Squish Manual

COLLABORATORS			
	TITLE : Squish Manual		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		April 10, 2009	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

List of Tables

Chapter 1

Welcome

This manual presents *froglogic Squish*®, a professional automated GUI testing framework. With *Squish* it is possible to test GUI applications based on toolkits such as Qt® by Trolltech®, the free Tk or Java™ and XView™ from Sun™.

Additionally, *Squish* can be used to test HTML-based web applications running in Microsoft Internet Explorer®, Firefox and other browsers of the Mozilla family, Apple's Safari™ and KDE™'s Konqueror.

Chapter 2

Release Notes

2.1 Version 3.4.4

2.1.1 General

- Added objectMap.load() function.
- Prevent crash that occurred when accessing global AUT properties (like Qt's qApp) after the AUT has been shut down.
- Fixed Perl return values in case of recursive calls caused by an event handler.
- Respect the 'Response Timeout' setting in all cases, to avoid that script commands get aborted after 5 minutes.
- Improved handling of breakpoints in shared script files.
- Added option --updatevp option to convertvp utility to update the image in the verification point.
- Properly escape object names and support unicode characters in Verification and Synchronization Points in Tcl test scripts.
- Fixed memory leak in testData.dataset() in Javascript and Python.
- Fixed test::warning() and test::fatal(), which were broken in some cases when using Qt3 and Perl.
- Support stringwise and numerical comparison operators (cmp, <=> etc.) on wrapped objects wherever the objects allow a meaningful comparison. Also be more verbose when being asked for a string representation of non-string objects.
- Fixed the side effects of the 3.4.3 protection against access to "dead" objects. On repeated AUT starts object class descriptions got mixed up.
- Don't let the Unix startup code get confused if one of the tools (e.g. squish) is searched in PATH when the this variable contains an entry that ends with the same name. This would be .../squish/ in this example.
- Filter out multiply recorded waitForApplicationLaunch() statements.

2.1.2 IDE

- Fixed opening Test Scripts on Windows Shares from Test Result messages.
- Fixed line selection when clicking on a Test Log item with colons in the message.
- Improved stability of the Squish IDE when running tests which print a lot of information to the standard output channel.
- Improve handling of test case templates which are stored in non-standard locations.

2.1.3 Qt-specific

- Support Qt 4.5.
- Improved algorithmic complexity and therefore speed of object name generation and lookup.
- Enable wrapping of constructors defined for QWidget sub-classes.
- Fixed replaying recorded scripts that used Qt3 tearoff menus.
- Fixed a crash on Mac OS X on exit of certain Qt applications with Qt 3.
- Fix replay of menu tests when clicking on menus that have a submenu.
- Preserve non-ASCII characters in existing scripts when a newly recorded script snippet is inserted.
- Improved support for translated strings.

2.1.4 Java-specific

- Support for tab list button from CTabFolder.
- Fixed truncatation of return value from a Java function that is of type float or double.
- Fixed a security exception when loading the Java wrapper library with webstart in some cases.
- Show AWT/Swing dialogs as toplevel objects in the Spy.
- Added new property 'basetype'.

2.1.5 Web-specific

- Support lazy hooking for IE when the body of the document that is to be hooked does not exist yet.
- Fixed choosing entries in <SELECT> elements in case the entry contains a single apostrophe or the backslash character.
- Fixed runtime dependency for testing Web applications with Firefox 3 on Windows.
- Fix retrieval of Web properties which contain a '%' in case the property value is longer than 150 characters.

2.1.6 Tk-specific

- Improved double-click emulation by sending also a release the second click as well.
- Improved menubar replay support by moving the mouse to the right top-level menu.

2.1.7 Source Builds

- Support compilation against Tcl 8.6.
- Fixed compilation with Qt 4.5 on Windows and Mac OS 10.5 (Leopard).
- Don't wrap QStyle sub-classes to make the wrapper more portable. They are not really required for testing purposes.
- Fix compile problem with Qt 4 and gcc 3.2.

2.2 Version 3.4.3

2.2.1 General

- Added a --apply-mask mode to the convertp utility.
- Added currentApplicationContext() function in all scripting languages. Will be used when recording in a multiple-application setting.
- Fixed reading of `.tsv` (tab-separated values format) files that have empty fields.
- Marked references to objects of an exited AUT as "dead" to prevent crashes on further access.
- Fixed a Unicode problem if non-ASCII characters are used in symbolic object names.
- Allow conversion of Tcl double values (or what the interpreter believes are doubles) to a string.
- Added a fallback error message for findObject() to avoid crashes on Solaris.
- Implemented .name and .caller properties on JavaScript function objects that allow assembling a better callstack for improved debugging. Also added a .lineNumber property to exception objects in addition to the .line property for compatibility to the Mozilla JavaScript engine.
- Let the Python test.verify() function to use the standard Python truth value rules rather than restricting it to just int and long values.
- Supress "different type" result details for Tcl command [test compare] when not applicable.
- Let Python's waitFor() gracefully handle expressions that return a non-integer value.
- Changed the JavaScript print() output so that it writes to stdout (instead of stderr) and uses the UTF-8 encoding to display non-ASCII characters properly.
- Fixed crashes that occurred as a result of syntax errors in some Python test scripts.
- Fixed a Unicode problem in the test result output that can occur in rare circumstances.
- Fixed a regression when waitForObjectItem() is called on object names that contain a space.
- Fixed a slight oversight in the improved error reporting. The unmatched properties are now reset when an error is reported and hence don't pollute later error messages.

2.2.2 IDE

- Made the "End recording" button work if testing with sub AUTs and there are sub AUTs running.
- Fixed focus problems that occurred when saving a file.
- Let the Perl variable watcher display the value of mixed string/int and string/double types. Hide reserved variables that are denoted by a leading caret.
- Prevented the creation of test cases with empty names.
- * src/gui/objectmapeditor.cpp: Allow dots when editing hierarchical names in the objectmap editor.
-
-
-
-
-

2.2.3 Qt-specific

- Instead of treating an access to the .text property of an invalid item view index as an invalid operation (that could cause a crash), an empty string is returned.
- Fixed a crash caused by an access to an invalid iterator.
- Avoid parse errors on Object Map name reuse if the map contains a mixture of multi-property and hierarchical names.
- Fixed a possible source for crashes on 64-bit Windows systems.
- Improved the detection of statically linked Qt applications.
- Raise the AUT on playback on Mac OS X with Qt 4.4.
- Fixed a Spy Unicode problem with Qt 4.
- Avoid wrong menu activation when replaying menu actions with Qt 4.4 on Mac OS X.
- Fix recording of QResizeEvent send during the application launch.

2.2.4 Java-specific

- Fixed the replaying of type("swing-object", "<Ctrl+X>").
- Fixed the wrong "selection" property of TreeItemProxy. It was always false.
- Don’t record mouseClick() on glass panes if these events are dispatched in the application and trigger a mouse event in the content pane. Mouse clicks on components below a glass pane will be replayed by sending clicks to the glass pane.
- Let the application context name property of sub-applications be the argv[0] system value rather than using "squish".
- Stopped FigureCanvas only looking at IFigure and ignoring SWT widget children.

2.2.5 Web-specific

- Fixed replaying text input on file input fields with Internet Explorer 6.
- Fixed the recording of backslashes in file input fields with Internet Explorer.
- Support isBrowserDialogOpen() for Mozilla on X11 systems like Linux.
- Fixed a communication problem that could cause objects not being found when testing with Safari.
- Fixed random crashes of the internal webhook tool when doing Safari testing.
- Fixed the recording and playback of text input on input fields with type "search" (they are understood by Safari).
- Support for Kiwi Matrix tables.
- Fixed the handling of strings containing the character sequence ’HTTP’.

2.2.6 Tk-specific

- Improved support for the Tk "scale" widget. Replaying scrollTo now uses real mouse events so custom bindings to mouse-press will also be triggered.
- Fixed infinite recursion when dealing with the VTK KWWidget combobox control.

2.2.7 Mac-specific (Cocoa/Carbon edition)

- Added support for Cocoa toolbars and toolbar items.
- Fixed the object picker (and highlighter) for Carbon applications: sometimes, it did not manage to find the right objects under the cursor.
- Fixed the problem that occurred in some Carbon applications when the application menu did not show up (e.g., in the Spy and during recording and playback).
- Added support for HIDataBrowser. Please view with the wrapperexplorer to see which APIs are supported by the HIDataBrowser and HIDataBrowserCell wrapper classes.
- Fixed the playback of menus which in some Carbon applications sometimes selected the wrong menu item.
- Fixed the recording of menu events on Mac OS 10.5 (Leopard) for Carbon applications.
- Mouse clicks are recorded after the Quit menu option is chosen so that actions on the modification alert can be recorded.
- Allow picking of objects in a sheet in the Spy.
- Fixed a race condition which caused random crashes on playback of waitForObject() on sheets (and may have been the cause of other crashes too).

2.2.8 Source Builds

- Fixed building with Qt 4.5.
- Fixed the 'Record and insert here' feature for when Squish is built without Tcl support.
- Fixed the compilation problem in xml2reportgenerator.cpp which occurred when building squishrunner if JavaScript support was disabled.

2.3 Version 3.4.2

2.3.1 General

- Fixed the variable watcher so that it shows the correct value for remote objects while stepping through a test script.
- Fixed the recording of waitForApplicationLaunch() calls in some situations where multiple AUTs are used.
- Made screenshot verifications work reliably in distributed tests.
- Set the name property of the application context objects that belong to the subprocesses Squish has hooked into.
- Fixed the recording of setApplicationContext() calls for Perl scripts.
- Highlight the if, else and elsif keywords in Perl scripts.
- Respect spaces in the suite.conf file's OBJECTMAP entry.
- Fixed drag and drop replay on Mac OS X.
- Empty entries like ";;" are no longer added to the AUT environment's PATH.

2.3.2 Qt-specific

- Corrected the source object of a drag&drop operation recording that got recorded wrongly sometimes.
- Fixed the replaying of activateItem() statements on items which have backslashes or underscore characters in their text.
- Qt 3 on Mac OS X 10.5: fixed a crash of the AUT on application exit.
- Added a workaround for a Qt 4 bug that causes problems with clickItem() on Q3ScrollView when scrolling is needed to reach the item.
- Made event handlers reliable when they are triggered multiple times by the same event.
- Support compiling against custom Qt 4 installations that have QToolTip, QCursor, QTableView, QTableWidget, QMenu, QContextMenu, and QTabBar disabled.
- Don't require otool to be installed on Mac OS X if you are testing a Qt application.
- Made the spinUp() and spinDown() functions style-independent for Qt 4. Also made recording less time-sensitive, i.e., accept clicks that take a bit longer.
- Prevented a crash when non-button objects are passed to the clickButton() function.
- Wrapped the new Qt 4.4 QPlainTextEdit class
- Fixed support for Q3ComboBox, both standalone and inside a Q3Table.
- Show the text of QStrings in the variable watcher for Qt 4.x and Qt 3.0.x applications.
- Fixed the replay of tests when using a statically linked Qt on Linux.
- Respect the copy action for drag and drop operations.

2.3.3 Java-specific

- Fixed the playback of special keys on JTextField.
- Swing's JTabbedPane now also supports clickTab() with the extra x, y, key-state, and button arguments. This script function gets recorded when the tab is clicked with a key modifier pressed and/or any other button than the left mouse button.
- Fixed screenshot verification points on Mac OS X.
- Made the logScreenshotOnFail work on Mac OS X.
- Added support to the extension API for RCP based applications. This requires JDK-1.5 or higher.
- Fixed tree item selections for items that have non-string user objects.
- Fixed JVM crash when querying the field or method list throws an exception.
- Added support for recording the result of the ColorDialog (just like with the File and Directory dialogs). The script function is chooseColor().
- Fixed the java.awt's Choice and List not sending an ItemEvent on selection change.
- Added a mouseClick(obj, x, y, clicks, state, button) overload for AWT/Swing.
- Fixed some corner-cases in the improved error reporting when object names were using a path instead of a description.
- Fixed binding to SWT objects that are handled via the extension API: Inspectable.isObjectReady() was never called, Inspectable.mapFromGlobal() result was handled as canvas coordinates, better defaults for InspectableAdapter.map{To,From}Global, using the Inspectable.getGlobalBounds, JavaExtensionDir didn't handle factory classes with a package prefix.

2.3.4 Web-specific

- Fixed the execution of the automateLogin() and isBrowserDialogOpen() functions when running Web tests with Firefox 3.
- Fixed a lot of potential crashes when invoking functions like grabWidget(), mouseClick(), clickItem(), and others, with null object arguments.
- Fixed the 'Record Script and Insert Here' feature so that it works properly with Web test suites.
- Fixed error findObject()'s error reporting for object names that don't use multiple properties.
- Fixed replay of listbox selections for GWT by sending a real change event.

2.3.5 Tk-specific

- Fixed the emission of the "application started" event which is needed for script recording when there are multiple AUTs.

2.4 Version 3.4.1

2.4.1 General

- Support Ctrl+A select all in the object map editor.
- Fixed various problems when executing Tcl scripts which involved the 'array set xyz { ... }' statement.
- Fixed occasional crash when right-clicking on test script code in the IDE while having selected not more than one line of text.
- Fixed an infinite loop at the end of executing certain test cases with the Valgrind add-on.
- Avoid a bug in the spy which occurred when selecting an unnamed object in the object spy.
- Fixed occasional creation of broken test case templates on Windows.
- Pass additional commandline arguments in startaut to the AUT on Windows as well.
- Properly show texts with '<' or '>' in the control bar while recording a test.
- Make removing breakpoints from a test script while the test is being executed work on Windows.
- Accelerate creation of verification points by about 600%.
- Avoid crashing the squishrunner when attempting to create a verification point while a previous VP is still being created.
- Fixed Perl debugger when compiled with Perl 5.10.
- Fixed setting of string properties on object via Perl.
- Fixed PATH_MAX declaration was not found on some Linux systems.
- Fixed void symbol conflict with applications that use the TinyXML parser.

2.4.2 Qt-specific

- Fixed support of scrolling in comboboxes using the scrollbar as well as the mouse wheel.
- Improved the monkey test example so that it's more fair when deciding between multiple available widgets in Qt4 applications.
- Fixed the monkey test example so that it properly selects menu items which have no accelerator key associated (and also ignore separator items).
- Fixed the monkey test example so that it properly interacts with QMenu objects in Qt 4 applications.
- Avoid that the monkey test example gets stuck sometimes when encountering popup widgets.
- Fixed a crash in dllpreload if the AUT and its commandline arguments got too long.
- Added the Unix utility 'whoami' to the list of ignored applications.
- Fixed crash when spying a QListView which has no model.
- Fix screenshots on Mac OS X for OpenGL widgets.

2.4.3 Java-specific

- Fixed getLocationOnScreen exception occuring both during recording and replaying.
- Added a --with-java-home switch which is equivalent to setting the JAVA_HOME environment variable for detecting the location of the Java development kit.
- Fixed SWT webstart applications using the JRE javaws tool. Unfortunately, the SQUISH_DISABLE_AWT environment variable has to be set to workaround the fact that the AWT classes are wrapped during the javaws startup and the SWT classes as soon as the webstart application is started.
- Fixed 'java.lang.SecurityException: class "com.froglogic.squish.swt.SquishClassLoader"'s signer information does not match signer information of other classes in the same package'

2.4.4 Web-specific

- Fixed detecting context switches in all cases with Safari.
- Added support for automateLogin on X11

2.4.5 Tk-specific

- Fixed double-click support on canvas based tk widgets.

2.4.6 Native Win32 Support

- Added support for various special key combinations like Shift+End or Shift+Home.
- Fixed replaying key combos involving the Ctrl key on native Windows controls.
- Record & replay pressing the context menu key on native Windows controls.

2.4.7 Native X11 Support

- Added support for sending special keys like **Tab** as native events for X11

2.5 Version 3.4.0

2.5.1 General

- Automatic reuse of Object Map names that use wildcards and regular expressions.
- Improved error messages when an object cannot be found on test execution.
- Wizard for the creation of data-driven tests.
- "Record at Cursor" mode for more efficient test script maintenance.
- The toolkit wrapper can be specified on an application-basis thus allowing for tests of applications using different GUI technologies within one script.
- Simplified navigation between script editor and Object Map.
- Ant built system plugin for Squish test execution.
- Integration plugin for the CruiseControl continuous integration framework.
- Added spinUp() and spinDown() functions for recording and playback of QSpinBox usage.
- Fixed a Perl syntax error in recorded sub-application scripts.
- Cleaned up HTML documentation search index
- More type tolerant Perl setter function for logScreenshotOnFail and logScreenshotOnError configuration.
- Fixed memory management problems of DOM nodes created by the XML object as well as database connections created by the SQL object.
- Don't report that the AUT crashed if it was explicitly killed.
- Implement more verbose error reporting when an object is not found. Now reports the properties that didn't match
- Fixed a JavaScript number-to-string conversion on SPARC and other big endian architectures.
- Allow copy and paste of property name/value in the spy.
- Allow passing Perl strings to functions that accept a numeric type.
- When right-clicking on a selected text in the script editor, offer an option to replace the selected text with a reference to a field of a test data record in case the selected script code is within a test data loop.
- Fixed clicking on Test Log items when the referenced script file contained a '-' or space character.
- Reduced size of object names placed into the clipboard upon copying out of the Spy.
- Prevent the AUT from crashing when doing object lookups with empty strings.
- Implemented support for recording little snippets of script code and inserting it at the current cursor position.
- Fixed invocation of the PDF manual viewer manual on Windows.
- Prevent crash if the Spy is during a "Check all object existence" run.
- Respect logScreenshot settings for test.fail() and test.fatal().
- Fixed crashes occurring after an application launched with startApplication() has exited.
- Support alternative locations for testcase templates by using the SQUISH_TESTCASE_TEMPLATES environment variable.
- Fixed initialization defaultApplicationContext() properties.
- Fixed picking of disabled plain Windows controls.
- Prevent crash in the Perl debugger that occurred when trying to display variables referencing an application context or a test data record.

2.5.2 Qt-specific

- Implemented Qt 4.4 support.
- Binary packages for MSVC 9 and 64-bit Linux.
- Expose QPainter API for test scripts.
- Add support for Input-Method events for Qt 4, so non-latin input is also recorded properly
- Use row and column in clickItem() on Qt 4 tables.
- Mac OS X: make the binary edition work with Qt frameworks that are not installed in a standard framework-search location.
- Support use of wildcards in item names when using the clickItem() function on Qt 4 itemviews like a QTreeWidget.
- Improved parsing of CSV files, using RFC 4180 as reference for the parser implementation.
- Fixed occurrence count recorded for Qt objects which don't have a QObject name set.
- Temporary way to specify custom controls (like OpenGL controls) that require QPixmap::grabWindow() instead of grabWidget() for taking a screenshot: the SQUISH_GRABWINDOW_CLASSES environment variable can be set to a comma-separated list of class names of QWidget sub-classes.
- Fixed recording of actions on native Windows controls with Qt 4.x
- Don't show null objects (i.e. a null pointer to a QWidget) in expandable state in the Variable Watcher of the IDE.
- Respect the --port=xxx argument of the startaut command.
- Fixed calls to functions with C/C++ enum parameters in JavaScript, Tcl and Perl. Was already working in Python.

2.5.3 Java-specific

- Implemented support for Eclipse 3.4 (Ganymede).
- Java record extension API to enable high-level record and Spy support of complex custom controls.
- Fixed item lookup differences between recording and replaying a clickItem() on a SWT Tree.
- Allow launching a SWT FileDialog, MessageBox or DirectoryDialog when the Squish IDE is in Spy mode.
- Adjust x coordinate for SWT Tree item click to the middle of the viewable item rectangle, making clickItem() more stable for differently styled Trees.
- Empty captions of SWT widgets were left out of the object property map, which made these multi-property names match unnecessary many objects. Now they are recorded as: caption=''
- Add 'expanded' property to TreeItem proxy for the Spy.
- Fixed JVM shutdown notification that made the Java process hang on exit in some setups.
- Improved test script speed when interacting with Java classes.
- Properly convert from Java unicode string to QString, so objects containing umlauts in the Spy.
- Added synthetic 'isvisible' property for SWT which internally calls Control.isVisible(). For new names use 'isvisible' in favor of 'visible'.
- Fixed 'Illegal constant pool index' error in the SWT FileOpen class.
- Fixed registered extra classes not being recognized by the script bindings.

2.5.4 Web-specific

- Fix recording of context switches to pages which have non-ASCII characters in their names.
- Provide an additional '--webbrowserargs' command line argument which can be used to pass additional arguments to the browser process started when performing Web tests.
- Fix playback of Web tests with Firefox when a different profile than the default one is used.
- Launch Internet Explorer in a new process to make sure that a new session is used.
- Fixed recording and replay of double and right mouse-button clicks.
- Support the isBrowserDialogOpen() script function on Mac OS X.
- Support the automateLogin() script function on Mac OS X.
- Support Safari 3.1.
- Fix retrieving objects with '%' in their value.
- Fixed entering of non-alphanumerical characters using typeText() with Firefox.
- Improved speed of the Spy when navigating through large web pages.

2.5.5 Tk-specific

- Fixed key event sending via the type() function.
- Record plain mouse press events on buttons that could e.g. be used to open a popup menu.
- Record button clicks with middle and right mouse button.
- Small performance improvement on recording.

2.5.6 Source Builds

- Allow compilation against Perl 5.10
- Restore support for Python 2.2.
- Avoid conflict between Perl flags and Solaris system headers.
- Fixed compile problem with Solaris CC
- Compile fix for strict compiler. abs() is defined in stdlib.h
- Explicitly enforce run-time linking on AIX in order to allow loading of native Python extensions. Contributed by customer.
- Fixed conflict with AIX system headers that was causing a compilation problem.

2.6 Version 3.3.1

2.6.1 General

- Prevent crash if the Spy is closed during a "Check all object existence" run.
- Respect logScreenshot settings for test.fail() and test.fatal().
- Fixed crashes when calling global functions or reading global script properties on an application that has already exited.

- Support alternative locations for testcase templates by using the SQUISH_TESTCASE_TEMPLATES environment variable.
- Fixed initialization of defaultApplicationContext() properties.
- Fixes for the recording and spying of native windows controls.
- Fixed crashes when inspecting invalid objects in the Variable Watcher.
- Handle negative timeouts more gracefully.
- Fixed drag and drop when dragging over very small distances.
- Let the Variable Watcher show Python objects of 'new-style' classes (which inherit the generic 'object').
- Some heuristics for a more relaxed detection of Unix shell scripts.
- Show the complete call trace in case a Python, JavaScript or TSL script fails.
- Fix a crash on Solaris.
- Fixed sending text to native Windows controls when the text contains umlauts and non-ASCII characters.
- Make the response timeout value editable in the server settings dialog.
- Don't show null objects (i.e. a null pointer to a QWidget) as expandable in the Variable Watcher of the IDE.

2.6.2 Qt-specific

- Fixed invocation of functions with enum parameters.
- Support mouseDrag() recording on all widget types.
- When using attachToApplication(), correctly detect the case that another testcase is already connected and throw an error.
- Respect the --port=xxx argument of the startaut command.

2.6.3 Java-specific

- Fixed 'Illegal constant pool index' error when recording File Open actions in an SWT application.
- Fix extra classes not being recognized by the script bindings.

2.6.4 Web-specific

- Fixed entering non-alphanumerical characters using typeText() with Firefox.
- Improved speed of the Spy when navigating through large web pages.
- Fix lookup of 'heuristic' names with dots in their value.
- Fixed typing text in web tests via setText() in case the given text contains backslashes.
- Fixed compile problem with Solaris CC

2.6.5 Source Builds

- Avoid conflict between Perl flags and Solaris system headers.
- Compile without QTabWidget support in Qt, i.e. QT_NO_TABWIDGET being defined.
- Fixed compile problem with AIX xlC.
- Fixed Qt visibility detection for the Intel compiler on Linux.
- Fixed configure switches for enabling/disabling the Windows wrapper.
- Compile fix for strict compiler. abs() is supposed to be defined in stdlib.h.

2.7 Version 3.3.0

2.7.1 General

- Ensure that the line of the breakpoint is visible once it is hit
- Added saveDesktopScreenshot() function
- Added OS.chdir(path) and OS.cwd() JavaScript functions.
- Windows only: fix a crash in the squishserver that occurred in rare cases at the end of a testcase.
- Bring the Squish IDE to the foreground once a breakpoint is hit.
- Fix incorrect file names and line numbers in error messages when exceptions were thrown from imported Python modules.
- Prevent Variable Watcher from crashing when displaying a variable after the AUT exited.
- Fix renaming of columns after right clicking on a column header if the table has been scrolled to the right.
- Properly enable/disable the Record button, menu entry and link on the suite info page depending on whether a default AUT is set or not.
- Don’t show JavaScript functions in the variable watcher.
- Select current AUT path as default path when browsing for a new AUT in the testsuite settings dialog.

2.7.2 Qt-specific

- Fix the monkey test example so that it clicks on tabs again.
- Make recording activateItem() calls on menus more robust.
- Properly wrap QFlags types to avoid ambiguous overloads with int overloads. Other template class fixes.
- Show the attributes of QPointF, QSizeF and QRectF properties in the Spy.
- Fix object lookup after an object was reparented.
- Fix for using Squish IDE with Qt4.4
- Fix recording of nested menu items inside a context menu without title.
- Improved Drag and Drop support.

2.7.3 Java-specific

- Added support for the IBM Java Virtual Machine.
- Added support for Java Web Start.
- Miscellaneous fixes for the GEF support.
- Added support for JFace actions.

2.7.4 Web-specific

- Fixed File Upload on X11 platforms. Also make it work if the button is scrolled out of the view.
- Improved the installer's detection of Firefox installations on Windows.
- Use newer version of XPath code. Statements like 'following-sibling::TABLE' should not result in an endless loop anymore.

2.7.5 Mac-specific

- Fixed most known hooking problems by avoiding up-front class initializations.

2.7.6 Source Builds

- Fixed compilation with Python installations that use the C type long for representing a Unicode character.
- Drastically shorten command line during Qt wrapper build to avoid errors from the Windows 2000 shell.
- Fixed compile and link errors on AIX
- A few percent speedup when building the Qt wrapper.

2.8 Version 3.3.0 Beta 1

2.8.1 General

- Highly improved Object Map editor
- Added object.children() and properties() functions that return a list of child objects and a property map, respectively. Available in all scripting languages.
- waitForObjectItem() and findObjectItem() functions for improved synchronization
- Support for live Object Map maintenance while Spy is running
- Support for an append mode in JavaScript's File.open() function.
- Support an optional SQUISH_LICENSEKEY_DIR environment variable which can be used to define where the license key file should be read (and written).
- Safe handling of NULL values in the Variable Watcher.
- Let the "Uncomment" feature of the script editor only remove the comment marker once so that this operation is symmetric to "Comment".
- Avoid crash in Variable Watcher in case it's triggered after the application exited and tries to access a value of the destructed application.
- Don't show the "File modification" message box in case an entry was added to the object map using the context menu in the Spy.

2.8.2 Qt-specific

- Dedicated QGraphicsView support. QGraphicsItems are exposed as fully-fledged objects including properties. The objects are recognized during recording and their state can be inspected and verified.
- Support for testing (record and replay) of native Windows controls (MFC, ActiveX, etc.) in Qt applications.
- New simplified and non-intrusive setup for tests attaching to a running application.
- Support testing of Qt Jambi applications.
- Wrap QListView and QFontMetrics(F) classes in the Qt 4 wrapper.
- Avoid crash with Qt 3.0.5 and earlier versions.
- Fixed name generation for objects that have property values having am &-characters in them.
- Consider 'objectName' for the symbolic name.
- Support the QFileOpenEvent event type in sendEvent() calls.

2.8.3 Java-specific

- Support testing of applications which mix AWT/Swing and SWT/RCP controls.
- Greatly simplified setup procedure for hooking into Java applications. Applications launched through a .bat file do not have to be patched anymore for example.
- Pre-built binary packages for Solaris.
- Support for record and replay of drag'n'drop and mouse drag operations.
- Dedicated record and replay support for Eclipse GEF-based controls
- Fixed access to object members via Perl test scripts.
- A plug-in that adds a Squish Test Type to the Eclipse Testing and Performance Testing Tools Platform (TPTP). It allows for Squish tests to be driven by tools based on the TPTP framework.

2.8.4 Web-specific

- Support record and replay of actions on Java applets embedded in a Web page.
- On Windows, support record and replay of basic interactions with native browser dialogs (e.g. certificate and password dialogs), native browser control elements and embedded ActiveX objects. This includes Flash/Flex plug-ins.
- Several small improvements for better synchronization and object identification.

2.8.5 Mac-specific

- New edition available for automated testing of Carbon and Cocoa applications on Mac OS X

2.8.6 Source Builds

- Compile with a Qt installation that was stripped down with QT_NO_MESSAGEBOX, QT_NO_MENU, QT_NO_MENUBAR and QT_NO_TOOLBUTTON being defined.
- Avoid compilation error by not using a possibly-reserved "constant" identifier.

2.9 Version 3.2.3

2.9.1 General

- Fixed reading Object Map with empty lines.
- Make sure that backslashes in symbolic names do not get lost.
- Don't pass open sockets to child processes.
- Fix problems with accessing wrapped functions after an application started via startApplication() was terminated.
- Fix sporadic crash after test suite removal or replacement.
- Keep pointing out file modifications in source()'d files when using Run and Record
- Improved usability of row selection and deletion in the Object Map editor
- Fixed setting of the pid property of the context returned by defaultApplicationContext().
- Fixed Current Working Directory path names with spaces in them.

2.9.2 Qt-specific

- Suppress mouse animation in cases where the events generated by the windowing system could conflict with operations like dragging of canvas elements.
- Fix clicking on tabs which have a backslash character in their name.
- Fix 'Object not found' errors when running the monkey test example.
- Qt 4 on Mac only: make recording and playback of menu actions work (i.e. turn off the native menu bar support).
- Fixed memory leaks that occurred with Qt 3.x and use of buddy* and window* object name properties.

2.9.3 Web-specific

- For image links which have no good properties, use the image's src in the generated name.
- Fixed a race condition leading to waitForContextExists() waiting unnecessariliy for 20 seconds
- Added openContextMenu() function
- Emulate onchange events correctly for select-multi input elements.
- Consider attributes for object identification.
- Fixed race condition leading to a crash of webhook after a test run.
- nativeMouseClick() now accepts a second 'button' argument.
- Fixed hooking into frames in some tricky situations.
- Sanity check on frame hookup.

2.9.4 XView-specific

- Fixed a settings-dependant crash on Solaris.

2.9.5 Source Builds

- Fixed Perl auto-detection on HP-UX
- Compile fix for a picky version of Sun CC.

2.10 Version 3.2.2

2.10.1 General

- Added File.readln() JavaScript function.
- Guess separator used for testdata correctly even for very short data tables.
- Fixed Perl bindings for template, pointer and reference C++ types.
- Compile fixes for Lynx OS. Some workarounds for for recently discovered compiler bugs are still pending. Please inquire with support for a patch.
- Don't let the IDE crash when creating a new testdata file and saving it before a second column was added.
- Fixed Solaris walkcontext() check.

2.10.2 Qt-specific

- Added --with-moc switch.
- Added QMenu.title descriptor. This seems got lost somehow between Squish 3.1 and 3.2.
- Fixed Qt 4 library detection on AIX.
- Deal with quoted built-in, pre-processor macros of latest Sun CC compiler.

2.10.3 Web-specific

- Fixed clickButton()
- On replay when an object is not found in the current context, try finding it in any other open context before throwing an error.
- Fixed hooking into windows opened by subwindows. *Squish* only hooked into windows opened by frames of subwindows so far. This will fix many issues with popup windows.
- Record and replay right-mouse-button clicks correctly.
- If, when recording, it took longer that 20 seconds for an object or context to be available, record this timeout in waitForObject() and waitForContextExists().
- When a frame, which is the current context, is reloaded, reset the context to the new frame window.
- Fixed case where waitForContextExists() waited 15 seconds if called in an unfortunate moment.

2.11 Version 3.2

Squish 3.2 is a minor release that brings new features, improvements for all editions. New features and some of the improvements are listed below. See the `ChangeLog` file in the package for a complete list.

2.11.1 General

- Property name matching supports regular expressions and wildcards. This allows for dynamic property values to be identified.
- References to other objects are possible in multi-property names to avoid repeating redundant information.
- The list of properties used for object identification is configurable.
- Improved documentation with more examples, in-depth information and tutorials
- Improved usablity of the [Test Suite|Record] sequence in the IDE.
- Implemented source() function for Perl script bindings.

2.11.2 Qt

- Generated multi-property names now reference container, window, buddy, leftWidget, aboveWidget, parentWidget by name and not using a set of identifying properties. This makes the names carry less redundant information and better maintainable.
- Use Qt's translator mechanism to do property value matching in object names and matching of item texts in functions such as activateItem() so test scripts can run on internationalized applications with no modifications.
- New Qt 4.3 classes like QWizard are wrapped
- Support Qt 3.3.8 by working around a regression that broke the debugger on Windows.
- Support Q3ListBox and Q3ComboBox

2.11.3 Web

- Support record and replay of interactions with FileUpload form elements.
- The Spy can be used without a running test
- Added a waitForContextExists() function and record that instead of setContext().
- Much improved Spy object picker.
- Screenshots can now be taken of individual elements in the page and not only of the whole page (as it was the case in previous versions)
- Added new event types AlertOpened, PromptOpened and ConfirmOpened than can be handled in the test script.
- Added a JavaScript extension API that allows user-defined support for custom AJAX widgets. That includes custom object identification for record and replay and property definitions.
- Added support for Firefox on Mac OS X.
- Fixed various errors in the interaction with Internet Explorer 7

2.11.4 Java

- More properties and relative widgets (buddies, etc.) are used in object names when recording to generate more robust names.
- Fixed script replay actions ignoring the return value whether the action succeeded or not.

2.11.5 Tk

- Added support for testing Tk-inter applications (e.g. PyTk)

2.12 Version 3.1.2

Version 3.1.2 is a maintenance release that incorporates a couple of bug fixes. See the `ChangeLog` file in the package for a detailed list. User visible changes and improvements are listed below.

2.12.1 General

- Fixed a race condition leading to occasional IDE crashes when removing a test suite.
- Fix a crash in screenshot comparison with masks if the image is smaller than the mask.
- More robust breakpoints in Python scripts located via SQUISH_SCRIPT_DIR.
- Improved function overload resolution of Perl and JavaScript bindings. Prefer a normal type conversion over a "tolerant" one.
- Support NULL pointer passing from Perl scripts.
- Don't crash when using an invalid report output format.
- Fix compile problems with Python 2.5.
- Added a "Copy Real Name" option to the Spy.
- Improved algorithm that is used to detect whether a file is binary or text. French accents and Umlauts won't confuse it anymore.
- Mac OS only: if you add an application path or add an attachable AUT via the test suite settings dialog, then show the last modal dialog in front.

2.12.2 Qt

- Fix over-aggressive mouse move event compression so that recording multiple subsequent moves of a dialog window gets played back properly.
- Fix setting breakpoints and using the spy with Qt 3.3.8 on Windows.
- Fixed Q3ListBox interaction crash.
- Wrap Q3ComboBox which which appeared in Qt 4.1
- Support Qt builds that have QWheelEvent disabled.

2.12.3 Java

- Support for Java testing on the Mac.
- Keyboard mapping fixes for both SWT and AWT/Swing.
- Replaying 'typeText()' fix in AWT/Swing, it simply called 'setText()'.
- Fix replaying menu selection when menu bar is a derived JMenuBar class.
- Filter out 'typeText()' on widgets not having the focus AWT/Swing.
- Menu selections and combobox selections are now recorded on the mouse button release event for AWT/Swing.
- Jvm crash workaround when calling 'toString()' on some SWT widgets.

2.12.4 Web

- Fix for hooking into pages which automatically redirect.
- Fixed hang in unload handling.
- Made isPageLoaded more robust.
- Fixed occasions where picking objects hung.
- Added dedicated support for Kiwi_TreeTable.
- Fixed error handling.
- Fixed recording selectOption() when the option text contains special characters.

2.12.5 Tk

- Generate better names for randomly-named TkInter widgets.
- Sanity check that allows to record on statically linked Tk apps.
- Fixed hooking for TkInter Python applications.

2.13 Version 3.1.1

Version 3.1.1 is a maintenance release that incorporates a couple of bug fixes. See the `ChangeLog` file in the package for a detailed list. User visible changes and improvements are listed below.

2.13.1 General

- A test suite has a default naming scheme property now. If not specified in the `suite.conf` file, it is set to old, hierarchical names. This way existing pre-3.1 test suites keep using hierarchical as default. This is also true for using the Spy when picking objects and inserting Verification Points. Modify the *`NAMINGSCHEME`* entry to *`MULTIPROP`* or *`HIERACHICAL`* in `suite.conf` to change the default naming scheme.

2.13.2 Qt

- The following classes that appeared in the Qt 4.2 release were added to the Qt Wrapper and are therefore available in test scripts: QGraphicsView and related classes, QCalendarWidget, QFontComboBox, QDialogButtonBox, QWidgetAction, QSystemTrayIcon, QDesktopServices and Q3(H|V)GroupBox.
- The multi-property object identification scheme added in version 3.1.0 has undergone some improvements to provide more robust names.
- When building a Squish for Qt package from sources a Qt setup with the source directory being different than the Qt installation directory is automatically detected. This enables builds with custom Qt and Qtopia installations without manually setting the *`QTSRCDIR`* build variable.

2.13.3 Java

- Important fix for *Squish* Spy for SWT, it had a serious threading problem in 3.1.0.
- Screenshots for a `JFrame`, or `Shell` in SWT, is now supported as well.
- Added `closeWindow(object)` to the Java™ convenient API for scripts, which closes a top level window or dialog. Also closing such a window with recording by the system window menu or close button, will now record it as a `closeWindow(-object)`.
- The object picker from spy now prevents that the click event, triggered by the actual picking, does not reach the AUT event processing.

2.13.4 Web

- Input into file upload elements (INPUT elements of type FILE) can be recorded and replayed now.
- During setup any running Mozilla and Firefox instances are automatically shut down to ensure a proper plug-in installation.
- DIV and SPAN elements that have a reasonable innerText property are now identified by this property rather than by fragile, hierarchical names.

2.14 Version 3.1.0

2.14.1 General

- Improved control bar: when you record a test script, the control bar shows the events it records and when you replay a test, it already shows the test log.
- Insertion of Synchronization Points via Point & Click: you can insert synchronization points (`waitForObject` statements) within the Spy. It works the same way as inserting a verification point.
- Show AUT's call stack on crash in test log: if the AUT crashes during test execution, a backtrace of the crash is included in the test log. This is currently not supported on Mac OS X.
- Live Script console in script debugger: if you execute a test script and hit a breakpoint, the Squish IDE opens a debugger pane next to the variable watcher. There you can execute script statements to debug your test scripts.
- Completely overhauled variable watcher window: the variable watcher displays more variables and it also displays the type of the variable.
- Improved automatic object naming: additionally to the hierarchical object identification, you can now choose to use multiple property object identification instead. This results in more robust object names in many applications. You can choose which object naming scheme to use when recording a new test script. It is no problem to use both, hierarchical names and multiple property names side-by-side.
- Excel import for test data: you can copy an Excel file to the test data directory and use it in the script like you would use any other test data file to read it and access the elements (see Section 7.11.2). It is not possible to edit the Excel test data files inside Squish, though.
- Excel export for test results: when calling squishrunner with the option `--reportgen` you can specify `xls` to generate the testresults as an Excel file.
- Perl support for test scripts: Perl is now a supported language to write test scripts.
- Test-Script template support in IDE

Chapter 3

Installation

This chapter describes the steps necessary to install *Squish* on Unix, Windows, Mac OS X and Embedded Linux. *Squish* can be used in many setups using binary packages, but might also need to be compiled from source.

