froglogic

Web GUI Testing with Froglogic Squish

October 2020, Edwards Life Sciences

Alan Ezust, Systems Engineer

froglogic

Agenda

Introduction
Squish GUI Tester
Resources
JavaScript Basics
Basic Squish Usage
Object Identification
Multi-touch Gesture Support
Squish Extensions
Accessing Application Internals
Test Synchronization
Test Framework
Data-driven Testing
Advanced Browser Hooking
Advanced Test Execution

froglogic

Agenda

Introduction

About froglogic

Squish GUI Tester

Resources

JavaScript Basics

Basic Squish Usage

Object Identification

Multi-touch Gesture Support

Squish Extensions

Accessing Application Internals

Test Synchronization

Test Framework

Data-driven Testing

Advanced Browser Hooking

Advanced Test Execution

About froglogic • Company Facts

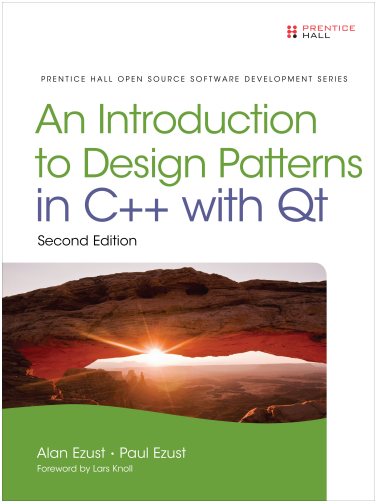
- Established in 2003
- Headquarters in Hamburg, Germany
- US Presence since 2009
- Company focus on software testing
  - Squish GUI Tester
  - Squish Coco

About froglogic • Customers

- Large and growing customer base in USA, Europe an Asia
- More than 3000 companies using Squish

About Alan Ezust

- alan@froglogic.com
- 9 years provider of training, support and consulting services for Qt and Squish
- Trainer of Qt, QML for http://qt.io
- Co-author of Qt, C++ Textbook
- M.Sc McGill University (Computer Science)
- From Boston, lives in Victoria, BC, Canada



froglogic

Activity • Introductions

froglogic

Agenda

Introduction
Squish GUI Tester
Resources
JavaScript Basics
Basic Squish Usage
Object Identification
Multi-touch Gesture Support
Squish Extensions
Accessing Application Internals
Test Synchronization
Test Framework
Data-driven Testing
Advanced Browser Hooking
Advanced Test Execution

froglogic

Squish GUI Tester • Technology Overview

Cross-platform Testing

Multi-Technology / Toolkit Support

froglogic

Squish GUI Tester • Feature Overview

Capture & Playback

Multiple Scripting Languages

Behaviour Driven Development

Verification & Validation

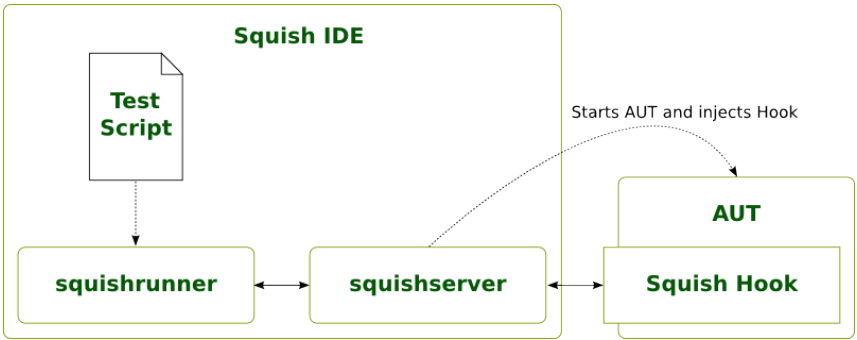
Test Creation Environment

Data-driven Testing

Distributed Batch Testing

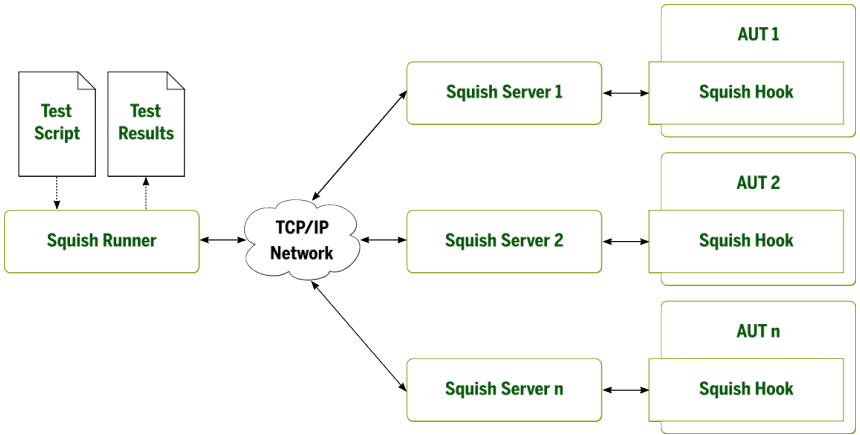
Integration Options

Squish GUI Tester • Architecture (Local Testing)



froglogic

Squish GUI Tester • Architecture (Distributed)



froglogic

Squish GUI Tester • Integrations

Integrate Squish with various build tools, CI servers, or ALM systems:

- Ant
- Maven
- Eclipse IDE, Eclipse Test & Performance Tools Platform (TPTP)
- Jenkins / Hudson
- CruiseControl
- Atlassian Bamboo
- Atlassian Jira
- HP Quality Center
- TestTrackTCM
- IBM Rational Quality Manager
- JetBrains TeamCity
- Microsoft Visual Studio / Team Foundation Server / Microsoft Test Manager

froglogic

Agenda

Introduction
Squish GUI Tester
Resources
- General Resources
- Scripting Resources

JavaScript Basics
Basic Squish Usage
Object Identification
Multi-touch Gesture Support
Squish Extensions
Accessing Application Internals
Test Synchronization
Test Framework
Data-driven Testing
Advanced Browser Hooking
Advanced Test Execution

froglogic

General Resources

- Squish Manual: http://doc.froglogic.com/squish/latest
- Knowledge Base: http://kb.froglogic.com
- Blog: http://blog.froglogic.com
- BDD Tutorial: http://bdd.tips
- Support: support@froglogic.com
  - Step-by-step description of how to reproduce an issue
  - Select Help | Collect Support Information for a ZIP file with log files and package information

froglogic

Scripting Resources · Squish for Web

- JavaScript
  - JavaScript Tutorial
  - Mozilla's Core JavaScript Reference
  - Notes on JavaScript Integration and Extension API in Squish
- Squish API
  - Configuration API
  - Edition-independent API
- Web Object API
  - API specific to Squish for Web
- JavaScript Tutorial

froglogic

Agenda

Introduction
Squish GUI Tester
Resources
JavaScript Basics
History
Basic Squish Usage
Object Identification
Multi-touch Gesture Support
Squish Extensions
Accessing Application Internals
Test Synchronization
Test Framework
Data-driven Testing
Advanced Browser Hooking
Advanced Test Execution

JavaScript Basics • History

- JavaScript is a scripting language
- Originally designed to be run in a Web browser for dynamic HTML
- Standardized in ECMAScript (ECMA 262) specification
  - Dynamic typing
  - Object-based
  - Run-time evaluation
  - OOP supported through prototypes
  - API for working with text, arrays, dates, regular expressions
  - No API for networking, I/O, storage, ... (except in Squish)
- Various implementations exist
  - Spidermonkey (Mozilla), V8 (Chrome), JavaScriptCore (Safari), ...
  - Squish
  - ...

froglogic

JavaScript Basics • Syntax

- JavaScript programs consist of
  - statements
  - variable declarations
  - function declarations

JavaScript Basics • Statements

- Most statements are expression statements, normally terminated with a **semicolon**:

```
1  test.log("Hello world!");
2  age = 18;
3  myObject.number = 5;
4  myObject.number++;
5  1 + 1;
```

- Other statements:

```
break, return, continue, ...
```

froglogic

JavaScript Basics · Comments, Whitespace and Line Breaks

- The JavaScript interpreter ignores comments completely
- Whitespace and Line Breaks can be used to format and indent your program

```
1 // Calling a method of an object
2 test.log("Hello world!");
3 // Assigning a number value to a variable
4 age = 18;
```

JavaScript Basics • Primitive Data Types

- Primitive data types:
  - Numbers, e.g. `1230`, `12.25`
  - Strings, e.g. `"This is a String"`
  - Boolean, e.g. `true` or `false`
- Trivial data types:
  - undefined: value of un-initialized variables
  - null: can be used as invalid value
- Objects (discussed later)

JavaScript Basics • Variables

- Variables are named containers to hold data values for later access

```
1 // Declaring a variable (value will be undefined)
2 var myData;
3 // Declaring a variable and initializing it with null object
4 var myData = null;
5 // Assigning a number value to a variable that has been declared earlier
6 myData = "I'm the string";
7 // Accessing a variable, i.e. to log its current value to Squish Test
      Result
8 test.log(myData)
```

froglogic

JavaScript Basics • Variables

- Variables can be declared only once using the `var` keyword
- JavaScript is an untyped language, which means that variables can hold values of any data type

```
1 var myData;
2 // Assigning a String value
3 myData = "Hello world!";
4 // Assigning a number value
5 myData = 5;
6 // Assigning a boolean value
7 myData = true;
```

JavaScript Basics • Variable Scope

- The scope of a variable is the part of the program in which it is defined
  - Global variables: global scope, accessible anywhere in the scripts
  - Local variables: declared and accessible in a specific function only

```
1 var globalVar = "I'm global";
2 function myFunction() {
3    var localVar = "I'm local"
4    test.log(globalVar);
5    test.log(localVar);
6 }
```

