classes

- OrderedCollection
- Array (fixed size OrderedCollection)
- Set
- Bag (counted items of Set)
- SortedCollection (order from chaos)
- Dictionary (key/value mappings)
- Interval (5 to: 100 by: 3)
Collection protocols

- **collect:**
  - fractions := (1 to: 10) collect: [:n | 1 / n]

- **do:**
  - fractions do: [:each | html div: each]

- **select:**
  - overQuarter := fractions select: [:f | f > 0.25]
Multicounter

• Just a bunch of counters
The Back Button

- Callback URLs modify instance vars
- What if we reinvoke the same URL?
- Might want original action on old value
  - Or maybe on new current value
- To act on old value, use #states
  - Values are associated with URLs
  - Frozen and thawed as necessary
- Otherwise, acts on current value
• Add class to any relevant brush:
  html div class: ‘entry’; with: ‘some text’.
• Style with CSS
  • External files can be edited by designer
• Simple styles for testing defined inline
  • String returned by #style on component
Forms

• Input fields painted with brushes
• Callbacks executed on form submission
• No need to name anything:
  html text: ‘name:’.
  html textInput callback: [:v | self name: v].
• Default values can be provided
• With right accessors, code is simple:
  html textInput on: #name.
LCM two numbers

- Build code to do this
Persistence Solutions

- Saving the image
- Writing objects
- Object prevayler (Sandstone, Prevayler)
- Object database (GemStone/S, Magma)
- Object/Relational Mapper (GLORP)
Why Persist?

- Objects naturally live in the image
- Image lives only when it’s alive
  - And only on one machine
- Data survive past a reboot
- Data shared with other live images
- Data shared with non-Smalltalk processes
Reachability

• Objects rarely stand alone
• Objects contain other interesting objects
  • ... which in turn contain others
• Might even be circular references
• Persistence must deal with this
Migration

- Classes will change through development
  - Add methods
  - Add instance variables
  - Change instance variables
  - Rename classes
- Instances in storage will need to migrate
- Some persistence schemes deal with this
Dirty Objects

- How are updates handled?
- Load object, make changes, now what?
- Some systems require notification
- Others notice automatically
- Not relevant for some
  - All saving is explicit
- Flushing only changes often more efficient
Strategies

- Save the image
- Serialize the objects
- Database access
- Object-Relational Mappers
- Image persistency
Saving the image

- Periodically create a snapshot
- Use a low-priority process with delay:
  ```smalltalk
  [[(Delay forSeconds: 300) wait.
    SmalltalkImage current saveSession.
  ] repeat.
  ] forkAt: Processor userBackgroundPriority.
  ```
- Simple
- Could save broken image
- Timestamp your image names
Objects

- Serialize your application objects
- Two main mechanisms
  - File Based
    - Simple
  - Server Based
    - Can often be clustered for scaling
File Based

- ReferenceStreams
- ImageSegments
- SIXX
- OmniBase
- MinneStore
- SandstoneDB
ReferenceStreams

• Subclass of DataStream (and Stream)
• Knows how to serialize objects
• Great example from Ramon Leon’s blog:
  (ReferenceStream newFileNamed:‘blog’)
    nextPut: self allPosts.
• Object returned from “self allPosts” saved
  • And all objects referenced therein
• Restore with:
  allPosts :=
    (ReferenceStream oldFileNamed:‘blog’) upToEnd.
    “do something with allPosts”
ImageSegments

- Store part of the live image on disk
- Automatic activation (I think)
- Documentation is obscure
- Avi Bryant is was using this for DabbleDB
SIXX

• Load “SIXX” and an XML parser
• Every object can turn into XML and back
• Customize the XML for your objects
• Example read/write:
  (SixxWriteStream newFileNamed: ‘blog’)
    nextPut: self allPosts.
  allPosts :=
    (SixxReadStream readOnlyFileNamed: ‘blog’)
    contents
• Sadly, they don’t inherit from streams
OmniBase

- Portable object database
- Claims:
  - Good concurrency support
  - Large collections
- Loaded into Squeak 3.8 but not later
- Any object (and descendants) can persist
- “Dirty” objects must be marked
- Transactions (MVCC) are supported
MinneStore

- Not compatible since Squeak 3.4
- Tests failed on 3.8
- minnestore.sourceforge.net claims:
  - English-like query language
  - Multiple indexes
  - Transactions (commit/rollback)
  - Automatic disk reclamation
- “main site” is now a parked domain
SandstoneDB

- Ramon Leon’s favorite tool
- Maybe because he wrote it
- Prevayler-style storage
- ActiveRecord API
- All active objects must live in memory
- Atomic save
Server Based

• Increased complexity
• Possibility of scaling to multiple processes
  • Perhaps even multiple machines
• Main players in this arena
  • GOODS
  • Magma
GOODS