In case it is not possible to use a binary package for your local setup, the section Section 3.2 explains how a default build from source works. A list of common options to the configure script is available in the Section 3.2.2 section.

Installing *Squish* for a non-default setup is a bit more complex. Please refer to the following sections for cases that are not possible to do with a default setup:

- Section 3.2.4
- Section 3.2.5
- Section 3.2.7
- Section 3.2.8

Finally, if you want to use a *Squish* installation on more than just one machine, please take a look at Section 3.3.

3.1 Installing from Binary Packages

Squish can be installed from binary packages, which simply need to be unpacked and then configured before they are ready for use. Binary packages are available for Linux®, OS X® and Microsoft™ Windows®. Since *Squish* integrates very tightly with your computer's system and with the application/website you want to test, different versions of each package are available. Hence, the first step when installing binary packages is finding out which package is the correct one.

3.1.1 Getting the Correct Package

The binary *Squish* packages are available in your Customer Area on the froglogic website. To find out which of these packages is the one you need, you first need to know whether you are going to test either a web application or an application written using the Qt® toolkit. Additionally, you need to know which operating system you are using, i.e. Linux, Mac OS X, or Microsoft Windows.

If you will be testing a Qt application, you need to collect some further technical information in order to be able to choose the correct package:

1. Which version of Qt is used by the application you want to test?
2. Which compiler (and which version of the compiler) was used to compile the Qt library (and the application which you want to test)?

If you don't know this yourself, it's likely that a developer will be able to give you this information. After you have collected this information, the following table can tell you the name of the package to be used for each edition/operating system/compiler/Qt version combination.

Note
Please note that should you be unable to find an entry in the table which matches your local setup, it is best that you build *Squish* from source. This ensures that it integrates correctly with your local software configuration. Building *Squish* from source is described in Section 3.2.

Note
Squish binary packages do not require installion of any files into some additional directory of your computer, so the directory you choose for decompressing the package can just as well be the one from which you will run *Squish*.

After you have decided which of these packages is appropriate for you, download it onto your computer and decompress it somewhere. Decompressing these files requires a ZIP decompressor for Windows. The Linux and Mac OS X packages use GNU gzip and GNU tar. In case you don't have a graphical interface for your respective compressor, change to the directory into which you downloaded the package and then execute the following command:

On Windows `C:\squish> unzip -o packagename`

On Linux `% tar zxf packagename`

On Mac OS X `% tar zxf packagename`

packagename is a placeholder for the actual name of your binary package, as given in the table above.

3.1.2 Configuring the Package

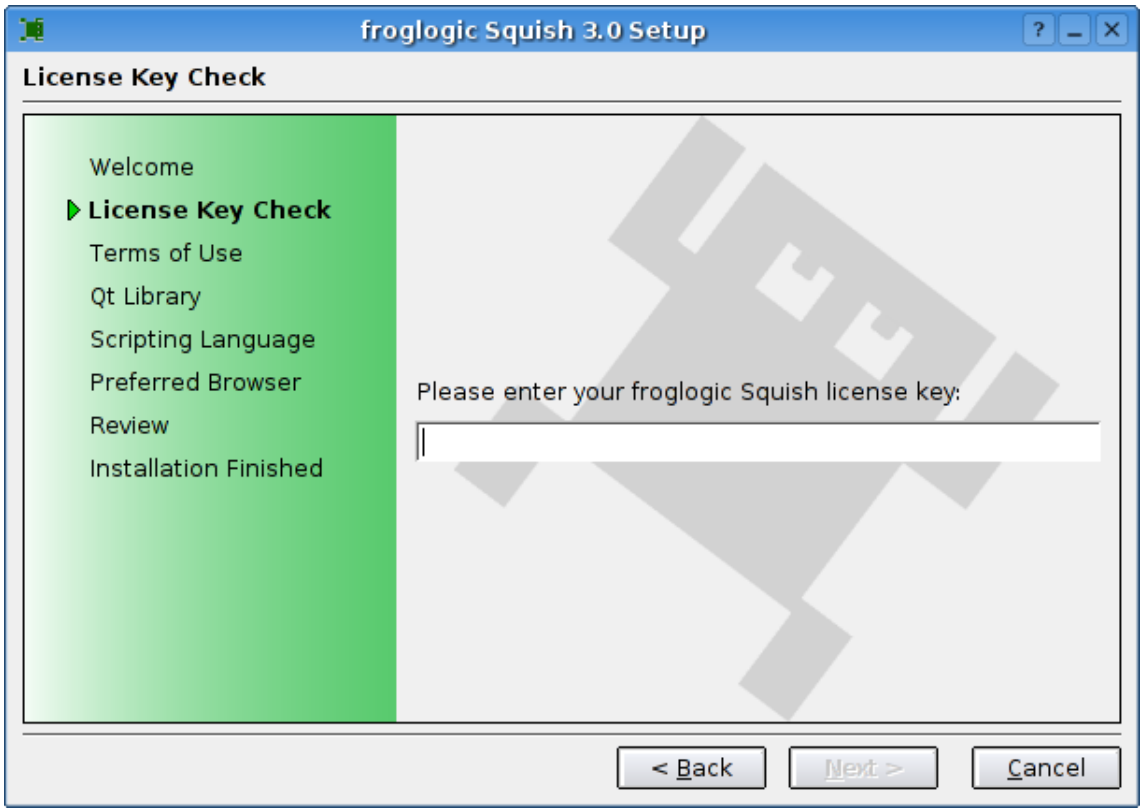
Before you can run *Squish* from a binary package, you need to run a special configuration tool contained within the package so that *Squish* knows where certain files reside on your hard disk. This configuration tool is called setup and is stored in the top-level directory of your package. Hence, after decompressing the package, you should be able to find an executable file called setup - start the configuration process by executing that program.

Note
In case you're using a binary package of *Squish* built for the Microsoft Visual Studio 2005 compiler (MSVC8), your system might lack certain runtime libraries. You can tell whether you lack them by starting the setup program: if it works, everything is fine. However, if you get an error message along the lines of 'This application has failed to start because the application configuration is incorrect' then you have to install the required runtime libraries first. To do so, simply run the vcredist_x86 program which is in the same directory as the setup program prior to starting the setup.

Note
If you want to go back and change a setting you made during the configuration, you can use the button labeled Back to navigate back to the respective page, adjust your configuration, then move on again using the Next button.

Operating System	Edition	Qt Version	Compiler	Package Name
Windows (32bit)	Qt Testing	3.3.5	MinGW	squish-3.1.0-qt335-win32-mingw.zip
Windows (32bit)	Qt Testing	3.3.6	MinGW	squish-3.1.0-qt336-win32-mingw.zip
Windows (32bit)	Qt Testing	4.1.x	MinGW	squish-3.1.0-qt41x-win32-mingw.zip
Windows (32bit)	Qt Testing	4.1.x	MinGW	squish-3.1.0-qt42x-win32-mingw.zip
Windows (32bit)	Qt Testing	3.3.5	MSVC 6	squish-3.1.0-qt335-win32-msvc6.zip
Windows (32bit)	Qt Testing	3.3.6	MSVC 6	squish-3.1.0-qt336-win32-msvc6.zip
Windows (32bit)	Qt Testing	4.1.x	MSVC 6	squish-3.1.0-qt41x-win32-msvc6.zip
Windows (32bit)	Qt Testing	4.1.x	MSVC 6	squish-3.1.0-qt42x-win32-msvc6.zip
Windows (32bit)	Qt Testing	3.3.5	MSVC 7.1 (.NET 2003)	squish-3.1.0-qt335-win32-msvc7.zip
Windows (32bit)	Qt Testing	3.3.6	MSVC 7.1 (.NET 2003)	squish-3.1.0-qt336-win32-msvc7.zip
Windows (32bit)	Qt Testing	4.1.x	MSVC 7.1 (.NET 2003)	squish-3.1.0-qt41x-win32-msvc7.zip
Windows (32bit)	Qt Testing	4.1.x	MSVC 7.1 (.NET 2003)	squish-3.1.0-qt42x-win32-msvc7.zip
Windows (32bit)	Qt Testing	3.3.5	MSVC 8.0 (2005)	squish-3.1.0-qt335-win32-msvc8.zip
Windows (32bit)	Qt Testing	3.3.6	MSVC 8.0 (2005)	squish-3.1.0-qt336-win32-msvc8.zip
Windows (32bit)	Qt Testing	4.1.x	MSVC 8.0 (2005)	squish-3.1.0-qt41x-win32-msvc8.zip
Windows (32bit)	Qt Testing	4.1.x	MSVC 8.0 (2005)	squish-3.1.0-qt42x-win32-msvc8.zip
Windows (64bit)	Qt Testing	3.3.5	MSVC 8.0 (2005)	squish-3.1.0-qt335-win64-msvc8.zip
Windows (64bit)	Qt Testing	3.3.6	MSVC 8.0 (2005)	squish-3.1.0-qt336-win64-msvc8.zip
Windows (64bit)	Qt Testing	4.1.x	MSVC 8.0 (2005)	squish-3.1.0-qt41x-win64-msvc8.zip
Windows (64bit)	Qt Testing	4.1.x	MSVC 8.0 (2005)	squish-3.1.0-qt42x-win64-msvc8.zip
Linux (32bit)	Qt Testing	3.3.5	any	squish-3.1.0-qt335-linux32.tar.gz

3.1.2.1 Entering the License Key

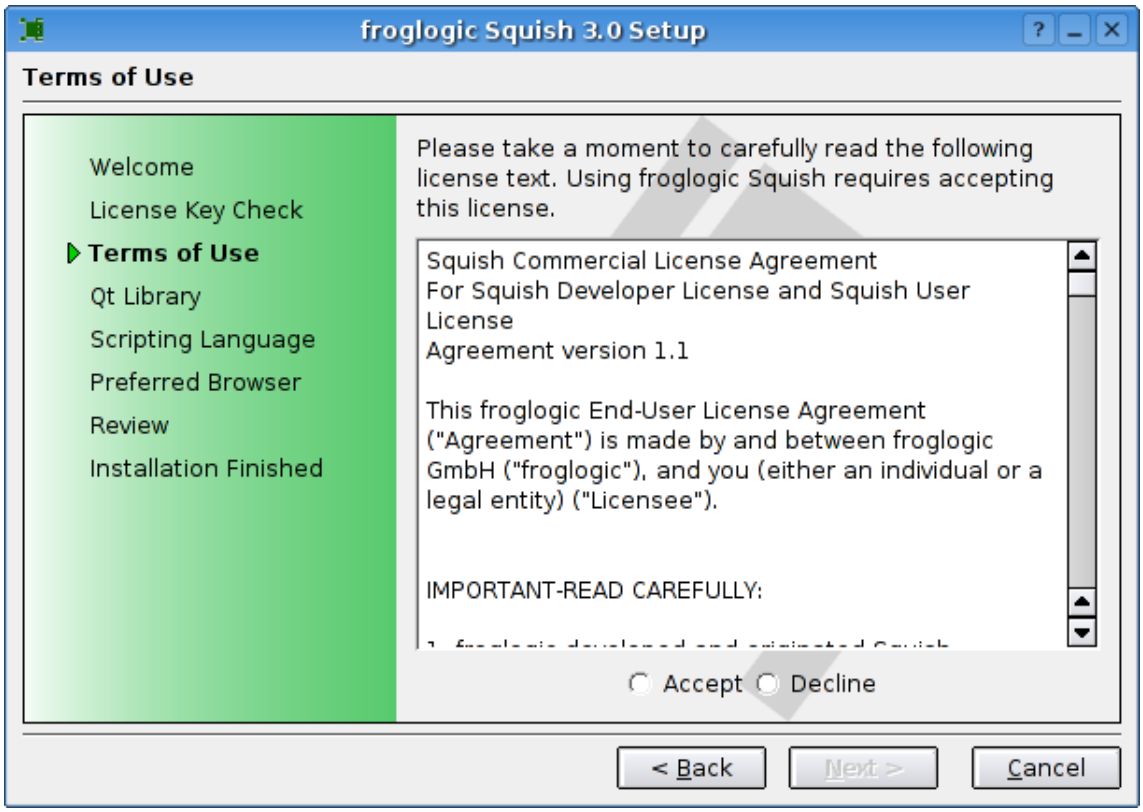


After acknowledging the welcome page, by pressing the button labeled Next on the bottom of the setup program, the next page will request you to enter your *Squish* license key as given to you by *froglogic*. If you already have a previous *Squish* installation on your computer, the license key used previously will be shown in the input field. Enter your license here exactly as given to you, including any dashes.

Note
Since the release of *Squish* 3.0 also introduced a new license key scheme, you should have received a new license key in your customer area on the froglogic website (called `squish-3-license`). In case your previous installation didn't already use this scheme, it is recommended that you use the new license key instead of any previous one, as used by *Squish* 1.x and 2.x.

Click the Next button to advance to the next step of the configuration process. If there are any problems with the license (for instance, if it has expired or is malformed) then a message box will be shown, notifying you of the issue.

3.1.2.2 Acknowledging the Terms of Use

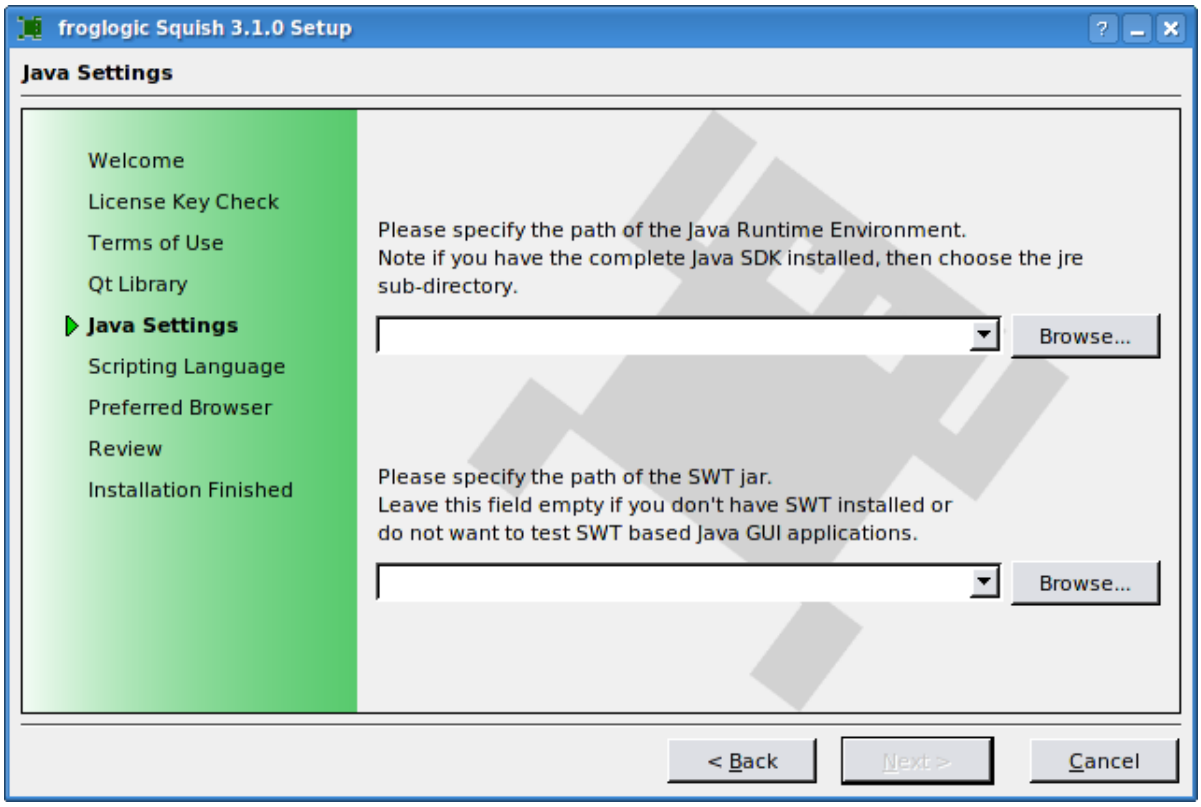


After entering your license key, you will be presented with the license under which you are permitted to use your copy of *Squish*. Please read the entire license text carefully before proceeding. Two buttons labeled Accept and Decline, below the license text, let you state whether you agree or disagree with these terms.

Note
The license text can change depending on which license key you entered.

Accepting the license text will allow you to click the Next button, so you can proceed to the next step of the configuration process.

3.1.2.3 Paths for Java™ Settings



In order to be able to test Java™ applications, you need to point *Squish* to the `jre` sub directory of your Java™ SDK installation, or if you only have the JRE installed, the root of that installation.

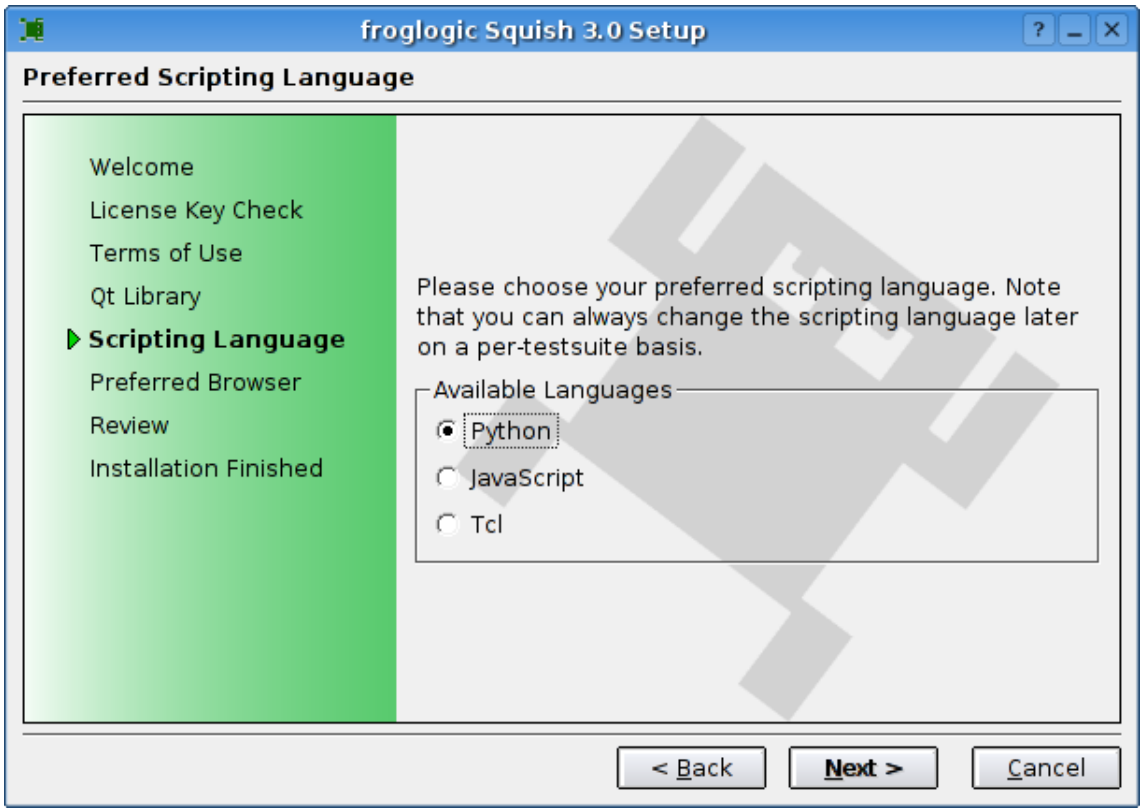
If you also want to test Java™ applications based on the SWT GUI toolkit, then the path to the SWT jar file as well.

Note
Beware that the SWT jar file needs the accompanying shared libraries (or dll's) to be found in your system path.

Note
If you don't know where the library is stored on your computer, your system administrator should be able to tell you.

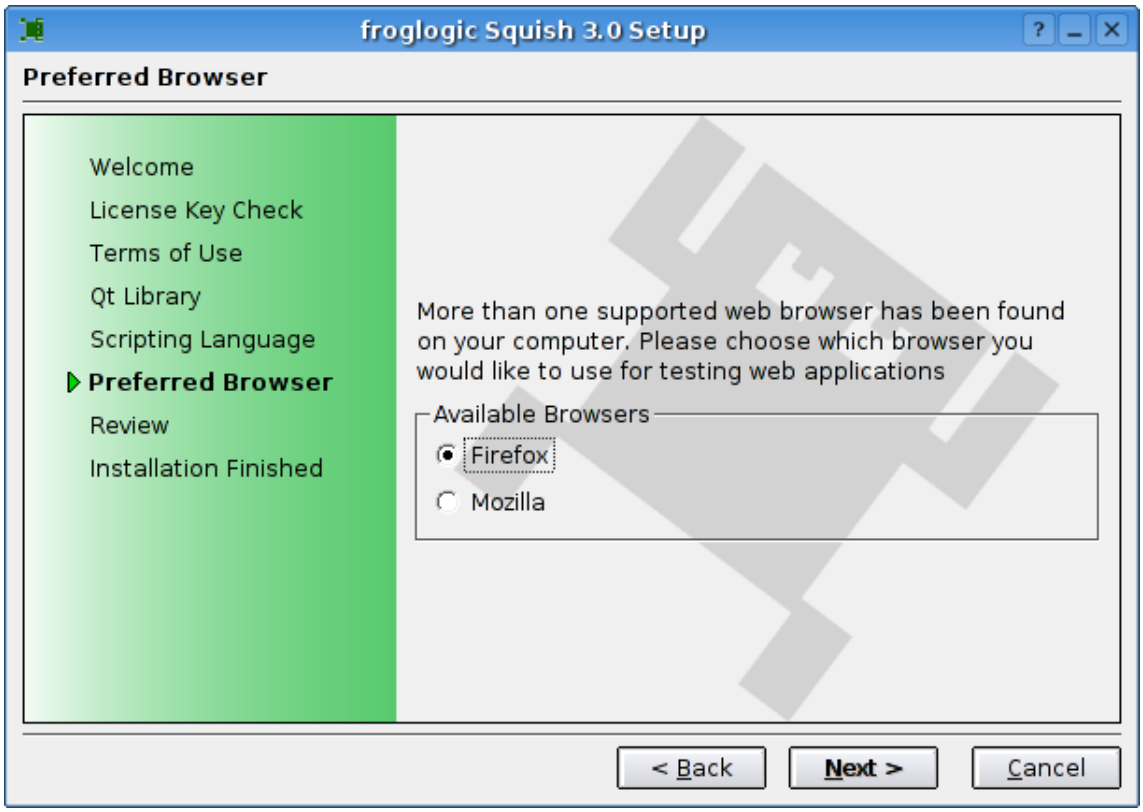
After specifying the path to the requested `jre` directory and optionally filled in a valid path to a SWT jar file, use the Next button to advance to the next step of the setup.

3.1.2.4 Preferred Scripting Language



Test scripts created with *Squish* can be formulated in a number of scripting languages, such as Python, Tcl or JavaScript. This page lets you define which language should be used by default when creating new test suites. Select your preferred scripting language here (or choose the default), then proceed to the next step using the Next button.

3.1.2.5 Preferred Web Browser



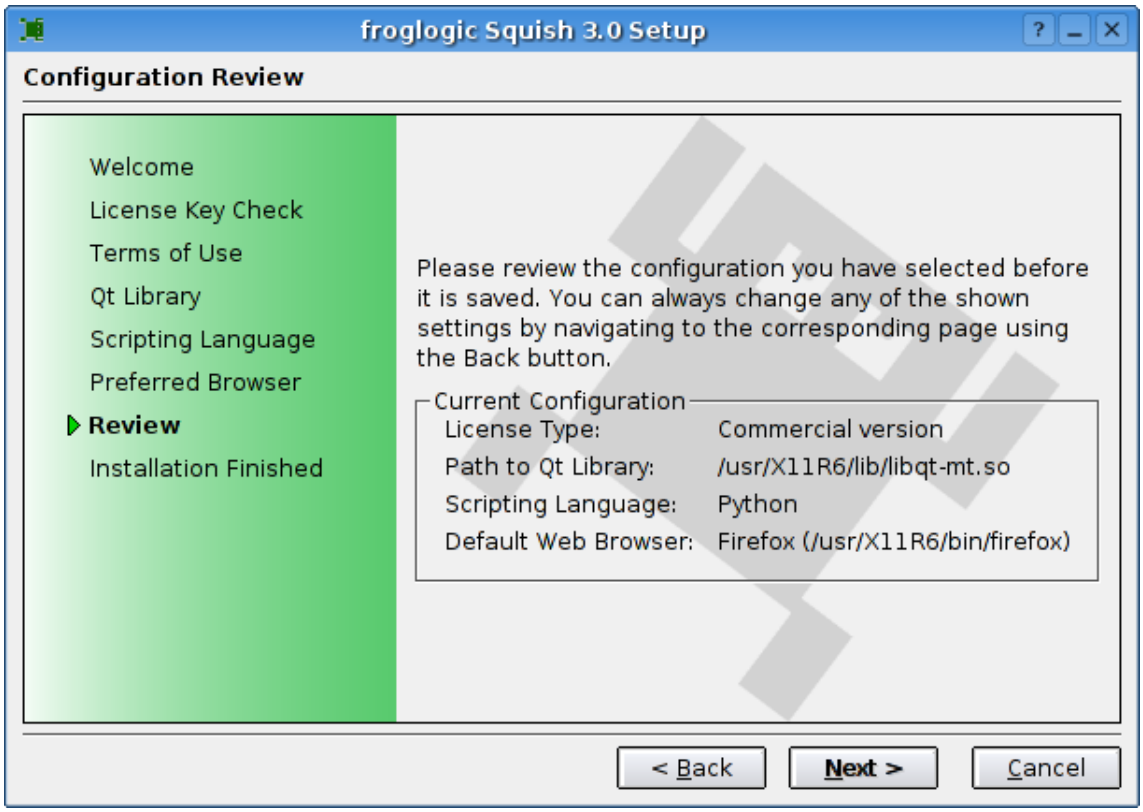
Tip This step is only necessary if you are configuring a *Squish* package set up for testing Web applications on Linux, and if your license key entitles you to test Web applications. If this is not the case, you won't be presented with this page and can safely skip to the next section.

When using *Squish* to test web applications, it is necessary to decide which browser should be used for recording and replaying the tests. On Windows and Mac OS X, the default browser is used (Internet Explorer® and Safari® respectively). However, Linux systems have no default browser, and if more than one supported browser is detected you need to choose which browser should be used.

Note
If you are using a Linux package and there was just one supported browser found, it will be chosen silently and this page will be skipped.

Just select your preferred web browser, or choose the default selection, then move on with the setup by clicking the Next button.

3.1.2.6 Configuration Review

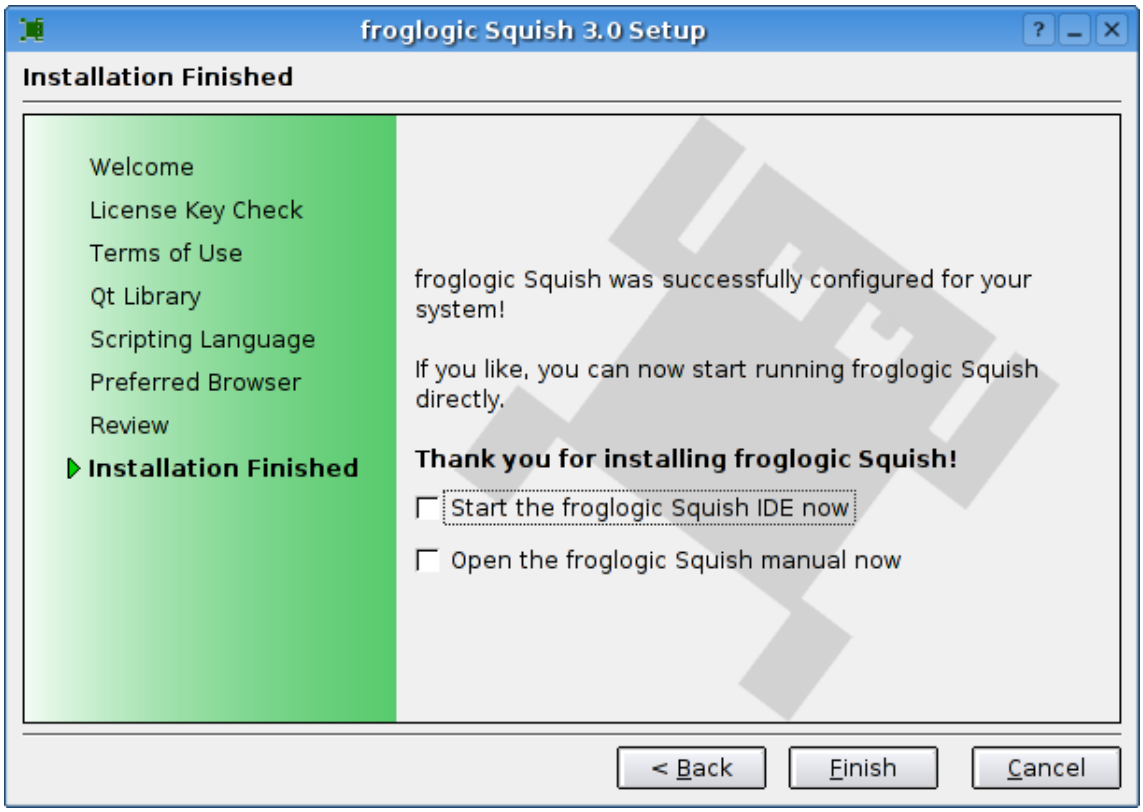


All required information has now been gathered by the setup program and is presented here for you to review, an example of which is shown in the picture above. As usual, if you notice any mistakes, you can use the Back button of the setup program to go back to the respective page and adjust your choices, then return to the review page by using the Next buttons.

Warning
Pressing the Next button on this page means that the configuration shown will be written out - a process which cannot be reversed (though you could re-run the setup program and overwrite the settings with new ones). Please ensure that the settings shown look sensible.

Proceeding to the next page, using the Next button, will save the configuration shown, and advance to the final page.

3.1.2.7 Concluding the Configuration



Congratulations! You have finished the installation, and all settings have been saved successfully. This page concludes the configuration of your *Squish* binary package.

You can now use the button labeled Finish to finish the installation and close the setup program. If you like, you can check any of the check boxes shown, should you subsequently want to open the Squish IDE or the *Squish* manual.

3.1.3 Using the Binary Package

After acquiring, decompressing and configuring your *Squish* binary package, you are ready to go. You can start the Squish IDE by navigating to the `bin` folder in the place where you unpacked your binary package, and then starting the program called `squish`.

If this is your first time using *Squish*, it might be a good idea to proceed to the tutorial so you can familiarize yourself with the software.

For testing a sample Java™ application, see Chapter 5.

3.2 Installing from Source Packages

Squish can - and might need to - be built from source, which means that you have to compile it. This ensures that *Squish* supports the Qt configuration your applications use. Once *Squish* is compiled, it can be distributed to different computers running the same operating system (see Section 3.3).

Compilation usually just requires a few simple steps. The sequence of which, for the most common cases, is described in Section 3.2.1. Deviations from the common scheme for non-standard configurations are covered in subsequent sections.

3.2.1 Quick Install

The Quick Install guide describes how to perform a standard installation of *Squish*. This covers most common cases and is the easiest way to install *Squish*.

Requirements

For a standard installation, certain requirements have to be met:

- Qt 3.1.0 or greater needs to be available.
- Qt needs to be configured and built with support for threads.
- Qt needs to be configured and built as a shared library.
- If you want to use Python as a scripting language for test scripts, a version of Python equal to or greater than 2.2.0 needs to be installed.
- If you want to use Tcl as a scripting language for test scripts, a version of Tcl equal to or greater than 8.0 needs to be installed.
- If Perl is the scripting language of your choice a Perl 5.8 installation is required on your system. The development release Perl 5.9 is known to work but not regularly being tested.

Windows	Linux/Unix	Mac OS X
We recommend installing the Python package available at http://www.python.org/ftp/python/-2.3.3/Python-2.3.3.exe or a more recent version from that site.	The Python interpreter executable has to be in the `PATH`. Most Unixes already have Python pre-installed. If this is not the case, however, pre-built packages for most Unix systems are available for download.	The pre-installed Python version fulfills this requirement.
We recommend installing ActiveState's Tcl build. You can download it from http://www.activestate.com/Products/-Download/-Download.plex?id=ActiveTcl. It comes with a Windows setup program. If you run this and accept all the standard settings it will be installed in `C:\Tcl`.	Most Unixes already have Tcl installed.	The pre-installed Tcl version fulfills this requirement.
We recommend installing ActiveState's Perl package. You can download it from http://www.activestate.com/Products/-Download/-Download.plex?id=ActivePerl. It comes with a Windows setup program. If you run this and accept all the standard settings it will be installed in `C:\Perl`.	Most Unixes already have Perl installed. The perl executable is searched in `PATH` unless overriden with the `--with-perl` configure switch.	The pre-installed Perl version fulfills this requirement.

If you want to use *Squish* to test Qt 3.0.x applications, see Section 3.2.4 and if you want to test Qt/Embedded 2.3 applications, see Section 3.2.5. QtopiaCore and Qtopia 4.x specific instructions can be found in Section 3.2.6.

Some parts of *Squish* require that they use the same Qt library as the application you want to test. If the Qt library your application uses does not fulfill the requirements above (i.e. is not a shared, multi-threaded Qt library), you have to do a split build of *Squish* (see Section 3.2.7 and Section 3.2.8).

There are no additional requirements if you want to use JavaScript as a scripting language for test scripts: the source code of a small JavaScript interpreter library is shipped with the *Squish* package. The library is built automatically unless JavaScript support is turned off using the `--disable-js` switch of the configure program.

Supported Platforms

Squish has been tested on the following platforms and compilers:

Windows	Linux/Unix	Mac OS X
Platforms: • Windows 2000 • Windows XP • Windows Vista Compilers: • Microsoft VC++ 6.0 • Microsoft VC++ 7.x • Microsoft VC++ 8.x • Microsoft VC++ 9.x • Intel C++ 7.x	• xlC 9.0.0.2 on AIX 6.1 • aCC A.05.52 on HP-UX 11i 11.22 • aCC A.03.45 on HP-UX 11.11 32bit and 64bit (with PHSS_30049 ld patch) • gcc 3.3.x on HP-UX 11.11 • GNU gcc 2.95.x on IRIX 6.5 (See Q: 10.1.2) • MIPSPro C++ 7.4m on IRIX 6.5 32bit and 64bit • GNU gcc (3.x and 4.x) on Linux • Intel C++ 7.x on Linux • GNU gcc (3.x and 4.x) on SUN Solaris 8 x86 • SunCC 5.5 on SUN Solaris 8 x86 • GNU gcc 3.4.4 on FreeBSD 6.x	• gcc 3.3 and 4.0 on Mac OS X 10.4 (Tiger)

Building *Squish*

We assume that you have unpacked the package into

Windows	Linux/Unix	Mac OS X
`C:\squish`	`/usr/local/squish`	`/usr/local/squish`

You can use any path, but this document uses this path for the sake of example.

The first step is to ensure that you have a license key. Download the file called `squish-3-license` from the download area where you obtained your *Squish* package. Copy this file to

`%HOMEPATH%\.squish-3-license`	`$HOME\.squish-3-license`	`$HOME.squish-3-license`

(notice the change of name to include a leading dot). Alternatively you can enter the license key contained in the file when configure asks for it.

Note
If you build *Squish* for another user, you must make sure that each user copies the license file into their *own* account area.

Now *Squish* is ready to be configured according to your system's settings.

Start by changing to *Squish*'s installation directory

Open a console (Command Prompt) and type: `C:\> cd \squish`	`$ cd /usr/local/squish`	`$ cd /usr/local/squish`

Now run the configuration script:

`C:\squish> configure`	`$ ./configure`	`$ ./configure`

The configure script searches for your C++ compiler, Qt, Tcl, Python, and other system components relevant to *Squish*. The output indicates which compiler has been detected. You should use the same compiler for *Squish*, Qt and your applications. If you want to force configure to use a different compiler from the one it automatically detects, set the `CXX` environment variable.

For example, if you want to use Intel C++ instead of MSVC++, run configure like this:
`C:\squish> set CXX=icl`
`C:\squish> configure`
	For example, if you want to use gcc instead of SunCC on Solaris, run configure like this:
	`$ CXX=g++ ./configure`
		For example, if you want to use gcc 3.3 instead of the default gcc version, run configure like this:
		`$ CXX=g++-3.3 ./configure`

Now configure asks you to run build to compile *Squish*.

`C:\squish> build`	`$ ./build`	`$ ./build`

This runs for quite a while, since it also creates wrappers for the Qt library. During certain steps of the build process you will probably see a lot of warnings from the squishidl tool. It is safe to ignore these warnings.

Once it has finished, *Squish* is installed. The binaries and libraries are now in:

`C:\squish\bin` `C:\squish\lib`	`/usr/local/squish/bin` `/usr/local/squish/lib`	`/usr/local/squish/bin` `/usr/local/squish/lib`

You might want to add the `bin` directory to your `PATH` environment variable so you can invoke the *Squish* tools without specifying the path:

Open the advanced system settings in the Windows control panel and add `C:\Squish\bin` to `PATH`.	Edit your `.bashrc` (or the appropriate file for the shell you are using) and extend the `PATH` by adding `/usr/local/squish/bin`. Alternatively, you may prefer to create symbolic links, in `/usr/local/bin`, to the executables in `/usr/local/squish/bin`.	Edit your `.bashrc` (or the appropriate file for the shell you are using) and extend the `PATH` by adding `/usr/local/squish/bin`. Alternatively, you may prefer to create symbolic links, in `/usr/local/bin`, to the executables in `/usr/local/squish/bin`. The Squish IDE itself is an application bundle (`/usr/local/lib/_squish.app`), but don't call it directly; rather start the Squish IDE from the `/usr/local/bin/squish` command line program. This ensures that the environment is set up correctly for *Squish*.

3.2.2 Configure Switches

When invoked, configure tries to automatically detect or guess the configuration of your system and the desired configuration of *Squish*. In some cases, this approach is not sufficient. That's why it is possible to override default settings by a set of switches.

A few examples are:

- `--enable-examples` – (see Partial Build for a complete explanation)
- `--with-tclconfig=/usr/lib/tcl8.3/tclConfig.sh` – overrides the automatic Tcl detection
- `--with-pydir=P:\Python` – points configure to the directory where `python.exe` or `bin/python` are installed
- `--with-qtdir=C:\Qt\3.3` – points to the Qt installation directory, overriding the `QTDIR` environment variable
- `--enable-64bit` – forces a 64-bit build on platforms where this is supported

For a complete list, please invoke `configure--help` from the command line.

Partial Build

The default behavior of the configure script is to select all components for a build. However, in some cases it may be desirable to only build a subset of components. Typical reasons are:

- a distributed installation with a minimal runtime installation on the target machine and execution tools, like the IDE, on the tester's machine
- a build with differently configured Qt libraries, such as described in Section 3.2.7 and Section 3.2.8

To support partial builds, configure supports a set of switches that will exclude individual components from the build. Currently the following components can be selected this way:

- `server` – the Squish Server application
- `runner` – the Squish Runner (client) application
- `ide` – the Squish IDE (graphical frontend)
- `idl` – the IDL compiler (needed for generating wrapper libraries)
- `wrappers` – Squish's framework wrapper libraries
- `examples` – the examples
- `tk` – basic Tk testing
- `rest` – everything not included in another component

Any of the above components can be used in `--enable-x` or `--disable-x` switches where `x` stands for the name of the component. If you are not interested in building the examples, you could configure and build *Squish* like this:

```
configure --disable-examples
build
```

You can combine these switches freely within the constraints of internal dependencies. This is where the `all` component comes in handy:

```
configure --disable-all --enable-server --enable-wrappers
```

This disables every component but the server and wrapper libraries.