JavaScript Basics • Functions

- A function is a piece of reusable code which can be called and executed anywhere in the program
  - Pre-defined functions
  - Custom functions

```
1 // JavaScript standard function
2 print("This is printed to stdout (Runner/Server Log)");
3 // Squish API function
4 test.log("Hello world!");
5 // Custom function implementation
6 function customFunction() {
7     mouseClick(...);
8 }
```

JavaScript Basics • Function declaration

```
1 function FUNCTIONNAME(PARAMETERLIST) {
2     STATEMENTS
3 }
```

- **FUNCTIONNAME**: Unique name
- **PARAMETERLIST**: list of (variable) names, separated by commas
- **STATEMENTS**: statements to be executed when the function is called

froglogic

JavaScript Basics • Function declaration

- Example of a custom function that does some logging

```
1 function formatAndLog(message) {
2     test.log("Log message is:" + message);
3 }
4
5 formatAndLog("Program started");
6 ...
7 formatAndLog("Program finished");
```

JavaScript Basics • Objects

- In JavaScript, everything is an object and derived from the type Object

JavaScript Basics • Objects

- All objects are derived from upper the mother of all objects, the `Object`

froglogic

Agenda

Introduction
Squish GUI Tester
Resources
JavaScript Basics
Basic Squish Usage
- Installation
- Manage AUTs
- Test Development Basics
- Test Structure
- Squish Configuration

Verification Points
Object Identification
Multi-touch Gesture Support
Squish Extensions
Accessing Application Internals
Test Synchronization
Test Framework
Data-driven Testing
Advanced Browser Hooking
Advanced Test Execution

Installation • Squish Packages

- Squish **Package Name Variables**
  - Platform

    win, linux, mac, ...
  - GUI Toolkit

    Qt, Java, Web, ...
  - 32 vs. 64 bit

    Browser, OS might still be 64 bit!
- Example Package Name:

  squish-6.3.1-web-windows.exe

Installation • Squish package download options

- Squish release packages:
  http://www.froglogic.com/secure
- Squish snapshot packages can be requested via
  support@froglogic.com

froglogic

Demonstration • Squish Installation

froglogic

Activity • Squish Installation

- Install Squish package through matching installer

froglogic

Demonstration • Example application

Activity • Example application

- Start the address book web example application
  1. Start the mini web server
  2. Open the address book application in a browser:

     http://localhost:9090/AddressBook.html

froglogic

Test Development Basics • Script-based Test Creation

- **Recording:**

  Record user interaction with AUT and let Squish generate the test script

- **Manual scripting:**

  Write each Test Case by programmatically scripting all contents

- **Hybrid approach:**

  Most common approach, using a combination of recording, refactoring and scripting

froglogic

Test Development Basics • BDD: Step-based Test Creation

Name steps before they are recorded.

- **Add** a new BDD test case (.feature file)
- Describe, or name **steps** of a scenario in Gherkin (natural language)
- Easily **reuse** steps from previous tests
- Click **Record missing steps in scenario**

Test Suites can have a mix of BDD and script-based tests.

Test Development Basics

- Creating a new Test Suite
  1. Select File|New Test Suite or click on the New Test Suite button in the toolbar

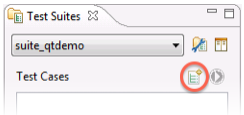
  2. Select a directory to save the Test Suite at
  3. Enter a name for the Test Suite
  4. Select the toolkit used by the application
  5. Select JavaScript as scripting language
  6. Click on Browse to select the application executable at

     Skip this step, URL will be entered on recording

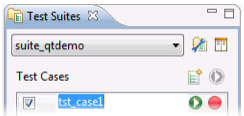
froglogic

Test Development Basics (cont.)

- Creating a new Test Case
  1. Select File|New Test Case or click on the button New Test Case in the Test Suites view

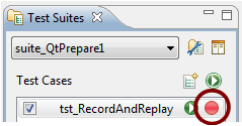


  2. Enter a Test Case name

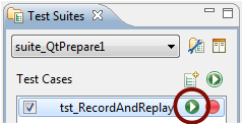


Test Development Basics (cont.)

- Recording a Test Case
  1. BDD: **Record Missing Steps in Scenario**
  2. Scripts: Click on the red Record button next to the Test Case
  3. Record the following workflow
     - 3.1 Create a new address book
     - 3.2 Create entries for a couple of persons
     - 3.3 Delete an entry



- Executing a Test Case
  1. Click on the Play button next to the Test Case



froglogic

Activity • Test Development Basics

- Record and playback a test in Squish IDE
  1. Create a Test Suite
     - 1.1 Select File|New Test Suite or click on the New Test Suite button in the toolbar
     - 1.2 Select a directory and a name for Test Suite
     - 1.3 Select JavaScript as scripting language
     - 1.4 Click on Browse to select the application executable at Skip this step, URL will be entered on recording
  2. Create a Test Case

     Select File|New Test Case or click on the button New Test Case in the Test Suites view
  3. Record and Playback
     - 3.1 To record click on the red Record button next to the Test Case
     - 3.2 To play click on the Play button next to the Test Case

froglogic

Test Development Basics • Recording Snippets

- Record a script snippet into an existing script to
  - Record smaller portions
  - Extend an existing script
  - Re-record parts of a test

froglogic

Recording Snippets(1)

- Recording snippets **without** prior test execution
  1. Point cursor at the line of code where the code should be placed
  2. Select Run|Record Snippet or right-click and select Record Snippet from the context menu
  3. Click on the Stop Recording button
- To avoid superfluously recorded lines, prepare the AUT before:
  1. Click Launch AUT button in the toolbar 
  2. Prepare AUT state
  3. Record snippet as before
  4. Click Quit AUT button in the toolbar 

froglogic

Activity • Recording Snippets(1)

- Extend an existing Test Case (**without** prior test execution)
  1. Record a new Test Case that enters one address book entry
  2. Extend the Test Case so that it adds two more entries by recording snippet without prior test execution
     - 2.1 Point cursor at the line of code where the code should be placed
     - 2.2 Click Launch AUT button in the toolbar 
     - 2.3 Prepare AUT state (create a new address book)
     - 2.4 Select Run|Record Snippet or right-click and select Record Snippet from the context menu
     - 2.5 Add two more entries to address book
     - 2.6 Click on the Stop Recording button
  3. Run the script to verify that it replays without errors

froglogic

Recording Snippets(2)

- Recording snippets **with** prior test execution to prepare the application state
  1. Set a breakpoint at the line where the code should be placed
     - Either double-click on the line number, or
     - Right-click on the line-number and select Add/Toggle Breakpoint
  2. Execute the Test Case until the breakpoint is reached an execution stops
  3. Select Run|Record Snippet or right-click and select Record Snippet from the context menu
  4. Click on the Stop Recording button
  5. Click on the Terminate or Resume button 

Activity • Recording Snippets(2) (with prior test execution)

- Extend an existing Test Case
  1. Record a new Test Case that enters one address book entry
  2. Extend the Test Case so that it adds two more entries by recording snippet with prior test execution
     - 2.1 Set a breakpoint at the line where the code should be placed
     - 2.2 Execute the Test Case until the breakpoint is reached an execution stops
     - 2.3 Select Run|Record Snippet or right-click and select Record Snippet from the context menu
     - 2.4 Add two more entries to address book
     - 2.5 Click on the Stop Recording button
     - 2.6 Click on the Terminate or Resume button
  3. Run the script to verify that it replays without errors

Test Structure • Test Suite

- Collection of Test Cases
- Configuration files
  - `suite.conf` - suite configuration
  - `config.xml` - optional configuration of test settings
  - `envvars` - optional environment variables for the AUT
- `shared/scripts/names.js` for object recognition
- Possibly: shared scripts, search images, verification points, test data
- All files are text files meant to be under revision control

froglogic

Test Structure • Test Case

- JavaScript test script
  - main()
    - Mandatory
    - Main entry point of the test
    - Generated on recording
  - init()/cleanup()
    - Optional
    - Automatically called before/after main()
  - Custom functions
    - Optional
    - Reusable, maintainable code

Squish Configuration

- `server.ini`

  Specific to Squish server installation (**not** IDE)

- **Squish IDE Preferences:**

  Specific to the Squish IDE

froglogic

Demonstration • Squish Configuration

froglogic

Squish Configuration • Server Settings

- Edit settings **specific to squishserver** (discussed later) via Edit|Server Settings
  - Browser
  - Application Behaviour
  - Playback
- Stored in plain-text configuration files
  - Windows:

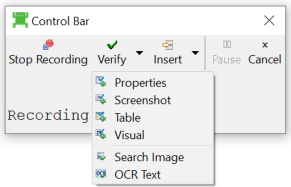
    `%APPDATA%/froglogic/Squish/ver1`
  - Linux/OS X:

    `~/.squish/ver1`
  - Override with environment variable `SQUISH_USER_SETTINGS_DIR`
    - Server settings are placed under that in `ver1/`

Verification Points • VP Types

A **Verification Point (VP)** is used to **verify the expected result(s)** of a given test scenario

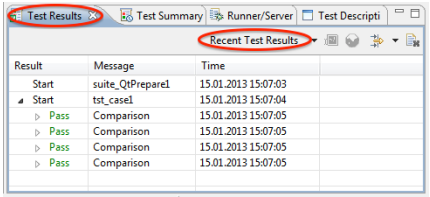
- **Verification Points** types
  - Object Properties
  - Screenshots
  - Tables
  - Visual
  - Image presence
  - OCR Text presence
- **Verification Points Creation**
  - Can be written manually
  - Can be generated using the **Control Bar** during recording



froglogic

Verification Points • Test Results

- The **results of a verification** are logged and shown in the Test Results view
  - **Result history** can be access via **Recent Test Results** combo box