- Language-neutral
- Even Perl, via Pogo in the CPAN
- Requires configuring a server (C++)
- Persistence by reachability
  - Add objects
  - Make changes
  - Updates get pushed at commit
- Commits may result in exceptions
  - Objects updated in conflicting ways
Magma

- Modern local or clustered persistence
- Simple transaction protocol
  - ACID fault tolerance
- Handles large indexed collections
- Live class migration
- Active development
- Production deployments
Using Magma

- Get a session
  - Remote connection via TCP
  - Local connection to a file (single-user)
- Use the session to make changes
  - Create a commit block
  - Via session root, update referenced data
  - Commit or rollback
- New reachable objects automatically added
Magma and Seaside

- Seaside glue via “Magma Seaside” package
- Add WAMagmaConfiguration
- Use resulting configuration screen
  - Configure location of server
  - Type of connection (single, shared, pool)
- Get at Magma with “self session magma”
- Can point Magma root at your app data
Database access

- SQLite
- PostgreSQL
- ODBC
- Other unmentionable databases
  - “Friends don’t let friends use MySQL”
  - “If you can afford a full-time DBA, you can afford Oracle”
SQLite

• Wrapper around fully featured SQLite lib
  • Transactions, triggers, stored procs, more!
• Uses FFI to talk to native SQLite lib
  • External dependency!
• Won’t load into Squeak 3.8, 3.9 or 3.10
• “Connect” to SQLite
• Use connection to execute SQL
  • Results returned as structure
• Doesn’t handle placeholders
PostgreSQL

- Conceptually similar to SQLite
- Lot more power
- Nearly complete PG protocol
  - Placeholders
  - Streaming results
  - Events
- Slightly outdated
  - Needed monkey patch to work with 8.2
ODBC

- Ditto
  - Create connection
  - Send SQL
  - Get back data
- Haven’t played with this
O-R Mappers

- GLORP
- Cincom’s ActiveRecord
- Roe
- CouchDB
- TokyoTyrant
- SqueakDBX
- Cloudfork-AWS
GLORP

• Maps objects to tables
• Large
• Mostly underdocumented
  • But it comes with a large test suite
• PostgreSQL seems to be primary target
  • But some of it must be outdated
• SqueakDBX layer also said to be compatible
GLORP Descriptor

- What a table contains
- What objects will participate
- How to map them
- Very flexible system
- Perhaps too flexible
- Tests provide a variety of examples
GLORP Mappings

- Simple mapping is one-to-one
  - Describe a table
  - Create an object that looks like a row
  - Describe that object and its mapping
- Complex mappings
  - One row becomes variant object type
  - Multiple rows combine for one value
- Possible to map existing objects to RDB
GLORP Basics

- Create a session object
  - Made from an accessor object
    - Made from a connection object
    - Made from a login object
- Within a unit of work, make changes
- Use session to query existing data
- Create new objects and add them
  - Sub-objects are automatically noted
- Commit or rollback the unit of work
ActiveRecord

- Cincom is building as part of WebVelocity
- Handles the “simple mapping” of GLORP
- Probes the database
  - Can also create the database
- Presumes certain naming conventions
  - Not good for legacy databases
- Probably handle 80% of basic persistence
- Open source “soon” we’re told
Roe

- Relational algebra evaluator
- Underdocumented
  - But at least it has tests
- Appears to be abandoned
  - But at least it talks to PostgreSQL
- Write SQL as Smalltalk using DNU-trapper
  - Evaluated expression pushed to RDB
- Handles queries and updates
  - but not inserts (I’m guessing here)
CouchDB

- Map Smalltalk objects to CouchDB
- Haven’t played with this
Tokyo Tyrant

- Interface to Toyko Cabinet
- Haven’t played with this
SqueakDBX

- Interface to OpenDBX
- DB-neutral abstraction for DDL and SQL
- Summer-of-code 2008 project
- Haven’t played with this
  - Although it sounds very promising
CloudforkAWS

- Data in the cloud
- Interface for Amazon S3
- Haven’t played with this
Image Persistency

- Have the entire image just persist
- Only one player in this space
  - That I’m aware of
- GemStone/S
- See Dale’s blog
GLASS

- GemStone/S object engine
- Linux, Apache, Smalltalk, Seaside
- All wrapped up in a VMWare appliance
- Free to use for small applications
  - Even commercial applications!
- Not open source though. :( 
- As hits increase, scale up for modest fee
Ajax

- JavaScript library integration
  - jQuery (now)
  - Scriptaculous (legacy)
- No need to write JavaScript
  - Everything is coded from Smalltalk!
- [demo]
Testing

- Component level testing
  - Using standard Smalltalk Unit Tests
- HTTP level testing
  - Seaside Testing package
  - Albatross (like Selenium)
More info

• http://seaside.st/
• http://MethodsAndMessages.vox.com/