Override Build Variables

configure will output the detected configuration into the files `config.h` and `Build.conf`. The former will be included by the source code files, the latter serves as an input file to the build tool. In case the automatic detection produced unsuitable results, you may choose to adapt these files before invoking build. The variables written to `Build.conf` can easily be overridden, without manually editing the file, by specifying arguments of the form *`VARIABLE=value`* or *`VARIABLE+=value`* at the configure command line. These arguments will replace and extend the named variables respectively.

An example that will override the default mode of `gcc` and enforce a 32-bit build on a 64-bit system:

```
configure "CXXFLAGS+=-m32" "LFLAGS+=-m32"
```

Here, the default optimization flag will be overriden:

```
configure "CXXOPT=-O1"
```

3.2.3 Installation for Testing Pure Qt 4 Applications

Testing a Qt 4.x application which does not use the Qt 3.x support functionality (a so called "pure" Qt 4.x) requires a split build of *Squish*. This means you must compile the components that come in direct contact with the AUT using your pure Qt 4.x library. The rest of Squish can be compiled with either Qt 3 or a Qt 4 installation that includes the Qt3Support module. For this, you have to run the commands configure and build twice: the first time to build the parts that will use the pure Qt 4 library and the second to build the parts that require Qt 3 or - alternatively - Qt3Support.

Here's a minimal example that assumes *Squish* to have been unpacked into two different directories:

Windows	Linux/Unix	Mac OS X
C:\Squish-Qt-4.0 C:\Squish-Qt-3.2	/home/user/squish-qt-4.0 /home/user/squish-qt-3.2	/home/user/squish-qt-4.0 /home/user/squish-qt-3.2

The pure Qt 4.0 and Qt 3.2 are installed in:

C:\Qt\4.0.1 C:\Qt\3.2.3	/usr/local/qt40 /usr/local/qt32	/usr/local/qt40 /usr/local/qt32

First, build only the Qt wrapper and the examples using Qt 4.0:

C:\> cd C:\Squish-Qt-4.0 C:\Squish-Qt-4.0> configure --enable-pure-qt4 --with-qtdir=C:\Qt\4.0.1 --disable-all --enable-idl --enable-wrappers --enable-examples	$ cd /home/user/squish-qt-4.0 $ ./configure --enable-pure-qt4 --with- -qtdir=/usr/local/qt40 --disable-all --enable-idl --enable-wrappers --enable-examples	$ cd /home/user/squish-qt-4.0 $ ./configure --enable-pure-qt4 --with- -qtdir=/usr/local/qt40 --disable-all --enable-idl --enable-wrappers --enable-examples

Verify that the configuration log output reads `Checking Qt version ..... 4.x` where x would be the respective Qt 4 release you are using. Now build the wrappers and the examples using Qt 4.0:

C:\Squish-Qt-4.0> build	$ ./build	$ ./build

Now change to the directory of the second Squish installation, point configure to the Qt 3.2 installation and select the remaining components:

C:\Squish-Qt-4.0> cd C:\Squish-Qt-3.2 C:\Squish-Qt-3.2> configure --with-qtdir=c:\Qt\3.2.3 --disable-idl --disable-wrappers --disable-examples --disable-explorer	$ cd /home/user/squish-qt-3.2 $ ./configure --with-qtdir=/usr/local/qt32 --disable-idl --disable-wrappers --disable-examples --disable-explorer	$ cd /home/user/squish-qt-3.2 $ ./configure --with-qtdir=/usr/local/qt32 --disable-idl --disable-wrappers --disable-examples --disable-explorer

Invoking

C:\Squish-Qt-3.2> build	$ ./build	$ ./build

a second time compiles the second group of binaries.

What's missing is the registration of the Qt wrapper. Normally this is done automatically, but when the split build was performed the tool for registration (squishrunner) was missing at the point when the wrapper was built. All you need to do is "rebuild" the target like this:

C:\Squish-Qt-3.2> build qtwrapperinit	$ ./build qtwrapperinit	$ ./build qtwrapperinit

If you like, you could even perform the above steps on different machines, if you prefer to execute the tests remotely.

Finally, copy over the required files which were built in the Qt4 directory over to your Qt 3.2 build directory:

C:\Squish-Qt-3.2> copy C:\Squish-Qt-4.0\bin\squishhook.dll bin\
C:\Squish-Qt-3.2> copy C:\Squish-Qt-4.0\bin\squishqtwrapper.dll bin\
C:\Squish-Qt-3.2> copy C:\Squish-Qt-4.0\bin\winhook.dll bin\
C:\Squish-Qt-3.2> copy C:\Squish-Qt-4.0\bin\dllpreload.exe bin\
C:\Squish-Qt-3.2> copy C:\Squish-Qt-4.0\lib\squishqtpre.dll lib\
C:\Squish-Qt-3.2> copy C:\Squish-Qt-4.0\src\wrappers\qt*.tcl lib\
	$ cp /home/user/squish-qt-4.0-/lib/libsquishhook.so /home/user/squish-qt-3.2-/lib
	$ cp /home/user/squish-qt-4.0/lib/libsquishqtwrapper.so /home/user/squish-qt-3.2-/lib
	$ cp /home/user/squish-qt-4.0-/lib/libsquishqtpre.so /home/user/squish-qt-3.2-/lib
	$ cp /home/user/squish-qt-4.0-/src/wrappers/qt/*.tcl /home/user/squish-qt-3.2-/lib
	$ cp /home/user/squish-qt-4.0/bin/isstaticapp /home/user/squish-qt-3.2-/bin
		$ cp /home/user/squish-qt-4.0-/lib/libsquishhook.so /home/user/squish-qt-3.2-/lib
		$ cp /home/user/squish-qt-4.0/lib/libsquishqtwrapper.so /home/user/squish-qt-3.2-/lib
		$ cp /home/user/squish-qt-4.0-/lib/libsquishqtpre.so /home/user/squish-qt-3.2-/lib
		$ cp /home/user/squish-qt-4.0-/src/wrappers/qt/*.tcl /home/user/squish-qt-3.2-/lib
		$ cp /home/user/squish-qt-4.0/bin/isstaticapp /home/user/squish-qt-3.2-/bin

Now, the build in your Qt 3.2 directory is fully usable for creating and running tests for applications which link against a pure Qt 4.x library.

3.2.4 Installation for Testing Qt 3.0 Applications

Testing a Qt 3.0.x application requires a split build of *Squish*. This means you must compile the components that come in direct contact with the AUT using a Qt 3.0 library. The rest of Squish must be compiled with a more recent version of the Qt library (Qt 3.1 or higher). For this, you have to run the commands configure and build twice: the first time to build the parts that require Qt 3.0 and the second to build the parts that require Qt 3.1 or higher.

Here's a minimal example that assumes *Squish* to have been unpacked into two different directories:

Windows	Linux/Unix	Mac OS X
C:\Squish-Qt-3.0 C:\Squish-Qt-3.2	/home/user/squish-qt-3.0 /home/user/squish-qt-3.2	/home/user/squish-qt-3.0 /home/user/squish-qt-3.2

Qt 3.0 and Qt 3.2 are installed in:

C:\Qt\3.0.7 C:\Qt\3.2.3	/usr/local/qt30 /usr/local/qt32	/usr/local/qt30 /usr/local/qt32

First, build only the server and the Qt wrapper using Qt 3.0:

C:\> cd C:\Squish-Qt-3.0 C:\Squish-Qt-3.0> configure --with-qtdir=c:\Qt\3.0.7 --disable-all --enable-server --enable-idl --enable-wrappers --enable-examples	$ cd /home/user/squish-qt-3.0 $ ./configure --with-qtd- ir=/usr/local/qt30 --disable-all --enable-server --enable-idl --enable-wrappers --enable-examples	$ cd /home/user/squish-qt-3.0 $ ./configure --with-qtd- ir=/usr/local/qt30 --disable-all --enable-server --enable-idl --enable-wrappers --enable-examples

Verify that the configuration log output reads `Checking Qt version ..... 3.0.x` where x would be the respective Qt maintenance release. Now build the server and the wrappers using Qt 3.0:

C:\Squish-Qt-3.0> build	$ ./build	$ ./build

Now change to the directory of the second Squish installation, point configure to the Qt 3.2 installation and select the remaining components:

C:\Squish-Qt-3.0> cd C:\Squish-Qt-3.2 C:\Squish-Qt-3.2> configure --with-qtdir=c:\Qt\3.2.3 --disable-server --disable-idl --disable-wrappers --disable-examples	$ cd /home/user/squish-qt-3.2 $ ./configure --with-qtd- ir=/usr/local/qt32 --disable-server --disable-idl --disable-wrappers --disable-examples	$ cd /home/user/squish-qt-3.2 $ ./configure --with-qtd- ir=/usr/local/qt32 --disable-server --disable-idl --disable-wrappers --disable-examples

Invoking

C:\Squish-Qt-3.2> build	$ ./build	$ ./build

a second time compiles the second group of binaries.

What's missing is the registration of the Qt wrapper. Normally this is done automatically, but when the split build was performed

the tool for registration (squishrunner) was missing at the point when the wrapper was built. All you need to do is "rebuild" the target like this:

`C:\Squish-Qt-3.2> build qtwrapperinit`	`$ ./build qtwrapperinit`	`$ ./build qtwrapperinit`

If you like, you could even perform the above steps on different machines, if you prefer to execute the tests remotely.

Usage Example

Here is a very brief summary of an example usage of this setup on a Unix system. Apart from the different path names, the approach is the same on Windows. We will assume an AUT with the path `/opt/ourcompany/ourapp`.

```
/home/user/squish-qt-3.0/bin/squishserver &
/home/user/squish-qt-3.0/bin/squishserver --config addAppPath /opt/ourcompany
```

At this point you could verify that the application will indeed be found in the given path:

```
/home/user/squish-qt-3.2/bin/squishrunner --info applications
```

The name of your application should be printed out – possibly along with other application names.

Now run the Squish IDE and uncheck the "Start squishserver locally on startup" option (Edit → Preferences), if this has not been done already.

```
/home/user/squish-qt-3.2/bin/squish
```

All that's left, before you can start creating new test cases, is the creation of a new test suite and the selection of ourapp in the suite's settings dialog.

3.2.5 Installation for testing Qt/Embedded 2.3 applications

Using *Squish* with Qt/E 2.3.x on X11 with QVFb

Installation

Unpack *Squish* into two directories, e.g. `/usr/local/squish-desktop`, `/usr/local/squish-embedded`.

In the `squish-desktop` directory we build the client-side parts of *Squish*. For this we need Qt version 3.1.x or higher (X11 version) with thread support turned on. For the rest of the example we assume that we are using Qt 3.2 and that it is installed in the directory `/usr/local/qt32`.

In the `squish-embedded` directory we build the parts of *Squish* that hook into the AUT as well as the server. For this build we need Qt/Embedded version 2.3.x, that has been compiled with thread support turned on and built as a shared library. The command line, with the parameters you passed to configure, should look something like

```
$ ./configure -depths 8,16,32 -qvfb -debug -thread -shared
```

For the rest of the example we assume that Qt/Embedded is installed in the directory `/usr/local/q23embedded`.

First we build the desktop version of *Squish*. Change into the directory `/usr/local/squish-desktop` and build it normally, linking against Qt 3.1.x or higher (see the Section 3.2.1 for a more detailed description):

```
$ cd /usr/local/squish-desktop
$ ./configure --with-qtdir=/usr/local/qt32
$ ./build
```

Next we have to build the Qt/Embedded version that hooks into your AUT and the server. For this, change to the `/usr/local/squish-embedded` directory and configure *Squish* with everything disabled except the server, the wrappers and the IDL compiler (we also enable the examples so that we have a small program for testing):

```
$ cd /usr/local/squish-embedded
$ ./configure --with-qtdir=/usr/local/qt23embedded --enable-debug --disable-all --enable- ↩
    server --enable-wrappers --enable-examples --disable-tk --enable-idl
```

then run

```
$ ./build
```

to build the Qt/Embedded version of *Squish*.

Testing the address book example

Now we can start with an initial test of a Qt/Embedded application. First add the AUT path of the address book example to the squishserver:

```
$ /usr/local/squish-embedded/bin/squishserver --config addAppPath /usr/local/squish- ↩
    embedded/examples/qt3/addressbook
```

Next open a command line shell from which you first start the Qt Virtual Framebuffer:

```
$ /usr/local/qt32/tools/qvfb/qvfb -width 1280 -height 1024 &
```

Then set `QTDIR` to point to your Qt/Embedded 2.3.x installation directory and start squishserver:

```
$ export QTDIR=/usr/local/qt23embedded
$ /usr/local/squish-embedded/bin/squishserver
```

Open another command line shell from which you can control the testing:

```
$ /usr/local/squish-desktop/bin/squish
```

This starts the Squish IDE. Now open the test suite `/usr/local/squish-embedded/examples/suite_addressbook/suite.conf`.

You also have to tell the Squish IDE to connect to the squishserver we started, rather than starting one of its own. Open Edit → Preferences and uncheck Start squishserver locally on startup. Make sure that squishserver host is "localhost" (or "127.0.0.1") and that the squishserver port is "4322" (See Section 7.7.2 for more details).

Finally we also have to tell Squish to start our AUT with the `-qws` argument: choose Test Suite → Settings and enter `-qws` as the Arguments for the AUT.

Now you can run tests or record them, as described in the manual. You will see that the AUT (address book example), which is compiled as a Qt/Embedded application, opens up in the Qt Virtual Framebuffer and you can record events and replay the test cases in there.

Using Squish with a cross-compiled Qt/E 2.3.x

Installation

The installation procedure for building *Squish* with a cross-compiled Qt/Embedded 2.3.x is almost identical to the procedure for Qt/Embedded with the Qt Virtual Framebuffer. The only differences are the use of a different squishidl and the setting of the `XCXX` environment variable, specifying the cross compiler.

Make sure that the cross compiled Qt 2.3.x/Embedded is built as a shared library with thread support turned on. For the rest of the example we assume it is installed in `/usr/local/qt23xcompiled`.

After you have built Squish with the desktop version of Qt, copy the squishidl tool over and set the `LD_LIBRARY_PATH` variable to Qt's library path, if the library is not found automatically (the squishidl tool is needed at compile-time and not at run-time during the test):

```
$ cd /usr/local/squish-embedded
$ mkdir bin
$ cp /usr/local/squish-desktop/bin/squishidl bin/
$ export LD_LIBRARY_PATH=/usr/local/qt32/lib
```

Next configure the embedded build. You have to set the environment variable `XCXX` to the cross-compiler you are using. This time we also disable the squishidl tool (we already copied it over):

```
$ XCXX=arm-linux-g++ ./configure --with-qtdir=/usr/local/qt23xcompiled --enable-debug -- ↩
    disable-all --enable-server --enable-wrappers --enable-examples --disable-tk
```

then run

```
$ ./build
```

to build the cross-compiled version of *Squish*.

Testing the address book example

This is essentially the same, except that you start the squishserver on the device and enter the IP address of the device, instead of "localhost", as the "squishserver host" in Squish IDE's preferences dialog.

To avoid problems during testing we recommend verifying the correctness of your Qt and AUT setup prior to test execution. Is `QTDIR` pointing to the Qt installation directory (must be set prior to launching the server)? Can you successfully run the AUT standalone?

3.2.6 Installation for testing of Qt/Embedded, QtopiaCore and Qtopia 4.x applications

3.2.6.1 Build instructions

Principally, the build instructions given for a build against Qt/Embedded 2.x apply to QtopiaCore and therefore to Qtopia 4.x as well. It's mainly the changed directory layout between the two versions that makes a difference for our purpose.

To build the parts of *Squish* that hook into the AUT unpack the source package into a directory, change to that directory and use configure to specify the location of the QtopiaCore installation. In the example

```
$ ./configure --with-qtdir=/home/user/qtopia-build/qtopiacore/target
--disable-all --enable-server --enable-examples --enable-idl
--enable-wrappers --enable-pure-qt4
```

the `--with-qtdir` switch points to the location of QtopiaCore installation. That is where the file `lib/libQtCore.so` is to be found for example.

Once the configure run has finished issue

```
$ ./build
```

to build bin/squishserver and most importantly the QtopiaCore-specific wrapper library. The client side tools (e.g. squishrunner and the IDE) can either be obtained by compiling a source package against a Qt/X11, Qt/Win or Qt/Mac installation or installing one of the pre-build binary versions of *Squish*.

3.2.6.2 Cross-compilation

A so-called cross-compilation is performed when building software on the local host system that will later run on a target system that has a different architecture. This is a common requirement when building software for special purpose hardware that is too limited in terms of cpu, memory or tools to serve as a development itself.

Part of the build process requires the execution of the squishidl tool that scans the Qt header files to provide test script bindings. The tool needs to run on the host system and therefore needs to be build with a native compiler. This is done in the first phase:

Unpack a *Squish* source package into a directory like `squish-desktop`. Change the directive

```
#if 0 // enable for cross-compilations
```

in `src/idl/cpp/cppinput.cpp` (at around line 1157) to read

```
#if 1
```

Then run:

```
$ ./configure --with-qtdir=/path/to/desktop/qt --disable-all --enable-idl
$ ./build
```

Now you will have the squishidl tool in the `bin` directory.

For the second phase - the actual cross-compilation perform the following steps:

Unpack the source package in a different directory. Like `squish-embedded`. Use a text editor to create a shell script named `squishidl` in the `bin` directory that has the following content:

```
SQUISH_DESKTOP=/path/to/squish-desktop
LD_LIBRARY_PATH=${SQUISH_DESKTOP}/lib:${LD_LIBRARY_PATH}
export LD_LIBRARY_PATH
${SQUISH_DESKTOP}/bin/squishidl "$@"
```

Make that shell script executable with **chmod +x bin/squishidl**. You can test whether this wrapper around the real tool works by issuing **bin/squishidl --help**.

Now, inquire with development to learn about the cross-compiler they use for application development. Specify its path to the `configure` via the `XCXX` environment variable and select to build only those parts that need to be installed on the target:

```
$ ./configure --with-qtdir=/path/to/qt-embedded
  --disable-all --enable-server
  --enable-wrappers --enable-pure-qt4
```

Once that run has completed start the build which will take a few minutes:

```
$ ./build
```

3.2.6.3 Installation on the target device

Once the executables and libraries are build copy over the directories `bin/`, `lib/` and `etc/` and their content to the target device.

The Qt directory in your build environment is likely to differ from the target system. As the executables use Qt themselves they need to be told where to find it. That can be accomplished by adapting the `LibraryPath` variable in `etc/paths.ini`.

Separate from that *Squish* might need to be told where to look for the Qt libraries used by the application. This might be the same set as used for the tool itself but can be different. By default *Squish* will look in the directory that is was told about during the configuration on the build system. There are two ways to tell it about the path on the target system: 1.) set the environment `SQUISH_LIBQTDIR` to Qt's `lib/` directory before starting squishserver. 2.) add an entry like `UserQtLibDirectory=-"/path/to/qt/lib"` to the `[General]` section of the `etc/squish.ini` file.

By default *Squish* expects the Qt libraries unversioned files `libQtCore.so` and `libQtGui.so` to be installed. These are typically used for development purposes. The target system might only have the versioned runtime library `libQtCore.so.4` installed, however. It is possible to modify the build to expect only the latter (contact support for help with that) but there is a simple alternative. Create the same symlinks that you will find on your PC by issuing the following commands on the system:

```
$ ln -s /path/to/qt/lib/libQtCore.so.4 /path/to/qt/lib/libQtCore.so
$ ln -s /path/to/qt/lib/libQtGui.so.4 /path/to/qt/lib/libQtGui.so
```

3.2.7 Installation for testing with a single-threaded Qt library

The easiest approach is to link the AUT and all of *Squish* to the same Qt library. If it is possible to link your AUT to a shared, multi-threaded Qt library, we recommend using this for both Squish and your AUT. Any other Qt configuration settings, (STL, etc.) don't matter.

If you don't want to use a multi-threaded Qt library for your AUT, you can use one Qt library for your AUT and a different one for Squish (some parts of *Squish*, mainly the squishserver, require a multi-threaded Qt library).

However, the parts of Squish that hook into your AUT (Squish's qtwrapper, hook and object model libraries) have to be built against the *exact* same Qt library as your AUT. Also make sure that the same C++ compiler is used. The rest of Squish can be linked with any Qt library.

So in order to support a single-threaded Qt library, you have to perform a split build of Squish, i.e. build some parts against a multi-threaded Qt library and some parts against the single-threaded Qt library that your AUT uses.

Note
If you want to use a single-threaded Qt library on Windows, please take a look at Q: 10.1.4.

For the example we assume that the single- and multi-threaded Qt libraries are in the following locations:

Windows	Linux/Unix	Mac OS X
C:\Qt\singlethreaded C:\Qt\multithreaded	/usr/local/qt-singlethreaded /usr/local/qt-multithreaded	/usr/local/qt-singlethreaded /usr/local/qt-multithreaded

First compile only the multi-threaded parts of *Squish* by running configure, disabling everything but the server:

C:\squish> configure --with-qtdir=C:\Qt\multithreaded --disable-all --enable-server C:\squish> build	$ ./configure --with-qtdir=/usr/local/qt-multithreaded --disable-all --enable-server $ ./build	$ ./configure --with-qtdir=/usr/local/qt-multithreaded --disable-all --enable-server $ ./build

Then compile the remaining parts of Squish with the single-threaded Qt by running configure again, this time disabling only the server:

C:\squish> configure --with-qtdir=C:\Qt\singlethreaded --disable-server C:\squish> build	$ ./configure --with-qtdir=/usr/local/qt-singlethreaded --disable-server $ ./build	$ ./configure --with-qtdir=/usr/local/qt-singlethreaded --disable-server $ ./build

Make sure that the AUT is linked against exactly the same single-threaded Qt library. If in doubt, use one of the following tools to verify the correct linkage:

Dependency Walker (depends.exe)	ldd or chatr	otool

3.2.8 Installation for testing with a static Qt library

If you want to link the application against a static version of Qt, this is only partly supported. For recording test cases, the application must be linked against a dynamic Qt library. For running tests on Unix, the application can be linked against a static

Qt library. On Windows and on Mac OS X this is not possible, so the AUT must always be linked against a shared Qt library.

Note
It is not possible to record any tests with a static Qt library, but it is possible to replay test scripts; but currently only on Linux/Unix. On Windows and Mac OS X it is not possible to use a static Qt library.

If you want to run tests on Unix on a statically linked application, there is one important requirement. When linking the application its symbols must be exported so that *Squish* can hook into the application. To do this using Linux/gcc, for example, specify `-rdynamic` on the link line. This switch is harmless, so you can leave it in permanently and ship the application linked this way. On some operating systems (e.g. Solaris) all the application's symbols are exported by default, so no special linker switch is necessary. In this case the configuration of the static Qt library doesn't matter at all.

3.2.9 Installation for testing with a renamed Qt library

Note
The following currently works only with Qt 3; installation of *Squish* for testing a Qt 4 application with a renamed Qt library is not supported at the moment. If you you need support for this, please contact us at squish@froglogic.com.

If you want to use *Squish* with a renamed Qt library, you have to use additional options to configure. On Unix and Mac OS X you have to use the `-with-qtlibname=`*`renamed Qt library`*.

On Windows, you have to specify three options (you have to specify all three options, even if one of the libraries is not renamed; in this case, just specify the original version):

- `--with-qtdllname=`*`path to Qt`*`/bin/`*`renamedqt-mt.dll`*
- `--with-qtlibname=`*`path to Qt`*`/lib/`*`renamedqt-mt.lib`*
- `--with-qtmainlibname=`*`path to Qt`*`/lib/`*`qtmain.lib`*

With the option `--with-qtdllname` you have to specify the path to the renamed Qt DLL, with `--with-qtlibname` you have to specify the path to the renamed Qt import library and with the option `--with-qtmainlibname` the path to the renamed `qtmain.lib` library.

The renamed Qt DLL and import library should contain the characters `mt` if you have a multi-threaded Qt library. configure uses this to detect if the Qt library is mult-threaded or single-threaded. If you don't follow this convention, the automatic detection fails and some parts of *Squish* are falsely disabled.

3.3 Distributing and Sharing an Installation

Motivation

Once you have completed the build steps described in the previous section you have a system ready to use on the local machine. You may repeat these steps for licensed users on as many machines as you like.

This section is devoted to users or administrators that wish to distribute or share a build performed on one machine to a different location and users on other machines, respectively. If access to a local installation is all you need, you can safely skip the rest of this section and proceed to the chapters about using *Squish*.

There are, of course, many reasons why you would want to reuse the built executables and libraries. These include:

- Division of labor between developers or administrators (who will *build Squish*) and testers (who will *use Squish*).
- Separation between machines equipped for product development and machines dedicated for testing.

- Reduced maintenance work required when upgrading the software.
- Saving of compile time
- Saving of disk space

Distributables

After completing the build, the distributables can be found in the top-level `bin`, `lib` and `etc` directories. `bin` contains the executables (and `.dll` files on Windows) while underlying libraries and script files reside in `lib`. `etc` houses installation configuration files (user specific settings are stored elsewhere as explained further below). Here is the complete list:

Windows	Linux/Unix	Mac OS X
bin\squish.exe bin\squishidl.exe bin\squishrunner.exe bin\squishserver.exe bin\dllpreload.exe bin\squishhook.dll bin\squishinterpreter.dll bin\squishobjectmodel.dll bin\squish*wrapper.dll bin\squishserverlib.dll bin\squishtest.dll bin\fkit.dll bin\kjs.dll bin\squishsw.dll bin\winhook.dll	bin/squish bin/squishidl bin/squishrunner bin/squishserver bin/isstaticapp	bin/squish bin/squishidl bin/squishrunner bin/squishserver bin/isstaticapp
lib_squish.exe lib_squishrunner.exe lib_squishserver.exe lib\squishqtpre.dll lib\keycompression.tcl lib\propertydelegates.tcl lib\uniqueproperties.tcl lib\uniqueproperties_qt4.tcl	lib/_squish lib/_squishrunner lib/_squishserver lib/libsquishhook.so lib/libsquishinterpreter.so lib/libsquishobjectmodel.so lib/libsquishqtpre.so lib/libsquish*wrapper.so lib/libsquishserverlib.so lib/libsquishtest.so lib/libfkit.so lib/libkjs.so lib/libsquishsw.so lib/keycompression.tcl lib/propertydelegates.tcl lib/uniqueproperties.tcl lib/uniqueproperties_qt4.tcl	lib/_squish.app lib/_squishrunner lib/_squishserver lib/libsquishhook.dylib lib/libsquishinterpreter.dylib lib/libsquishobjectmodel.dylib lib/libsquishqtpre.dylib lib/libsquish*wrapper.dylib lib/libsquishserverlib.dylib lib/libsquishtest.dylib lib/libfkit.dylib lib/libkjs.dylib lib/libsquishsw.dylib lib/keycompression.tcl lib/propertydelegates.tcl lib/uniqueproperties.tcl lib/uniqueproperties_qt4.tcl

Windows	Linux/Unix	Mac OS X
etc\paths.ini etc\squish.ini etc*wrapper_descriptors.xml	etc/paths.ini etc/squish.ini etc/*wrapper_descriptors.xml	etc/paths.ini etc/squish.ini etc/*wrapper_descriptors.xml

The above can be broken down into groups of files (with some overlap), being part of the server, runner, IDE and wrapper generator. These groups can be distributed seperately if desired.

Distribution

To distribute the build to another location on the same machine, or a different one, just copy over all (or only the required) files from the three directories listed above. While the original build path has been hard-coded into the binaries, they will also work in a different location. To enable the executables to locate their sub-modules, it is important that the relative position remains the same. This means that you can copy the contents of `bin`, `lib` and `etc` to whatever directory prefix you want, as long as they keep their parallel position.

When *Squish* hooks into the AUT, it loads the Qt library. The Qt library (with the full path) is hard-coded into the binaries. If the Qt library is in a different location on the target machine, you must set the environment variable `SQUISH_LIBQTDIR` to point to the `lib` directory of your Qt library.

Shared Installation

To allow all licensed users to access *Squish* over a shared network location, make the `bin`, `lib` and `etc` directories accessible to them. This can be done via a share on Windows or via an NFS mount on Unix, for example. As mentioned earlier, the `bin`, `lib` and `etc` directories have to maintain their parallel position.

User specific settings will be stored locally on a per-user basis in the application data directory on Windows (`%APPDATA%\froglogic\Squish`) or the `~/.squish` directory on Unix and Mac OS, respectively.

Chapter 4

Concepts and Making an Application Testable

4.1 Squish Concepts

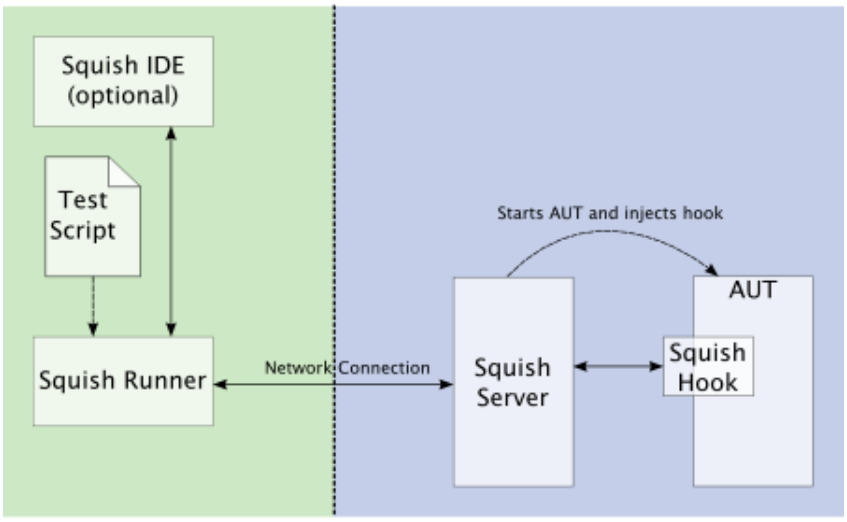
When testing an application we have two main components:

- An application under test (AUT)
- A test script that exercises the AUT

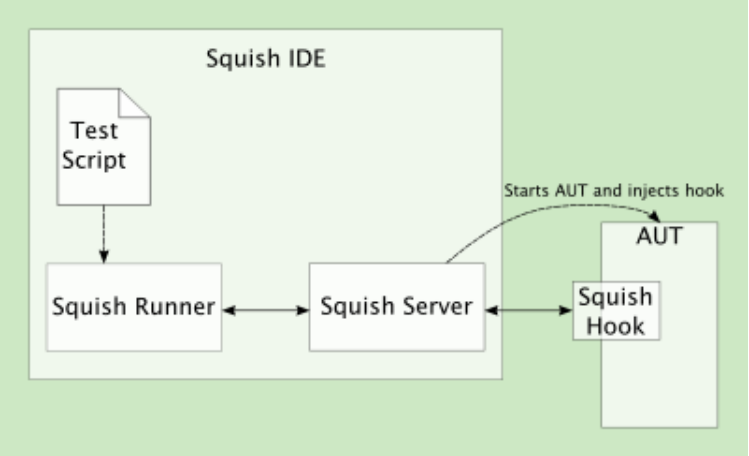
One fundamental aspect of *Squish*'s approach is that the test script and the AUT always run in two separate processes. So if the AUT crashes, it should never crash the test script execution. Other benefits of this approach are that it allows you to store the test scripts at a central location, and that you can test applications on different machines and platforms remotely. This is especially useful when testing GUI applications on embedded devices.

Squish runs a small server (squishserver) that handles the communication between the test script and the AUT. The test script is executed by the squishrunner tool, and connects to the squishserver. The AUT uses *Squish*'s hook library to connect to the server. This library allows the test script to inspect the application remotely via a socket connection.

The following diagram illustrates how the individual script components work together.



From the script writer's perspective this separation is not visible at all, since all the communication is handled transparently behind the scenes. When using the Squish IDE, all components are used implicitly as the following diagram shows.



To be able to access the AUT's functionality from test scripts, the following is necessary:

- Bindings to the GUI toolkit used by the application to obtain information about the object hierarchy, standard components, and their properties, functions, etc.
- *Optionally*: bindings to the API of the application or additional components in situations where the toolkit's bindings don't provide sufficient functionality for testing.

The former is provided by *Squish*, which comes with generic toolkit bindings. The latter is application specific and rarely necessary.

4.2 Making an application testable

In the usual case, nothing special needs to be done to make an application testable, since the toolkit's API (e.g. Qt) provides enough functionality to implement and record test scripts. The connection to the squishserver is also established automatically, when *Squish* starts the AUT.

Chapter 5

Tutorial: Creating the First Test with *Squish*/Java™

This chapter will use the Squish IDE to create, run, and extend a test of a sample Swing application. We will use the Calculator example from Java Swing Tutorial | Swing Examples to demonstrate how *Squish* is used to test Java applications.

Even though we will use the Squish IDE, a graphical front end to the *Squish* testing framework, anything we will do can also be accomplished using command line tools which is interesting when thinking about automating your tests (doing nightly runs of your regression test suite, for example). Whenever we describe how to do something using the Squish IDE we will also show how to do it with the command line tools. For full details on *Squish*'s command line tools, please refer to the Chapter 8.

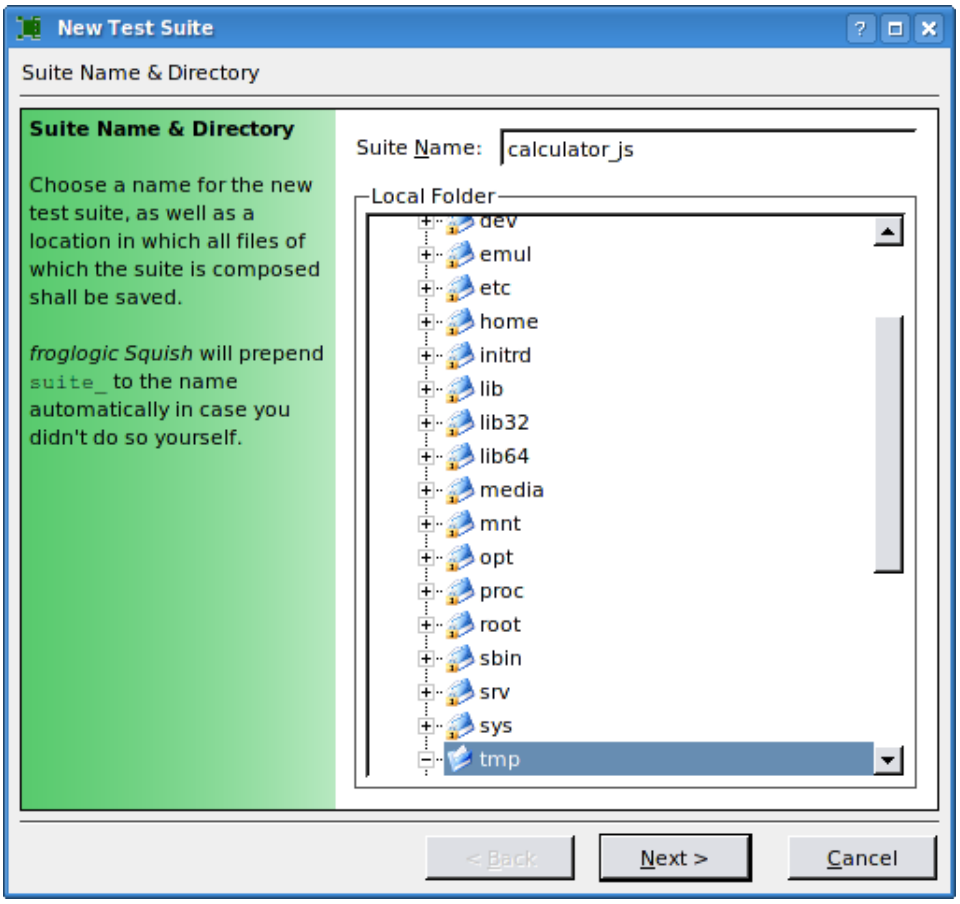
Tip
The complete test suite we create in this tutorial is available in `squish/examples/java/suite_calculator_js`, so you don't necessarily have to type everything shown here yourself.

For testing applications with *Squish*, a special server application called squishserver must be running (for further information, you might want to have a look at the section Chapter 4). If you use the Squish IDE, you don't need to worry about this since the Squish IDE will start a server automatically and care about everything. However, not using the Squish IDE means that you have to start squishserver manually before you can record or run any tests. Please see the respective section in the command line reference Section 8.2.2 for information how to do this.

5.1 Creating a Test Suite

The first step is to create a new test suite. To do so, start the Squish IDE, and then select File → New Test Suite (or press Ctrl+Shift+N) to open a convenient wizard which will walk you through the process of creating a new test suite.

5.1.1 Choosing a Name



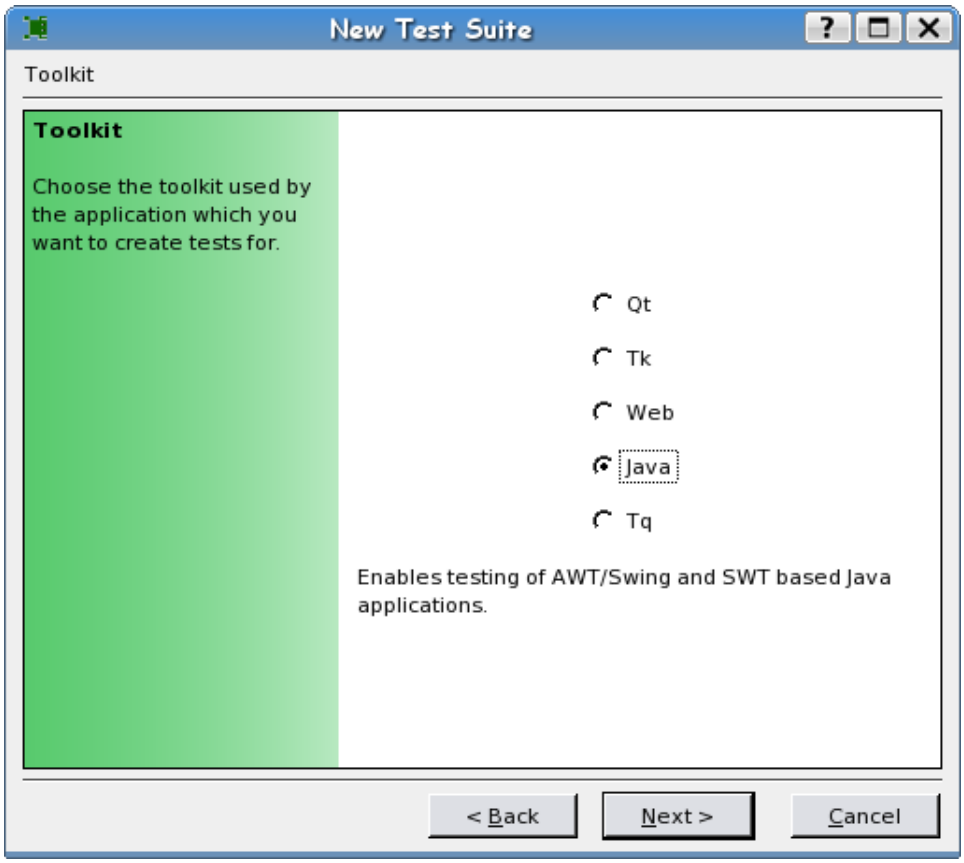
The first page of the wizard used for creating new test suites.