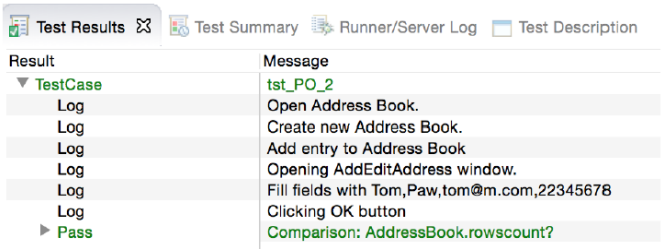
- Default Test Result **directory**:
  - Windows: `Documents/Squish Test Results`
    (customize via Edit|Preferences|Squish|Logging)
  - Linux/OS X: `~/.squish/Squish Test Results`
    (customize via Squish|Preferences|Squish|Logging)

froglogic

Verification Points • Additional log messages

- Provide more information by logging additional messages to the Test Result:

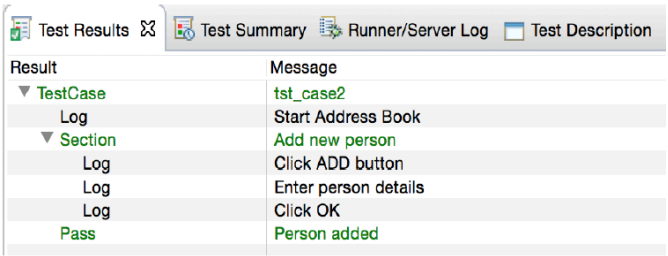
```
test.log(message);
```



froglogic

Verification Points • Additional log messages - Test Sections

- Group test results into logical units
- Start section - test.startSection(title)
- End section - test.endSection()



Verification Points • Properties Verification Point

Properties Verification Points **compare properties of an object** with an expected value.

- Properties Verification Point **sub-types**:
  - **Scriptified Properties VP**:
    - One `test.compare(...);` statement for each property
    - VP data is in the script itself
  - Non-Scriptified **Properties VP**:
    - One `test.vp(...);` statement for all properties
    - VP data is stored in an associated VP file in the Test Suite directory

froglogic

Demonstration • Verification Points

froglogic

Demonstration • Properties Verification Point

froglogic

Activity • Properties Verification Points

- Create Properties Verification Points to verify entries of the address book table
  1. Add two persons to address book
  2. Verify text property of each table cell (surname, name, email and phone):
     - 2.1 Create a Scriptified Properties VP to verifty a first person
     - 2.2 Create a (non-Scriptified) Properties VP to verify a second person

froglogic

Demonstration • Additional Logging with Properties VPs

Verification Points • Logging Screenshots

- Configure Squish to create screenshots at VPs or in case of general script errors
  - Screenshots are saved in the Test Result directory
  - Screenshots can be opened from the Test Results view
  - Screenshots are visible in Web Report
- Enable Test Suite Settings|Test Settings
- Enable at the beginning of a Test Case:

```
1 // Create screenshot for failing VPs
2 testSettings.logScreenshotOnFail = true;
3 // Create screenshot for passing VPs
4 testSettings.logScreenshotOnPass = true;
5 // Create screenshot in case of general script errors
6 testSettings.logScreenshotOnError = true;
```

froglogic

Demonstration • Logging Screenshots

Verification Points • Screenshot Verification Points

- **Verification of a screenshot** of a specific object
  - Comparison modes:
    - **Strict**: strict image comparison
    - **Pixel**: tolerance values configurable
    - **Histogram**: histogram-based image comparison
    - **Correlation**: statistical correlation between images
- **Configuration** of screenshot comparison
  - **Mask portions** of the image to include or exclude from comparison
  - The screenshot and information pertaining to the verification point is stored in an **associated VP file** in the Test Suite directory
    - Test Case Resources|VPs
    - Test Suite Resources|VPs

froglogic

Demonstration • Screenshot Verification Points

froglogic

Activity • Screenshot Verification Points

- Create a Screenshot Verification Point that verifies a screenshot of a button of the Web example application

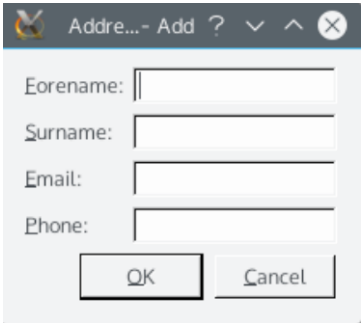
froglogic

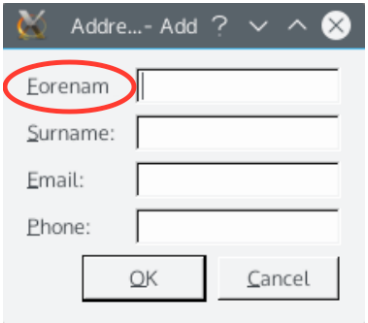
Visual Verifications • Property vs. Screenshot Verifications

- **Object Property Verifications**
  - Verifies specific content that is displayed through HMI
  - Does not verify visual appearance
  - Reliable approach for functional testing
- **Screenshot Verifications**
  - Sees the application as the user does
  - But: verifies many things at once, flattens the object hierarchy

froglogic

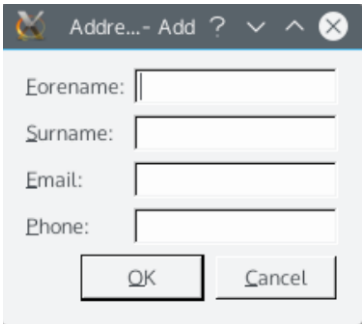
Visual Verifications • Content

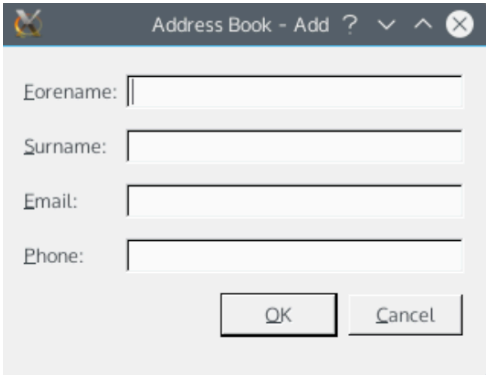




froglogic

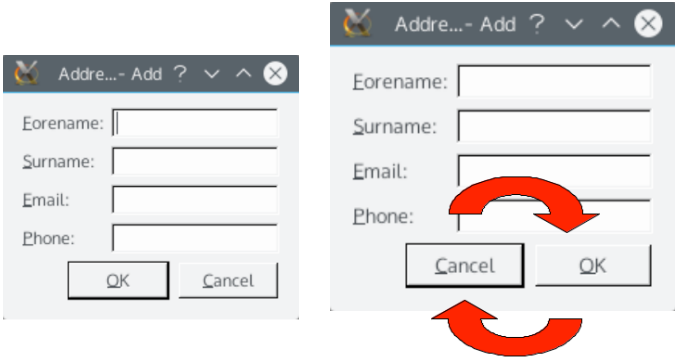
Visual Verifications • Geometry





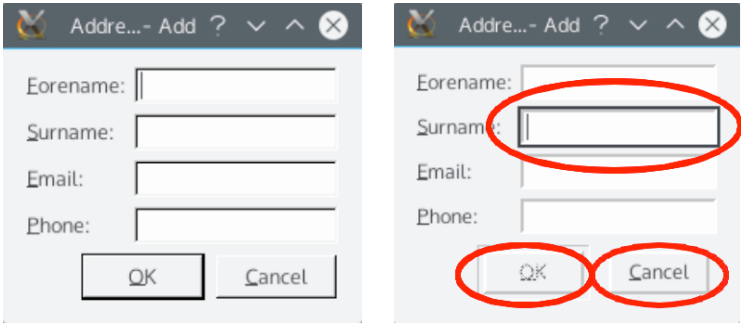
froglogic

Visual Verifications • Topology



froglogic

Visual Verifications • Visual Appearance/Rendering

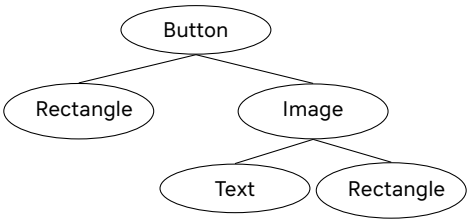


froglogic

Visual Verifications • Complex controls

Use Visual Verifications to verify:

- Complex controls
- Group of controls
- Dialogs, Panels



froglogic

Demonstration • Visual Verification Point

Activity • Insert Visual Verification Point

- Perform Visual Verification for Squish Addressbook buttons bar
  1. Start recording a new Test Case
  2. Insert a Visual Verification Point from the Control Bar
  3. Use the Picker Tool to pick a buttons bar
  4. In the Application Objects select the checkbox in front of picked object
  5. Click “Save and Insert Verification” button
  6. Finish recording
  7. Run the script and check the Test Result

Verification Points • Image Verification Points

- Verify that image is present on the display(s)
- One `test.imagePresent(imageFile, [parameterMap], [searchRegion]);` statement for image to verify
- Search for the image is performed left to right, top to bottom
- The first successful match is selected (use occurrence parameter to change it)
- Available parameters:
  - occurrence
  - timeout [ms], interval [ms]
- Search region:
  - entire desktop (default)
  - AUT object
  - ScreenRectangle object

froglogic

Demonstration • Image Verification Points

Verification Points • OCR Text Verification Points

- Perform optical character recognition to verify that text is present on the display(s). For each text to verify use one statement

```
test.ocrTextPresent(text, [parameterMap], [searchRegion]);
```

- The first successful match is selected (use occurrence parameter to change it)
- Available parameters:
  - occurrence (default is 1)
  - timeout [ms] (default is 20s), interval [ms]
  - language, profiles, options
- Search region:
  - entire desktop (default)
  - AUT object
  - ScreenRectangle object

froglogic

Demonstration • OCR Text Verification Points

froglogic

Verification Points • Custom Verification Points

- Verification Points can also be scripted manually using the verification methods of the global test object:
  - Comparing two values: `test.compare(value1, value2, [message]);`
  - Comparing two files:
    - `test.compareTextFiles(expectedFilePath, actualFilePath, [options]);`
    - `test.compareXMLFiles(expectedFilePath, actualFilePath, [options]);`
    - `test.compareJSONFiles(expectedFilePath, actualFilePath);`
  - Comparing a boolean expression: `test.verify(condition, [message]);`
  - Log a fail message to the Test Result: `test.fail(message, [detail]);`
  - ...