The first page of the wizard lets you choose where the test suite is going to be stored. Simply choose a directory in the directory tree view, and enter a name for the new test suite in the input field above. The screen shot above shows that calculator_js was chosen.

Note
If your suite name does not start with 'suite_' then the wizard will prepend that prefix automatically, since *Squish* expects suite names to start with that text.

After deciding on a location for the suite, press the button labeled Next to progress to the next page.

5.1.2 Selecting a Toolkit



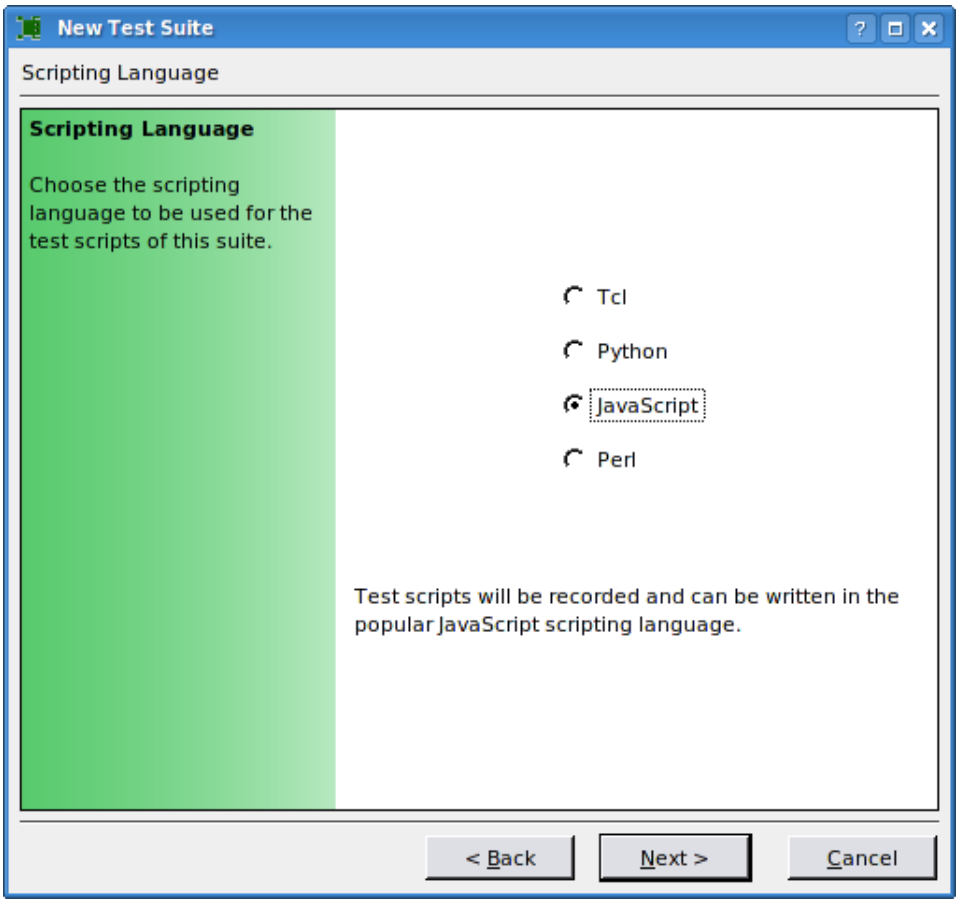
The second page of the wizard used for creating new test suites.

This next page allows you to choose what kind of application you want to test in this test suite. Since we want to test a Java™ application, we select the Java option here and then advance to the next step of the test suite creation using the Next button.

Squish supports both AWT/Swing GUI applications as well as SWT based ones.

Note
In case your license key does not include support for testing Java™ applications, or if you're using a binary package of *Squish* which does not include support for testing Java™ applications, then there won't be an option labeled Java on this page. Please contact *froglogic* at sales@froglogic.com for finding out how to get support for testing Java™ applications.

5.1.3 Choosing a Scripting Language



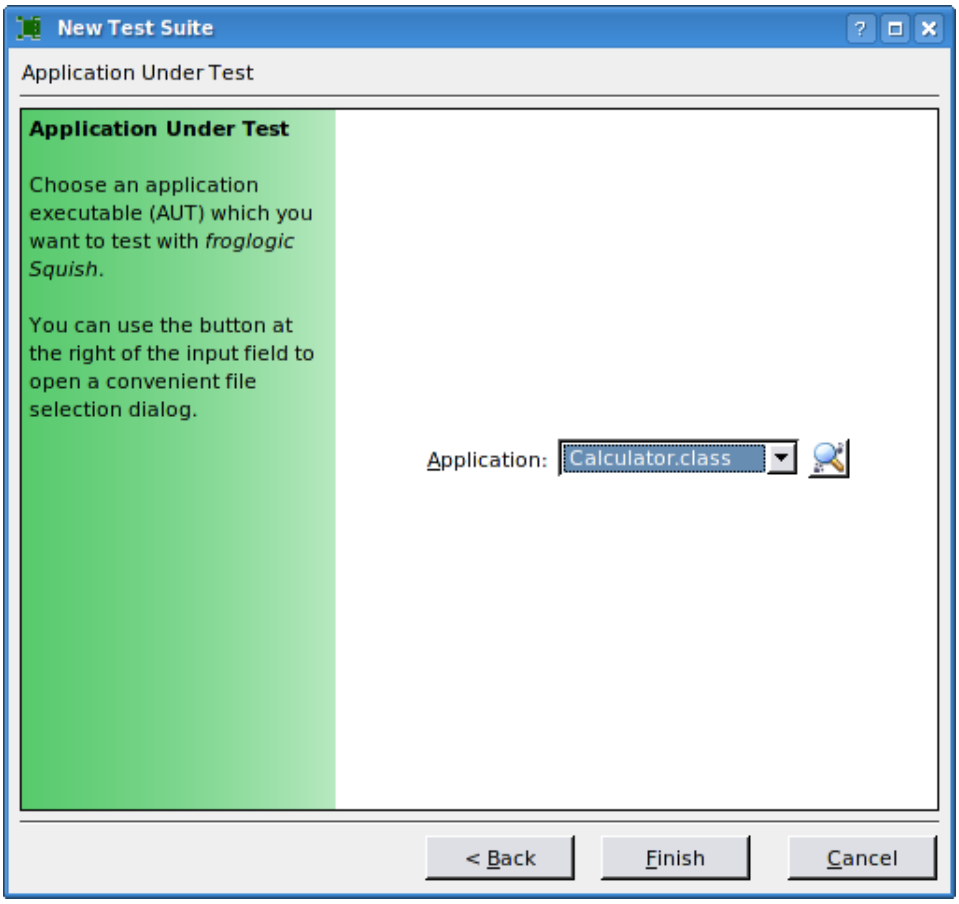
The third page of the wizard used for creating new test suites.

Now you get to choose between different scripting languages used for authoring test scripts for your application. The functionality of *Squish* is independent of the scripting language used, but your personal preference and experience may vary. For this tutorial, we'll choose JavaScript by selecting the JavaScript option.

Tip
A comparison between the scripting languages is available in the chapter Section 8.1.1.

5.1.4 Selecting the AUT

Enter the final step and specify the application you want to test with this test suite. Choose the browse button () and select the `Calculator.class` file. You find it in `squish/examples/java/calculator`. This is the AUT *Squish* launches for this test suite.

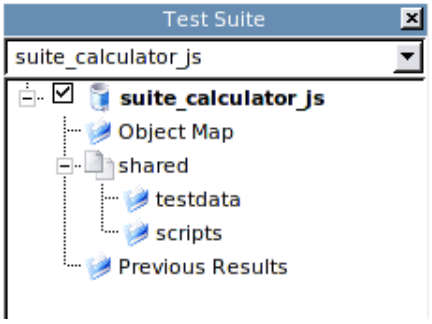


Additional arguments, the working directory of this application or custom environment variables can be configured later in the test suite's settings, see Section 7.2.1 for more on this subject. For the calculator example none are required.

Note
Instead of a Java .class file you can also choose one of the following file-types when specifying your AUT:

- JAR-file: The main JAR file of your Java application.
- Native EXE: A native executable which launches your Java application. This is usually the case for RCP/SWT applications and often also for Swing/Netbeans applications where a native launcher application is built.
- Shell script or batch file: A shell script or batch file wherein you launch the Java application yourself.

This page concludes the creation of our test suite. Clicking the Finish button closes the wizard and will show our new test suite in the tree view at the left side of the Squish IDE, similar as in this screen shot:



An empty testing suite as shown in the tree view of the Squish IDE.

Now we can start adding test cases to our suite.

5.2 Recording a Test

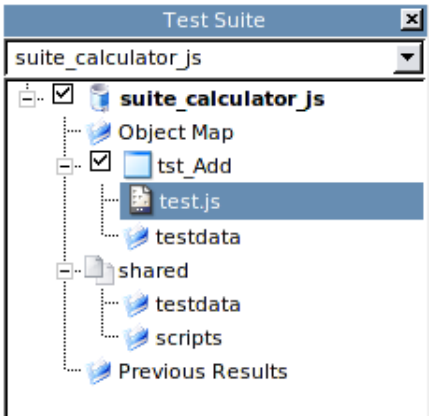
In *Squish*, tests can either be written by hand, they can be recorded - or you can use a mixture of the two. We'll start with creating a new test case and then recording a script of user interactions. Afterwards, we will augment the script with our customizations.

5.2.1 Creating a Test Case

First, let's create a new test case. To do so, open the context menu of our newly created suite_calculator_js suite by clicking on the item labeled suite_calculator_js with the right mouse button. Then, select New Test Case... to create a new test case. An input field will be shown, waiting for you to enter a name for the new test case. For this tutorial, we will go for 'Add' as the name for our first test case.

Tip
Another way to create a new test case is to select File → New Test Case... or press Ctrl+N. This will create a new test case in the currently active test suite (the one shown in the tree at the left side of the Squish IDE).

After creating a new test case, your tree view at the left hand side of the Squish IDE should look similar to what is shown in the next picture.

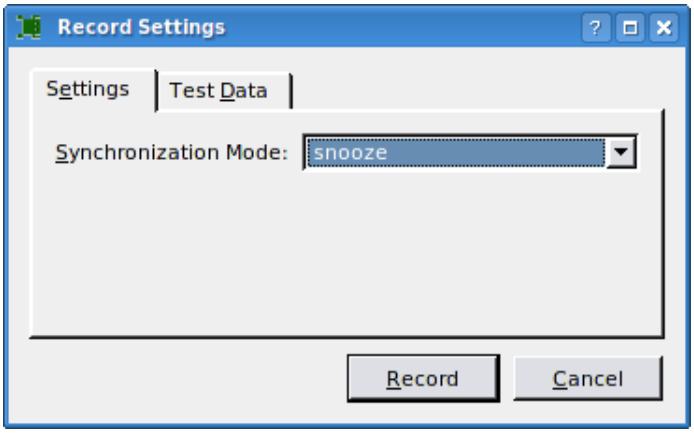


The Java testing suite with one test case called tst_Add as shown in the tree view of the Squish IDE.

Now everything is ready to record a test.

5.2.2 Recording User Input

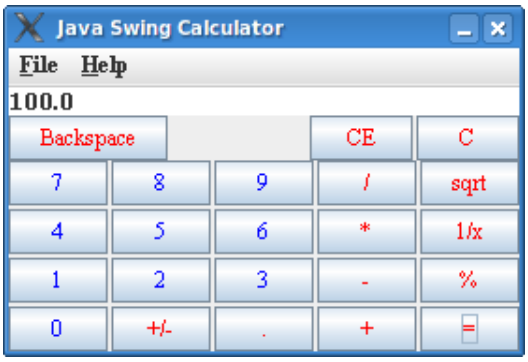
To record a test, either select Record... from the context menu of the test case item in the tree (the one labeled tst_Add in our example), or go via the menu bar by selecting Test Suite → Record Test Case → tst_Add.



The generic record settings dialog shown when recording.

This will minimize the Squish IDE into a small control window, shown in the upper left corner. A dialog with the caption Record Settings will pop up, allowing you to fine-tune some recording settings such as whether to use any initial test data and what kind of synchronization method to use. For many situations, including ours, just sticking to the defaults and accepting them by clicking the Record button in the dialog is fine.

Now we can test the calculator adding capabilities. For this tutorial we entered 47, then the plus button, entering 53 and finally the equal sign.



The calculator.

When you are done, close the calculator application to bring the Squish IDE back, showing a script of the recorded actions you did.

Now, let's have a look at the generated script and what it does.

5.2.3 A Generated Script

The actions resulted in the following JavaScript script. Depending on how much the actions you did on the application, your script might differ more or less from the one shown here. Before we go on, it might be a good idea to look at this script for a minute or two, trying to figure out what all the calls do.

```
function main() ❶
{
    snooze(1.5); ❷
    clickButton(":frame0.4_javax.swing.JButton"); ❸
    snooze(0.7);
    clickButton(":frame0.7_javax.swing.JButton");
    snooze(1.0);
    clickButton(":frame0.+_javax.swing.JButton");
    snooze(0.8);
    clickButton(":frame0.5_javax.swing.JButton");
    snooze(0.6);
    clickButton(":frame0.3_javax.swing.JButton");
    snooze(0.8);
    clickButton(":frame0.=_javax.swing.JButton");
    snooze(1.1);
    activateItem(":frame0_javax.swing.JMenuBar", "File"); ❹
    snooze(1.0);
    activateItem(":frame0.File_javax.swing.JMenu", "Exit");
}
```

Here's a short explanation what's happening:

❶ The recording always produces a function called `main`. A `main` function is mandatory in *Squish* test cases.

❷ The `snooze` statements are used to wait for a while before going on. Since some time elapses between the different user actions, a snooze is recorded (it takes the number of seconds to wait, one and a half second in this case) and later played back.

You will notice that there are a lot of snooze statements in recorded scripts. You can change the amount of time which they wait before execution of the script proceeds (or you can remove them altogether) if you like, but please note that this could break the script. For instance, after clicking on a menu, you should wait some time for the menu to show, which may take a bit longer the first time it is opened.

3. We click on a button. Note the naming of the button. It starts with the name of the frame, added the caption and finally appended the class name.

4. A menu item was activated.

To check that everything works as expected so far, you could now run this script and watch *Squish* replaying your actions. To do this, just press the little 'Play' button () in the toolbar of the Squish IDE.

5.3 Introducing Verification Points

Now that we have a first script recorded, we need to augment it with some checks verifying some properties of the calculator applicaton so that we actually test something. With *Squish*, such checks are done via 'Verification Points'.

A verification point consists of at least one comparison which looks whether some given condition was met. For instance, a verification point could test whether a given button is enabled or not and whether the caption of the button equals a given string.

In *Squish* there are multiple ways to implement verification points. The most generic and powerful way is to implement verification points manually as script commands. Another, easier approach is to use the Squish IDE to create verification points via point & click.

Inserting verification points via the Squish IDE is much simpler and only in rare cases it is necessary to manually write verification point statements. So we use the Squish IDE here. For more information about manually writing verification points, consult the chapters Section 7.1.2 and Section 7.10.

5.3.1 Verifying the Outcome using Verification Points

We may want to check the outcome of the calculation we did. If the application is further developed, we want to catch possible regressions (which is in this example rather embarrassing if the addition would not result in 100).

We check the value of the calculation after the equal sign is pressed This is done by setting a break point on the line of the script where the verification point should be inserted. This is done by clicking on the line number next to the line with the left mouse button.

```
8    clickButton(":frame0.+_javax.swing.JButton");
9    snooze(0.8);
10   clickButton(":frame0.5_javax.swing.JButton");
11   snooze(0.6);
12   clickButton(":frame0.3_javax.swing.JButton");
13   snooze(0.8);
14   clickButton(":frame0.=_javax.swing.JButton");
15   snooze(1.1);
16   activateItem(":frame0_javax.swing.JMenuBar", "File");
17   snooze(1.0);
18   activateItem(":frame0.File_javax.swing.JMenu", "Exit");
19
20 }
```

Setting a new break point in the editor of the Squish IDE.

Now, run your test suite by pressing the run button () in the toolbar of the Squish IDE. *Squish* will the minimize itself and starts the calculator application, replaying the recorded actions.

As soon as the break point is hit, execution of the script is suspended and *Squish* restores itself again. You can see the editor in the Squish IDE, showing that it's halted at the last line. Now we're at the right position in our script to insert a verification

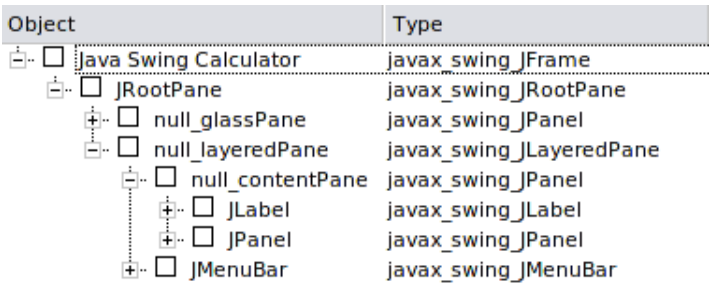
point. Below the editor, you can find a monitor showing all variables in your script file and their values. Since we don't have any variables defined, this window most likely just shows the text No variables in scope.

Note
In case you did not choose JavaScript as the scripting language to use when creating the test suite (see Section 5.1.3), it might be that you do see some variables since some languages have some global variables by default. This is not the case for JavaScript though.

5.3.2 Selecting Properties to Check

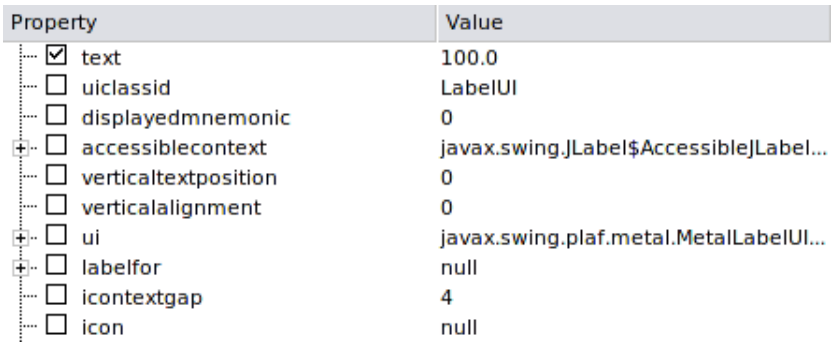
To select the properties of the Java™ application which we want the verification point to check, we use the so-called 'Spy'. You can invoke the Spy by clicking on the corresponding button () in the toolbar of the Squish IDE. The variable watcher will disappear and get replaced with a similar-looking component which is divided vertically. This is the Spy. It's possible that you might need to wait for a few moments for the left hand side of the spy to get populated.

In the left hand side of the spy, you can see a tree showing the Java™ frame object. Expanding the tree shows the component hierarchy in this application. Selecting one object, will populate the right hand side of the spy with a list of all testable properties of that object. In order to be able to test our case, we need to find the label that shows the result.



The component hierarchy of the calculator application.

For purpose of a tutorial, we pick the right object from the application by using the object picker () from the toolbar by clicking on it. The Squish IDE will minimize itself and now click on the label of the calculator that shows the result of '100.0' - and voila! The Squish IDE shows up again, showing just that one object in the spy's object tree. Select the object by clicking on it to see all the properties:



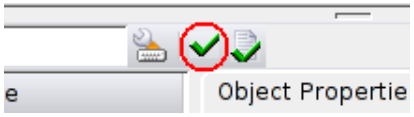
The properties of the result label.

The `text` propery has the value we need for verification.

Note
The property `text` is dynamic generated by *Squish* and uses the `getText` and `setText` internally.

5.3.3 Injecting a Verification Point Into the Script

Select the checkbox before the property `text`. The verification point is injected into the script using the little green check mark icon at the top of the spy, right next to the input field for the name of the verification point:



The button in the spy used for adding new verification points.

Click that button and you should see a new verification point called VP1 show up in the tree view at the left hand side of the Squish IDE. To actually insert the verification point into the script, close the spy again by clicking on the spy button () in the toolbar again, and then continuing execution of the script beyond our break point by clicking on the run button ().

When the script finishes, *Squish* inserts a new statement into the script to test the new verification point: `test.vp("VP1")`. The `test.vp` function is used for testing verification points and expects to be given the name of the verification point to test. Since the new verification point is called 'VP1', we pass that to `test.vp`.

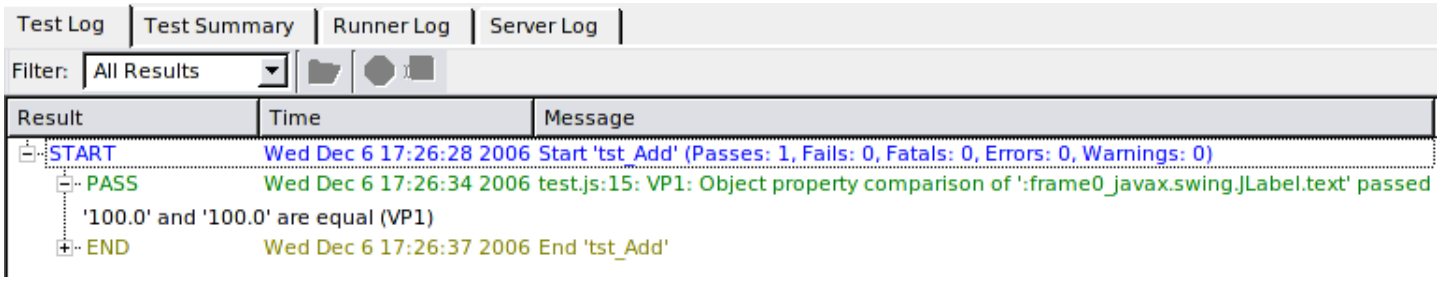
```
11  snooze(0.6);
12  clickButton(":frameO.3_javax.swing.JButton");
13  snooze(0.8);
14  clickButton(":frameO.=_javax.swing.JButton");
15  test.vp("VP1");
16  snooze(1.1);
17  activateItem(":frameO_javax.swing.JMenuBar", "File");
```

The newly inserted test.vp call.

5.3.4 Verification Points in Action

Now, let's see whether the test really worked. To do so, we first need to remove the break point again we defined a few moments ago. Just click on the little red icon with the white cross in front of the newly inserted `test.vp` call to remove the break point. Now, run the test again using the run button ().

If everything goes right, the test finishes, and you should see a positive test result in the test results view, which is at the very bottom of the Squish IDE:



Our verification point succeeded!

This is a detailed report of the test run and would also contain occurring failures, errors, etc. If you click on a test log item, the Squish IDE highlights the script line which generated the test result.

Tip

The interface to the test results in Squish is very flexible. By implementing custom report generators it is possible to process test results in many different ways, for example to store them in a data base, or to output them as HTML files. You can also save the test results in the Squish IDE as XML by right clicking on the test results and choosing Save Results As.

If you run tests on the command line using squishrunner, you can also export the results in different formats and save them to files. See the chapters Section 7.7.3 and Section 7.1.2 for more information.

5.4 Conclusion

This concludes the tutorial. A test for a sample application was created, executed and augmented. Still, there is much to learn of course, since the API which *Squish* provides gives so much more power, not to speak of the vast expressiveness of any of the scripting languages supported by *Squish*. The other tutorials and the Chapter 7 give introductions and details about more features of Squish and topics of automated GUI testing.

Chapter 6

Tutorial: Setting Up Automated Batch Testing

This tutorial walks through the steps necessary to set up running automated tests and to process the results.

We start by creating a script which processes *Squish*'s XML test results and generates a HTML file with the information. In the second step we create a script which automatically runs test suites and processes the test results automatically.

6.1 Processing Test Results

6.1.1 Generating HTML Test Results

One important part of automating test runs is to present the test results to the testing team. *Squish* can store the test results in a XML format that is suitable for arbitrary processing. In this chapter we use a Python script to convert the XML to HTML.

But the XML format allows various conversion, e.g. you can integrate the test results into a test management system. Any post-processing is done in very similar ways. With the information in this chapter it is possible to create any kind of presentation of the test results.

In order to convert the *Squish*'s XML format, one has to know its format; it is described in detail in the chapter Section 7.7.3 in the Chapter 7.

As an example we take the XML test result when running a sample `suite_addressbook_py` test suite which contains two test cases for whether entering data into a simple addressbook database application works as expected.

To generate the XML report, we run the test suite using squishrunner (assuming that squishserver is running):

```
squishrunner --testsuite suite_addressbook_py --reportgen xml,/tmp/results.xml
```

The possible values for *`reportgen`* are *`xml`* (XML), *`xls`* (Excel) and *`stdout`* (ASCII table).

This produces a XML result file similar to the one below:

```
<SquishReport version="1.2" >
    <summary fatals="0" testcases="2" expected_fails="0" unexpected_passes="0" warnings="0" ↩
        tests="20" errors="0" fails="0" passes="20" />
    <testresult message="Start 'suite_addressbook_py'" result="START" time="2006-05-22T15 ↩
        :27:55" >Test 'suite_addressbook_py' started</testresult>
    <testresult fatals="2424884" expected_fails="0" unexpected_passes="0" warnings="0"  ↩
        errors="46" message="Start 'tst_add_address'" result="START_TEST_CASE" fails="0"  ↩
        time="2006-05-22T15:28:01" passes="13691" >Test Case 'tst_add_address' started</ ↩
        testresult>
    <testresult line="test.py:22" message="VP1: Object property comparison of 'Addressbook. ↩
        ABCentralWidget1.QListView1.childCount' passed" result="PASS" time="2006-05-22T15 ↩
        :28:17" >'1' and '1' are equal (VP1)</testresult>
```

```
    <testresult line="test.py:22" message="VP1: Object property comparison of 'Addressbook. ↩
        ABCentralWidget1.QListView1.item_1.text0' passed" result="PASS" time="2006-05-22T15 ↩
        :28:17" >'Max' and 'Max' are equal (VP1)</testresult>
    <testresult line="test.py:22" message="VP1: Object property comparison of 'Addressbook. ↩
        ABCentralWidget1.QListView1.item_1.text3' passed" result="PASS" time="2006-05-22T15 ↩
        :28:17" >'max@mustermann.net' and 'max@mustermann.net' are equal (VP1)</testresult>
    <testresult line="test.py:22" message="VP1: Object property comparison of 'Addressbook. ↩
        ABCentralWidget1.QListView1.item_1.text2' passed" result="PASS" time="2006-05-22T15 ↩
        :28:17" >'Bakerstreet 55' and 'Bakerstreet 55' are equal (VP1)</testresult>
    <testresult line="test.py:22" message="VP1: Object property comparison of 'Addressbook. ↩
        ABCentralWidget1.QListView1.item_1.text1' passed" result="PASS" time="2006-05-22T15 ↩
        :28:17" >'Mustermann' and 'Mustermann' are equal (VP1)</testresult>
    <testresult message="End 'tst_add_address'" result="END_TEST_CASE" time="2006-05-22T15 ↩
        :28:18" >End of test case 'tst_add_address'</testresult>
    <testresult fatals="2424884" expected_fails="0" unexpected_passes="0" warnings="0"  ↩
        errors="46" message="Start 'tst_add_address_datadriven'" result="START_TEST_CASE"  ↩
        fails="0" time="2006-05-22T15:28:22" passes="13691" >Test Case ' ↩
        tst_add_address_datadriven' started</testresult>
    <testresult line="test.py:29" message="VP1 [addresses.tsv:Number:1]: Object property  ↩
        comparison of 'Addressbook.ABCentralWidget1.QListView1.childCount' passed" result=" ↩
        PASS" time="2006-05-22T15:28:35" >'1' and '1' are equal (VP1 [addresses.tsv:Number ↩
        :1])</testresult>
    <testresult line="test.py:29" message="VP1 [addresses.tsv:First Name:1]: Object  ↩
        property comparison of 'Addressbook.ABCentralWidget1.QListView1.item_1.text0' passed ↩
        " result="PASS" time="2006-05-22T15:28:35" >'Max' and 'Max' are equal (VP1 [ ↩
        addresses.tsv:First Name:1])</testresult>
    <testresult line="test.py:29" message="VP1 [addresses.tsv:Last Name:1]: Object property ↩
         comparison of 'Addressbook.ABCentralWidget1.QListView1.item_1.text1' passed" result ↩
        ="PASS" time="2006-05-22T15:28:35" >'Mustermann' and 'Mustermann' are equal (VP1 [ ↩
        addresses.tsv:Last Name:1])</testresult>
    <testresult line="test.py:29" message="VP1 [addresses.tsv:Address:1]: Object property  ↩
        comparison of 'Addressbook.ABCentralWidget1.QListView1.item_1.text2' passed" result ↩
        ="PASS" time="2006-05-22T15:28:35" >'Bakerstreet 55' and 'Bakerstreet 55' are equal  ↩
        (VP1 [addresses.tsv:Address:1])</testresult>
    <testresult line="test.py:29" message="VP1 [addresses.tsv:E-Mail:1]: Object property  ↩
        comparison of 'Addressbook.ABCentralWidget1.QListView1.item_1.text3' passed" result ↩
        ="PASS" time="2006-05-22T15:28:35" >'max@mustermann.net' and 'max@mustermann.net'  ↩
        are equal (VP1 [addresses.tsv:E-Mail:1])</testresult>
    <testresult line="test.py:29" message="VP1 [addresses.tsv:Number:2]: Object property  ↩
        comparison of 'Addressbook.ABCentralWidget1.QListView1.childCount' passed" result=" ↩
        PASS" time="2006-05-22T15:28:44" >'2' and '2' are equal (VP1 [addresses.tsv:Number ↩
        :2])</testresult>
    <testresult line="test.py:29" message="VP1 [addresses.tsv:First Name:2]: Object  ↩
        property comparison of 'Addressbook.ABCentralWidget1.QListView1.item_1.text0' passed ↩
        " result="PASS" time="2006-05-22T15:28:44" >'John' and 'John' are equal (VP1 [ ↩
        addresses.tsv:First Name:2])</testresult>
    <testresult line="test.py:29" message="VP1 [addresses.tsv:Last Name:2]: Object property ↩
         comparison of 'Addressbook.ABCentralWidget1.QListView1.item_1.text1' passed" result ↩
        ="PASS" time="2006-05-22T15:28:44" >'Kelly' and 'Kelly' are equal (VP1 [addresses. ↩
        tsv:Last Name:2])</testresult>
    <testresult line="test.py:29" message="VP1 [addresses.tsv:Address:2]: Object property  ↩
        comparison of 'Addressbook.ABCentralWidget1.QListView1.item_1.text2' passed" result ↩
        ="PASS" time="2006-05-22T15:28:44" >'Rhodeo Drv. 678' and 'Rhodeo Drv. 678' are  ↩
        equal (VP1 [addresses.tsv:Address:2])</testresult>
    <testresult line="test.py:29" message="VP1 [addresses.tsv:E-Mail:2]: Object property  ↩
        comparison of 'Addressbook.ABCentralWidget1.QListView1.item_1.text3' passed" result ↩
        ="PASS" time="2006-05-22T15:28:44" >'jkelly@acompany.com' and 'jkelly@acompany.com'  ↩
        are equal (VP1 [addresses.tsv:E-Mail:2])</testresult>
    <testresult line="test.py:29" message="VP1 [addresses.tsv:Number:3]: Object property  ↩
        comparison of 'Addressbook.ABCentralWidget1.QListView1.childCount' passed" result=" ↩
        PASS" time="2006-05-22T15:28:54" >'3' and '3' are equal (VP1 [addresses.tsv:Number ↩
        :3])</testresult>
```

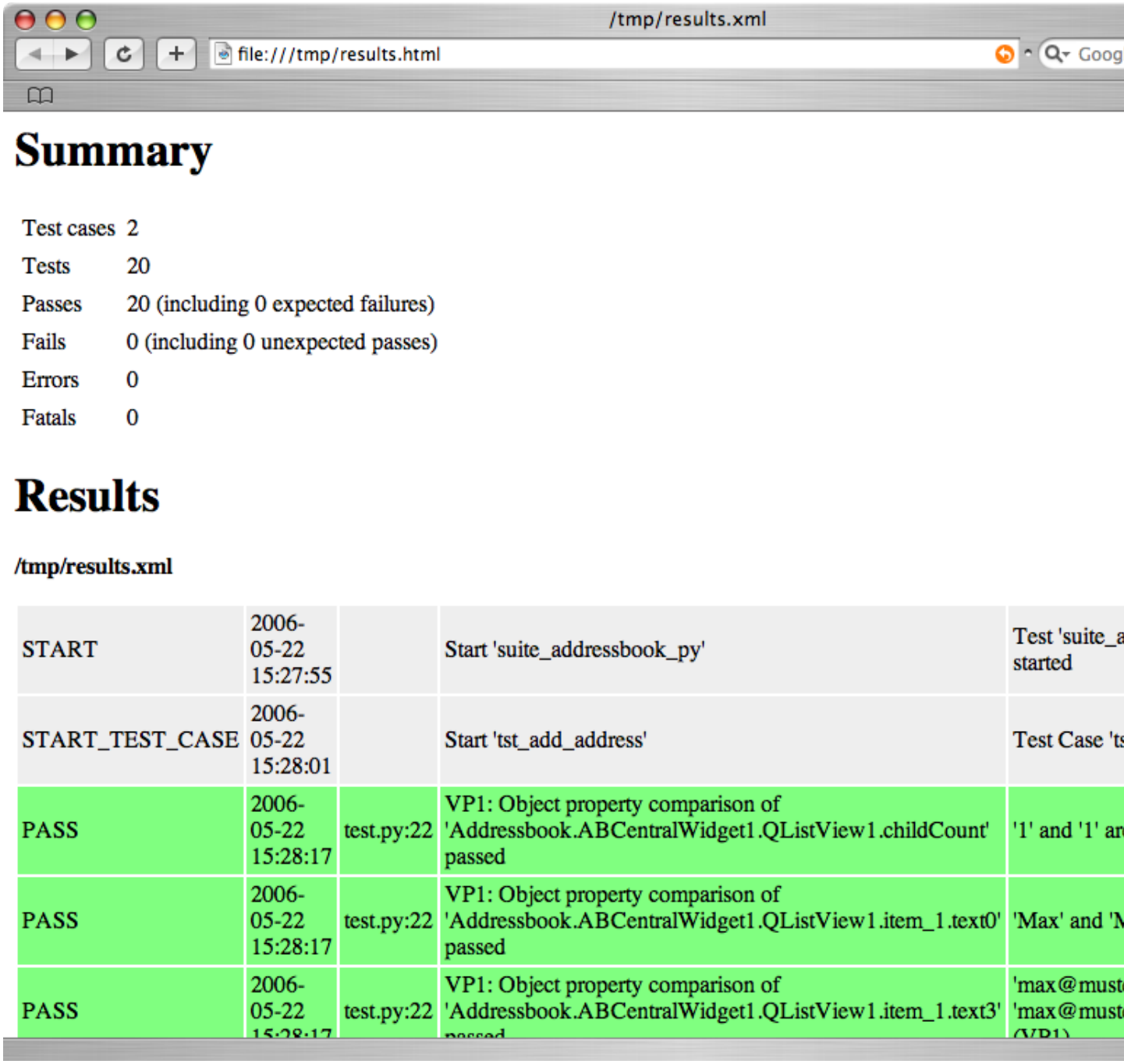
```
    <testresult line="test.py:29" message="VP1 [addresses.tsv:First Name:3]: Object  ↩
        property comparison of 'Addressbook.ABCentralWidget1.QListView1.item_1.text0' passed ↩
        " result="PASS" time="2006-05-22T15:28:54" >'Joe' and 'Joe' are equal (VP1 [ ↩
        addresses.tsv:First Name:3])</testresult>
    <testresult line="test.py:29" message="VP1 [addresses.tsv:Last Name:3]: Object property ↩
         comparison of 'Addressbook.ABCentralWidget1.QListView1.item_1.text1' passed" result ↩
        ="PASS" time="2006-05-22T15:28:54" >'Smith' and 'Smith' are equal (VP1 [addresses. ↩
        tsv:Last Name:3])</testresult>
    <testresult line="test.py:29" message="VP1 [addresses.tsv:Address:3]: Object property  ↩
        comparison of 'Addressbook.ABCentralWidget1.QListView1.item_1.text2' passed" result ↩
        ="PASS" time="2006-05-22T15:28:54" >'Queens Blvd. 37' and 'Queens Blvd. 37' are  ↩
        equal (VP1 [addresses.tsv:Address:3])</testresult>
    <testresult line="test.py:29" message="VP1 [addresses.tsv:E-Mail:3]: Object property  ↩
        comparison of 'Addressbook.ABCentralWidget1.QListView1.item_1.text3' passed" result ↩
        ="PASS" time="2006-05-22T15:28:54" >'joe@smith.com' and 'joe@smith.com' are equal ( ↩
        VP1 [addresses.tsv:E-Mail:3])</testresult>
    <testresult message="End 'tst_add_address_datadriven'" result="END_TEST_CASE" time ↩
        ="2006-05-22T15:28:54" >End of test case 'tst_add_address_datadriven'</testresult>
    <testresult message="End 'suite_addressbook_py'" result="END" time="2006-05-22T15 ↩
        :28:55" >End of test 'suite_addressbook_py'</testresult>
</SquishReport>
```

The complete Python script to convert the XML file to HTML is included in *Squish* under `examples/regressiontesting/xmlresult2html.py`.

We can now run the script to convert the `results.xml` to HTML:

```
xmlresult2html.py /tmp/results.xml > results.html
```

This way the resulting HTML is redirected into the file `results.html` which we can display now in a web browser:



This completes the first step: We can now convert a XML test report to a user presentable HTML file.

6.1.2 Walkthrough the Script Code

You can find the complete Python script in the directory `examples/regressiontesting/xmlresult2html.py`. We just present the most important parts: the `ResultParser` class parses the XML result into a Python struct:

```
class ResultParser:
    """This class parses one XML test result file and stores it in Python
    structures. The attribute summary is a Python dictionary with the summary
    of the test result file. The attribute details is a list of dictionaries
    for the detailed test results."""
    def __init__(self, filename):
```

```
        self.filename = filename
        self.summary = {'testcases':         0,
                        'tests':             0,
                        'passes':            0,
                        'fails':             0,
                        'errors':            0,
                        'fatals':            0,
                        'unexpected_passes': 0,
                        'expected_fails':    0,
                        }
        self.details = []

        # parse the document
        doc = xml.dom.minidom.parse(filename)
        self.__walkNodes(doc)
        doc.unlink()

    def __walkNodes(self, node):
        """This private function walks each node in the document and processes
        the node. The walk is depth first recursive."""
        if node.nodeName == 'summary':
            for attr in self.summary.keys():
                if attr in node.attributes.keys():
                    self.summary[attr] = int(node.getAttribute(attr))
        elif node.nodeName == 'testresult':
            result = {}
            for attr in ('result', 'time', 'line', 'message', 'detail',):
                if attr == 'detail':
                    if node.firstChild:
                        result[attr] = node.firstChild.nodeValue. \
                                       replace('\\n', '\n'). \
                                       replace('\\t', '\t')
                    else:
                        result[attr] = ''
                elif attr == 'time':
                    str = node.getAttribute(attr)
                    dt = str.replace('T', ' '). \
                             replace('-', ' '). \
                             replace(':', ' ').split()
                    if len(dt) == 6:
                        result[attr] = datetime.datetime(int(dt[0]), int(dt[1]),
                                                         int(dt[2]), int(dt[3]),
                                                         int(dt[4]), int(dt[5]))
                    else:
                        result[attr] = ''
                else:
                    result[attr] = node.getAttribute(attr)
            self.details.append(result)
        # walk child nodes
        child = node.firstChild
        while child:
            self.__walkNodes(child)
            child = child.nextSibling
```

The `HtmlConverter` class converts the Python struct to HTML:

```
class HtmlConverter:
    """This class converts a list of test results (as ResultParser objects) to
    HTML. The attribute html contains the HTML version of the converted
    results."""
```

```
    def __init__(self, title, results):
        summary = {'testcases':         0,
                   'tests':             0,
                   'passes':            0,
                   'fails':             0,
                   'errors':            0,
                   'fatals':            0,
                   'unexpected_passes': 0,
                   'expected_fails':    0,
                   }
        detailsHtml = ''
        for r in results:
            # aggregate the keys of the summary
            for attr in summary.keys():
                summary[attr] += r.summary.get(attr, 0)

            # convert the details to HTML
            detailsHtml += htmlResultsBegin % {'filename': r.filename}
            for d in r.details:
                # convert newlines and tab to HTML
                d['detail'] = d['detail']. \
                              replace('\n', '<br>').replace('\t', ' '*4)
                detailsHtml += htmlResults.get(d['result'],
                                               htmlResults['default']) % d
            detailsHtml += htmlResultsEnd

        # put the parts together to one HTML document
        self.html  = htmlHeader % {'title': title} + \
                     htmlSummary % summary + \
                     detailsHtml + \
                     htmlFooter
```

The above code uses some variables with the HTML code for certain parts of the resulting document. For example the variable `htmlResults['PASS']` contains the HTML code for one test result entry that passed:

```
htmlResults['PASS'] = """\
<tr bgcolor="#80FF80">
    <td>%(result)s</td>
    <td>%(time)s</td>
    <td>%(line)s</td>
    <td>%(message)s</td>
    <td>%(detail)s</td>
</tr>
"""
```

The code contains placeholders like `%(result)s` that are filled with Python's `%` operator. Here are all variables that are used:

```
# The following HTML snippets can be modified to customize the look of the
# resulting page. The script substitutes the "%(title)s" parts with the data
# from the XML testresults.
htmlHeader = """\
<html>
    <head>
        <title>%(title)s</title>
    </head>
    <body>
"""
```

```
htmlFooter = """\
    </body>
</html>
"""

htmlSummary = """\
<h1>Summary</h1>
<table border="0" cellpadding="3">
    <tr>
        <td>Test cases</td><td>%(testcases)s</td>
    </tr>
    <tr>
        <td>Tests</td><td>%(tests)s</td>
    </tr>
    <tr>
        <td>Passes</td><td>%(passes)s (including %(expected_fails)s
                          expected failures)</td>
    </tr>
    <tr>
        <td>Fails</td><td>%(fails)s (including %(unexpected_passes)s
                         unexpected passes)</td>
    </tr>
    <tr>
        <td>Errors</td><td>%(errors)s</td>
    </tr>
    <tr>
        <td>Fatals</td><td>%(fatals)s</td>
    </tr>
</table>
<h1>Results</h1>
"""

htmlResultsBegin = """\
<p><b>%(filename)s</b></p>
<table border="0" cellpadding="3" width="100%%">
"""

htmlResultsEnd = """\
</table>
"""

htmlResults = {}
htmlResults['PASS'] = """\
<tr bgcolor="#80FF80">
    <td>%(result)s</td>
    <td>%(time)s</td>
    <td>%(line)s</td>
    <td>%(message)s</td>
    <td>%(detail)s</td>
</tr>
"""
htmlResults['XFAIL'] = htmlResults['PASS']
htmlResults['FAIL'] = """\
<tr bgcolor="#FF8080">
    <td>%(result)s</td>
    <td>%(time)s</td>
    <td>%(line)s</td>
    <td>%(message)s</td>
    <td>%(detail)s</td>
</tr>
"""
htmlResults['XPASS'] = htmlResults['FAIL']
```

```
htmlResults['ERROR'] = htmlResults['FAIL']
htmlResults['FATAL'] = htmlResults['FAIL']
htmlResults['default'] = """\
<tr bgcolor="#EEEEEE">
    <td>%(result)s</td>
    <td>%(time)s</td>
    <td>%(line)s</td>
    <td>%(message)s</td>
    <td>%(detail)s</td>
</tr>
"""
```

The remaining part of the script is used when the script was called from the commandline (as opposed to being included with the `import` statement). It just passes all the XML files given on the commandline to the `ResultParser` class and converts them to HTML with the `HtmlConverter` class:

```
if __name__ == '__main__':
    """The following code is executed if this script was called from the
    commandline."""

    import sys
    # open input file
    if len(sys.argv) < 2:
        raise RuntimeError, 'Missing a file name argument'

    # parse the XML files
    title = ''
    results = []
    for filename in sys.argv[1:]:
        title += filename + ' '
        results.append(ResultParser(filename))

    # convert the result to HTML
    converter = HtmlConverter(title, results)
    print converter.html.encode('utf-8')
```

6.2 Automatically Running Tests

The next step is to automate running the tests and automatically call the script to convert the XML reports into HTML. Also, an index page should be generated, so a test engineer can view the index page and click on links to see the full reports.

For this we create a script `runtests.py` which takes over this task. The script is printed below and can also be found in `examples/regressiontesting/runtests.py`.

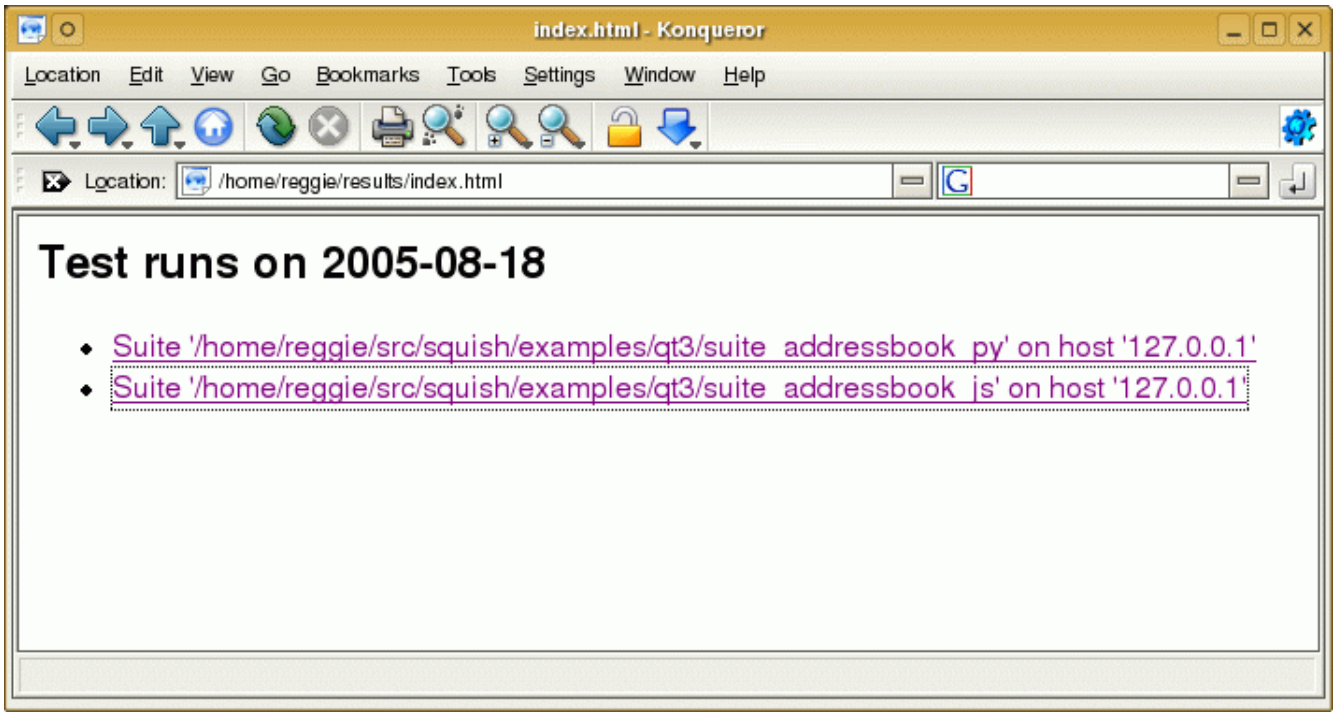
```
#! /usr/bin/env python

# import necessary modules
from os import system
from os import path
from datetime import date


# these variables need to be adjusted:
# bin directory of Squish
squishbin = "/home/reggie/src/squish/bin/"
# directory to which the HTML results should be written
```

```
outdir = "/home/reggie/results"
# list of test suites to execute
suites = ["/home/reggie/src/squish/examples/qt3/suite_addressbook_py",
          "/home/reggie/src/squish/examples/qt3/suite_addressbook_js"]
# list of hosts on which the suites should be executed
hosts = ["127.0.0.1"]


# this functions runs the test suite 'suite' on the host 'host' and converts the
# XML result into HTML and saves that to 'htmloutput'
def runTest(suite, host, htmloutput):
    print("run " + suite + " on " + host)
    print(squishbin + "squishrunner --host " + host + " --testsuite " + suite + " -- ↩
        reportgen xml,/tmp/results.xml")
    system(squishbin + "squishrunner --host " + host + " --testsuite " + suite + " -- ↩
        reportgen xml,/tmp/results.xml")
    system(squishbin + "../examples/regressiontesting/xmlresult2html.py /tmp/results.xml >" ↩
         + htmloutput)


index = "<h2>Test runs on " + str(date.today()) + "</h2>\n<ul>\n"

# loop over all test suites and hosts and call 'runTest'
for s in suites:
    sn = path.basename(s)
    for h in hosts:
        file = outdir + "/" + str(date.today()) + "_" + sn + "_" + h + ".html"
        runTest(s, h, file)
        index += "<li><a href=\"" + file + "\">Suite '" + s + "' on host '" + h + "'</a>\n"

index += "</ul>\n"


# write links to the generated HTML reports into the index.html so one
# can view a summary of all results
if path.isfile(outdir + "/index.html"):
    ifile = open(outdir + "/index.html", "r")
    index += ifile.read()
    ifile.close()

ifile = open(outdir + "/index.html", "w")
ifile.write(index)
ifile.close()
```

The comments in the script explain the code quite well. Basically one defines a list of test suites and hosts and then, using a loop, the test suites are run on all hosts using squishrunner. The resulting XML file is automatically converted into HTML. The file names for the HTML result contain the date, test suite and host to be unique.

When running this script it is assumed that on all specified hosts squishserver is running, and the AUT paths are set up so the needed AUTs for the test suites can be found.

Here is the generated index page from one test run:



The last step would be to have the `runtests.py` script ran automatically for example once every night to ensure that no regressions have been introduced into the AUT. On Unix this can be done by setting up a cron job which executes the script. On Windows it is possible to set up a Windows Service to run the script regularly.

For detailed documentation about cron jobs or Windows Services, search for the relevant information on the Internet or contact *froglogic*'s commercial support to assist you.

6.3 Conclusion

This tutorial showed how to set up automated test runs using *Squish*'s command line tools and two simple Python scripts. Of course a lot more can be done in the way how results are presented, statistics are calculated, etc. By interpreting the XML results all this is possible and using the information provided in this tutorial you should be able to improve the scripts to satisfy your needs

Chapter 7

User's Guide

The Squish User's Guide explains all aspects and features of *Squish* in detail.

7.1 Scripting

This chapter discusses Squish's scripting support, the different supported scripting languages and the script APIs which are available when working with test scripts.

7.1.1 Accessing Java™ API

One of *Squish*'s most useful features is the ability to access the toolkit's API from test scripts. This gives test engineers an amount of flexibility allowing to test about anything in the AUT.

With *Squish*'s Java™-specific API it is possible to find and query objects, access fields and methods. When we talk about properties, we mean fields in Java™ classes combined with member functions that have a

```
Type getSomething();
boolean isSomething();
void setSomething( Type t );
```

pattern. In the above code the property would have been called `something`. When both `getXxx` and `isXxx` are there, then *Squish* will use the `getXxx` variant. The existence of `getFoo`/`isFoo` and/or `setFoo` sets whether the property is read-only or read-write, write-only properties are not generated by *Squish*.

In addition, *Squish* provides a Section 7.1.1.4 to execute common actions on GUI applications such as clicking a button or entering some text.

The chapter Section 7.1.6 later in this manual shows using different examples how to use the scripting Java™ API to access and test complex Java™ GUI elements.

7.1.1.1 Finding and Querying Java™ Objects

Squish provides a function called `findObject` which returns the object for a given qualified object name.

There are two notations for an object qualified name:

- Multiple properties: A list of property/value pairs in curly braces defines the object. *Squish* will search the GUI parent child hierarchy until it finds a matching object. Example: `{type='javax.swing.JButton' caption='*' windowType='Calculator' windowName='frame0' windowCaption='Java Swing Calculator'}`
- Hierarchical notation: From the top Frame (or Shell in SWT) the path to the object, all parent GUI elements, are noted and separated by a dot. Example: `:frame0.JRootPane.null_layeredPane.null_contentPane.JLabel`.

To find out the name of an object, you can use the Spy to introspect the document of the Web application. See the chapter Section 7.8.4 for details.

Squish prefers the multiple property notation where ever possible. This notation has the advantage that scripts wont break if you need another layer (like a `JPanel`) in between somewhere.

The multiple property notation may be followed by a hierarchical notation.

As described in Section 7.4, the multiple property notation names automatically get a shorter name in the 'Object map'. In the above example that shorter name could be `:frame0.*_javax.swing.JButton`. You may use either the short or original name to reference the object.

Querying on object using `findObject` in Python you would use the following code:

```
multiply = findObject(":frame0.*_javax.swing.JButton")
```

If `findObject` can't find the specified object, a script error is thrown which stops the script execution. In some situations it might be desirable to first check if the object exists and execute an action on the object. For this purpose, *Squish* provides an `object.exists` function.

As example, we want to find the radio button like above, and click it if it exists. In Python this functionality would be implemented like this:

```
if object.exists(":frame0.*_javax.swing.JButton"):
    multiply = findObject(":frame0.*_javax.swing.JButton")
    clickButton(multiply)
```

This allows test engineers to query and access the complete object tree of the Web document.

7.1.1.2 Calling Functions on Java Objects

With *Squish* it is possible to call every public function on any Java object. Section 7.1.1.1 describes how objects can be found. The following example shows how you can create an object in JavaScript:

```
var s = new java_lang_String("abc");
```

Note
The notation of Java™ objects are with a '_' instead of '.' for package directories separation. This is because a '.' has a meaning in most of the script languages that *Squish* support.

In the example below we change the button text of the multiply button of the calculator demo application. In JavaScript this can be written as:

```
var multiply = findObject(":frame0.*_javax.swing.JButton");
multiply.setText( "x" );
```

Static functions can also be called. The next example shows a JavaScript example of calling the static `Integer.parseInt-(String s)`:

```
var i = java_lang_Integer.parseInt("12");
```

7.1.1.3 Accessing Properties of Java™ Objects

Java™ objects can have fields. Public fields are accessible in *Squish* like in the next JavaScript example:

```
var p = new java_awt_Point( 5, 8 );
print( p.x );
```

In addition to public fields, *Squish* adds synthetic properties derived from method names with the `Type getSomething()`, `boolean isSomething()` and `void setSomething(Type t)` pattern (see Section 7.1.1 for details). In the example where we changed the button text with `setText("x")`, we could have written in JavaScript:

```
var multiply = findObject(":frame0.*_javax.swing.JButton");
multiply.text = "x";
```

And *Squish* will call the corresponding `setText("x")` for us.

Note
These extra generated properties allows you to add more verifications points in the test scripts, see Section 7.10 for more information.

Squish knows about a limited set of classes. If you get errors accessing a class method, referring to a super class, then it is likely this class is not wrapped by *Squish*. See Section 7.2.7 how to extend the set of known classes.

7.1.1.4 Java™ Convenience API

This section describes the script API *Squish* offers on top of Java™'s API to make it easy to perform common user actions such as clicking a button, entering text, etc. A complete list of this API is available in the chapter Section 8.1.3 in the Chapter 8. Here we will show some examples.

In the example below, we click on a button, type some text and activate a menu. As mentioned, there are more convenience functions, but the examples below should give a good overview of how it works.

Here is the Python code for those examples:

```
clickButton(":frame0_Notepad$1")
type(":frame0_javax.swing.JTextArea", "Some text")
activateItem(":frame0_javax.swing.JMenuBar", "File")
activateItem(":frame0.File_javax.swing.JMenu", "Exit")
```

In Perl or JavaScript this looks like this:

```
clickButton(":frame0_Notepad$1");
type(":frame0_javax.swing.JTextArea", "Some text");
activateItem(":frame0_javax.swing.JMenuBar", "File");
activateItem(":frame0.File_javax.swing.JMenu", "Exit");
```

7.1.2 Test Statements

This section discusses the API *Squish* offers to perform tests which will create test results. Verification points use this test API. Different approaches and topics on verification points are discussed in the chapter Section 7.10. Working with the test result log is discussed in the chapter Section 7.7.3.

To compare two values and write the result to the test log, the `test.compare` is used. To just verify a boolean value, `test.verify` is used. To write some information to the test log, the `test.log` function can be used. If you want to emphasize a message, you might want to mark it as a warning message, which can be done with the `test.warning` function.

Here is a small example for Python:

```
lineedit = findObject("Addressbook.ABCentralWidget1.FirstName")
test.verify(lineedit.enabled)
test.compare(lineedit.text, "Max")
test.log("Important note", "This is an important note about the test")
test.warning("Suspicious warning", "This test is incomplete and should be extended!")
```

In JavaScript we of course can do the same:

```
lineedit = findObject("Addressbook.ABCentralWidget1.FirstName");
test.verify(lineedit.enabled);
test.compare(lineedit.text, "Max");
test.log("Important note", "This is an important note about the test");
test.warning("Suspicious warning", "This test is incomplete and should be extended!");
```

In Tcl it works too:

```
set lineedit [findObject "Addressbook.ABCentralWidget1.FirstName"]
test verify [property get $lineedit enabled]
test compare [property get $lineedit text] "Max"
test log "Important note" "This is an important note about the test"
test warning "Suspicious warning" "This test is incomplete and should be extended!"
```

There are of course more test commands available. A complete listing and API documentation can be found in the chapter Section 8.1.2 in the Chapter 8.

7.1.3 Event Handlers

In *Squish* test scripts it is also possible to react on events in your AUT. For example, this is useful to react on unexpected dialogs popping up such as error message boxes.

The concept is to specify an event handler function for a certain event on a certain object, all objects or object type. The function to do this is called `installEventHandler`.

In this chapter we will give the code examples only in one script language. As we will discuss three different types of event handlers, there will be a code example in each script language.

A complete listing and API documentation of event handlers can be found in the chapter Section 8.1.2 in the Chapter 8.

7.1.3.1 Global Events

A global event is for example the event `MessageBoxOpened` which is sent when a message box opens. This can be used to e.g. react on unexpected message boxes which pop up. The event handler below will log the message box's text in the test results and then close the message box so the test can continue. In Python this would look like this:

```
def handleMessageBox(msgBox):
    label = msgBox.child("messageBoxText")
    label = cast(label, QLabel)
    text = label.text
    caption = msgBox.caption
    test.log("Message box '" + str(caption) + "' opened", str(text))
    msgBox.close()

def main():
    installEventHandler("MessageBoxOpened", "handleMessageBox")
```

Another special event is `Crash`. This way it is possible to install an event handler which is called when the AUT crashes to do e.g. cleanups or restart the AUT. In JavaScript, this would look like:

```
function handleCrash()
{
    test.log("Deleting lock files after application crash");
    deleteLockFiles();
}

function main()
{
    installEventHandler("Crash", "handleCrash");
}
```

A third special event are `Timeout` events. These events are triggered whenever the AUT fails to respond to some *Squish* command within five minutes. This can happen if the application got stuck in an endless loop, or if there is some other reason which keeps it from being able to respond. You can install an event handler for this event in order to handle this event gracefully.

7.1.3.2 Events for Object Types

It is also possible to react on specific events on a specified object type. For example, we can install an event handler which is always called when a `QMouseEvent` occurs on a `QCheckBox`. This means, every time the test performs e.g. a click on any check box in the application, the event handler would be called. In JavaScript we could implement it as follows:

```
function handleCheckBox(obj) {
    test.log("QCheckBox clicked", "check box '" + objectName(obj) + "' clicked")
}

function main() {
    installEventHandler("QCheckBox", "QMouseEvent", "handleCheckBox");
}
```

7.1.3.3 Object Events

The third kind of event handling is to react on events on a specific instance of an object. As an example we could install an event handler which is called every time a line editor receives a `QKeyEvent`. This means the event handler will be called every time the test types some text into the line editor. In Tcl we would implement it like this:

```
proc handleKeyEvent {obj} {
    set firstName [property get $obj text]
    test log "First Name Changed" "First Name changed to $firstName"
}

proc main {} {
    set lineedit [findObject "Addressbook.ABCentralWidget.FirstName"]
    installEventHandler $lineedit QKeyEvent handleKeyEvent
}
```

7.1.4 Synchronization Points

When recording a script in *Squish*, the event recorder automatically inserts `snooze` statements into the script. Those statements make the script to wait for a specified amount of seconds. This is important to ensure that a script is replayed in the same speed as it was recorded. So if the user waited for e.g. a window to pop up, the script will do the same.

Using `snooze` statements is the simplest way of synchronizing the AUT and the script. But in many cases, just waiting for a certain amount of time isn't sufficient. E.g. when a script is recorded on a fast machine and later replayed on a slow computer a `snooze` might wait not long enough.

Therefore, in the record settings dialog, you can choose to not insert `snooze` statements but `waitForObject` statements instead. This way, before every action recorded, a `waitForObject` statement will be recorded for the object to be accessed. So on replay, *Squish* will then, instead of just waiting for a given amount of time, wait for the given object to exist and be accessible.

Alternatively, *Squish* offers a `waitFor` function. This function waits until a given condition becomes true, or optionally, until a specified time out expired.

The condition can be anything from a property to a complex script statement. The following Python code would wait until the dialog "Addressbook.FileSave" pops up. In case it doesn't pop up after 5 seconds, an error will be thrown:

```
ok = waitFor("object.exists('Addressbook.FileSave')", 5000)
if not ok:
    test.fatal("Dialog Addressbook.FileSave didn't pop up'")
```

The following JavaScript snippet will wait until the file "addresses.addr" exists in the AUT's directory (which is the current working directory). Since no timeout is specified, this call will wait forever in case the file gets never created:

```
waitFor("QFile.exists('addresses.tsv')");
```

The following Tcl code will wait for a maximum of 2 seconds for the Add button to become disabled:

```
set addButton [findObject "Addressbook.ABCentralWidget1.AddButton"]
set ok [waitFor {property get $addButton enabled} 2000]
if {$ok == false} {
    test fatal "Add button still enabled after 2 seconds"
}
```

The examples above have shown different variations of synchronization points. As the condition which is passed to the `waitFor` function can be any script code which can be evaluated, there are no limits to synchronization points.

Event-Driven Test

By combining the `waitFor` function and Section 7.1.3 it is possible to wait with the script execution until a specific event has been handled. This can be done by installing an event handler, setting a global variable (e.g. `eventHandled`) to false and passing this global variable as condition to the `waitFor`. In the event handler, just set `eventHandled` to true and you are done.

This way it is event possible to create completely event driven tests, where the main function of the test just installs event handlers and starts a `waitFor` and all testing is done in the event handlers.

7.1.5 Testing Multiple AUTs from a Single Test Script, Working with ApplicationContext

Usually, as also shown in the tutorials, one application under test is specified for a test suite. This AUT is then executed and accessed by each test case. In *Squish* it is also possible to start multiple applications and access and test all of them. This allows to test the interaction between different applications or multiple instances of the same application which is necessary to test client/server systems.

For each started application, an `ApplicationContext` object is created which is used as a handle to the application. Using the application context it is also possible to query information such as the command line, the running state, etc. Using a default application context it is also possible to query such information from a single AUT. So this information is also interesting for single-AUT tests.

7.1.5.1 Starting and Accessing Multiple AUTs

When testing multiple applications from one test script, the first step is to specify no application in the test suite settings of the test suite. To do that in the Squish IDE open Test Suite → Settings and go to the page Settings. Under AUT simply choose <No AUT>.

The script command to start an application is `startApplication`. This command starts the given application (assuming it is located in an application path- see Section 7.2) using the given command line arguments and returns its *context*. The context is a handle which refers to the application.

Optionally, as second and third parameter a host and port can be passed to `startApplication`. This way, `startApplication` will connect to the squishserver on the specified host listening to the specified port instead of using the default host and port (as specified in the Squish IDE's settings or squishrunner's command line). This allows to control multiple applications on multiple computers from one test script.

Special care has to be taken if the application is using a different GUI toolkit than the default toolkit of the test suite. The testSettings object allows for a configuration of the toolkit wrapper on a per-AUT basis. See Section 8.1.2.11 for usage of the setWrappersForApplication() function.

Using `setApplicationContext` the currently active application can be set. This means, that script function calls will be executed in this currently active application. When calling `startApplication`, the current context is automatically set to the newly started application.

The function `applicationContextList` returns a list of all application contexts. Using `defaultApplicationContext` the context of the default application (the one specified in the test suite settings), if there is one, is returned. . Using `currentApplicat` the current context is returned.

Note
If you want to record and access applications which are started by the AUT and not by *Squish* please see [?].

The next section will discuss the usage of `ApplicationContext` objects in more detail. Now we will look at an example of working with multiple applications in a test script.

Suppose we have an chat client/server system with an application called `chatserver` which needs to be running for the communication and two clients called `chatclientqt` and `chatclientjava` which can be used for chatting. In the test we first start the chat server. Then we start two clients which automatically connect to the chat server. We then type something in the message editor of the first client and check that the second client received the message. We will use Python for the test. The comments in the program code give some more details:

```
# start server so clients can connect to it
startApplication("chatserver")

# start first client, using Qt toolkit
client1 = startApplication("chatclientqt","Qt")

# start second client, using Java toolkit
client2 = startApplication("chatclientjava","Java")

# need to switch to client1 to be able to access it
setApplicationContext(client1)

# type something into the message editor
editor = findObject("ChatWindow.messageEditor")
type(editor, "Message for client 2")

# switch context to client2 to be able to verify
# that the message was received
setApplicationContext(client2)

# check that the message was received
msgView = findObject("ChatWindow.messageView")
test.compare(msgView.text, "Message for client 2")
```

7.1.5.2 Working with ApplicationContext

Using the `ApplicationContext` object information about the application it refers to can be retrieved. The application context of the running AUT as defined in the test suite settings can be retrieved using the function `defaultApplicationContext`. When starting an application using `startApplication`, this function returns the application context for the newly started AUT.

A complete list of properties and functions provided by the `ApplicationContext` object can be found in the chapter Section 8.1.2 in the Chapter 8. Here are some examples.

To print out the command line and current working directory of the test's default application the following Python code can be used:

```
ctx = defaultApplicationContext()
print(ctx.commandLine)
print(ctx.cwd)
```

To start the application `myapp` and calculate its maximum memory usage as long it is running, we could use the following JavaScript code (*The `memoryUsage` property is only available in conjunction with the Squish memory module add-on*):

```
ctx = startApplication("myapp");
var maxcpu = 0;
while(ctx.isRunning) {
    maxcpu = Math.max(ctx.usedMemory, maxcpu);
}
```

To print out the process identifier of the application `myapp` and to check for a maximum time of 10 seconds if the application is responsive (not frozen), the following Tcl code would work:

```
set ctx [startApplication "myapp"]
puts [applicationContext $ctx pid]
set frozen [applicationContext $ctx isFrozen 10]
test compare $frozen false
```

To add everything which has been written to STDOUT and STDERR by the application to the test log, classifying all STDERR messages as warnings, the following Python code would work:

```
ctx = startApplication("myapp")
test.log("STDOUT", ctx.readStdout())
test.warning("STDERR", ctx.readStderr())
```

7.1.6 Testing Java™ applications

This chapter shows using small example code snippets in JavaScript how to test specific Java widgets in an application and verify that their properties have the expected values and the widgets contain the expected contents.

The hardest part when implementing test scripts is to create the test verifications. As shown in the chapter Section 5.3 and in the tutorial Chapter 5, most of that can be done using the Spy and its point & click interface. But in some cases it might be most desirable to implement a verification point using script code to gain more flexibility.

Often the biggest issue there is to find out how to retrieve the information from widgets which should be tested. These examples show how to do this on different widgets of different complexity. Using the principles shown in this chapter it should be possible to test any widget in your application under test.

7.1.6.1 Accessing widgets

To test and verify a widget and its properties or contents, first we need access to the widget in the test script. To obtain a reference to the widget, the `findObject` function is used. This functions locates a widgets of the given name and returns a reference to it.

For this purpose we need to know the name of the widget we want to test. To find out the name of a widget the Spy tool comes in very handy (for more details about using Spy see Section 7.8.4.

The steps to find out the name are the following:

- Start the Squish IDE and make the test suite we are working in active
- Start the Spy on the application under test
- Switch to the AUT and work through the GUI until the widget we want to test is visible (e.g. open the dialog it is contained in)
- Switch back to the Squish IDE and switch the Spy into Pick mode
- Switch to the AUT and click on the widget you want to test
- Switch back to the Squish IDE. In the Spy object view the selected widget and its tree will be displayed. Right-click onto the object name and choose "Copy to clipboard".
- Exit the Spy

Now the object name we were looking for is saved in the clipboard and we can paste it into the script as the argument to `findObject`.

For more details about the `findObject` functions, see Section 7.1.1.1.

In the following sections we always assume that we have access to the widget we want to test.

7.1.6.2 Testing Widget States

Very often the state of a widget, such as its visibility, needs to be tested. Those states can be queried on a widget by testing the value of its respective properties. Here are some examples in the various scripting languages that *Squish* supports:

Verify that the widget has the input focus in Python:

```
test.compare(widget.focusowner, True)
```

Verify that the widget is visible in JavaScript:

```
test.compare(widget.visible, true);
```

Verify that the widget is enabled in Tcl:

```
test compare [property get $widget enabled] true
```

7.1.6.3 Testing CheckBox

A common test is to verify that the state of a checkbox reflects the expected settings.

7.1.6.3.1 AWT CheckBox

This can be tested by querying the value of the checkboxes `state` property in JavaScript:

```
test.compare(checkbox.state, true);
```

7.1.6.3.2 Swing JCheckBox

This can be tested by querying the value of the checkboxes `selected` property in JavaScript:

```
test.compare(checkbox.selected, true);
```

7.1.6.3.3 SWT Button of type `CHECK` or `RADIO`

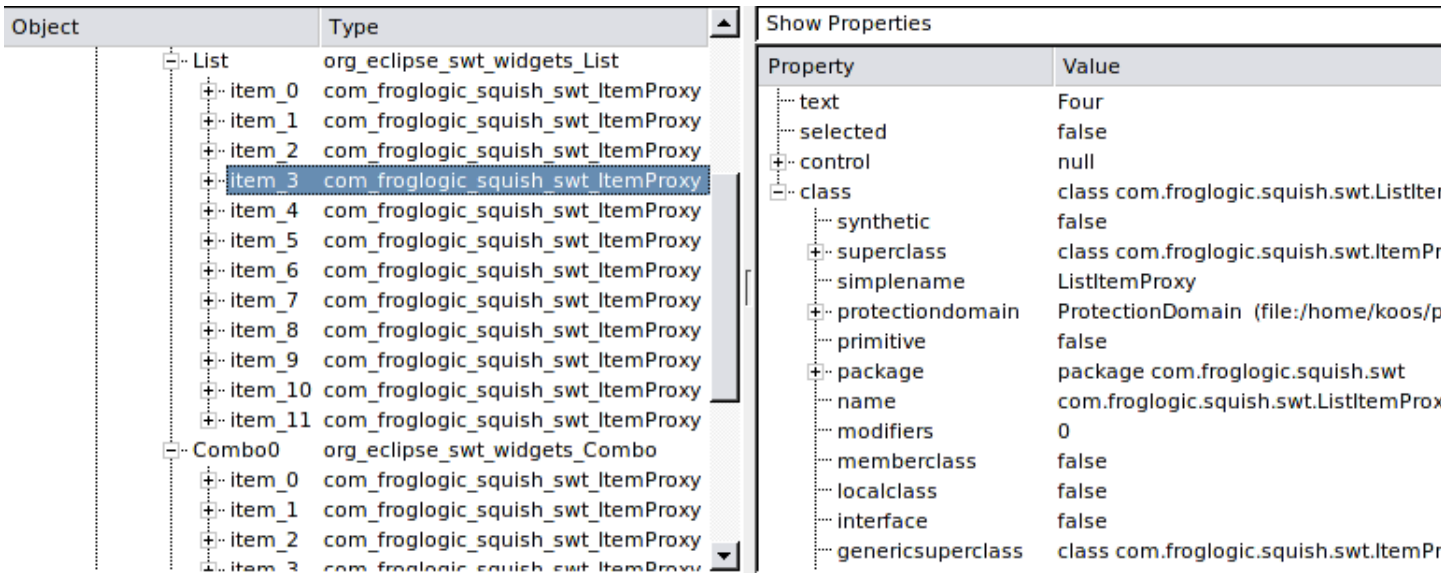
This can be tested by querying the value of the checkboxes `selection` property in JavaScript:

```
test.compare(checkbox.selection, true);
```

7.1.6.4 Testing List and ComboBoxes

Although the different GUI Java™ toolkits that *Squish* supports have different implementations for list and comboboxes, their items are wrapped by *Squish* in `ItemProxy` objects. These objects act as child objects of the list or combobox object. The child objects are called `item_0`, `item_1` and so on.

The `ItemProxy` objects have the properties `text` and `selected`. For the SWT based List and Combo widgets, there is an extra property called `control`. The AWT/Swing based ones have an extra property called `component`.



In the spy you get for each item an ItemProxy derived object.

In SWT, the List and Combo widget have the method `getSelectionIndex()`. *Squish* generates from this method a property calls `selectionindex`. So for the SWT toolkit, these two tests are equal:

```
test.compare(list.selectionindex, 2);
test.compare(list.item_2.selected, true);
```

Note however that the second test gives a script error if the list has less then three items. And that the first test only gives one of the selections if the list supports multiple selections. In that case you can only use the generated `ItemProxy` objects for verification points.

7.1.6.5 Testing GEF applications

Squish exposes Figures and FigureCanvas from the GEF (Graphical Editing Framework) library. The Figure hierarchy of the FigureCanvas is exposed and can be inspected in the spy. Properties of the Figure items can be tested like this:

```
item = findObject("<name of Figure item>")
test.compare(item.visible, true);
test.compare(item.enabled, true);
```

7.1.7 Semi-Automatic Tests: Querying User Input

While *Squish* allows to automate the GUI testing, in some cases it might still be of interest to have testers to input some data. E.g. when testing a software which works together with a hardware device, it might be necessary to ask a user when running the test if the device's state has been changed using the software as expected.

Let's assume we create a test for a printer software. The tests exercises the software and one of the results should be that the printer printed a page with the contents "This is a test."

Since *Squish* can't verify that, we could put the following code into the Python test script so the person running the test has to confirm that the page was printed correctly:

Note
The code below uses Qt API (QMessageBox) to display a message box. For editions other than Squish for Qt, different APIs (such as a JavaScript alert box or a Tk message box displayed using `evalJS` or `tcleval` could be used.

```
def main():
    ....

def main():
    ok = QMessageBox.question(0, "Require User Input",
                              "Did the printer print a page with the contents 'This is a  ↩
                                  test'?",
                              "Yes", "No") == 0
    if ok:
        test.passes("Printing worked", "Printer printed test page correctly")
    else:
        test.fail("Printing failed", "Printer didn't printed test page")
    ....
```

Another example would be to query the tester for the number of pages printed. In JavaScript this can be done like this:

```
function main()
{
    ....
    var num = QInputDialog.getInteger( "Require User Input", "How many pages have been  ↩
        printed?" );
    if ( num == 0 ) {
        test.fail( "No pages printed");
    } else {
        test.pass(num + " pages printed");
    }
    ....
}
```

The same in Tcl:

```
proc main {} {
    ....
    set num [invoke QInputDialog getInteger "Require User Input" "How many pages have been  ↩
        printed?"]
    if {$num == 0} {
        test fail "No pages printed"
    } else {
        test pass "$num pages printed"
    }
    ....
```

This section showed again that using *Squish*'s powerful script bindings classes like QMessageBox and QInputDialog (or the respective DOM/HTML, Tk or XView APIs depending of the used Squish edition) can be used in test scripts to allow querying for user input.

7.1.8 Automatic Screenshots on Test Failures and Errors

To make tracing of test failures and errors easier, it is possible to instruct *Squish* to take a screenshot on test failures or test errors. This way it is possible to look at a screenshot of the complete desktop taken at the time the error occurred. This is especially helpful for debugging of test failures and errors of automated, unattended test runs.

To enable this feature, it is necessary to set the test script properties `logScreenshotOnFail` and/or `logScreenshotOnError` of the global `testSettings` object to value indicating a True boolean value in your scripting language.

Once this property is activated, all test results of following fails or errors will contain a path to image file that contains the saved screenshot. In the Squish IDE the screenshots can be viewed directly by right-clicking on the respective test result item.

See Section 8.1.2.11 for the language-specific syntax.

7.2 AUTs and Settings

This chapter discusses many aspects of applications under test, how they are located and can be specified and special settings.

7.2.1 AUT Class Name and Classpath for Java™

A Java™ application can be configured in the testuite settings dialog. There are three different paths one can take to set the AUT.

Leave the Application as `<No Application>` and put the class name containing the `main` in the Main Class edit box.

Alternatively, a `.jar` archive can be set in the application field provided that the archive lists the main class in a suitable `MANIFEST` included within. Please note however that SWT based applications will not work with this setup and it is therefore advisable not to use this method, Instead add the .jar archive to the `CLASSPATH` and set the main class name as described above.

And finally one can set a script or executable as AUT in the Application field. Testing RCP applications however, require only to set the launcher executable in this field.

Next section goes deeper in the subject of setting the AUT this way.

`CLASSPATH` can be configured in the Classpath edit box or outside of *Squish* as environment variable. Do not use any quotes, *Squish* get confused if paths are quoted.

Note
Use either a `CLASSPATH` environment variable set outside of *Squish* or use the Classpath edit box.

7.2.2 AUT Paths and Mapped AUTs

In *Squish* the application under test is not specified with its absolute path in the test suite. The reason for this is, that the test runner (squishrunner) doesn't directly execute the AUT, but that the squishserver will be instructed to start the AUT.

squishserver can be running on multiple machines where the AUT is located in different location. Therefore squishrunner only passes the name of the AUT's executable to squishserver and it is squishserver's responsibility to locate the executable and start it.

There are two different approaches to let squishserver know where an executable is located:

7.2.2.1 AUT Path

The AUT Path is a variable in squishserver's settings which specifies a list of paths. When starting an AUT, squishserver searches in all of the specified paths for the executable and as a matching executable can be found, it will be started.

To add an path to squishserver's AUT paths, call

```
squishserver --config addAppPath <path>
```

To remove a path again, call

```
squishserver --config removeAppPath <path>
```

The same can be done in the Squish IDE's test suite settings dialog under Test Suite → Settings. To add an AUT path, click on the Manage... button in the General tab. In the Manage AUTs dialog, first select the AUT Paths tree item, and then click the Add... button and choose the directory you want to add. To remove a path, select the path to be removed and click the Remove button.