Verification Points · Custom Verification Points

- Example of a custom Verification Point:

```
1 if (object.exists(names.surname_LineEdit)) {
2     test.pass("Surname field exists");
3 }
```

- To get a list of all VP methods available, use the search field of the Squish HMTL help and search for “test.”

froglogic

Verification Points • Comparing files

- Comparing two Text files:

```
Boolean test.compareTextFiles(expectedFilePath, actualFilePath, [options]);
```

  - strictLineEndings

- Comparing two XML files:

```
Boolean test.compareXMLFiles(expectedFilePath, actualFilePath, [options]);
```

  - ignoreNodeOrder, ignoreAttributeOrder, ignoreComments, ignoreCDATA
  - whitespaceInTextHandling, whitespaceInValuesHandling
  - matchSpecialTags
  - filteredElements, filteredAttributes

- Comparing two JSON files:

```
Boolean test.compareJSONFiles(expectedFilePath, actualFilePath);
```

froglogic

Demonstration • Comparing files

froglogic

Agenda

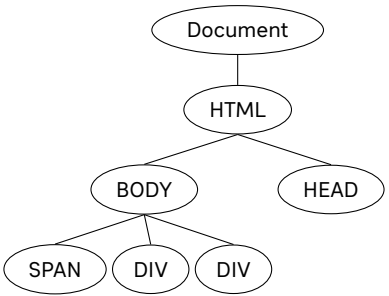
Introduction
Squish GUI Tester
Resources
JavaScript Basics
Basic Squish Usage
Object Identification
- Object Map
- Object Lookup Errors
- Symbolic vs. Real Names
- Scripted Object Map
- Best Practices
- Object Name Generation

Multi-touch Gesture Support
Squish Extensions
Accessing Application Internals
Test Synchronization
Test Framework
Data-driven Testing
Advanced Browser Hooking
Advanced Test Execution

froglogic

Object Map · Application Object Tree

- The objects (or HTML elements) of a web application are organized in a **hierarchical tree structure**, called the **DOM tree hierarchical tree structure**
  - Each node in the tree is an HTML element with a set of **object properties**:
    tagName, type, innerText, x, y, ...
  - The **object tree changes** depending on the state of the application



Object Map • Contents of the Object Map

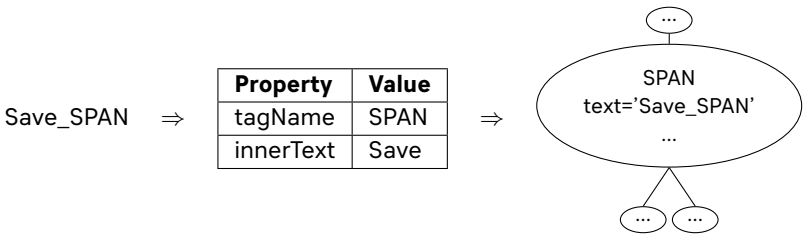
- Each Test Suite manages an object repository, the **Object Map**, that contains a **mapping for the application object tree**
  - Each Object Map entry is identified by a **Symbolic Name**, for example:

    names.Save_SPAN
  - For each Symbolic Name, the Object Map stores a **list of properties and values**, for example:

Property	Value
tagName	SPAN
innerText	Save

Object Map · Object Lookup with the Object Map

- The following process takes place when Squish executes a script and tries to find an object:
  1. Symbolic Name found in test script
  2. Lookup Symbolic properties and values in Object Map
  3. Search for object that matches properties and values



Object Map • Object Lookup Functions (1)

- The following lookup functions can be used in a script
  - `Object waitForObject(objectName, [timeout]);`
    `Object waitForObjectItem(objectName, itemText, [timeout]);`
    - Object **must be accessible** (existing, enabled, visible)
    - Default timeout is 20000 ms
  - `Object waitForObjectExists(objectName, [timeout]);`
    - Object **must exist**
    - Default timeout is 20000 ms

froglogic

Object Map • Object Lookup Functions (2)

- The following lookup functions can be used in a script
  - `Object findObject(objectName);`
    - Object **must exist**
    - Returns a reference to the object
  - `SequenceOfObjects findAllObjects(objectName);`
    - Object **must exist**
    - Returns a list of object references
  - `Boolean object.exists(objectName);`
    - Checks for object existence

Activity • Object Lookup Functions

- Declare a variable and store an object reference returned by

```
Object waitForObject(objectOrName);
```

- Set a breakpoint in the test script and run it
- Use the Test Debugging perspective to inspect an object reference and to execute the script step-wise by using the controls of the Debug view

froglogic

Demonstration • Object Map Editor

Object Map • Symbolic Names

- A Symbolic Name
  - is a variable
  - thus its name **can be changed**

froglogic

Object Map • Property List

- Properties are just a **sub-set of the application object properties**
- The property list **can be modified** to change the way the application object is recognized
- Minimum **requirements** of the property list:

  Squish for Web:
  - The **tagName** property is mandatory
  - The **tagName** value has to be uppercase
  - One additional property has to be used

froglogic

Object Map • Object Map Population

- Adding Object Map entries (Symbolic Names + property list):
  - Entries are generated automatically for application objects that have been interacted with while recording
  - Another option is to select objects individually to create entries when scripting manually
  - Existing entries are reused if possible

froglogic

Demonstration • Object Map Population

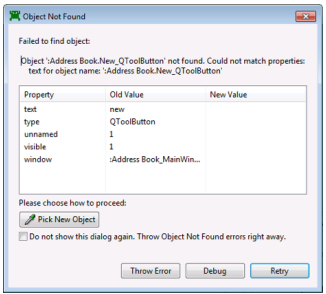
froglogic

Activity • Object Map Population

- Add a new object map entry by copying a symbolic name
  - Launch the AUT
  - Use the Pick tool and select Edit button from web application
  - Trigger a context menu on the selected object in the Application Object tree
  - Copy Symbolic Name
  - Check a status displayed in the status bar of the Squish IDE
  - Check the content of the Object Map

Object Lookup Errors • Object Not Found Dialog

- The Squish IDE will display the Object Not Found dialog in case no object with the given properties can be found within the timeout
  - Instant debugging of an object lookup error
  - Switching to the Squish Test Debugging perspective



froglogic

Demonstration • Object Not Found Dialog

froglogic

Activity • Object Not Found Dialog

- Use the Object Not Found dialog to update the property list for an object map entry
  - Modify one of the Object Map entries' text property
  - Run a test that uses the modified entry
  - Update the entry with Object Not Found Dialog

Object Lookup Errors • Handling Object Lookup Errors

- In case an object lookup error is expected for some reason, it can be handled in the script itself
  - `Boolean object.exists(objectName);`
  - Exception handling

Object Lookup Errors • Handling Object Lookup Errors

- Check for object existence to prevent an expected object lookup error:

```
1 if (object.exists(names.newButton){
2    clickButton(waitForObject(names.newButton));
3 } else {
4    test.log("Button not there, continue anyway");
5 }
```

- Still to solve: synchronization issues (discussed later)

Object Lookup Errors • Handling Object Lookup Errors

- Handle object lookup error through script exception handling:

```
1 try {
2    checkBox = waitForObject(
3       names.MakePaymentCheckSignedQCheckBox);
4    test.pass("Found the checkbox as expected");
5 } catch (err) {
6    test.fail("Unexpectedly failed to find the checkbox", err);
7 }
```

Symbolic vs. Real Names · Object Map Revisited

- The content of the Object Map is stored in a script file, located in the Test Suite's shared resources:

  `<SUITEDIR>/shared/scripts/names.js`

- Every script-based object map is made of the same components:
  - A separate module **objectmaphelper** is loaded.
  - A name space is created, commonly called **Names** or **names**.
  - The object map entries.

froglogic

Demonstration • Object Map Revisited

froglogic

Symbolic vs. Real Names · Real Names

- A Real Name lists a number of **properties and values**
  - Real Names are **dictionaries**
  - Dictionary keys are property names
  - Dictionary value can be:
    - a property value
    - a variable
    - a dictionary
- Example:

```
{"tagName":"DIV", "innerText":"Hello World!"}
```

froglogic

Symbolic vs. Real Names · Object Name Equivalence

- Symbolic Names and Real Names are equivalent, both types can be used in scripts
  - Using Symbolic Names in a script, Squish will use the Real Name from the Object Map

```
waitForObject(names.DIVHelloWorld);
```

  - Using Real Names in a script, Squish won't use the Object Map at all

```
waitForObject({"tagName":"DIV", "innerText"="Hello World"});
```

froglogic

Demonstration • Accessing Object Names through Script

froglogic

Demonstration • Getting Real Names from AUT

froglogic

Symbolic vs. Real Names • Dynamic Object Lookup

- **Symbolic Names**
  - have **fixed property values** that cannot be parametrized in the script
- **Real Names**
  - are just dictionaries that can be built at run-time, allowing to **specify an object dynamically**
  - do not depend on an Object Map, thus they can be **useful in Global Scripts** (discussed later)

froglogic

Demonstration • Dynamic Object Lookup

froglogic

Activity • Dynamic Object Lookup

Create a function that can be used to **delete an entry based on the surname**

1. Record a whole workflow
   - 1.1 Create new address book and add a couple of persons
   - 1.2 Click on a cell with a surname
   - 1.3 Remove this entry
2. Declare a function `deleteEntry(surname)`
   - 2.1 Use a **Real Name** and use it to select and delete the table row with the given surname
   - 2.2 Obtain a Real Name of a cell with a surname (use Picker tool)
   - 2.3 Modify the Real Name to parameterize the **title** property, using the function parameter
   - 2.4 Use `mouseClick(...)` and `waitForObject(...)` to select the table cell

Scripted Object Map · Converting Object Map

In the past Object Maps were text files:

- Symbolic names were Strings starting with a colon
- Real names were specially formatted Strings
- Each Object Map entry was a single line with symbolic and real names separated by a tab

froglogic

Scripted Object Map · Converting Object Map

In the past Object Maps were text files:

- Symbolic names were Strings starting with a colon
- Real names were specially formatted Strings
- Each Object Map entry was a single line with symbolic and real names separated by a tab

Currently Object Maps are **script files**:

- where **symbolic names** are variables
- where **real names** are values of those variables
- that can implement additional logic
- that have a better IDE support

froglogic

Demonstration • Converting Object Map

Scripted Object Map • Object Map Features

For Script-Based Object Map the IDE offers the same features as for other scripts:

- Auto-completion for Symbolic Names
- Renaming Symbolic names
- Find all references and declarations of a Symbolic Name
- Add documentation to Symbolic Names

froglogic

Demonstration • Object Map Features

froglogic

Best Practices · Working with Changing Properties

- Scenario: A changing version number:
  - MyApplication 1.0
  - MyApplication 1.1
  - MyAppli…

froglogic

Best Practices · Working with Changing Properties

- Scenario: A changing version number:
  - MyApplication 1.0
  - MyApplication 1.1
  - MyAppli...
- Solutions to handle such a scenario:
  - Wildcards
  - Regular Expressions

Best Practices · Working with Changing Properties

- Use the following wildcard characters in property values to match a variable text:

?	Matches any single character
*	Matches any number of character, including none
\	Prepend to the characters above for escaping

- To learn how to create regular expressions for property values, refer to the Qt Documentation on Regular Expression

froglogic

Demonstration • Wildcard and Regular Expression

froglogic

Best Practices · Working with Changing Properties

- **Wildcards** can also be used **in Real Names**:

  As a property value use a `Wildcard("<expression_to_evaluate>")` call.

- Example:

```
{"type":"MainWindow", "windowTitle":"myApp V1.2"}
```

becomes

```
{"type":"MainWindow", "windowTitle": new Wildcard("myApp*")}
```

Best Practices · Working with Changing Properties

- **Regular expressions** can also be used **in Real Names**:

  As a property value use a `RegularExpression("<expression_to_evaluate>")` call.

- Example:

```
{"type":"MainWindow", "windowTitle":"myApp V1.2"}
```

becomes

```
{"type":"MainWindow", "windowTitle": new RegularExpression("myApp
V[0-9]+\.[0-9]+")}
```

froglogic

Best Practices · Occurrence Property

- What happens if two (or even more) objects have the same property set?
  - Squish starts counting
  - The counter will be added in the pseudo-property occurrence

Best Practices · Occurrence Property

- **Real Name** examples for a number objects with the **same property set**:
  - 1st object: `{"type":"javax.swing.JTextField", "text":"Okay"}`
  - 2nd object: `{"type":"javax.swing.JTextField", "text":"Okay", 'occurrence':2}`
  - 3rd object: `{"type":"javax.swing.JTextField", "text":"Okay", 'occurrence':3}`
  - ...
- Typical **Symbolic Name** that indicates usage of the **occurrence property**:
  - names.okayJTextField
  - names.okayJTextField_2
  - names.okayJTextField_3
  - ...

froglogic

Best Practices · Occurrence Property

- While the **occurrence property is a workaround** for the problem that the objects do not have unique properties, it makes the **object recognition fragile**
  - In a different Test Case, the counter for the objects might be different due to a different workflow
  - Redesign of the user interface can affect the value of the occurrence property
  - Even invisible internal changes of the AUT can have an impact
- To prevent Squish from using the occurrence property, **unique identifiers** can be set through the toolkit the AUT depends on
  - Drawback: access to source code of the application is required

froglogic

Best Practices · Explicitly Naming Objects

- To assure that Squish generates unique object names, a **unique identifier** can be set:
- **Squish for Web**:
  - Set the **id property**
  - Workaround: use the **class attribute** as a property
    - **Only one class name** is allowed, for example:
      HTML element: <div class=ßubmenu»...
      → {'tagName':'DIV', 'className':'submenu'}

froglogic

Agenda

Introduction
Squish GUI Tester
Resources
JavaScript Basics
Basic Squish Usage
Object Identification
Multi-touch Gesture Support
Gesture Recording
Squish Extensions
Accessing Application Internals
Test Synchronization
Test Framework
Data-driven Testing
Advanced Browser Hooking
Advanced Test Execution

froglogic

Multi-touch Gesture Support • Gesture Types

- Squish supports all different types of gestures
  - Single-touch and multi-touch
  - Curved path recognition
  - Touch-size pressure

froglogic

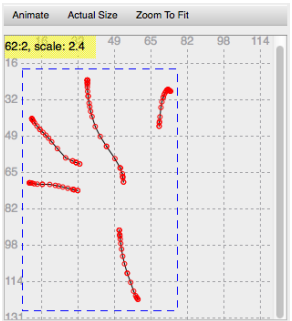
Multi-touch Gesture Support • Single-touch gestures w/o path recognition

- Recorded through specialized Squish API
  - Based on common object recognition
  - Relative positions and offsets
- Examples from the Squish for Android API:

```
touchAndDrag(objectOrName, x, y, dx, dy);
longPressAndDrag(objectOrName, x, y, dx, dy); ...
```

Multi-touch Gesture Support • Multi-touch gestures and path recognition

- For complex gestures are defined in gesture files
  - Generated during recordings
  - Created or manipulated via GestureBuilder API
  - Located in the Test Case or Test Suite Resources



- Gesture execution

```
gesture(obj, readGesture("Gesture1"));
```

froglogic

Demonstration • Gesture Editor

froglogic

Demonstration • Gesture Creation

froglogic

Agenda

Introduction
Squish GUI Tester
Resources
JavaScript Basics
Basic Squish Usage
Object Identification
Multi-touch Gesture Support

Squish Extensions
Accessing Application Internals
Test Synchronization
Test Framework
Data-driven Testing
Advanced Browser Hooking
Advanced Test Execution

froglogic

Squish Extensions · Standard vs. Custom Controls

- Squish supports all **standard controls** of the toolkit
- Support for **custom controls** can be added through **Squish Extension**
  - Squish for Windows SDK
  - Squish for Qt SDK
  - Squish for Java Extension API

froglogic

Agenda

Introduction
Squish GUI Tester
Resources
JavaScript Basics
Basic Squish Usage
Object Identification
Multi-touch Gesture Support
Squish Extensions
Accessing Application Internals
- Object Properties
- Object Methods
- DOM Properties
- DOM Functions
- CSS Attributes
- Browser JavaScript Interpreter

Test Synchronization
Test Framework
Data-driven Testing
Advanced Browser Hooking
Advanced Test Execution

froglogic

Accessing Application Internals • DOM Properties

- All **standard DOM properties** of HTML elements can be accessed through object references, for example

```
text = waitForObject(names.Some_DIV).innerText;
```

- **Non-standard HTML properties** can be accesses as well, for example

```
myProp = waitForObject(names.Some_DIV).property(myProp);
```

froglogic

Accessing Application Internals • DOM Functions

- Equivalent functions to **standard DOM functions** exist, for example

```
parent = waitForObject(names.Some_DIV).parentElement();
```

- All these functions are part of the Squish Web Object API

Accessing Application Internals · CSS Attributes

- **CSS attributes** can be accessed through the HTML_Style object

```
1 style = waitForObjectExists(names.Some_DIV).style();
2 bgColor = style.value("backgroundColor");
3 display = style.value("display");
```

Note: for **composite attributes** (such as background-color), Squish adopts the widely-used JavaScript convention of capitalizing the letter following a hyphen and then dropping hyphens (for example backgroundColor)

Accessing Application Internals • Browser JavaScript Interpreter

- Squish can **execute arbitrary JavaScript code** in the Web Browser's JavaScript interpreter and retrieve the results as
  - **String values**:

    ```
    Object evalJS(code);
    ```

  - **Object reference** to the Browser's interpreter:

    ```
    JsObject retrieveJSObject(code);
    ```

```
1 var style_display = evalJS("var d = document.getElementById('busyDIV');
    d ? d.style.display : ''");
2 var jsobject = retrieveJSObject("var globalObject = { 'id': 'obj1',
    'name': function() { return 'name1'; };globalObject;");
```

froglogic

Agenda

Introduction
Squish GUI Tester
Resources
JavaScript Basics
Basic Squish Usage
Object Identification
Multi-touch Gesture Support
Squish Extensions
Accessing Application Internals
Test Synchronization
- Time-based Synchronization
- Object-based Synchronization
- Conditional Synchronization
- Image-based Synchronization
- Text-based Synchronization
- Event Handling