The test suite settings dialog also shows all executables which squishserver can find and could start.

7.2.2.2 Mapped AUT

Another approach is to directly map the name of an AUT to a given path. So when squishserver looks for the path of an executable, it looks if a path is mapped to it and starts it directly without searching all AUT paths.

This way it is possible to make sure that the correct application will be started and it can't happen that accidentally a wrong executable with the same name in a different path is picked up.

To map the application `myaut` to the path `/home/squishuser/bin/`, call

```
squishserver --config addAUT myaut /home/squishuser/bin
```

This way, if squishserver is instructed to execute the AUT `myaut`, it will not search all AUT paths for the executable but directly start `/home/squishuser/bin/myaut`.

```
squishserver --config removeAUT myaut /home/squishuser/bin
```

The same can be done in the Squish IDE's test suite settings dialog under Test Suite → Settings. To add an AUT directly mapped to a path, click on the Manage... button in the General tab. In the Manage AUTs dialog, first select the Mapped AUTs tree item, and then click the Add... button. Now select the application, and the mapping between the selected AUT and its path will be added to squishserver. To remove a mapping, select the AUT and click the Remove button.

The test suite settings dialog also shows all executables which squishserver can find and could start. In there it is also visible if an application has been mapped to a path or if it was found in an AUT path.

7.2.3 Settings Groups

Another concept offered by *Squish* are settings groups. This way it is possible to save specific settings (such as an AUT path) under a settings group.

To explain the benefit of that we will look at a usage example which often occurs:

We want to create tests for an application called `mycad`. There exist two versions of the application, one in `/usr/local/mycad_1_0/bin` and the new version 1.1 is located in `/usr/local/mycad_1_1/bin`.

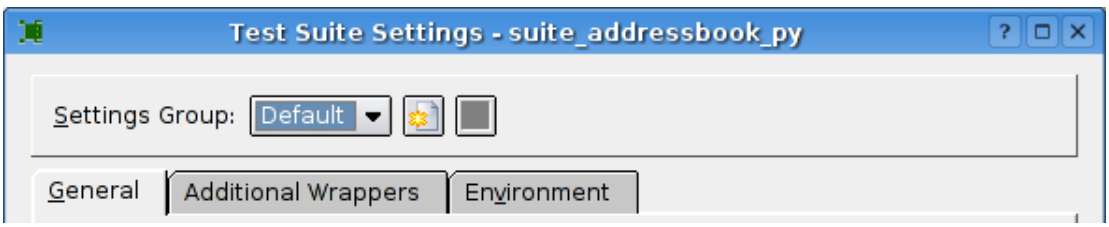
The tests we create should be ran against both versions. But without modifying the AUT paths or AUT mapping between each test run, it isn't possible to tell squishserver which version we want to launch.

In addition, only in version 1.1 we need to pass the argument `--guifrontend` to the application when starting it.

To solve this kind of problems, *Squish* introduces settings groups.

Setting up everything with settings groups is much easier in the Squish IDE so we discuss this way of doing it. First we create two settings groups: v1_0 and v1_1. To do this we open the test suite settings via Test Suite → Settings in the Squish IDE.

At the right top corner we can see a combobox which currently displays the default settings group and a button to add a new settings group:



By clicking this button, the two new settings groups can be added. Now all settings which will be specified (AUT, AUT path, arguments, etc.) will be added under the currently active settings group. To change the currently active settings group, select one in the combobox at the top right

When recording or replaying a test, it also is necessary to let the Squish IDE know which settings group to use. This way we can specify if e.g. version 1.0 or 1.1 of the AUT should be started. This can be done in the test suite settings dialog as well - just select the settings group to use in the combobox at the top.

For command-line users

On the command line, to specify a settings group when recording a test case, use

```
squishrunner --settingsGroup group --testsuite suite_directory --record test_case_name
```

To run a test suite in a specific settings group, use

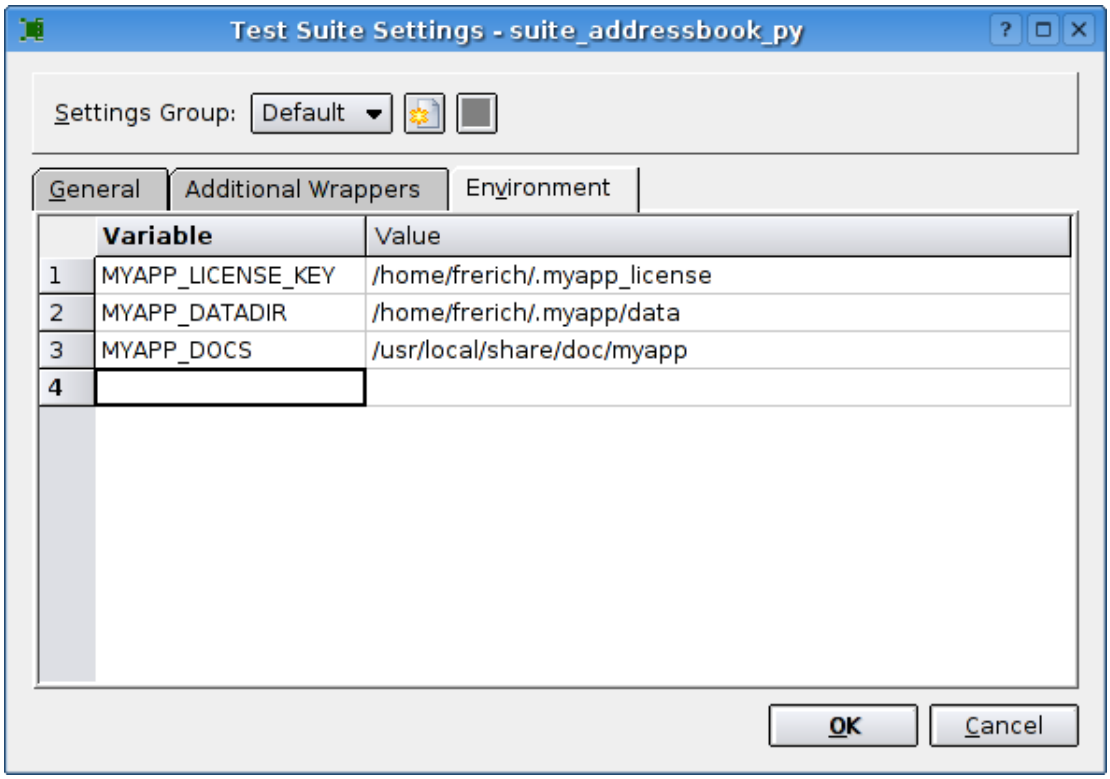
```
squishrunner --settingsGroup group --testsuite suite_directory
```

7.2.4 Setting Environment variables for the AUT

In some cases an application needs a special environment to be set up before it can start. This might include setting some environment variables, running a helper application and checking for some other conditions (enough memory available, etc.). Usually this is done by having a shell script or Windows batch file which does this pre-initialization work and then starts the actual application binary. See [?] for details.

If only some environment variables need to be set up, the preferred way to do this when testing and application with *Squish* is to specify the environment variables in the test suite settings and use the application binary directly as AUT.

To specify environment variables, open the test suite settings in the Squish IDE via Test Suite → Settings and switch to the Environment Variables tab. In there it is possible to specify environment variables:



This way the specified environment variables will be set before squishserver starts the AUT.

7.2.5 Testing Java Applets

For testing Java™ applets, just specify in your test suite settings the Java `appletviewer` as AUT (application under test) and specify the URL of the HTML file containing the applet as argument to the AUT. No more special setup is required.

Additionally, using *Squish* for Web it is also possible to test Java applets embedded in a web browser. See also [?]

7.2.6 Testing Java Web Start

For testing Java™ Java applications launched via Java Web Start, just specify in your test suite settings the Java `javaws` as AUT (application under test) and specify the URL to the JNLP file of your web start application as argument to the AUT. No more special setup is required.

7.2.7 Wrapping custom Java™ classes

Squish wraps a predefined set of Java™ classes. Only from wrapped classes, scripts can access properties and methods on object instances. Likewise the spy can only show properties of wrapped classes. If a class isn't wrapped, only properties and methods from super classes can be used that are wrapped.

To extend the list of wrapped classes, the classes have to be listed in a `.ini` file. And the wrapper of the Java™ process has to be told where to find this file.

7.2.7.1 Writing the `.ini` file

The format is like in the example below. For a class `com.example.product.Example` and `com.example.product.Main`.

```
[general]
AutClasses="com/example/product/Example", "com/example/product/Main"
```

The class names must be relative to the `classpath`.

When an extra class is wrapped, all super classes, classes from method arguments and return type classes will be wrapped as well. If such a wrapping fails however, it will become an `Unknown` type.

Additionally, an extra `classpath` can be set that might be needed if the application extends its `classpath` during execution. For instance, testing Java™ applets with Web testing, the `classpath` is extended by the `ARCHIVE` attribute of the `APPLET` HTML tag. In the previous example, a `classpath` setting is added like below.

```
[general]
AutClasses="com/example/product/Example", "com/example/product/Main"
AutClassBaseUrl="http://www.example.com/product/"
AutClassPath="demo.jar"
```

The `classpath` entry is like usual notation, with ';' (Windows) or ':' (Mac/Linux/Unix) separators. The complete entry has to be quoted.

The option `AutClassBaseUrl` is optional and when set, adds a prefix to all paths in the `AutClassPath` entry plus the `AutClassBaseUrl` itself is added to the `classpath`.

7.2.7.2 Register the `.ini` file

7.2.7.2.1 Using *Squish* server

The squishserver can then be used to register the created `.ini` file. The command

```
squishserver --config setConfig <name-of-AUT> <path-of-ini-file>
```

will register the file. Note the name of the application in the above command is the exact name found in the Test Suite Settings dialog. Removing this registration is done by issuing this command with the `removeConfig` option, thus

```
squishserver --config removeConfig <name-of-AUT> <path-of-ini-file>
```

7.2.7.2.2 Using an environment variable

An environment variable can be set to specify the `.ini` file location. The environment variable is `SQUISH_WRAPPER_CONFIG` and can be set in the testsuite settings dialog.

7.2.8 Configuring the recognition of native Windows controls

By default, *Squish* will only record actions on native controls (such as embedded ActiveX controls) if it can clearly identify the controls as such. However, with Java™, it's not possible for *Squish* to tell for sure whether something is a native control or a Java™ control. Hence, it is necessary to tell *Squish* about the native controls being used, otherwise no actions on the control will be recorded.

Note
The support for interacting with native Windows control is in a relatively early stage. The support for more sophisticated and stable automation of native Windows GUIs is scheduled to be released with *Squish* 4.0 but may require purchasing a special Windows testing license.

Much like in the section Section 7.2.7, this is done by configuring an `.ini` file appropriately. For instance, to describe that a control called `MyCalendarCtrl` is a native control, create a configuration file with the following content:

```
[Windows Wrapper]
Whitelist="MyCalendarCtrl"
```

Tip
If you don't know the class name of the native control, the developers of the application can properly tell. Alternatively, you can use tools like Spy++ (ships with Microsoft Visual Studio) or AutoIt Window Info.

If you have an existing `.ini` file (possibly because you instructed *Squish* to recognize custom Java™ classes as described in the section Section 7.2.7), you can simply add the above lines to your existing file.

If you want to add more entries to the *Squish* whitelist, simply separate their names with a comma, as in this example:

```
[Windows Wrapper]
Whitelist="MyCalendarCtrl","AnotherControl","InternetExplorerCtrl"
```

After creating the configuration file, the only thing left to do is to tell *Squish* which application this configuration file is for. To do so, simply follow the steps described in the section Section 7.2.7.2.

7.3 Testcase Templates

After creating a number of test cases in *Squish*, you might notice that they all have some basic structure in common. For instance, they might all have some startup code in their main test script. To avoid that you have to copy and paste that common code whenever you create a new testcase, you can use test case templates.

7.3.1 Creating a New Template

To create a new test case template, you first need to decide which existing test case shall serve as a template. To do so, choose any test case item in your tree of test suites at the left hand side of the Squish IDE, right click on it to open the context menu, then click on the menu item labeled Use As Testcase Template.

A small dialog will pop up, asking you for a name for the template - choose something descriptive here, then press the OK button. In case you already have a test case template with the chosen name, the dialog will tell you so and ask you again.

7.3.2 Reusing a Template

To use your test case template instead of starting a new test case from scratch, simply select File → New Test Case From Template... from the menu bar (or choose the menu item with the same name from the context menu of test suite items).

Assuming you have defined any test case templates before (if you haven't, a message box will tell you so), a little dialog will become visible, asking you to select one of the available templates for the creation of the new test case. Select the template you want to use, then press the OK button.

A new test case item will be added to the currently active test suite, and you have to enter a name for the new test case. Press Enter after entering a name for the test case to finish the creation.

7.3.3 Choosing a Custom Location for Storing Templates

By default, test case templates are stored in `%APPDATA%\froglogic\Squish\templates`, (Windows) or `~/.squish/templates` (Unix & Mac OS). This allows reusing test case templates between different test suites. However, it might be desireable to store test case templates together with the suites. For instance, a test suite might be stored in a version control system and all test case templates should go with the suite so that every user who downloads the test suite from the version control system gets to use the very same test case templates as well.

This can be accomplished by making sure that the `SQUISH_TESTCASE_TEMPLATES_DIR` is set as the Squish IDE is started. In case this variable is set, the default location will be ignored and instead the given directory will be expected to contain the test case templates.

7.4 Object Map

This chapter will introduce the concept of an automatic object map (also called GUI object map in QA literature) and how *Squish* implements this concept.

7.4.1 The Concept of the Object Map

The object map is a tool with the goal to ease the maintenance of test scripts when the application under test changes its object hierarchy or object names.

In a test script, objects are accessed by looking up the object via its fully qualified name or its multi-property name. This is called the real name.

If the name of the object, one of its parent object (for qualified names) or one of the identifying properties (multi-property names) changes, this real name is not valid anymore.

To solve this problems, objects are not looked up directly by its *real* name anymore. Instead in the test scripts, *symbolic* names are used. Then, when the object is looked up by its name, the *real* name for the given *symbolic* name is looked up in the *object map*. This way, if the name of an object changes because of any of the given reasons, it is enough to correct one entry in the object map and all test scripts will keep working.

For more information on *Squish*'s object naming mechanism see also Section 7.6

7.4.2 Squish's Object Map

The concept provided above is offered by *Squish* as well to make the QA engineers' lives easier when the AUT changes in the described manners.

7.4.2.1 Creating an Object Map

The object map for an application is created automatically when recording test cases in *Squish*. The map is saved in the file `objects.map` in the test suite's root directory and applies to all tests of the test suite. Alternatively it is possible to specify the location of the object map using the `OBJECTMAP` key in the `suite.conf` of the test suite. The value of this can be an absolute or relative path. Specifying a different location of an object map can be useful if e.g. multiple test suites should share the same object map file.

When recording a test, for each object accessed in the generated script code, an entry is stored in the object map. Initially the symbolic name and the real name of the widget are identical. The only way to distinguish a symbolic object name is its prefix. Concretely, this means a symbolic object name is prefixed by a colon.

Therefore, in the generated test script only symbolic object names will be used by using this prefix. So when *Squish* executes a test script and an object should be looked up by its name, *Squish* will retrieve the real name from the object name if this name is prefixed with a colon and will do the runtime object lookup with the real name. If the given object name does not start with a colon, the object is looked up directly with this name.

This means it is possible to mix symbolic and real object names in test scripts without conflicts.

7.4.2.2 Editing an Object Map

The best way to edit an object map is to click on it in the Squish IDE in the test suite view and edit it in the object map editor which will open up automatically.

If you manually write script code and want to use symbolic names for objects which have not been added to the object map yet, you can insert those entries by editing the object map table.

If the object name for an object changed and you want to correct its real name, you can do this by looking for this object in the object map editor and by editing the real name of the object. A good way to look up the new object name is to use the Spy as discussed Section 7.8.4. You can add objects from the Spy to the object map using the Spy's context menu.

When opening the object map editor while the Spy is actively spying the application under test, it is possible to check which names in the object map are valid. To do this, right-click on a symbolic name in the object map editor and choose the option to check one or all names.

It is also possible to edit the symbolic name of an object. If you do this, you will also have to edit the scripts using these names, since the symbolic name is the one which is used for the lookup.

7.5 Improving Object Identification

Starting with Squish 3.2, the syntax of multiple property names has been enhanced to allow more flexible matching of objects. For instance, lets assume that you have the following real name for a MainWindow object in your application with the caption 'UsefulApp v1.3 - All rights reserved':

```
{caption='UsefulApp v1.3 - All rights reserved' type='MainWindow'}
```

This multiple property name will match the first object in your application which is of type MainWindow and whose caption matches the one given above.

Such names are perfectly fine in many situations and they are generated automatically when recording a test (see Section 7.6 for more information on that topic), but over time, as your application evolves, you might find this to be insufficient. For example, imagine a new version of the UsefulApp program gets released - version 1.4 - and you attempt to run your test. Since your programs version number was increased, the caption of the MainWindow object now read 'UsefulApp v1.4 - All rights reserved'. This unfortunately makes many tests fail, since our real name given above still expects the caption of the 1.3 main window.

To solve this in an elegant manner, Squish allows you to use other matching modes when comparing the property values given in a real name with the property values of some object in the application. Some real name like

```
{text='Watermelon'}
```

simply identifies the object whose text equals ‘Watermelon’. This text equality is expressed by the ’=’ sign. However, it is possible to use other matching modes as well. For instance, the following matches any object whose text starts with ‘Wat’, followed by zero or more arbitrary characters, followed by ‘lon’:

```
{text?='Wat*lon'}
```

Note how the ’=’ was changed into a ’?=’ to express that we don’t want to simply compare to character strings for equality but that the given string is to be treated as a so-called ’wildcard’ string. Hence, the asterisk (’*’) stands for ’zero or more arbitrary characters’ and not a plain ’*’. If we had used ’=’ instead of ’?=’, then the asterisk wouldn’t have had any special role, and the real name would have identified the object whose text equals ’Wat*lon’.

Hence, we can solve our original problem easily by replacing the hardcoded version number in the real name with an asterisk, so that any version will match:

```
{caption?='UsefulApp v* - All rights reserved' type='MainWindow'}
```

Finally, there’s a third matching mode available, which treats the given string as a so-called regular expression. Please see the following reference table for more information on that:

PropertyName=PropertyValue Performs a straight character string comparison. A given object must match *PropertyValue* exactly to match.

PropertyName?=PropertyValue Performs a wildcard match. In this matching mode, the given *PropertyValue* can contain certain meta characters:

? This matches any single character.

***** This matches zero or more of any characters.

[...] (i.e. [abc123]) This matches a given set of characters. The example string [abc123] matches any of the characters ’a’, ’b’, ’c’, ’1’, ’2’ or ’3’.

If you want to match any of these meta characters directly (i.e. you want to match a plain question mark), you have to escape them by preceding them with a backslash character (i.e. ‘text?=’Is this true\?”).

For example, the following multiple property name will match all objects whose text starts with either ’a’ or ’A’ followed by an arbitrary character and ending in ‘tion’:

```
{text='[aA]?*tion'}
```

PropertyName~=PropertyValue Performs a regular expression match, treating the given *PropertyValue* string as a regular expression.

This is an even more powerful method to specify string patterns, more powerful than the above mentioned wildcard strings. Regular expressions feature more meta characters, however they might become very hard to read when overusing the features.

You can find a lot of information about regular expressions, including a referencee of the syntax, many examples and tutorials and links to other pages at Regular-Expressions.info.

7.6 Customizing Object Name Generation

7.6.1 (Insufficient) Object Names

When recording a test case or picking objects in the Spy, *Squish* will automatically create a name for the object to identify it. Depending on the configured naming scheme, this name can either be a hierarchical name or a so-called multiple property name. The latter is a set of *propertyName=value* pairs enclosed in curly braces. For instance, a multiple property name which identifies the object whose type is Button and whose text is ‘Hello’ looks like

```
{type='Button' text='Hello'}
```

Thus, when creating a name for an object, *Squish* needs to decide which properties to use for identifying the object. However, it can be difficult for *Squish* to decide on a set of properties which is both

Unambiguous There mustn't be more than one object in the application which matches the given set of property/value pairs, so that the name identifies just the object you want to.

Minimal You don't want to have more properties in the object name than necessary so that the name is robust in the face of application changes. For instance, if the object name included all properties of the respective object, any change to the object which affects the value of a property would break the test script since the object name no longer finds the original object.

Squish has a set of heuristics built-in to determine what properties to use, but sometimes this fails. For instance when writing a Qt test, *Squish* will use the 'name' property of objects in the application to identify them if it exists. However, you might find that for your particular application, this is not a good choice since the name is a more or less random value on every run so you would like to remove this property from the list which *Squish* uses internally.

Starting with *Squish* 3.2, the property list used to create the real names can be configured by editing some straightforward XML files. How to use these files to customize the name creation process will be explained in the following two sections.

7.6.2 Defining Property Sets

7.6.2.1 Descriptor File Locations

The list of properties used for the real name of some object are defined by XML files. For each wrapper which your application uses there can be up to two XML files (so called 'descriptor files') which have to adhere to a special naming scheme so that *Squish* finds them:

`Squish_Directory/etc/wrapper_descriptors.xml` This XML file contains the default properties which *Squish* uses for an object's name. Depending on the type of package you use, you might find `qtwrapper_descriptors.xml`, `swtwrapper_descriptors.xml` or others in the `etc` subdirectory of your *Squish* installation.

`Squish_Directory` stands for the installation where you installed your *Squish* package.

`Squish_User_Settings/wrapper_user_descriptors.xml` This file is used for your own extensions. In case you want to modify the default behaviour, either by overriding the existing behaviour or by adding new properties, you can use this file. The file format is the same as for the default descriptors file mentioned above.

`Squish_User_Settings` stands for the directory where user specific settings will be stored. On Windows, this is the directory `%APPDATA%\froglogic\Squish`, on Unix and Mac OS it's `~/.squish`. If you set the environment variable `SQUISH_USER_SETTINGS_DIR` to point to a different directory, that directory is used insted for storing the user settings (it is used for all user settings; not only for descriptor files).

In both files, a list of types is defined together with the names of the properties to be used when generating names for objects of that particular type. The file format used to do so is the same in both files and they're treated the same, except that the descriptor file for user extensions (`wrapper_user_descriptors.xml`) can not only add additional type descriptors, but it can also override any descriptors in the main XML file.

7.6.2.2 Descriptor File Format

The list of types and the properties to be used for each object are written down using a rather straightforward XML format. Here's a quick example for some fictive application toolkit which has a Button type:

```
<objectdescriptors>
  <descriptor>
    <type name="Button"/>
    <realidentifiers>
      <property>caption</property>
    </realidentifiers>
  </descriptor>
</objectdescriptors>
```

This descriptor file defines just one descriptor which says that for all objects of type Button, the 'caption' property should be used for the real name.

Note
This means that not only objects which are instances of the Button class will get the caption property in their real name, but also instances of classes which inherit Button! When deciding which descriptors to use for generating the name for a given object, *Squish* takes the inheritance hierarchy into account and will use all descriptors whose type name shows up in the inheritance hierarchy of the given object.

In this example, only the name of the type was used for identifying the name. However, it is also possible to introduce so-called constraints which make sure that not all objects of a particular type are applicable but only those which match certain constraints. Another example illustrates this:

```
<objectdescriptors>
  <descriptor>
    <type name="Button">
      <constraint name="visible">false</constraint>
    </type>
    <realidentifiers>
      <property>caption</property>
      <property>tooltip</property>
    </realidentifiers>
  </descriptor>

  <descriptor>
    <type name="Button"/>
    <realidentifiers>
      <property>caption</property>
      <property>xpos</property>
      <property>ypos</property>
    </realidentifiers>
  </descriptor>
</objectdescriptor>
```

Here, two descriptors are defined. Both of them are about objects of the Button class, however the first one is only applicable if the 'visible' property of the respective object has the value 'false'. When using this descriptor file to generate the real name for a Button object which is invisible, the properties 'caption' and 'tooltip' will show up in the real name. However, for visible buttons, the second descriptor will be used and hence the properties 'caption', 'xpos' and 'ypos' will be used in the object name.

You could also list more than one constraint. In that case, all of the listed constraints have to match before the descriptor is used. In general, when facing multiple descriptors which all reference the same type name, *Squish* will try to use the best match; that is, the descriptor with the highest number of constraints of which all match.

7.6.3 Advanced Property Set Definitions

7.6.3.1 The Catch-All Descriptor

In addition to the normal descriptors, which match an object by the type name, there's a special case descriptor which is always used no matter what the name of the object's type is. However, you can use constraints for this special type as well. Here's how to use it:

```
<objectdescriptors>
  <descriptor>
    <type name="*"/>
    <realidentifiers>
      <property>id</property>
    </realidentifiers>
  </descriptor>
```

```
  <descriptor>
    <type name="Button"/>
    <realidentifiers>
      <property>caption</property>
    </realidentifiers>
  </descriptor>
</objectdescriptor>
```

In this example, a catch-all descriptor (which is denoted by using an asterisk as the type name) is defined. This means that for all objects, no matter what their type name is, the id will be used in the property name. If the given object does not have an id property, it will be silently ignored but not trigger an error.

Note
It does not hurt to list properties in the catch-all descriptor (or any other descriptor, for that matter) which don't exist on some particular object. For instance, even though the catch-all descriptor would be used for *any* object, that object doesn't have to have an 'id' property. If it doesn't that property will be silently ignored.

The catch-all descriptor is also taken into account when merging multiple descriptors. For instance, if the above example would be used to generate the name for an object of type Button, the properties 'caption' *and* 'id' would be used in the real name.

7.6.3.2 Groups of Exclusive Properties

Let's say that you have a descriptor file to generate names for your 'Button' widgets like this:

```
<objectdescriptors>
  <descriptor>
    <type name="Button"/>
    <realidentifiers>
      <property>id</property>
      <property>caption</property>
      <property>text</property>
      <property>enabled</property>
    </realidentifiers>
  </descriptor>
</objectdescriptor>
```

With this descriptor, the properties 'id', 'caption', 'text' and 'enabled' will be used in the real name of 'Button' objects. However, this might be too much. For instance, even though you do want the 'id' and the 'enabled' property in the real name, you do not need the 'text' property in case a 'caption' is available. The 'text' property should only be used when there is no 'caption' available.

To solve this common problem, you can use so-called property groups. The idea is to put all properties which exclude each other (in this example, the 'caption' and the 'text' properties) into a group. When generating a name, *Squish* will try one property in the group after the other. As soon as it finds a property which exists on the given object, all other properties in the group are ignored.

Here's how to do that in XML:

```
<objectdescriptors>
  <descriptor>
    <type name="Button"/>
    <realidentifiers>
      <property>id</property>
      <group>
        <property>caption</property>
        <property>text</property>
      </group>
```

```
      <property>enabled</property>
    </realidentifiers>
  </descriptor>
</objectdescriptor>
```

Now, names for Button objects will *either* contain the caption of a property *or* the text - but never both.

7.6.3.3 Object References

One feature of object names in *Squish* is that they can not only contain property/value pairs but also references to related objects. For instance, it is possible to identify an input field by specifying the label which is next to the input field (such a label is often called the 'buddy' of the input field).

Here's an example object name for a fictive input field which is not only identified by its own properties but also by the buddy label:

```
{type='LineEdit' size='32' numbersAllowed='false' buddy={type='Label' text='Last Name'}}
```

Note how the value of the 'buddy' property is really just another object name, nested in the object name of the input field. The only notational difference between normal properties and such object references is that with normal properties (like 'size' in the example above) the value of the property is enclosed in apostrophes. With object references, this is not the case.

Specifying that some property is really a reference to another object in the XML file is quite straightforward:

```
<objectdescriptors>
  <descriptor>
    <type name="LineEdit"/>
    <realidentifiers>
      <property>type</property>
      <property>size</property>
      <property>numbersAllowed</property>
      <object>buddy</object>
    </realidentifiers>
  </descriptor>
</objectdescriptor>
```

The only difference with the buddy property is that it's enclosed in 'object' tags instead of the normal 'property' tags.

7.7 Automated Batch Testing

This chapter discusses all aspects of automating testing, so called batch testing. This includes automatically executing tests, distributing tests to different machines and finally processing the results from the test runs.

7.7.1 Automated Test Runs

To allow automating test runs, *Squish* provides command line tools which control the whole procedure. The tool to execute tests is squishrunner. To be able to start an AUT and communicate with it, squishserver has to be running.

A simple Unix shell script to execute the complete test suite `/home/squish/suite_myapp` and save its results to `/home/squish/results<date>.xml` would be:

```
#!/bin/sh

# start squishserver
squishserver &

# generate filename for logfile
LOGFILE=/home/squish/results`date +%d%m%Y`.xml
```

```
# execute the test
squishrunner --testsuite /home/squish/suite_myapp --reportgen xml,$LOGFILE

# exit squishserver again
squishserver --stop
```

The same as a Windows batch file wouldn't look very different:

```
REM start squishserver
start /B squishserver

REM generate filename for logfile
set LOGFILE=c:\squishhome\results.xml

REM execute the test
squishrunner --testsuite c:\squishhome\suite_myapp --reportgen xml,%LOGFILE%

REM exit squishserver again
squishserver --stop
```

Both sample scripts assume that the squishserver and squishrunner executables can be found in the system search path as denoted by the `PATH` environment variable.

To avoid having to maintain a Unix shell script and a Windows batch file, another option would be to implement this script e.g. using Python which would allow a platform independent implementation.

Such a script automatically executes a test with all required initializations and cleanups. For details on how to process the resulting XML file, see Section 7.7.3.

The next step would be to have this test script run automatically e.g. once a day. On Unix this can be done by setting up a cron job which executes the script. On Windows it is possible to set up a Windows Service to run the script regularly.

For detailed documentation about cron jobs or Windows Services, search for the relevant information on the Internet or contact *froglogic*'s commercial support to assist you.

7.7.2 Distributed Tests

So far everything in this manual assumed local testing. This meant that the squishserver, squishrunner, and the AUT are all running on the same machine. This chapter shows how to remotely run a test on a different machine. For example, let's assume that we work and test on computer A, and want to test the AUT located on computer B.

We begin by installing *Squish* and the AUT (except for Web testing, where this is not necessary), on the target computer (computer B). Now, if we are using an edition other than Squish for Web, on computer B, we must tell the squishserver where the AUT is located. This is achieved by running the following command:

```
squishserver --config addAppPath <path_to_addressbook>
```

Later we will connect from computer A to squishserver on computer B. By default the squishserver only accepts connections from the local machine. Accepting connections from elsewhere might compromise security, so if we want to connect to the squishserver from another machine we must register the machine which will try to establish a connection for executing the tests (computer A in our example) with the machine running the AUT and squishserver (computer B). This way we can ensure that only trusted machines can communicate with each other using the squishserver.

To do this, on the AUT's machine (computer B) we create a plain text file called `/etc/squishserverrc` (on Unix or Mac) or `c:\squishserverrc` (on Windows). If you don't have write permissions to `/etc` or `c:\`, you can also put this file into `SQUISH_ROOT/etc/squishserverrc` on either platform. The file should have the following contents:

```
ALLOWED_HOSTS = <ip_addr_of_computer_A>
```

<ip_addr_of_computer_A> must be the IP address or host name of computer A. For example on our network the line is:

```
ALLOWED_HOSTS = 192.168.0.3
```

but this will most certainly differ on your network.

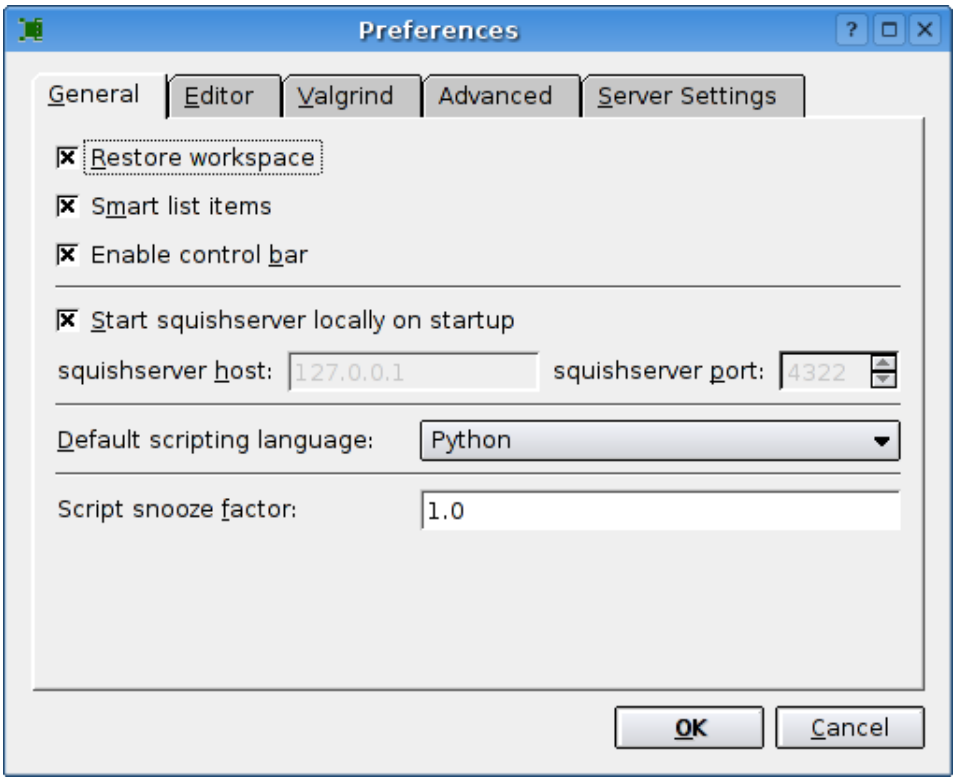
`ALLOWED_HOSTS` also accepts IP addresses with wildcards. To e.g. allow all machines from the 192.168.0 range to connect to this squishserver, you can specify 192.168.0.*. To specify multiple IP addresses which should be allowed to connect to the squishserver, you can specify all IP addresses in the `ALLOWED_HOSTS` separated by spaces.

Once we have registered computer A, we can run the squishserver on computer B, ready to listen to connections e.g. from computer A.

We are now ready to create test cases on computer A and have them executed on computer B. So we have to start squishserver on computer B (calling it with the default options starts it on port 4322 - see Section 8.2.2 for a list of available options):

```
squishserver
```

By default for convenience, the Squish IDE starts squishserver locally on startup and connects to this local squishserver to execute the tests. But it is also possible to connect to the squishserver on a remote machine, such as computer B, from withing Squish IDE. We can control this behavior through the preferences dialog. Click Edit → Preferences to invoke the preferences dialog. Uncheck the Start squishserver locally on startup checkbox, and enter the IP address of the AUT's machine (computer B) in the squishserver host line edit. The port spinbox should not need to be changed.



Now we can execute the test suite as usual. We will see that the AUT will not be started locally, but on computer B instead. After the test has finished, the results will be visible in the Squish IDE on computer A as usual.

We can also do remote testing using the command line. The command is the same as described in the previous sections, only this time we must also specify a host name using the `--host` option:

```
squishrunner --host computerB.froglogic.com --testsuite suite_addressbook
```

This way it is possible to create, edit and run tests on a remote machine via Squish IDE and also to control them from the command line or a shell script using squishrunner. In conjunction with Section 7.7.1 it is possible to automate testing of an applications located on different machines and platforms using shell scripts or batch files.

7.7.3 Processing Test Results

In the previous section we saw how to execute an AUT and run its tests on a target machine under the direction of a master machine and also how to automatically execute test runs using scripts and batch files. In this section we will look at processing the test results from automatic test runs.

By default, squishrunner prints the test results to `stdout`. Although it isn't difficult to parse this output, squishrunner also includes a report generator which can output the results in XML or Excel file format. There are modules for nearly every scripting language available to parse XML, so it is quite easy to post-process the test results and convert them to the format you require.

To make squishrunner use the XML report generator, specify `--reportgen xml` on the command line. If you want to get the XML output written into a file instead of `stdout`, specify `--reportgen xml,`*`filename`*, e.g.:

```
squishrunner --host computerB.froglogic.com --testsuite suite_addressbook_py --reportgen
   xml,/tmp/results.xml
```

For Excel output, use the `xls` option for `--reportgen` instead. So the command then looks like:

```
squishrunner --host computerB.froglogic.com --testsuite suite_addressbook_py --reportgen
   xls,/tmp/results.xls
```

The XML format is very simple. The document starts with the tag `SquishReport`. All other tags are children of this one. The following child tags are possible:

- `summary`: (only once) with the following attributes
  - `testcases`: Number of test cases executed
  - `tests`: Number of test commands executed
  - `fatals`: Number of fatal results
  - `errors`: Number of failed test commands
  - `passes`: Number of passed test commands
  - `expected_fails`: Number of expected failures
  - `unexpected_passes`: Number of unexpected passes
- `testresult`: (any number of times) with the following attributes
  - `message`: Short result message
  - `result`: Result type, this can be one of
    * `START`
    * `START_TEST_CASE`
    * `END`
    * `END_TEST_CASE`
    * `PASS`
    * `FAIL`
    * `FATAL`
    * `ERROR`
    * `LOG`
    * `MEMLOG`

    If the result type is `START_TEST_CASE` then the `testresult` element has all the attributes available with the `summary` element (`fails`, `passes`, `warnings` and so on), reflecting how many of the comparisons in that particular test case failed/passed etc..
  - `time`: The timestamp of the result in ISO format.
  - `line`: Line number information including source file name.

The detailed message of the result is saved in the text node of the element.

Here is an example report of a test suite run. This test suite had just one test case, in which four comparisons passed and one failed:

```
<SquishReport version="1.2" >
    <summary fatals="0" testcases="1" expected_fails="0" unexpected_passes="0" warnings="0" ↩
        tests="5" errors="0" fails="1" passes="4" />
    <testresult message="Start 'suite_addressbook_js'" result="START" time="2006-06-06T10 ↩
        :33:31" >Test 'suite_addressbook_js' started</testresult>
    <testresult fatals="0" expected_fails="0" unexpected_passes="0" warnings="0" errors="0" ↩
        message="Start 'tst_add_address'       " result="START_TEST_CASE" fails="1" time ↩
        ="2006-06-06T10:33:32" passes="4" >Test 'tst_add_address' started</testresult>
    <testresult line="test.js:23" message="VP1: Object property comparison of ':Addressbook ↩
        .ABCentralWidget1.QListView1.childCount' passed" result="PASS" time="2006-06-06T10 ↩
        :33:44" >'1' and '1' are equal (VP1)</testresult>
    <testresult line="test.js:23" message="VP1: Object property comparison of ':Addressbook ↩
        .ABCentralWidget1.QListView1.item_1.text0' failed" result="FAIL" time="2006-06-06T10 ↩
        :33:44" >'Max' and 'Maxi' are not equal (VP1)</testresult>
    <testresult line="test.js:23" message="VP1: Object property comparison of ':Addressbook ↩
        .ABCentralWidget1.QListView1.item_1.text3' passed" result="PASS" time="2006-06-06T10 ↩
        :33:44" >'max@mustermann.net' and 'max@mustermann.net' are equal (VP1)</testresult>
    <testresult line="test.js:23" message="VP1: Object property comparison of ':Addressbook ↩
        .ABCentralWidget1.QListView1.item_1.text2' passed" result="PASS" time="2006-06-06T10 ↩
        :33:44" >'Bakerstreet 55' and 'Bakerstreet 55' are equal (VP1)</testresult>
    <testresult line="test.js:23" message="VP1: Object property comparison of ':Addressbook ↩
        .ABCentralWidget1.QListView1.item_1.text1' passed" result="PASS" time="2006-06-06T10 ↩
        :33:44" >'Mustermann' and 'Mustermann' are equal (VP1)</testresult>
    <testresult message="End 'tst_add_address'         " result="END_TEST_CASE" time ↩
        ="2006-06-06T10:33:46" >End of test 'tst_add_address'</testresult>
    <testresult message="End 'suite_addressbook_js'" result="END" time="2006-06-06T10 ↩
        :33:46" >End of test 'suite_addressbook_js'</testresult>
</SquishReport>
```

In `examples/regressiontesting` you can find some example scripts which execute the addressbook test suite on different machines and present the daily output on a Web page by post processing the XML and generating HTML. The tutorial Chapter 6 shows step-by-step using this example how to automate test runs and process the test results.