Test Framework
Data-driven Testing
Advanced Browser Hooking
Advanced Test Execution

froglogic

Test Synchronization

- **Test script execution and the AUT have to be synchronized** to create robust test scripts
- Different parameters can cause **synchronization issues**:
  - (Re-)Painting of the User Interface
  - (Dynamic) reloading of the web app source
  - Background operations in progress
  - Network latencies
  - System resources like CPU or RAM influence performance
- Problem: determining how long will the operations mentioned above take?

froglogic

Test Synchronization • Synchronization Points

- Test script execution and the AUT can be synchronized by creating **Synchronization Points**
- There are different types of Synchronization Points:
  - Time-based
  - Object-based
  - Conditional
  - Image-based
  - Text-based

Synchronization Points • Time-based Synchronization

- Time-based synchronization means to simply let the script **wait for some time**

```
snooze(seconds);
```

- **Disadvantages**:
  - snooze has a **fixed timeout**, thus the script possibly waits longer than necessary
  - A timeout that might be long enough on one machine or in a specific situation **can still be too short** due to changing conditions:
    - Slower testing machines
    - Slower network connections
    - Higher system load

Synchronization Points • Object-based Synchronization

- Object-based synchronization means to **wait for objects** to show up in the user interface
- Squish already records a lot of object-based synchronized code using the following functions with **built-in synchronization**:
  - `Object waitForObject(objectOrName, [timeoutMSec]);`
  - `Object waitForObjectItem(objectOrName, itemText, [timeoutMSec]);`
  - `Object waitForObjectExists(objectOrName, [timeoutMSec]);`
- **Advantages**:
  - The functions above do not wait longer than needed
  - Shorter test runs compared to time-based synchronization

Synchronization Points • Object-based Synchronization

- Synchronization with **waitUntilObjectReady()**
- waitUntilObjectReady() is a callback function
  - Called just before waitForObject() returns
  - gets and argument (object which is ready)
- When to use it?
  - Synchronize tests with objects that may not quite be “ready” immediately after the waitForObject() function returns.
  - Give extra time to your AUT to process all pending events before getting new ones from Squish.
- waitUntilObjectItemReady() is a callback function for waitForObjectItem()

froglogic

Demonstration • Time-based vs. Object-based Synchronization

froglogic

Synchronization Points · Conditional Synchronization

- Various conditions can be used for synchronization as well

```
Boolean waitFor(condition, [timeoutMSec]);
```

- A Conditional Synchronization Point
  - continuously checks for the given condition
  - returns either if the condition becomes true, or if the timeout is reached
  - does not have a default timeout, waitFor returns if the condition becomes true only and may run forever!
  - returns the condition result

froglogic

Synchronization Points · Conditional Synchronization

- Examples for Conditional Synchronization Points:

```
1 // Continue Test Case execution after object text property has a value
      'Succeeded'
2 obj = waitForObjectExists(names.CustomObj);
3 if ( waitFor("obj.text == 'Succeeded'") == True){
4    clickButton(waitForObject(names.NewButton))
5 }
```

Test Synchronization • Clearing Web Object Cache

- In same cases, the **browser does not notify Squish** about **dynamic updates** of the DOM tree. This may cause **object lookup problems** because of Squish's internal **caching machanism**.
- **Solution: clearing the internal object cache** Squish will query the browser for up-to-date objects instead of using cached object references:

```
clearObjectCache()
```

Synchronization Points • Image-based Synchronization

- Image-based synchronization means to **wait for images** to show up on the screen.

```
ScreenRectangle waitForImage(imageFile, [parameterMap], [searchRegion]);
```

- All available screens making up the desktop will be searched. For searches on multi-screen setup on macOS please inquire with technical support.
- Available parameters:
  - occurrence
  - timeout [ms], interval [ms]
- Search region:
  - entire desktop
  - AUT object
  - ScreenRectangle object

Synchronization Points • Image-based Synchronization

- Examples for Image-based Synchronization Points:

```
1   // Click on icon.png
2   mouseClick(waitForImage("icon.png"));
3
4   // Click on second icon.png, wait maximum 10 seconds
5   mouseClick(waitForImage("icon.png", {occurrence: 2, timeout: 10000}));
6
7   // Click on icon.png, found with tolerance of 99.7
8   mouseClick(waitForImage("icon.png", {tolerant: true, threshold:
        99.7}));
```

froglogic

Demonstration • Image-based Synchronization

froglogic

Activity • Image-based Synchronization

- Create a test case that opens application other than AUT

Synchronization Points • Text-based Synchronization

- Text-based synchronization means to **wait for text** to show up on the screen

```
ScreenRectangle waitForOcrText(text, [parameterMap], [searchRegion]);
```

- OCR done over all available screens making up the desktop
- Available parameters:
  - occurrence
  - timeout [ms], interval [ms]
  - language
  - profiles, options
- Search region:
  - entire desktop
  - AUT object
  - ScreenRectangle object

Synchronization Points • Text-based Synchronization

- Examples for Text-based Synchronization Points:

```
1  // Click on second 'Click me' occurrence, wait maximum 10 seconds
2  mouseClick(waitForOcrText("Click me", {occurrence: 2, timeout:
       10000}));
3
4  // Click on text on the MyWidget object
5  mouseClick( waitForOcrText( "Click me", {}, waitForObject(
       names.MyWidget ) ) );
6
7  // Click on text in the specified area
8  mouseClick( waitForOcrText( "Click me", {}, new
       UiTypes.ScreenRectangle( 100, 100, 200, 300 ) ) );
```

froglogic

Demonstration • Text-based Synchronization

Event Handling

- Test script **execution is linear**:

```
1 startApplication("AddressBook.class");
2 activateItem(waitForObjectItem(names.Address_Book_JMenuBar, "File"));
3 activateItem(waitForObjectItem(names.File_javax_JMenu, "Open..."));
4 ...
```

- But sometimes events occur unexpectedly, for example error dialogs
  - Problem: handling such an event in a linear script
  - Solution: let Squish execute an event handler function in a non-linear way
  - Example: if a dialog shows up, execute a function to automate the dialog

froglogic

Event Handling • Supported Event Types for Web Applications

Event Type	Event (JavaScript method)
AlertOpened	occures when an alert box is opened (window.alert)
BodyUnloaded	occurs when the Squish the browser loads or reloads a page.
ConfirmOpened	occurs when a confirm dialog is shown (window.confirm)
Crash	occurs if the AUT crashes
ModalDialogOpened	occurs when a modal dialog is shown (showModalDialog)
ModelessDialogOpened	occurs when a modeless dialog is shown (showModelessDialog)
PromptOpened	occurs when a prompt box is shown (window.prompt)
Timeout	occurs when the Squish response timeout is reached
WindowOpened	occurs when a new window is opened (window.open)

Event Handling • Implementing an Event Handler

- An event handler is just a **script function**:

```
1 function dialogHandler(dialog) {
2     test.log("Handling a dialog");
3 }
```

froglogic

Event Handling · Installing an Event Handler

- **Event Handlers have to be installed** after the application startup
- Installing a **Global Event Handler**:
  - `installEventHandler(eventName, handlerFunctionName);`
- Installing an Event Handler for specific classes or objects:
  - `installEventHandler(className, eventName, handlerFunctionName);`
  - `installEventHandler(object, eventName, handlerFunctionName);`

```
1 installEventHandler("DialogOpened", "dialogHandler");
```

Event Handling · Uninstalling an Event Handler

- Installed Event Handlers are automatically uninstalled at Test Case end
- Event Handlers can be uninstalled manually
  - `uninstallEventHandler(eventName, handlerFunctionName);`
  - `uninstallEventHandler(className, eventName, handlerFunctionName);`
  - `uninstallEventHandler(object, eventName, handlerFunctionName);`

froglogic

Agenda

Introduction
Squish GUI Tester
Resources
JavaScript Basics
Basic Squish Usage
Object Identification
Multi-touch Gesture Support
Squish Extensions
Accessing Application Internals
Test Synchronization

Test Framework
- Capture & Replay vs. Script
- Refactoring
- Shared Scripts
- Object Map
- Test Framework Structure
- Page Objects design pattern

Data-driven Testing
Advanced Browser Hooking
Advanced Test Execution

froglogic

Test Framework • Problems with Capture & Replay

- Capture & Replay is an **convenient way to create tests**
- But Capture & Replay leads to **certain disadvantages**
  - Code duplication
  - Long test scripts with poor readability
  - Hard to maintain, e.g. in case of AUT changes
  - Inflexible approach

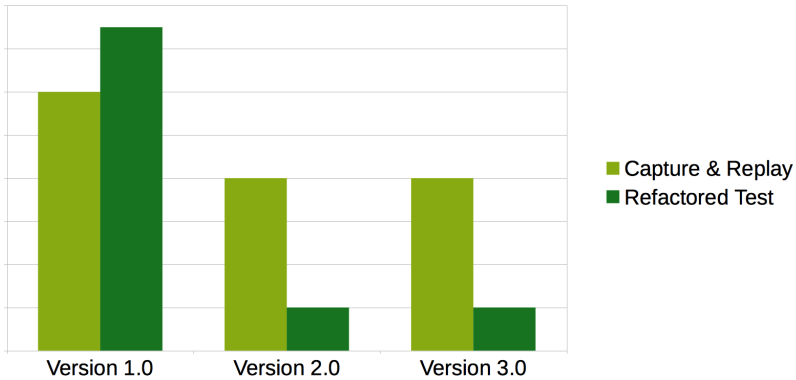
froglogic

Test Framework • Test Case Refactoring

- To overcome the disadvantages of pure Capture & Replay, a **test framework** should be created
  1. **Initial script recording** through Capture & Replay
  2. **Refactoring** of recorded scripts
  3. Make **common functions** available as a **test framework**
  4. Use test framework as a **construction kit for further Test Cases**

froglogic

Test Framework • Test Maintenance Costs



froglogic

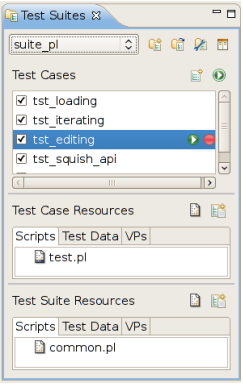
Test Framework · Shared Script Locations