7.8 Editing and Debugging Test Scripts

Squish doesn't only provide the tools to record and edit tests. In addition debugging and inspection tools are available which assist test engineers when post-editing and debugging test scripts.

7.8.1 Script Debugger

The script debugger in the Squish IDE allows to set break points in the script. This can be useful to insert verification points (see [?]) or to record from a breakpoint as discussed later in this chapter. But it also can be used to step through the test script using Test Suite → Step and Test Suite → Step Into.

To halt a test run without setting a break point, Test Suite → Break can be used. To continue a test run, use Test Suite → Execute again.

Sometimes during test execution it is interesting to follow the currently executed line in the Squish IDE. For this purpose it is possible to specify in the Editor's tab of the preferences dialog Edit → Preferences to highlight the current line during test execution by checking the corresponding check box.

7.8.2 Recording After a Breakpoint

In some cases it is required to extend an existing test case with more actions. It would be tedious to record everything from scratch just to e.g. click an additional button in the test. Sometimes it is easily possible to just manually write a few lines, but often it would be easier to record the new actions.

This approach is also very useful if e.g. each test needs to run the same initialization actions on the AUT (opening a file or so) which is provided by a shared script. A new test case could then be created by writing a script which loads the shared script (see Section 7.11.1 and executes this initialization actions. After that a new test should be recorded.

This is possible in *Squish* by setting a breakpoint in a test script located where the newly recording actions should be inserted. Then you execute the test until this breakpoint is hit. After that you can choose Test Suite → Record which starts recording on the running AUT. By choosing Test Suite → End recording recording will be stopped and the newly recorded actions will be inserted into the test script at the breakpoint.

7.8.3 Recording and Inserting at Cursor

In some cases it is needed to extend an existing test case. Often in applications its needed to some setup steps before the extra actions can be recorded. The section Section 7.8.2 explains how to do that by first executing the existing script. Sometimes this takes time due to the various verifications beeing done and also the AUT might allow shortcuts. So it would be nice if one could just start the application and work with it and at some point tell *Squish* to record from this point and insert the recorded actions into a testcase.

Squish allows to do that by right-clicking into the test script and choosing the menu item Record script and insert here. Then Squish starts the AUT and opens the *Squish* control bar. Then you can work with the AUT as if *Squish* wasn't running and when you want to start recording you can select the Start Recording toolbutton in the control bar. It uses the same icon as the normal record button in the IDE. When you're done recording your actions simply hit the stop button in the control bar and in your script you'll see new code inserted at the point where your cursor was.

7.8.4 Spy

In the chapter Section 7.10 and the chapter [?] in the tutorial [?] the Spy was used to insert verification points.

The Spy can be used stand-alone (without a test script) in all editions except Squish for Web to inspect the application, its objects and properties. For Squish for Web, to use the Spy, you need a test script which opens an URL. Set a breakpoint after the `loadUrl` call, and execute the test script until that point to use Spy.

Using Spy is especially useful when manually writing script code to find out the names of objects and which properties of certain objects are accessible.

To start the Spy, activate Spy → Spy AUT in the menu of the Squish IDE. Now the AUT of the currently active test suite will be started and the Spy window appears in the Squish IDE, or in Squish for Web, choose the menu option after you started the test script and hit the breakpoint.

There are three different modes in the Spy

- *Run:* The Spy is active and you can navigate in the object and property hierarchy views. In this mode all user events to the AUT are blocked.
- *Pause:* The Spy is inactive and all events to the AUT are processed normally. You usually switch to this mode when you want to e.g spy on a dialog in the AUT. For this reason you will pause the Spy, open the dialog in the AUT and then choose Run in the Spy again which will let you navigate the object hierarchy and properties of the now opened dialog.
- *Pick Object:* The Spy switches the AUT into a picking mode. In this mode you can select a widget in the AUT by moving the mouse over the widgets and clicking on it. If you selected a widget in the AUT, its object hierarchy and properties will get displayed in the Spy's views.

To exit the Spy, choose Spy → Spy AUT again.

Initially the Spy is in the *Run* state. To e.g. find out the name of a widget in the AUT, switch to the *Pick Object* state by activating Spy → Pick Object in the menu of the Squish IDE. Now we can switch the focus to the AUT which has been started. When moving the mouse over the AUT's windows, the objects under the mouse pointer are selected with a red frame and the name and type of the hovered widgets are displayed in a tool tip. So we move the mouse over the desired widget in the AUT and click on it.

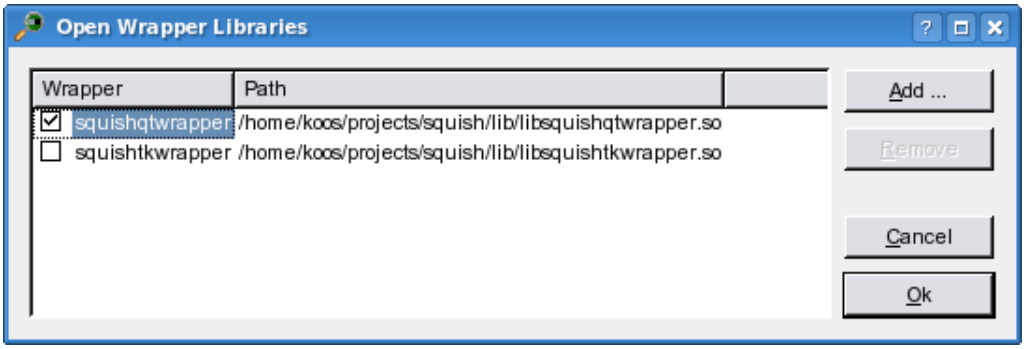
The Spy will automatically switch back to *Run* mode and display the selected object, its children and properties in the Spy views. By right clicking on the object in the Spy view, you can copy the name to the clipboard and paste it into the test script where needed (e.g. for a `findObject` call) or add the object directly to the object map. See also Section 7.4.2.2.

7.9 Explore Wrapper Libraries

Depending on your *Squish* edition, you can explore all the bindings (wrappers) that are available for *Squish*. *Squish* comes with its own set of wrapper libraries, but if you have bindings for an AUT, like explained in [?], then you can explore those too.

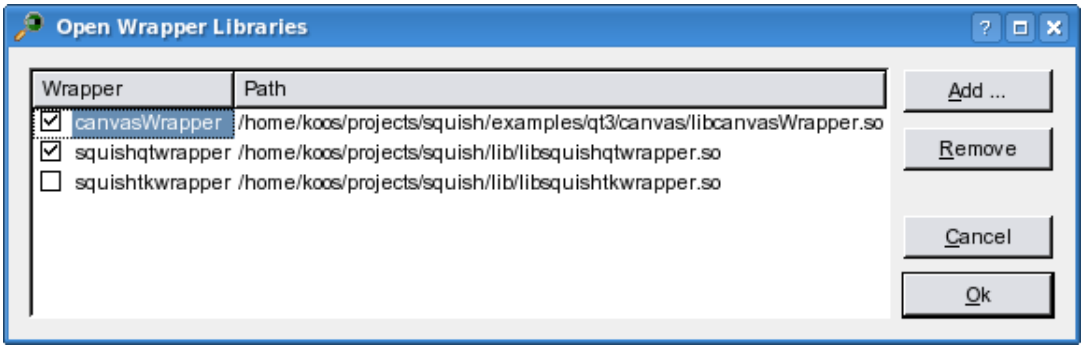
7.9.1 Opening a Wrapper Library

The Wrapper Explorer is started by choosing File → Explore Wrapper Library... in the Squish IDE. It will first open a dialog where you can select which wrapper library/libraries to explore.



Bindings created for your own AUT, can be added by clicking the Add... button. The file selection dialog which pops up will only show library files. However you still have to make sure you select only wrapper libraries. Their filename contains the string `Wrapper`.

After choosing Open the library will be added to the list in the Open Wrapper Libraries dialog.

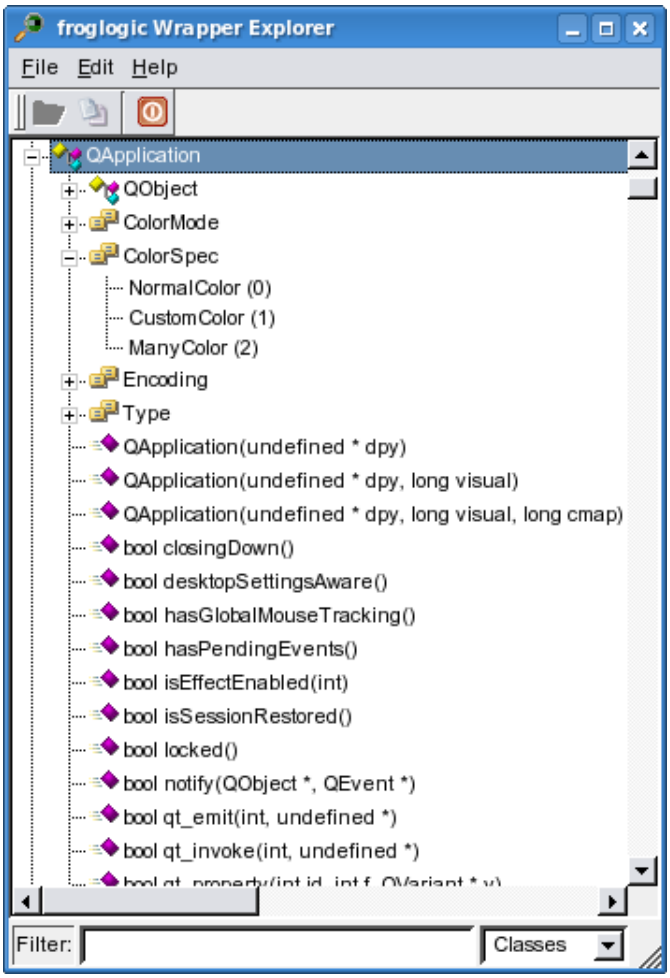


Wrapper libraries added will stay in this list for future use, or remove it with Remove.

Finally clicking Ok will load all checked libraries.

7.9.2 Exploring the Wrappers

After opening one or more libraries, you see the classes that are wrapped as a hierarchical tree.



Opening a class will show its parent class(es), declared enum(s), constructor(s), member function(s) and property(s) resp. if any declared.

For faster navigation, there is a Filter text field at the bottom of the application. When text is entered here, it will filter on the tree. If all entered text is in lowercase, the filter is case insensitive. In case Classes is selected in the bottom right list, the filter works only on class names. Use this for quickly finding a particular class. For fast navigation for class members, change the list in the right bottom to Members.

Selected items can be copied to the system's clipboard as text keeping its indention it has in the tree. Items become selected when clicked. Selection can be extended. On Windows™ one can select a second item by holding the **Ctrl** when clicking this second item. Holding the **Shift** will also select all items between the clicked one and the last selected item above it. Finally you can choose Edit → Select All to select all items.

Note
Child items are determined on first time expansion. Selecting all items will thus be different depending on which items are filled in.

7.10 Verification Points

Verification points are one of the central aspects of automated testing. One part is to automatically drive the AUT. The other part is to verify that the application reacted as expected. This is done using verification points which verify that property values, the screen output or similar is as expected.

In *Squish* it is possible to insert verification points via a point & click interface in the Squish IDE, or to code them manually via script statements. Both approaches are presented in this chapter.

7.10.1 Object Property Verifications

The most common type of verifications is to compare the value of an object's property with an expected value. For example, after a text was entered, one could compare that the line editor's text property returns the expected value.

7.10.1.1 Basic Properties

Object property comparisons can be inserted for all properties of QObject based objects. This includes all widgets of the Qt toolkit and custom widgets based on QWidget.

7.10.1.2 Item Views and Menus

Squish provides also extensions so items of view classes (such as QListView or Qt 4's item views) and menus can be accessed like any other QObject's property.

This means a QListView or QPopupMenu, for example, have an `item_N` child for each list view or menu item. This item in return has properties such as the item's text and the item's state (enabled, disabled, checked, etc.). Items can have child items again (such as nested list view items).

This allows to access view items and menu items from Squish IDE's point & click interface. So verification points which check if, for example, a list view contains the expected items or a menu's item is checked as expected can be easily inserted.

7.10.2 Screenshot Verifications

This section discusses how to create and work with screenshot verification points.

7.10.2.1 Creating Screenshot Verifications

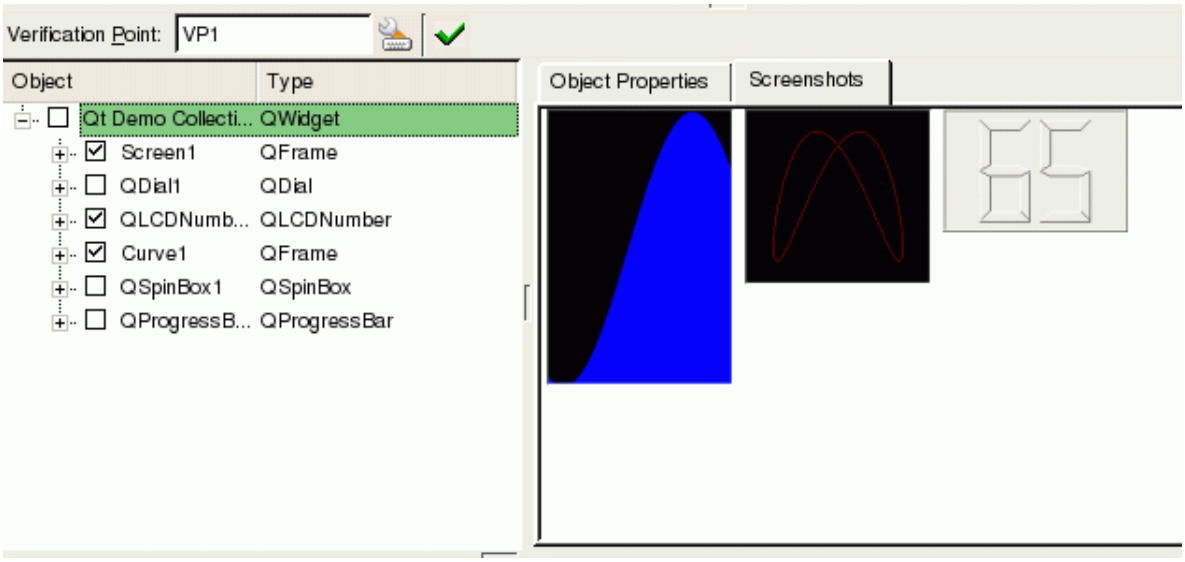
Another common type of verifications is to compare the the visual output of a widget (or set of widgets) with an expected image. This means a screenshot of the desired widget(s) is taken and compared to a screenshot which has been taken when the test was created.

Generally it is advisable to use object property comparisons whenever possible to compare actual values and not screen output. The reason for this is that due to different resolutions and platforms, esp. changing fonts often lead to differing images while the actual content would be correct. This way tests will fail even though the output is correct.

A case where screenshot comparisons are a good solution is when comparing that a graph or diagram is drawn correctly.

Inserting such a screenshot verification point in *Squish* is just as easy as inserting an object property comparison. The steps to it are the same as described in [?]. Only instead of selecting object properties, in the right pane of Spy you have to click on the tab page Screenshots.

Now when marking objects in the left pane as checked, a screenshot will be taken and a preview of it will be placed in the right pane:



When now inserting the verification point, the taken screenshots will be saved as the expected results. When later executing this verification point in the test script, screenshots of the same widgets will be taken during the test run and compared against the saved screenshots.

In case a screenshot verification fails, *Squish* saves the new and differing image for later inspection. The Squish IDE also allows to view the differences between the expected and new image in the test log view via Show Image Diff.

The differences are shown in different ways such as a subtraction of the images, a side-by-side view and an animation where the expected and new result are displayed quickly one after the other in a loop making differences visible. It then is also possible to accept the new image as new expected result.

7.10.2.2 Image Masks

Squish provides a tool which allows to make screenshot verifications more robust. This is done using image masks which allow to specify areas in the screenshot which should be ignored or respected in the comparison.

Image masks can be created and modified for a screenshot in the verification point editor in the Squish IDE There it is possible to insert, modify and remove positive and negative masks.

7.10.3 Scripting

While inserting verification points can be done without writing a single line of script code, it is also possible to code them in script. This is specially useful when a verification point needs to be more flexible, such as iterating over all items of a list view.

It never hurts to initially create all verification points using the point & click interface in the Squish IDE. Such a verification point is always saved as an external file, and a script line such as

```
test.vp("VP-name")
```

To convert such a statement into pure script code, you can right click on this statement in the Squish IDE and choose Scriptify Verification Point in the context menu. This way, for example, the line

```
    test.vp("VP1")
```

becomes

```
    # Verification Point 'VP1'
    test.compare(findObject(":Addressbook.ABCentralWidget1.QListView1").childCount, 1)
    test.compare(findObject(":Addressbook.ABCentralWidget1.QListView1.item_1").text0, "Max ↩
        ")
    test.compare(findObject(":Addressbook.ABCentralWidget1.QListView1.item_1").text3, " ↩
        max@mustermann.net")
```

```
    test.compare(findObject(":Addressbook.ABCentralWidget1.QListView1.item_1").text2, " ↩
        Bakerstreet 55")
    test.compare(findObject(":Addressbook.ABCentralWidget1.QListView1.item_1").text1, " ↩
        Mustermann")
```

To scriptify all verification points of a file, you can right click somewhere into the file and choose Scriptify All Verification Points.

For more details about the script API for test statements, have a look at Section 7.1.2. Here is a small example in JavaScript which shows a dynamic verification point which iterates over all items of a Qt list view and checks that the item text is not empty. Such a verification point can't be created using the Squish IDE since the number of items is unknown and can be different on each test run. Such flexibility is only provided by using script languages:

```
function main() {
    # get the list view
    var listview = findObject("Addressbook.addressList");

    # get the first item of the list view
    var item = listView.firstChild();
    var cnt = 0;

    # loop as long as there is a valid item
    while (!isNull(item)) {
        cnt++;
        # check that the text of column 0 is not empty
        test.compare(item.text(0).isEmpty(), false);
        # go to next item
        item = item.nextSibling();
    }
    # store in the test results the number of checked items
    test.log("Checked " + cnt + " items");
}
```

7.11 Shared Data and Scripts

This chapter discusses how to split tests into multiple files and how to share, access and use script and data files.

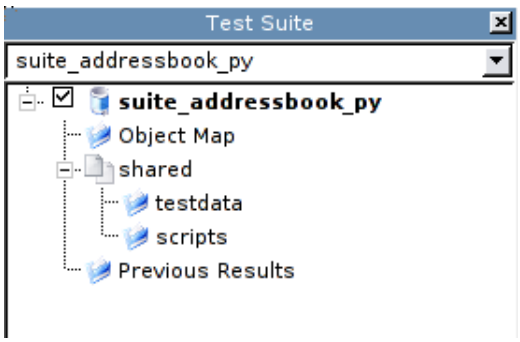
7.11.1 Storing and Locating Data- and Scriptfiles

Each test case contains a default script file called `test.lang` (where *`lang`* can be one of `py`, `js`, `pl` or `tcl`).

When creating more sophisticated tests, often many test cases may use common functionality. In *Squish* this is done by putting such functionality into a function in a separate script file which then can be used by multiple test cases.

Shared scripts are located in the `shared/scripts` folder of the test suite.

To create such a script file, just right click on scripts under shared in the test suite view and choose New Script



Then enter the name of the new file and press Enter. After that you can edit the new script file in the Squish IDE.

If you want to import an existing script into the shared script's folder of the test suite, just choose Import Script in the context menu instead.

For command-line users

Creating a shared script on the command line or using a different editor is just as easy: First create the directories `shared` and `scripts` in the test suite's directory and then create the new script file using the editor or your choice in the new location.

Sharing between test suites

In some cases it might be desirable to share script files between test suites even. To do this, you can put the script file in any directory and have the environment variable `SQUISH_SCRIPT_DIR` point to that directory.

After a shared script file has been created, it needs to be included by the test script of the test case.

To locate a script file, the `findFile` function can be used. The first argument to it is the type of file. In the case of a script file, we will use *script*. The second argument is the filename of the script. This function will search all possible locations (`shared/scripts` and `SQUISH_SCRIPT_DIR` and return the path of the file.

To include and evaluate a script file, the `source` function will be used.

Example 7.1 Include a Shared Script File

As an example we assume that we created a function `enterAddresses` which enters some addresses into the address book AUT. We have put this function into the file `enter_addr.<lang>` under `shared/scripts`.
To include and call this function in the main script file of a test case, we would write:

```
def main():
    ....
    source(findFile("scripts", "enter_addr.py"))
    enterAddresses()
    ....
```

```
function main() {
    ....
    source(findFile("scripts", "enter_addr.js"));
    enterAddresses();
    ....
}
```

```
sub main {
    ....
    source(findFile("scripts", "enter_addr.pl"));
    enterAddresses();
    ....
}
```

```
proc main{} {
    ....
    source [findFile scripts "enter_addr.tcl"]
    enterAddresses
    ....
}
```

Test scripts are a special kind of test data. You can also use generic test data of any type such as a table with input and expected output data for example.

If a test data file should be only used by a single test case, it can be put into the `testdata` directory of the test case. If the test data should be shared so it can be used by multiple test cases, the data file can be put into the `shared/testdata` folder of the test suite

Additionally it is possible to create sub directories under the `testdata` directories to structure the test data.

Creating or importing a test data file is done as described above for the script files, just that the `testdata` directory is used.

To retrieve the filename of a test data file in a script, the `findFile` function can be used by passing *testdata* as first argument to it.

7.11.2 Data-Driven Testing

Data-driven testing is an approach where the test data (input and expected output) is separated from the test script code which contains the logic of the test. This means that the test data is read from a file or a database which contains data records. The test or a certain part of it is run for each data record.

The rationale behind this approach is to allow modifying a test (e.g. adding new data which the test should be run for) without having to modify the test's logic (test script). This enables to have test engineers without coding skills work on the content of a test (the data), while engineers with coding skills create the test logic (scripts).

Squish offers a special API to utilize test data for data-driven testing. Creating test data files has been discussed in the previous section. This section presents the script API to read and interpret test data.

Test data always contains data in a tabular format. By default *Squish* can read tab separated value files (TSV). In such a file, records are separated by new lines and fields are separated by tabs. The first record is used to describe the columns. Here is an exemplary TSV data file:

```
First Name  Last Name Address E-Mail  Number
Max Mustermann  Bakerstreet 55  max@mustermann.net  1
John  Kelly Rhodeo Drv. 678 jkelly@acompany.com 2
Joe Smith Queens Blvd. 37 joe@smith.com 3
```

This first line describes the table columns (First Name, etc). The other lines contain the actual data records. Fields are separated by tabs.

To open that file, read each record and print out the values, we can use the following Python code:

```
    for address_record in testData.dataset("addresses.tsv"):
        firstName = testData.field(address_record, "First Name")
        lastName = testData.field(address_record, "Last Name")
        address = testData.field(address_record, "Address")
        email = testData.field(address_record, "E-Mail")

        print("First Name:" + firstName)
        print("Last Name:" + lastName)
        print("Address:" + address)
        print("E-Mail:" + email)
```

In JavaScript the same looks like this:

```
    var set = testData.dataset("addresses.tsv");
    for (var address_record in set) {
        firstName = testData.field(set[address_record], "First Name");
        lastName = testData.field(set[address_record], "Last Name");
        address = testData.field(set[address_record], "Address");
        email = testData.field(set[address_record], "E-Mail");

        print("First Name:" + firstName);
        print("Last Name:" + lastName);
        print("Address:" + address);
        print("E-Mail:" + email);
```

Finally the same can be done in Tcl:

```
    foreach address_record [testData dataset "addresses.tsv"] {
  set firstName [testData field $address_record "First Name"]
  set lastName [testData field $address_record "Last Name"]
  set address [testData field $address_record "Address"]
  set email [testData field $address_record "E-Mail"]

        puts "First Name: $firstName"
        puts "Last Name: $lastName""
        puts "Address: $address""
        puts "E-Mail: $email""
```

So using `testData.dataset` (`(filename)`) a test data file is opened and a reference to the data file object is returned. The file is located using `findFile` internally, so for locating the file the rules as described in the previous chapter apply.

Using a `for` loop it is possible to iterate over each record of the data set. Using `testData.field` it is possible to read the value of a specified fields of the current record.

Using this API it is possible to execute script code for all records of a data file and use this external data instead of hardcoded data in the script.

Finally it is also possible to import existing test data files (Excel and TSV files). You can either import the file using the context menu by right-clicking on the testdata item in the Squish IDE's test suite view or by just copying the file into the test data directory.

7.11.3 Using Test Data in the AUT

So far this chapter only talked about using test data in test scripts. Another usage case of test data is to provide files which the AUT itself should use or to retrieve files which the AUT created for verification.

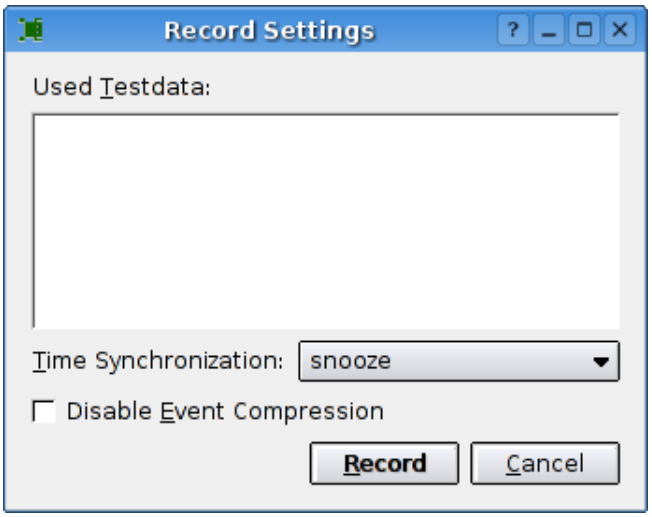
As a first example let's assume that the addressbook AUT should load a file called `customers.adr` before executing further actions. The best way to store the `customers.adr` file is again the test case's or test suite's `testdata` directory.

To allow the AUT to open this file without the need to specify the path to this file (which can be different depending on the machine the test runs on), it is possible to copy the data file into the AUT's current working directory using `testData.put`. Additionally *Squish* will automatically clean up the data (remove the file) after the test completed.

Here is an example in Python which copies the file into the AUT's current working directory and then instructs the AUT to open this file:

```
def main():
    ....
    testData.put("customers.adr")
    findObject("Addressbook").fileOpen("customers.adr")
```

Often it is already necessary to have this file available in the AUT's current working directory when recording a test case. Given that the file already exists in a `testdata` directory, when choosing Test Suite → Record Test Case it is possible to choose test data files which should be copied into the AUT's working directory in the record settings dialog:



When recording a test case this way, the `testData.put` will be generated automatically in the recorded script.

Other case is to verify a file which has been generated by the AUT. Let's assume the addressbook AUT create a file `newcustomers.adr` after saving the addresses. In the script we want to compare this file with the existing `customers.adr`.

To retrieve the file from the AUT's current directory, we can use `testData.get`. Let's look the the following Tcl script as an example:

```
proc main {} {
    ....

    # call saveAs in the AUT
    set mainwindow [findObject "Addressbook"]
    invoke $mainwindow saveAs "newcustomers.adr"

    # retrieve saved file
    testData get "newcustomers.adr"

    # open and compare files
    set f1 [open [findFile testdata "customers.adr]]
    set f2 [open [findFile testdata "newcustomers.adr]]
    test compare [read $f1] [read $f2]

    ....
}
```

Additionally, using `testData.exists` it is possible to verify if a file exists in the AUT's working directory. Using `testData.delete` it is possible to remove a file from the AUT's working directory.

7.12 Java Extension API for custom widgets

7.12.1 Introduction

Squish understands all AWT/SWT widgets that are normal components, however some custom widgets can choose to represent its children as non AWT/SWT components. For example, imagine a canvas with items on it, these can be just normal Java objects. Or take a Gantt diagram, it can decide to render the contents itself, with the data stored in some model. Using the Java extension API, you can teach *Squish* more about your AWT/SWT application and expose more functionality.

In order to tell *Squish* what functionality to expose, there is the concept of the Inspectable. An Inspectable is a class that handles questions *Squish* has for a certain class type. For example the Inspectable could handle canvas item types, and answer questions like, what are your bounds, what is your parent and return all the children. The exact questions are listed in the Inspectable interface. Once registered, *Squish* queries each Inspectable whether its knows the type it is handling at the moment. If so, it

uses its interface. This allows *Squish* to know for example when determining what object is hit after a mouse click, even for non AWT/SWT components, because the matching Inspectable does know which of its children is being hit.

All the Inspectables for the types you want to teach *Squish* about are placed together in what we call a wrapper. A clean way to do this is to put these classes in a separate jar, though if preferred they can be added to the existing jar of the application. The canvastest example shows how to do the former.

The following added functionality is possible for non AWT/SWT components by using the extension API:

- Allow identification in the scripts.
- Use verification points on them.
- Show them in the overall Spy hierarchy.
- Pick them using the object picker.

7.12.2 Custom canvas Example

To show how a wrapper to extend an existing AWT application works, we use the canvastest example which is shipped together with the Java package. You can find it in squish/examples/java/canvastest.

The source files CanvasTest.java, MyCanvas.java, MyCanvasItem.java, MyCanvasGroup.java, MyCanvasShape.java, MyRectCanvasItem.java, MyCircleCanvasItem.java are part of a AWT application that provides a simple canvas with multiple items.

To teach *Squish* about the application, the source file MyCanvasFactory.java has been made. It tells *Squish* which items to expose as *Squish* objects, so that they can be picked and inspected in the Spy, and will be identified when clicked on during recording.

To really enable the wrapper for the canvas one extra step has to be made; the location of the wrapper has to be made known through the included ini file like this:

```
squishserver --config setConfig CanvasTest.jar /abs/path/to/squish/examples/canvastest/ ↩
    Extension.ini
```

To try the AUT and the wrapper we made an existing testsuite. You can find it in squish/examples/java/suite_canvastest_js.

7.12.3 Extension API documentation

See Part I for an overview of the extension API.

7.12.3.1 Creating the jar file

The manifest file must contain an `Extension` entry that has the class as value having the `public static void init( InspectableRegistry registry )` method. This function is the entry point for the extension and should register the factory class in the `registry`.

For applications that load their classes with one or more classloaders that differ from the system classloader, an entry `LoadAtClass` must be added. All RCP fall in this category.

The value of this entry is the class that will trigger the extension registration, as soon as that class gets loaded by a classloader. This may require some trail and error, because all the fields of the factory classes that gets registered, must be known by the classloader allready.

Chapter 8

Reference Guide

8.1 Script API

This chapter documents the script APIs which can be used by *Squish* test scripts. First the differences between the supported scripting languages will be discussed.

8.1.1 Equivalent Script API

Squish supports the same APIs in all scripting languages. Here we will discuss the differences between the scripting languages.

This table shows equivalent script functions for the most basic tasks. It gives merely a quick overview to find out what the syntactical differences are.

All entries assume an example C++ class C defined as

```
   class C
   {
   public:
       C();
C(const char *s);

       void doA(bool b);
       static void doB();

       int p;
       ...
   }
```

and a class instance named `c`.

Feature	Tcl	Python	JavaScript	Perl
Construct a default object	`setc[constructC]`	`c=C()`	`varc=newC();`	`my$c=C->new();`
Construction with argument	`setc[constructC"apple"]`	`c=C("apple")`	`varc= newC("apple");`	`my$c=C-> new("apple");`
Get a property	`setx[propertyget$cp]`	`x=c.p`	`x=c.p;`	`$x=$c->p;`
Set a property	`propertyset$cp10`	`c.p=10`	`c.p=10;`	`$c->p(10);`
Call a member function	`invoke$cdoAtrue`	`c.doA(True)`	`c.doA(true);`	`$c->doA(1);`
Call a class (static) function	`invokeCdoB`	`C.doB()`	`C.doB();`	`C::doB();`

Feature	Tcl	Python	JavaScript	Perl
Test for equality	`testcompare[propertyget$cp] 2`	`test.compare(c.p,2)`	`test.compare(c.p,2);`	`test::compare($c->p, 2);`
Compare wrapped string-like object with native string	`compare$s"Max"`	`s=="Max"`	`s=="Max"`	`$seq"Max"`
Convert to native string	`sets[toString$val]`	`s=str(val)`	`s=String(val);`	`$s=$val.""`
Send key presses	`invoketype"namewidget""Max"`	`type("namewidget", "Max")`	`type("namewidget", "Max");`	`type("namewidget", "Max");`
Native boolean values	`true,false`	`True,False`	`true,false`	`-(use1and0)`

In the following sections the APIs will be described generally. Examples will be given for Python. Using the table above it will be easy to convert the respective code to any other scripting language.

8.1.2 Squish API

This chapter introduces *Squish*'s API modules. These extensions are useful for controlling the test framework and the AUT from test scripts.

8.1.2.1 Object Hierarchy

object.exists(objectName);

Returns whether the object with the name *`objectName`* exists.

findObject(objectName);

Finds and returns the object via the path specified by *`objectName`*. The object name contains dots to represent the object hierarchy. The names of the single objects are not necessarily the QObject names, but a unique name. This can also be another unique property like the caption, a groupbox title, etc. In the worst case this is the class name plus a number (e.g. QLineEdit5).

object.children(object);

Returns a list of child objects of *`object`*. The list will be of type Array, Tuple or similar depending on the scripting language. See the examples below for possible ways to access the result.

```
    JavaScript

var obj = findObject(":MyWidget");
var children = object.children(obj);
// iterate over array
for (var i in children) {
    var child = children[i];
    ...
}
```

```
    Python

obj = findObject(":MyWidget")
children = object.children(obj)
# iterate over tuple
for child in children:
    ...
```

```
    Perl

my $obj = findObject(":MyWidget");
my @children = object::children($obj);
# iterate over array
foreach $child (@children) {
    ...
}
```

```
    Tcl

set obj [findObject ":MyWidget"]
set children [object children $obj]
foreach child $children {
    ...
}
```

object.properties(object);

Returns a key/value map with all properties of *object*. The return type is an array in JavaScript, a dict in Python and a hash in Perl. The Tcl syntax is slightly different: the result is written into an array whose name is specified as an additional argument. The behavior of referencing an array in more than one call is undefined so better invoke array unset in between. See below for examples.

```
    JavaScript

var w = findObject(":MyWidget");
var props = object.properties(w);
for (var n in props) {
    test.log("Property " + n + ": " + props[n]);
}
```

```
    Python

w = findObject(":MyWidget")
props = object.properties(w)
for key, value in props.iteritems():
        test.log("Property " + key + ": " + str(value))
```

```
    Perl

my $w = findObject(":MyWidget");
my %props = object::properties($w);
while (($key, $value) = each(%props)) {
        test::log("Property: " . $key . ": " . $str);
}
```

```
    Tcl

set w [findObject ":MyWidget"]
object properties $w props
foreach key [array names props] {
  test log "Property $key: [toString $props($key)]"
}
```

8.1.2.2 Constructors, Functions and Properties

Squish objects blend in with native objects of the scripting language quite smoothly. This means that you can use standard language features to construct objects of the wrapped types, invoke member functions or get, set or iterate over properties on these objects.

The best way to show how that works is to look at the example code below:

```
# create an object of type QPoint
p = QPoint(10, 20)

# read a property of a widget
x = mywidget.x

# set a property of a widget
mywidget.y = 33

# call a member function
mywidget.setCaption("My Widget")

# call a static function
QApplication.setOverrideCursor(Qt.busyCursor)
```

8.1.2.3 Debugging

isNull(object);

Returns True (1) if *object* is null (a NULL pointer in C), False (0) otherwise.

typeName(object);

Returns the class name of the *object*.

8.1.2.4 Conversions

cast(object, type);

Casts the *object* to the type *type* and returns a reference to that object. *type* can either be specified by name (as a string) or via a type object.

This modifies the *object* itself. If the cast fails, 0 is returned.

8.1.2.5 Verification Functions

This set of functions implements functions that can be used to verify that the AUT's state is as expected during a test run.

test.verify(condition, [info]);

Verifies that *condition* is True (1). If the condition is True a test result of type PASS is added to the test log, otherwise a test result of type FAIL is added.

Returns True on PASS and False on FAIL.

You may use the optional *info* parameter to specify an arbitrary string (comment, bug number, test plan reference etc.) that will be added to the test result.

test.compare(val1, val2, [info]);

Compares *val1* and *val2* and if they are equal a test result of type PASS is added to the test log, otherwise a test result of type FAIL is added.

Returns True on PASS and False on FAIL.

You may use the optional *info* parameter to specify an arbitrary string (comment, bug number, test plan reference etc.) that will be added to the test result.

test.xverify(condition, [info]);

This function is used for expected failures in test scripts.

Verifies that *condition* is not True (1). If the condition is False a test result of type XFAIL (eXpected FAILure) is added to the test log, otherwise a test result of type XPASS (uneXpected PASS) is added.

Returns the boolean value of *condition*.

You may use the optional *info* parameter to specify an arbitrary string (comment, bug number, test plan reference etc.) that will be added to the test result.

test.xcompare(val1, val2, [info]);

This function is used for expected failures in test scripts.

Compares *val1* and *val2* and if they are equal a test result of type XFAIL (eXpected FAILure) is added to the test log, otherwise a test result of type XPASS (uneXpected PASS) is added.

Returns True if *val1* and *val2* are equal.

You may use the optional *info* parameter to specify an arbitrary string (comment, bug number, test plan reference etc.) that will be added to the test result.

test.exception(code, [info]);

If executing *code* throws an exception, a test result of type PASS is added to the test log, otherwise a test result of type FAIL is added.

Note
code must be of type string.

You may use the optional *info* parameter to specify an arbitrary string (comment, bug number, test plan reference etc.) that will be added to the test result.

test.log(message, [detail]);

test.warning(message, [detail]);

test.pass(message, [detail]);

Note
As `pass` is a reserved word in Python, the Python version of this functions is called `test.passes()`.

---

test.fail(message, [detail]);

test.fatal(message, [detail]);

Adds a test result of type PASS, FAIL, FATAL, LOG or WARNING with the *message* and an optional *detail* to the test log.

test.vp(name, [record]);

Executes the verification point *vp*. If the verification point references a data set, *record* references the record of the data set which should be used.