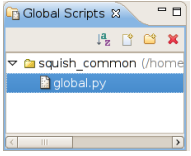
- Choose the **location/type** of a Shared Script to set the **scope of its functions**:
  - Test Case Resources:

    suite_mySuite/tst_myCase
  - Test Suite Resources:

    suite_mySuite/shared/scripts
  - Global Scripts:

    User-defined location





Test Framework · Importing a Shared Script

- **Shared Scripts have to be imported** in a Test Case to make its functions available by calling `source(filepath);`
- Avoid hard-coded file path by using `String findFile(where, filename);`, which will search for the file in the **search order** below:

  Test Case Resources ⇒ Test Suite Resources ⇒ Global Scripts

- Example:

```
1 source(findFile("scripts", "shared.js"));
```

froglogic

Demonstration • Shared Scripts

froglogic

Activity • Shared Scripts

Create a generic function to add a new person to Address Book as part of a test framework in a **Shared Script**

1. Record a basic Test Case with adding a person to address book
2. In the Test Case, begin by extracting a generic function
   - Function interface: **addEntry(forename,surname,email,phone)**
   - Move the code for adding the entry to the function
   - Replace hard-coded input values by the parameters
3. Create a Shared Script in the Test Suite Resources and move the function into this script
4. In the Test Case, **import the Shared Script and call the function**:
   - Add an import statement using the Squish API functions source(...) and findFile(...)
   - Call the function using a set of data

froglogic

Scripted Object Map · Splitting an Object Map

Managing object names can pose to be a real challenge when

- working with tests exercising complex applications
- working with tests which execute multiple applications
- working with tests exercising early development phase application

froglogic

Scripted Object Map · Splitting an Object Map

Managing object names can pose to be a real challenge when

- working with tests exercising complex applications
- working with tests which execute multiple applications
- working with tests exercising early development phase application

The solution might be splitting an Object Map into smaller files based on:

- business logic
- application elements
- object types
- ...

Scripted Object Map · Splitting an Object Map

Split Object Map is fully supported by the Object Map Editor

- all entries are displayed in a one view
- all entries can be edited
- new entries are created in the `names.js` file

froglogic

Demonstration • Splitting an Object Map

froglogic

Activity • Splitting an Object Map

Split an object map based on application views

- Create `mainWindowNames.js` and `addPersonNames.js` shared scripts
- include them in the Object Map
- move object map entries to the corresponding files

froglogic

Scripted Object Map · Object Property Value Translations

Most software comes with translations for user-visible texts.

- Many objects are recognized by text or caption property.
- Object Map can grow to a huge size
- Handling separate/split object map for each language multiplies maintenance costs

Scripted Object Map · Object Property Value Translations

Scripting To The Rescue

```
1 function tr(text) {
2     return {
3         "Open": "Otworz",
4         "New": "Nowy"
5     }[text]
6 }
7
8 export var addressBookOpenQToolButton = {"text": tr("Open"), "type":
      "QToolButton", "unnamed": 1, "visible": 1, "window":
      addressBookMainWindow};
```

Scripted Object Map · Generic Object Map Entries

Object Map can be full of entries with the same sets of properties and similar values

```
1 export var address_Book_New_QToolButton = {"text": "New", "type":
      "QToolButton", "unnamed": 1, "visible": 1};
2 export var address_Book_Add_QToolButton = {"text": "Add", "type":
      "QToolButton", "unnamed": 1, "visible": 1};
3 export var address_Book_Remove_QToolButton = {"text": "Remove", "type":
      "QToolButton", "unnamed": 1, "visible": 1};
```

Scripted Object Map · Generic Object Map Entries

Script-based object maps allow resolving duplication between object names by defining functions which act as parameterized object names.

```
1 function toolButton(text) {
2     return {"text": text, "type": "QToolButton", "unnamed": 1,
          "visible": 1};
3 }
4
5 export var address_Book_New_QToolButton = toolButton("New");
6 export var address_Book_Add_QToolButton = toolButton("Add");
7 export var address_Book_Remove_QToolButton = toolButton("Remove");
```

froglogic

Activity • Generic Object Map Entries

Create a function that returns real names for labels at Addressbook - Add dialog

- in the Object Map create **label(text)** function
- function should return parameterized Real Name for object of type label
- for each label object in Object Map replace real names with the function calls

Check that your soultion is correct

- check modified entries in the Object Map Editor
- successfully replay any Test Case where data in Addressbook - Add dialog is filled

froglogic

Test Framework • Test Framework Structure

- There is **no general rule** on how a test framework should be structured, except that general rules of software development apply:

  Don’t repeat yourself, **avoid code duplication**!

- Functions of a test framework can encapsulate code for
  - Complex controls
  - Whole dialogs
  - Complete workflows
  - Helper tasks
- The structure depends on the complexity of the application, the tests and also on organizational aspects.

froglogic

Test Framework • Page Objects design pattern

- Object Oriented framework
- **PageObject** – a class/object representing part of AUT
- Split framework into two layers
  - **business layer**
    - code at very abstract level (interface to interact with AUT)
    - focused on functionality (“what to do” not “how to do”)
    - self-explanatory naming
  - **technical layer**
    - interaction with AUT using Squish API

Squish API - is about Qt widgets (findObject, clickButton)

PageObject API - is about the AUT functionality

Test Framework · Page Objects design pattern

- **Basic rules**
  - Do not expose the internals of the page
  - Methods represent the actions the page offers
  - Do not make assertions inside Page Objects
- Demonstration
  - What **functionality/services** does the AddressBook AUT offer?
  - What **higher level elements** is the AddressBook AUT built from?

froglogic

Test Framework • Page Objects design pattern

- Page Object methods should return and accept:
  - **simple types** - strings, dates, numbers, Booleans
  - **Domain Objects**
- List of values should be returned as list of primitive types
- Use **Domain Objects** - test are more expressive and readable

Activity • Page Objects

Extend an example from Demonstration. Verify that pressing a Cancel button during adding new entry does not increase number of entries in Address Book.

1. Extend `dialog.py` file in Test Suite resources:
   - Add new class variable `CANCEL_BUTTON` for `DialogPO` class
   - Add new function `cancel()` in a module
2. Extend `add_person_dialog.py`
   - Add `from pageobjects.dialog import cancel` at the top
3. In Test Case call method `add_person_dialog.cancel()`
4. Call `addressbook.get_entries_number()` to get number of entries in Address Book

froglogic

Agenda

Introduction
Squish GUI Tester
Resources
JavaScript Basics
Basic Squish Usage
Object Identification
Multi-touch Gesture Support
Squish Extensions
Accessing Application Internals
Test Synchronization
Test Framework
Data-driven Testing
- Test Data
- Data Access

Advanced Browser Hooking
Advanced Test Execution

froglogic

Data-driven Testing · Separate Data and Implementation

- Data-driven testing can be used to **separate the input or verification data from the implementation** of the Test Case
  - Avoid hard-coded values in scripts
  - Vary test data using the same test implementation
  - Extend tests by adding test data to the data source

froglogic

Data-driven Testing · Data File Types

- Squish supports using different table-based data sources:
  - Excel
  - CSV
  - TSV
- Data tables can be
  - created using the Squish table editor
  - externally created and imported (e.g. using Microsoft Excel)

Data-driven Testing · Data Location

- Test data can be stored at different locations:
  - Test Case Resources

    ```
    suite_mySuite/tst_myCase/testdata
    ```
  - Test Suite Resources

    ```
    suite_mySuite/shared/testdata
    ```
  - Custom directories

froglogic

Demonstration • Test Data

Data-driven Testing · Data Access

- In Test Cases, test data is accessible through Test Data Functions of the Squish API:

```
1 for (var record in dataset) {
2     var forename = testData.field(dataset[record], "forename");
3     var surname = testData.field(dataset[record], "surname");
4     var email = testData.field(dataset[record], "email");
5     var phone = testData.field(dataset[record], "phone");
6     ...
7 }
```

froglogic

Demonstration • Data Access

froglogic

Activity • Data-driven Testing

- Create a data-driven test that reads input values (contacts) to be entered from a data table
  - Use Make Code Data Driven from the context menu to generate the loop
  - Use the shared function created earlier to add entries

froglogic

Agenda

Introduction
Squish GUI Tester
Resources
JavaScript Basics
Basic Squish Usage
Object Identification
Multi-touch Gesture Support
Squish Extensions
Accessing Application Internals
Test Synchronization
Test Framework
Data-driven Testing
Advanced Browser Hooking
    Squish Web Proxy
Advanced Test Execution

froglogic

Advanced Browser Hooking • Squish Web Proxy

- Squish can test web applications running on **various browser on mobile devices**. For this approach, the **Squish Web Proxy** has to be used.
  1. Start the Squish Web Proxy
  2. Configure browser to load content via proxy
  3. Configure squishserver to connect via proxy
- Limitations:
  - Automation of native file dialogs is unsupported
  - File upload dialogs are unsupported
  - Parallel recording / playback is unsupported
  - Screenshot Verifications on remote machines are unsupported
  - HTTPS connections are unsupported
  - Pop-up blockers have to be disabled
  - No gesture support on iOS

froglogic

Demonstration • Squish Web Proxy

froglogic

Agenda

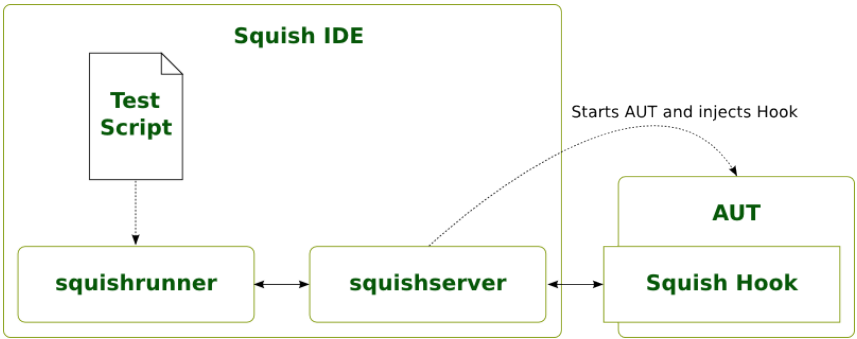
Introduction
Squish GUI Tester
Resources
JavaScript Basics
Basic Squish Usage
Object Identification
Multi-touch Gesture Support
Squish Extensions
Accessing Application Internals
Test Synchronization
Test Framework
Data-driven Testing
Advanced Browser Hooking

Advanced Test Execution

- Command-line Tools
- Tagged execution
- Automated Test Execution
- Squish Integration

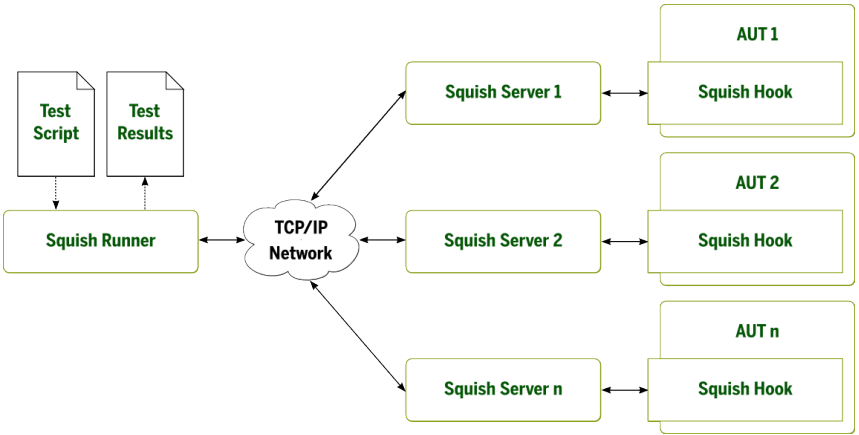
froglogic

Command-line Tools • Architecture (IDE)



froglogic

Command-line Tools • Architecture (Distributed)



froglogic

Command-line Tools • squishrunner & squishserver

- Squish uses two main components for test execution
  - **squishrunner**
    - The squishrunner process interprets the test script and sends the commands of the script to the squishserver process
    - `<SQUISHDIR>/bin/squishrunner(.exe)`
  - **squishserver**
    - The squishserver process starts the AUT, hooks into it and finally executes the commands received by the squishrunner process
    - `<SQUISHDIR>/bin/squishserver(.exe)`

froglogic

Command-line Tools • squishrunner & squishserver

- Executing tests locally from the command-line
  1. Start squishserver, e.g.

     ```
     # squishserver --port {PORTNUMBER}
     ```

  2. Start squishrunner, e.g.

     ```
     # squishrunner --port {PORTNUMBER} --testsuite {TESTSUITEDIR}
     ```

     or

     ```
     # squishrunner --port {PORTNUMBER} --testsuite {TESTSUITEDIR} --testcase
     {TESTCASENAME}
     ```

  Notes:

  - For a list of available command-line option, execute
    - `# squishserver --help`
    - `# squishrunner --help`
  - To stop squishserver, press <CTRL+C>

froglogic

Activity • Command-line Tools

Execute a test locally using the command-line tools

1. Quit the Squish IDE
2. Start squishserver
3. Start squishrunner

```
# squishrunner --testsuite <test-suite-directory> --testcase <testcase-name>
```

Command-line Tools • Test Results

- By default, squishrunner will log a summary of the Test Result only
- Alternative **configuration of the Test Result** format when starting squishrunner:

```
# squishrunner [--reportgen report-generator[,filename|directory]]
```

report-generator		
xml3	Squish 6 format (default)	directory
html	HTML output (includes JSON)	directory
xml2.2, xml2.1, xml2, xml	older Squish formats	filename
xmljunit	JUnit	filename
xls	Excel format	filename
json	JavaScript Object Notation	directory
stdout	command line	none

- Example, log HTML output

```
# squishrunner --testsuite suite_addressbook_py --reportgen
html,/tmp/results
```

Command-line Tools • Random execution

- Test Cases can be executed in a random order
  - helps with finding unintended dependencies between test cases
  - mitigate Pesticide Paradox
- `# --random` option for squishrunner
- The used sequence number is printed in the report as log message
- `# --random {SEQUENCE-NUMBER}` to reproduce a specific order

froglogic

Tagged execution • Setting a tag

- By default, **all** Test Cases from Test Suite are executed
- `# --testcase` parameter can be used to execute a **single** Test Case
- `# --tags` parameter can be used to execute a **subset** of Test Cases matching given tag
- Assigning a tag to Test Case
  1. Go to Test Management perspective
  2. Open Test Description label
  3. Click on pencil icon on a right
  4. Browse to selected Test Case and enter tag into Tags field

Tagged execution • Execution

- Executing only Test Cases with tag @Lion

```
# squishrunner --testsuite <path_to_suite> --tags @Lion
```

- Executing only Test Cases with tag @Lion **OR** @Ant

```
# squishrunner --testsuite <path_to_suite> --tags @Lion,@Ant
```

- Executing only Test Cases with tag @Lion **AND** @Ant

```
# squishrunner --testsuite <path_to_suite> --tags @Lion --tags @Ant
```

- Executing only Test Cases **WITHOUT** tag @Lion
- `# squishrunner --testsuite <path_to_suite> --tags ~@Lion`

Distributed Testing · Access control

- By default, squishserver will accept connections from squishrunner running on the same machine only
- Additional machines can be allowed by configuration
  - Edit the file `<SQUISHDIR>/etc/squishserverrc`
  - Enter a comma-separated list of IP addresses of the machines that may include the wildcard character *, e.g.

    ALLOWED_HOSTS = 192.168.1.*

Distributed Testing · Configuring squishserver

- Configuration of the default squishserver instance (host and port) squishrunner will connect to
  - Squish IDE:
    - Windows/Linux: Edit|Preferences|Squish|Remote Testing
    - OS X: Squish|Preferences|Squish|Remote Testing
  - squishrunner: specify the command-line options --host {host} and --port {port}
- To specify the squishserver instance in the Test Case itself, additional arguments can be passed:
  - `ApplicationContext startApplication(autName, host, port)`
  - `ApplicationContext attachToApplication(autName, host, port)`

Activity • Distributed Testing

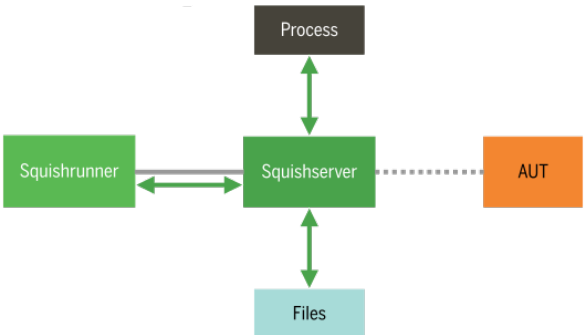
Execute a test on a remote machine

1. Configure access control
   1.1 Open the file <SQUISHDIR>/etc/squishserverrc
   1.2 Set ALLOWED_HOSTS = *
2. Start squishserver from command-line
3. Start squishrunner (IDE or command-line)

froglogic

Distributed Testing · Remote System API

- Perform actions on remote system (where squishserver is running)
  - file system accesses (e.g. reading and writing files)
  - executing commands remotely



froglogic

Activity • Distributed Testing

Remote System API

Execute a test on a **remote machine** that saves Address Book to a file (remote) and then copies this file to local system and checks it's content

1. Add a person to Address Book
2. Save AddressBook
3. Copy *.adr file from remote system to local system

```
RemoteSystem.download(remotePath, localPath)
```

4. Verify content of *.adr file

froglogic

Automated Test Execution • Batch Testing

- Automated batch testing using the command-line tools (pseudo-code):

```
start squishserver
for each test
    start squishrunner
stop squishserver
```

- For Linux/Unix and Windows example scripts, refer to Automated Batch Testing

froglogic

Squish Integration • Squish Plug-ins

- Integrate Squish with third-party build tools, CI servers, or ALM systems:
  - Ant
  - Maven
  - Eclipse IDE
  - Eclipse Test & Performance Tools Platform (TPTP)
  - Jenkins
  - CruiseControl
  - TestTrackTCM
  - Quality Center
  - IBM Rational Quality Manager
  - JetBrains TeamCity
  - Microsoft Visual Studio / Team Foundation Server / Microsoft Test Manager
  - Atlassian Jira
  - Atlassian Bamboo
  - Robot Framework

froglogic

Squish Integration • Jenkins Plugin

- Tight integration into the Jenkins CI
  - Schedule Squish test execution as a Jenkins build step
  - Check Test Results through Jenkins' web interface
  - Define post-test actions, e.g. notifications, statistics, etc.

froglogic

Demonstration • Jenkins Plugin

Thank you very much for your attention!

squish@froglogic.com