Returns True on PASS and False on FAIL.

saveDesktopScreenshot(filename);

Grabs an image of the screen the AUT is running on and saves it to disk. The filename should have a suffix denoting the image format to be used. An exemplary Python statement:

```
saveDesktopScreenshot("screen.png")
```

8.1.2.6 Test data Handling

findFile(where, filename);

Looks for the file *filename*. If the first argument is "scripts", the search begins in the test case's scripts directory and if it isn't found there, the shared scripts directory of the test suite is searched. If the first argument is "testdata", the search begins in the test case's testdata directory, and if the file isn't there, then the test suite's shared testdata directory is searched.

If the file is found the absolute file name is returned, otherwise an error is thrown.

testData.put(filename);

Copies the test data file *filename* from the test case's testdata directory or the test suite's shared testdata directory into the AUT's current working directory.

testData.get(filename);

Copies the file *filename* from the AUT's current working directory to the test case's testdata directory.

testData.remove(filename);

Removes the file *filename* from the AUT's current working directory.

testData.exists(filename);

Returns whether the file *filename* exists in the AUT's current working directory.

testData.dataset(filename);

testData.field(datasetrecord, fieldname|fieldindex);

These functions are used for data-driven testing. They allow opening a *dataset* (e.g. a TSV file) and retrieving fields from the *datarecord* by *fieldindex* or *fieldname*.

testData.fieldNames(datasetrecord);

This function returns a tuple containing the field names of the specified record.

testData.create(where, filename);

This function returns a valid filename you can write to. The file will be located in the shared test data directory when *where* is "shared" or in the local test data directory when *where* is "local".

This function should be used when you are running a script in a learning mode to create test data dynamically.

8.1.2.7 Object Map Handling

objectMap.add(object);

Adds *object* to the object map.

objectMap.symbolicName(object|realName);

Returns the symbolic name for the given *object*, or the object referenced by *realName*.

objectMap.realName(object|symbolicName);

Returns the real name for the given *object*, or the object referenced by *symbolicName*.

objectMap.symbolicNames();

Returns a tuple containing all mapped symbolic names. The element order is not defined.

objectMap.load(filename);

Load an object map from a given *filename*. The current object map is unloaded.

8.1.2.8 Application Context

startApplication(aut, [host], [port], [timeout]);

Starts the specified application and returns a reference to its context. *aut* must be an application whose path is already registered to squishserver. In addition to the application name command line parameters can be specified in the string passed to this function.

Optionally, as second and third parameter a *host* and *port* can be passed to this function. This way instead of connecting to the default host and port (as specified in the Squish IDE's settings or squishrunner's command line), a connection to squishserver on the specified host listening to the specified port will be established. This allows to control multiple applications on multiple computers from one test script.

Also optionally, as fourth parameter a timeout (in seconds) can be passed to specify how long *Squish* should wait for the application to come up before throwing an error. This value overrides squishrunner's default AUT timeout.

setApplicationContext(ctx);

This function can be used to switch the current application context. The *ctx* handle is is either the one returned from a setApplicationContext(), defaultApplicationContext() or applicationContextList() call. If the argument is omitted the default context is activated.

defaultApplicationContext();

Returns a handle to the default application context.

currentApplicationContext();

Returns a handle to the current application context.

applicationContextList();

Returns a list of handles to all existing application contexts.

The ApplicationContext object provides the following properties and functions:

- `name` Name of the application.
- `commandLine` Complete command line the application was started with.
- `pid` The process identifier of the application.
- `cwd` The current working directory of the application.
- `isRunning` Returns whether the application is still running.
- `startTime` Returns the time when the application was started.
- `usedMemory` Amount of memory used by the application (*This property is only available in conjunction with the Squish memory module add-on*).
- `isFrozen(timeout)` Returns False if the application answers before the timeout (in seconds) expired.
- `readStdout()` Reads everything which has been written to STDOUT by the application since the last call to readStdout() and returns it.
- `readStderr()` Reads everything which has been written to STDERR by the application since the last call to readStderr() and returns it.

8.1.2.9 Event Handlers

installEventHandler([object|class], event, handlerFunction);

Installs an event handler on an object *class*, an *object* or globally (if no *class* or *object* is specified). The script function *handlerFunction* will be called when the event of the type *event* occurs.

event can be any event type such as QKeyEvent (Qt), ButtonPressed (XView, Tk), etc. *Squish* adds the following convenience event types:

- *MessageBoxOpened*
- *DialogOpened*
- *MainWindowOpened*
- *ToplevelWidgetOpened*
- *Crash*

8.1.2.10 Information

`squishinfo.major`

A numeric property holding the *Squish* major version number.

`squishinfo.minor`

A numeric property holding the *Squish* minor version number.

`squishinfo.patch`

A numeric property holding the *Squish* patch level version number.

`squishinfo.version`

A numeric property holding a unique number representing the *Squish* version. The number is made up of a hexadecimal representation of major, minor, patch and release level. The individual values are shifted and packed into a single 4-byte number. The release levels Alpha, Beta and Final are denoted by the values 0xAA, 0xBB and 0xFF, respectively. As an example, 0x030001FF stands for the final 3.0.1 release.

This kind of representation looks odd on first sight but simplifies version tests in script code. Here's some JavaScript sample code that does a single comparison for conditional code:

```
if (squishinfo.version >= 0x030100AA) {
   // code meant for version 3.1.0-alpha or higher
}
```

`squishinfo.version_str`

A property of type string holding the human-readable *Squish* version number, e.g. "3.0.0-alpha".

`squishinfo.settingsGroup`

A property that denotes the currently active settings group of the test.

`squishinfo.testCase`

A property that returns the path to the current test case.

8.1.2.11 Test Settings

`testSettings.logScreenshotOnFail`

`testSettings.logScreenshotOnError`

Above properties control whether test failures and errors will automatically trigger a screenshot of the current desktop to be taken. By default, this feature is off. To activate it assign a value evaluating to the True boolean value.

Example: Let's assume that we ocassionally experience test failures when verifying the state of a specific control in our nightly tests. We cannot explain the cause of the problem but suspect that the overall state of our application's GUI is broken due yet unknown external factors. To provide development with first information via a visual inspection we enable automatic screenshot capturing for only this sequence of tests.

JavaScript

```
testSettings.logScreenshotOnFail = true;
// ... perform test ...
testSettings.logScreenshotOnFail = false;
```

Python

```
testSettings.logScreenshotOnFail = True
# ... perform test ...
testSettings.logScreenshotOnFail = False
```

Perl

```
testSettings->logScreenshotOnFail(1);
# ... perform test ...
testSettings->logScreenshotOnFail(0);
```

Tcl

```
testSettings set logScreenshotOnFail 1
# ... perform test ...
testSettings set logScreenshotOnFail 0
```

testSettings.setWrappersForApplication(application, wrapperList);

Associates a list of wrappers with the specified application. The list has to include at least one of the main toolkit wrappers like Java. Optionally, it can include wrappers created for an application specifically (not available in all editions).

testSettings.getWrappersForApplication(application);

Returns the list of wrappers associated with the specified application.

Here is an example that sets the Qt toolkit wrapper as well as a custom application wrapper:

JavaScript

```
testSettings.setWrappersForApplication("MyApp", ["Qt", "CanvasWrapper"]);
startApplication("MyApp");
```

Python

```
testSettings.setWrappersForApplication("MyApp", ("Qt", "CanvasWrapper"));
startApplication("MyApp")
```

Perl

```
testSettings->setWrappersForApplication("MyApp", ("Qt", "CanvasWrapper"));
startApplication("MyApp");
```

Tcl

```
testSettings setWrappersForApplication MyApp { Qt CanvasWrapper }
startApplication "MyApp"
```

8.1.2.12 Miscellaneous

source(filename);

Reads and evaluate the contents of *filename*. Use this function to parse shared script files.

snooze(secs);

Sleep for the specified nominal number of seconds. Fractions of seconds can be specified as well by using decimal numbers. The effective delay will depend on the current *snoozeFactor*.

waitFor(condition, [timeout]);

Evaluates *condition* until it becomes true or the waiting times out after the number of specified milliseconds. The *timeout* parameter is optional. This is useful if you want to synchronize your script execution to a certain state in the AUT.

waitForObject(objectname, [timeout]);

Waits until the object with the name *objectname* exists and is accessible. This is useful if you want to synchronize your script execution. By default, it waits for maximum 20 seconds. With the optional *timeout* parameter a different timeout (in milliseconds) can be specified.

waitForObjectItem(objectname, itemtext, [timeout]);

Waits until the object with the name *objectname* exists and is accessible and the list of items contains *itemtext*. This is an extension to `waitForObject` for widgets having items. By default, it waits for maximum 20 seconds. With the optional *timeout* parameter a different timeout (in milliseconds) can be specified.

waitForApplicationLaunch();

Waits for a new application, which *Squish* will hook into, to start. This is used by *Squish* for test scripts which access multiple applications which are started by the AUT.

sendNativeEvent(window, sequence);

Send a *sequence* of low-level native events to the specified window with the name *window*. This function may be of use when interacting with windows from another application or controls implemented with a foreign toolkit. For interaction with standard application controls use of type() and other function is recommended.

Example:

```
sendNativeEvent("Login", "username");
```

Currently, the only supported type of events are simple keystrokes. Support for complex keystrokes and mouse events might be added in a future version. Please contact technical support to voice your demand.

nativeType(keys);

Simulates user-input using the operating system facilities. One Windows, this can be used to interact with native Windows message boxes for example. The keyboard input will be sent to whatever control has the keyboard focus.

This function can also simulate keyboard input of special keys. Finally, it's also possible to send combinations of key presses using this function. To do so, separate the keys to be pressed with '+' signs. Here are some examples of a few different key combinations:

```
    JavaScript

nativeType( "Hello" );
nativeType( "<Return>" );
nativeType( "<Alt+F4>" );
nativeType( "<Alt+Tab>" );
nativeType( "<Ctrl+C>" );
```

```
    Python

nativeType( "Hello" )
nativeType( "<Return>" )
```

```
nativeType( "<Alt+F4>" )
nativeType( "<Alt+Tab>" )
nativeType( "<Ctrl+C>" )
```

```
    Perl

nativeType( "Hello" );
nativeType( "<Return>" );
nativeType( "<Alt+F4>" )
nativeType( "<Alt+Tab>" )
nativeType( "<Ctrl+C>" )
```

```
    Tcl

invoke nativeType "Hello"
invoke nativeType "<Return>"
invoke nativeType "<Alt+F4>"
invoke nativeType "<Alt+Tab>"
invoke nativeType "<Ctrl+C>"
```

Here's the complete list of currently supported special keys which (in addition to normal alphanumeric characters) can be used with the nativeType function:

- Shift
- Ctrl
- Alt
- Return
- Escape
- Backspace
- Tab
- Pause
- PageUp
- PageDown
- Home
- End
- Left
- Right
- Up
- Down
- Insert
- Delete
- Print
- NumPad0
- NumPad1

- NumPad2
- NumPad3
- NumPad4
- NumPad5
- NumPad6
- NumPad7
- NumPad8
- NumPad9
- MenuLeft
- MenuRight
- F1
- F2
- F3
- F4
- F5
- F6
- F7
- F8
- F9
- F10
- F11
- F12
- F13
- F14
- F15
- F16
- F17
- F18
- F19
- F20
- F21
- F22
- F23
- F24
- NumLock

- ScrollLock

setRecordMouseDrag(className, recordMouseDrag);

This function can be used in init files to disable the recording of mouseDrag() statements for certain widget classes. The class to disable this on is given by *className* and to disable recording of mouseDrag operations on that widget *recordMousedrag* should be set to false

The init files are Tcl scripts that are registered with the server and then read before starting any tests. How to register such an init script is explained in Section 8.2.1.4.

Example:

```
setRecordMouseDrag ScribbleArea false
```

Then we can tell Squish to execute this file at startup:

```
squishrunner --config addInitScript Qt <absolute_path>/myinit.tcl
```

8.1.3 Java™ Convenience API

In the following functions, a mouse button can be set with a convenient enumeration. The enumeration values are

Button	AWT/Swing	SWT	Value
No mouse button	AWT.NoButton	SWT.NoButton	0
Left mouse button	AWT.Button1	SWT.Button1	1
Middle mouse button	AWT.Button2	SWT.Button2	2
Right mouse button	AWT.Button3	SWT.Button3	3

Keyboard modifier keys don't have an enumeration and are toolkit dependent. The next table lists the mask values of them for your convenience

Modifier	AWT/Swing	Value	SWT	Value
No modifier		0		0
Shift	java.awt.Event.SHIFT_MASK	1	org.eclipse.swt.SWT.SHIFT	131072
Control	java.awt.Event.CTRL_MASK	2	org.eclipse.swt.SWT.CTRL	262144
Meta	java.awt.Event.META_MASK	4		
Alt	java.awt.Event.ALT_MASK	8	org.eclipse.swt.SWT.ALT	65536
Command			org.eclipse.swt.SWT.COMMAND	4194304

Note
In all functions below, *object* can be a reference to an object or the name of an object as string.

clickButton(object);

Clicks the button referenced by *object*.

type(object, text);

Types *text* into the editable widget referenced by *object*. If the text is surrounded by < and >, the contents is interpreted as a key combination, e.g *<Ctrl+Return>*. Note the SWT buttons from a ToolBar are selected with the clickItem function below.

activateItem(object, item);

Activates the menu item with the item text *item* in the popup menu or menu bar referenced by *object*.

mouseClick(object, x, y, state, button);

Sends a mouse click at position *x* and *y* with the button *button* and modifier state *state* to the widget referenced by *object*.

mouseClick(object, x, y, clicks, state, button);

Like the previous function with an extra *clicks* argument. This overload is for AWT/Swing. The additional argument represents the `MouseEvent.getClickCount` from Java™

doubleClick(object, x, y, state, button);

Sends a mouse double click at position *x* and *y* with the button *button* and modifier state *state* to the widget referenced by *object*.

mouseMove(object, x, y);

Moves the mouse to position (*x*, *y*) relative to top-left of widget referenced by *object*. This function can be useful if you want to trigger events that need a mouse at a certain position. Tooltips are an example of this, they only show when the mouse is at a certain area.

mouseDrag(object, x, y, dx, dy, state, button);

Performs a mouse drag operation on the widget referenced by *object* with a horizontal distance of *dx* and a vertical distance of *dy* with the button *button* and modifier state *state*. Parameters *x* and *y* indicate the relative distance from the widget top-left origin where the drag must start.

dragItemBy(object, x, y, dx, dy, state, button);

Moves the item referenced by *object* a horizontal distance of *dx* and a vertical distance of *dy* with the button *button* and modifier state *state*. Parameters *x* and *y* indicate the distance from the object top-left bounding box value where the drag was started and thus are relative values.

dragAndDrop(source, x1, y1, target, x2, y2, operation);

Starts a drag on the widget referenced by *source* at the position *x1* and *y1* and releases the drop on the widget referenced by *target* at the position *x2* and *y2*. *operation* is one of `DropNone`, `DropCopy`, `DropMove`, `DropLink`, `DropMoveTarget` or `DropDefault`. Parameters *x1*, *y1*, *x2* and *y2* indicate the distance from the object top-left bounding box value where the drag was started and thus are relative values.

clickItem(object, item, x, y, state, button);

Sends a mouse click at position *x* and *y* (in item's coordinates) with the button *button* and modifier state *state* to the item with the name *item* to the view widget referenced by *object*. Supported view widgets are List, Combo and ToolBar for the SWT toolkit and JList, JTable, JTree, List and Choice for the AWT/Swing toolkit.

clickTreeHandle(object, item);

Sends a mouse click to the expand/collapse locations of an *item*, which should be an item in a tree widget that has child items. The tree widgets is referenced by *object*. Supported tree widgets are Tree for the SWT toolkit and JTree for the AWT/Swing toolkit.

closeWindow(object);

Closes the `Window` (or `Shell` in SWT) referenced by *object* as if it was closed by its window system menu.

doubleClickItem(object, item, x, y, state, button);

Sends a mouse double click at position *x* and *y* (in item's coordinates) with the button *button* and modifier state *state* to the item with the name *item* to the view widget referenced by *object*. Supported view widgets are List, Combo and ToolBar for the SWT toolkit and JList, JTable, JTree, List and Choice for the AWT/Swing toolkit.

clickTab(object, tab);

Clicks on the tab with the text *tab* on the tab widget referenced by *object*.

clickTab(object, tab, x, y, state, button);

Only for the `org.eclipse.swt.custom.CTabFolder` and `javax.swing.JTabbedPane` variants. Clicks on the tab with the text *`tab`* at position *`x`*, *`y`* with mouse button *`button`* and with keyboard modifiers *`state`* pressed on the tab widget referenced by *`object`*.

grabWidget(object);

Takes a screenshot of the window referenced by *`object`* and returns it.

activateAction(objectname);

For SWT only. Activates the JFace action referenced by the object with the name *`objectname`*.

8.1.4 Java™ Hardcoded synthetic Properties

In addition to public Java™ class fields, sythetic properties generated from `getXxx`/`isXxx` functions, some additional properties are defined to better identify objects. See Section 7.6.2 for further reading on this subject.

Object window

Holds the toplevel window wherein this object resides.

Object container

Holds a parent Container/Composite that hold this object. These are typically tab pages and the menubar.

String type

Holds the class name of the object. Note that *Squish* replaces dots with underscores.

String caption

Holds the title, caption or text of this object if this object has typically a text to display.

Object leftWidget

Object aboveWidget

Holds the object left of this widget in the same logical parent Container/Composite. This should be used for objects that don't have a caption but often come with an accompagning widget, like edit boxes have often a label.

String buttonType

Holds the type of SWT buttons. Introduced because buttons with arrows are poorly described without this property.

String arrowDirection

Holds the direction of SWT buttons that have arrow style enabled.

String menuStyle

Holds the style of SWT menu, whether it is a menubar, menu popup or menu dropdown.

String firstItemText

Holds the first found text of child SWT ToolItem objects. If non found, then tooltip text is considered.

8.1.5 Python Notes

8.1.5.1 Modules

The extension modules are loaded automatically by internally executing the equivalent of the statements

```
import test
import testData
import object
import objectMap
from squish import *
```

before the script is executed. It's therefore not necessary to `import` them by yourself unless you are developing your own standalone module.

Note the drawback of above wild card import of the wrapped API: native Python functions like `int()` and `type()` will be shadowed as their names conflict with the wrapped `int` C type and the `type()` function used to send key events to the AUT. To access the built-in functions just `import` the `__builtin__` module and access them with their fully qualified name, e.g. `__builtin__.int()`.

8.1.5.2 Language Documentation

Besides the Squish extension API the full set of Python language features and modules is available for scripting. The Python Documentation page lists a few books and also has the current documentation available for download and online browsing.

8.1.6 Tcl Notes

8.1.6.1 Language Documentation

For documentation on the Tcl language and standard extension packages see the Tcl/Tk Documentation page on the Tcl home page. Next to links to the man pages you'll find a tutorial there.

8.1.7 JavaScript Notes

8.1.7.1 Language Documentation

Theoretically you could pick up *any* JavaScript book or online resource to learn about the language or look up a function. Unfortunately most authors don't draw a clear line between features that are part of the language proper and those specific to a web browser. When browsing such HTML programming documentation beware of any example code that operates on the `window` or `document` object.

When keeping this distinction in mind you'll still be able to draw the required information from a variety of books and online resources. Here are some online resources that we have collected for your convenience:

- Mozilla's Core JavaScript Reference
- Microsoft's JScript Language Reference at msdn.microsoft.com
- DevGuru JavaScript Quick Reference at www.devguru.com

8.1.7.2 Language Core

The JavaScript engine shipped with Squish is compliant with the ECMAScript language specification (ECMA 262 Edition 3). This corresponds with Netscape's JavaScript 1.5 and Microsoft's JScript 5.5. The language core that is defined in this standard encompasses operators, control structures, objects and other features commonly found in a scripting language. A handful of built-in classes provide functions for the most common data handling. These objects are:

- Object
- String
- Number
- Boolean
- RegExp
- Date

- Function
- Error

You may notice the lack of classes such as file or network I/O typically found in languages like Python and Tcl. This is because pure JavaScript (or rather ECMAScript) has been designed to be a small language that can safely be embedded into an application. It's meant to be extended by custom API specific to the embedding application, though. In the case of web browsers there are hundreds of functions and properties that allow manipulation of HTML documents via the DOM API for example. In the case of Squish we have added numerous functions specific to testing. We also added some general purposes classes described below. If you need anything else please file an enhancement request.

8.1.7.3 File Object

File.exists(path);

Returns true when a file or directory *path* does exist on the local hard disk; false otherwise.

File.remove(path);

Attempts to remove the file with *path* from the local hard disk. Returns true on sucess; false otherwise.

File.open(path, [mode]);

Will attempt to open file *path* and return a File instance. In case of failure an exception will be thrown. The *mode* can either be "r" for read access, "w" for write access or "a" for append mode where all data is written to the end of the file. If undefined it will default to "r".

Functions like `read()`, `write()` and `access()` can be called on the returned File instance.

read();

Reads in and returns the content of a file. The file content is assumed to be UTF-8 encoded.

Example:

```
var f = File.open("input.txt", "r");
var t = f.read();
inputField.setText(t);
```

readln();

Reads in and returns the next line of a file (without any end of line characters). When reading beyond the end of the file null is returned.

write(string);

Writes the *string* to the file using UTF-8 encoding. The file has to be opened in write-mode.

close();

Closes the file. Afterwards no data can be read or written anymore.

8.1.7.4 OS Object

`OS.name` The *name* property of the OS object holds the name of the operating system that squishrunner is running on. On Microsoft Windows systems the value will be "Windows". On Unix-like systems the value will be identical to the output of `uname -s`, i.e. "Linux", "Solaris", "Darwin" etc.

OS.system(command);

Executes *command* by calling the system command line shell. In case of a successfull execution the return status of the command will be returned. On errors -1 will be returned.

command can not only hold the name of an executable but also command line arguments to be interpreted by the application or shell.

Example: execute an external application and direct the output to a file:

```
var ret = OS.system("readresult.exe > output.txt");
if (ret != 0)
   test.warn("readresult failed");
```

Example: Send an e-mail on a Unix system

```
var msg = "This is a mail from Squish";
OS.system("echo '" + msg + "' | mail -s Subject bugs@example.com");
```

OS.capture(command);

Analogous to `OS.system()` will execute *`command`* from a system shell. The commands Stdout output will be read and returned.

Example:

```
var files = OS.capture("ls out/*.dat");
for (var f in files) {
   ...
}
```

OS.getenv(name);

Obtains the current value of the environment variable, *`name`*.

Example:

```
var homeDir = OS.getenv("HOME");
test.log("Current user's home directory: " + homeDir);
```

OS.setenv(name, value);

Inserts the environment variable *`name`* into the current environment. If the variable *`name`* does not exist, it is inserted with the given *`value`*. If the variable does exist, its value is overwritten.

Example:

```
var preferableEditor = "vim";
OS.setenv("EDITOR", preferableEditor);
```

OS.pause(msecs);

Pauses the script for the specified number of milliseconds. Unlike `snooze()` the delay is fixed and not influenced by the current *`snoozeFactor`* setting.

OS.listDir(path);

Returns the list of files and directories in the directory specified by *`path`*. The special directories `.` and `..` are excluded.

Here is an example that logs all files that were generated in an output directory:

```
    var files = OS.listDir("output");
    for (var f in files)
      test.log("Found generated file: " + files[f]);
```

OS.cwd();

Returns the current working directory.

Example:

```
var cwd = OS.cwd();
test.log("Current working directory: " + cwd);
```

OS.chdir(path);

Changes the working directory to the directory specified by *path*.

Example:

```
var dir = "output";
OS.chdir(dir);
```

8.1.7.5 XML Object

XML.parse(markup);

Attempts to parse the given string of XML markup. In case of success, a document node is returned which represents the root of the document ree. In case of an error, an exception is thrown. `node.isNull` Returns whether this node is a null node. `node.nodeName` Returns the name of the node. For elements, it's the tag name. For the document node, the string '<anonymous xml document>' is returned. `node.nodeType` Returns the type of the nodename of the node. Possible return values are:

- `XML.DocumentNode`
- `XML.ElementNode`
- `XML.CommentNode`
- `XML.TextNode`
- `XML.DeclarationNode`
- `XML.UnknownType`

`node.parentNode` Returns the parent node of this node. If there is no parent node (which is true for the document node) then a null node is returned. `node.firstChild` Returns the first child node of this node. If this node does not have any children, a null node is returned. `node.nextSibling` Returns the next sibling node of this node. This this node does not have any siblings, a null node is returned. `node.textContent` Returns the text contained within this node. Note that this function does not recursively collate the text of all child nodes (assuming that `aElem` always points to the 'a' element):

```
    // XML markup is '<a>Hello<a>'
    test.log( aElem.textContent ); // prints 'Hello' to the test log

    // XML markup is '<a><b>Hello</b></a>'
    test.log( aElem.textContent ); // prints an empty string to the test log
```

Note
This function can only be called on element nodes (where `nodeType` returns `XML.ElementNode`). Calling it on non-element nodes will cause an exception to be thrown.

node.hasAttribute(attrName);

Returns a boolean indicating whether the given node has an attribute called `attrName`.

Note
This function can only be called on element nodes (where `nodeType` returns `XML.ElementNode`). Calling it on non-element nodes will cause an exception to be thrown.

node.getAttribute(attrName);

Returns a string with the value of the attribute with the name `attrName`. If there is no such attribute with that name, the behaviour is undefined. If `attrName` is an empty string, an exception is thrown.

Note
This function can only be called on element nodes (where `nodeType` returns `XML.ElementNode`). Calling it on non-element nodes will cause an exception to be thrown.

8.1.7.6 SQL Object

SQL.connect(informationObject);

Attempts to create a connection to some SQL database. The information required is expected to be passed via an object whose properties are then interpreted. Here's a sample invocation showing how to connect to a MySQL server on the host 'dulsberg':

```
var c = SQL.connect( { Driver: "MySQL",
                       Host: "dulsberg",
                       Port: 1342,
                       Database: "mydatabase",
                       UserName: "test",
                       Password: "secretPhrase" } );
```

The settings have the following meanings:

Driver The driver to use when connecting to the database. Possible values are:

ODBC	ODBC Driver (includes Microsoft SQL Server)
Oracle*	Oracle Call Interface Driver
PostgreSQL	PostgreSQL v6.x and v7.x Driver
Sybase*	Sybase Adaptive Server
MySQL	MySQL Driver
DB2*	IBM DB2, v7.1 and higher
SQLite*	SQLite Driver
IBase*	Borland Interbase Driver

Note
The drivers marked with an asterisk (*) are not supported in *Squish* binary packages. This is usually because the license of the database vendor does not permit redistributing their client libraries without owning a license of their product. What you can do is to either try the ODBC driver, or build *Squish* yourself against a Qt library which includes support for your particular SQL database.

Host The host name (or IP address) of the computer on which the SQL database is installed.

Port The port on the remote computer to which the connection should be established. If this is omitted, the default port (depending on the driver) is used.

Database Specify the name of the database to which a connection should be established here.

DataSource This setting is only necessary when using the ODBC driver, in which case this is expected to specify the Data Source Name (DSN) to use.

UserName The user name to use when logging into the SQL server.

Password The password to use when logging into the SQL server. If omitted, an empty password is assumed.

In case there was some problem while connecting to the SQL server, an exception is thrown. If there is no error, an object of type 'SQLConnection' is returned which can be used to evaluate SQL statements. The connection object provides the following functions:

***connection*.close**();

Closes the given SQL connection.

***connection*.execute**(statement);

Executes an SQL statement on the given connection object *connection*. In case of an error, an exception is thrown.

***connection*.query**(statement);

Queries an SQL database referenced by the given connection object *connection*. In case of an error, an exception is thrown. Otherwise an object of type 'SQLResult' is returned which can be used to iterate over the returned data. Here is an example how to execute a SQL query on some connection object:

```
var result = connection.query( "SELECT last_name, first_name FROM persons WHERE country  ↩
    LIKE 'A%';" );
```

The result objects returned by this function provide the following functions and properties: *result*.isValid Returns whether this result object is in a valid state. Some functions return invalid result objects to flag an error or otherwise special condition (see below).

***result*.toNext**();

Selects the next row in this SQL result. Marks *result* as invalid in case there is no other row to navigate to. This function and the isValid property above let you iterate over the rows returned by some SQL query easily, as in this example:

```
var result = connection.query( "SELECT last_name, first_name FROM persons WHERE country  ↩
    LIKE 'A%';" );
while ( result.isValid ) {
    // do something with the result
    result.toNext();
}
```

***result*.toFirst**();

Navigate to the first row in this result. This is useful if you want to iterate over the rows many times: after processing the rows using toNext, call toFirst to jump back to the first result so that you can use toNext again. *result*.size Returns the number of rows which this result contains.

***result*.value**(fieldNumber);

***result*.value**(columnName);

These two functions can be used to get the data in the given column of the current row. The column can either be addressed using a number (counting from zero), or using the name of the column (case insensitive). The data in the given column is implicitely converted into a string. Note that you can also use the bracket-syntax as a shortcut. This little example will execute a sample SQL query and then iterate over the returned data rows, printing the first and last name of all matching entries in the persons table:

```
var result = connection.query( "SELECT last_name, first_name FROM persons WHERE country  ↩
    LIKE 'A%';" );
while ( result.isValid ) {
    var s = "First name: ";
    s += result.value( 1 );   // same as result["first_name"] or result.value( "first_name" ↩
        );
    s += ", Last name: ";
    s += result["last_name"]; // same as result.value( 0 ) or result.value( "last_name" );
    test.log( s );
    result.toNext();
}
```

8.1.8 Perl Notes

8.1.8.1 Language Documentation

The Perl documentation contains a manual page perlintro that can serve as a good starting point. Links to other manual pages, tutorials and FAQs can be found on the Online Documentation and the perldoc page.

There is a great numer of Perl books available for purchase. A listing can be found at the Perl Books page. Among the most popular ones are probably "Learning Perl" and "Programming Perl" both published by O'Reilly & Associates.

8.2 Command Line Reference

8.2.1 Squishrunner

squishrunner is used to execute and record test cases and test suites, and to set and retrieve the settings that squishrunner uses at runtime. squishrunner's command line interface looks like this:

```
squishrunner [server options] command [command specific options]
```

The optional server options are:

```
--host <hostname>
```

By default squishrunner connects to the squishserver on the local host (127.0.0.1). If squishrunner should connect to a squishserver running on a different machine, this can be specified with this option.

```
--port <port>
```

By default squishrunner tries to connect to the squishserver on port 4322 which is also the default port squishserver listens to. If you set up squishserver to listen on another port, use this option to tell squishrunner which port it should use for connecting to squishserver.

After the server options one of the following commands *must* be specified:

- Section 8.2.1.1
- Section 8.2.1.2
- Section 8.2.1.3
- Section 8.2.1.4
- Section 8.2.1.5

8.2.1.1 --testsuite

In this mode squishrunner executes a test suite or records a test case in a test suite. In this mode a settingsGroup to be used can be specified before the *--testsuite* switch using *--settingsGroup <settingsGroup>*.

Example:

```
squishrunner --testsuite /home/reggie/squish_addressbook_py --testcase tst_add_address
```

The first argument after this command must be the directory that contains the test suite. In this mode, squishrunner reads the file `suite.conf` in the test suite's root directory. The following settings are read:

- *AUT=<application [args]>* The application which will be started for each test case and optional command line arguments which will be used when executing the AUT.
- *LANGUAGE=<lang>* The scripting language which is used for interpreting the test scripts. Currently this may be *Python*, *JavaScript* or *Tcl*.
- *WRAPPERS=<wrappers>* List of wrappers which must be loaded for the tests. Currently supported: *Qt* and *XView*. Additionally a number of bindings libraries (application wrappers) to be loaded my be specified.

- *ENVVARS=<parameter>* File which should be read and parsed to set environment variables before executing the AUT. The file must contain *VARIABLE=VALUE* pairs which define the environment variables.
- *OBJECTMAP=<parameter>* File that contains the objectmap. If this entry is not present, it defaults to `objects.map`.

The values in the `suite.conf` can also reference environment variables. The syntax for this is *$(ENVVAR)*. So when such an entry is read, it is expanded to the value of the environment variable *ENVVAR*.

There are two different usages of this mode of squishrunner.

- Section 8.2.1.1.1
- Section 8.2.1.1.2

8.2.1.1.1 Executing a Test Case

To execute all of a test suite's test cases, no more arguments need to be specified. If only specified test cases from the suite should be executed, a sequence of one or more *--testcase <testcase>* options must be specified, where <testcase> identifies a particular test case.

To influence the delay triggered by `snooze` script calls can be specified with *--snoozeFactor <factor>*. A value smaller than 1 will cause shorter delays; a value greater than 1 will cause longer delays. A factor of 0 will lead to the fastest possible execution.

Optionally a different report generator to process the test results can be specified with the *--reportgen <[reportgen],[filename]>* option, e.g. *--reportgen xml,/tmp/results.xml*.

The possible values for *[reportgen]* are *xml* (XML), *xls* (Excel) and *stdout* (ASCII table).

8.2.1.1.2 Recording a Test Case

To record a test case in a test suite, *--record <testcase>* must be specified where <testcase> is the name of the test case which should be generated. The name *must* start with the prefix *tst_*.

If the test case needs some test data, the test data files can be specified with *--testdata <file>*, where file must exist either in the shared testdata directory or the test case's testdata directory.

Optionally, event compression during recording, which detects common event sequences and generates high-level API calls, can be switched off with *--disableEventCompression*. This is not recommended.

8.2.1.2 --testcase

In this mode squishrunner executes or records one specified test case.

Example:

```
squishrunner --testcase tst_add_address --wrapper Qt --aut addressbook
```

The first argument after this command must be the test case's directory.

If you want to record a test case, specify the *--record* option after the *--testcase* option. If a test case is to be recorded and requires test data, the test data files can be specified with *--testdata <file>*, where <file> must exist in the shared testdata directory or in the test suite's testdata directory, or in the test case's testdata directory.

The next options must specify which application should be started to execute or record the test case and the wrappers to be used. This is done with the *--aut <application> <arguments>* and one or more *--wrappers <wrapper>* options. Note that all arguments after *--aut* are interpreted as the AUT name and its command line argument so always keep it last. Otherwise you'll see squishrunner options being passed to your application.

If some environment variables should be set before executing the AUT, it is possible to specify a file which defines these variables with the *--envvars <filename>* option. The file must contain *VARIABLE=VALUE* pairs which define the environment variables. Alternatively environment variables can be specified directly on the command line via *--envvar VAR=value*.

If you are recording a test case, event compression during recording, which detects common event sequences and generates high-level API calls, can be switched off with *--disableEventCompression*. This is not recommended.

To influence the delay triggered by `snooze` script calls can be specified with *--snoozeFactor <factor>*. A value smaller than 1 will cause shorter delays; a value greater than 1 will cause longer delays. A factor of 0 will lead to the fastest possible execution.

If you are executing a test case and want to use a different report generator to process the test results, this can be specified with the *--reportgen <[reportgen],[filename]>* option, e.g. *--reportgen xml,/tmp/results.xml*.

The possible values for *[reportgen]* are *xml* (XML), *xls* (Excel) and *stdout* (ASCII table).

If you specify the *--debug* switch, squishrunner will start an interactive command line script debugger which allows setting break points, stepping through the code, etc. The interface is similar to *gdb's* but is currently internal and not documented. We recommend using the Squish IDE to debug test scripts as described in Section 7.8.1 in the Chapter 7.

8.2.1.3 --info

In this mode squishrunner can be queried for various information.

Example:

```
squishrunner --info applications
```

The second argument defines what squishrunner will be asked about:

- *wrappers* Prints out a list of installed wrappers,
- *applications* Connects to the squishserver and prints out a list of all the applications which are located in the squishserver's application paths and can be tested by Squish with the current settings.
- *AUTTimeout* Prints out how many seconds squishrunner will wait before timing out with a test failure if the AUT doesn't respond after being started.
- *cursorAnimation* Returns whether the mouse cursor should be animated (visually moved between positions) during script playback.
- *pauseHotkey* Returns the hotkey which should be used to pause test execution.
- *settingsKey* Prints out the settings key for this Squish installation.

8.2.1.4 --config

This option allows squishrunner's settings to be changed.

Example:

```
squishrunner --config setAUTTimeout 600
```

The following arguments are recognized:

- *setBaseDir <wrapper> <dir>* Creates a new wrapper called <wrapper> with the base directory <dir>.
- *addInitScript <wrapper> <script>* Adds the script <script> as an init script for the wrapper <wrapper>. <script> must be either an absolute file name or relative to the wrapper's base directory. All the init scripts for a wrapper are executed before the test script of a test case which uses the wrapper.
- *setAUTTimeout <seconds>* Specifies how many seconds squishrunner will wait before timing out with a test failure if the AUT doesn't respond after being started.
- *setCursorAnimation [on|off]* Specifies whether the mouse cursor should be animated (visually moved between positions) during script playback.
- *setPauseHotkey <key>* Specifies the hotkey which should be used to pause test execution.

8.2.1.5 --spy

In this mode squishrunner starts an interactive command line spy.

Example:

```
squishrunner --spy --testsuite /home/reggie/squish_addressbook_py
```

The spy shell is started on the already running AUT given its ID via *--aut <aut-id>* or by starting the AUT of the test suite given by *--testsuite <suite_dir>*.

The interactive commandline Spy frontend is internal and we recommend using the Spy via the Squish IDE as described in Section 7.8.4 in the Chapter 7.

8.2.2 Squishserver

squishserver, is responsible for the communication between the squishrunner and the application under test. squishserver must run on the same machine as the AUT, although squishrunner can run on a different machine.

If squishserver is started without any command line options, it will listen to connections on port 4322 and will only accept connections from the local host (127.0.0.1). To allow connections from other hosts, those hosts must be specified. For this purpose, squishserver reads a file called `squishserverrc` in `/etc` (on Unix/Linux/Mac) or `C:\` (on Windows). This file can specify the IP addresses from which connections are allowed using the line *ALLOWED_HOSTS=<ip-addr1> <ip-addr2> ... <ip-addrn>*. The port number that squishserver listens to can be changed by creating a *PORT=<port>* line in the squishserverrc file.

The following command line options are supported by squishserver:

- *--port <port>* Specifies the port which squishserver should listen to. This will override any *PORT=<port>* setting in the squishserverrc file.
- *--config* If squishserver is started with this option, it will simply update its configuration and terminate; it will not listen for connections.

  In this mode a settingsGroup to be used can be specified before the *--config* switch using *--settingsGroup <settingsGroup>*

  The following arguments are recognized after *--config*:

  - *addAppPath <path>* Adds <path> to the list of paths in which squishserver will look for applications when starting an AUT.
  - *addAUT <aut> <path>* Adds the application <aut> to be mapped with the fixed path <path> to be known by squishserver.
  - *removeAppPath <path>* Removes <path> from the list of paths in which squishserver will look for applications when starting an AUT.
  - *removeAUT <aut> <path>* Removes the application <aut> which has been mapped with the fixed path <path> to be known by squishserver.
- *--stop* The server will be asked to shut itself down. Unless overridden with *--port* the server will be contacted at the default port.
- *--daemon* The server will be run in the background. This means it will detach itself from the controlling terminal and stop printing anything to standard output and standard error. Note that anything the AUT prints to these channels will currently get lost, too.

  This switch is not supported on Windows.

8.2.3 Squishidl

squishidl is a tool which parses C++ header files and generates C++ introspection and automation code. For each C++ header file one `.h` and `.cpp` file with the automation and introspection code will be created. For this reason, squishidl must be run on every header file twice.

To generate the header file, run squishidl with the *`--decl`* option, and specify the C++ header that must be parsed with the *`-i`* option. For example:

```
squishidl --decl -i myapp.h -o generated.h
```

To generate the cpp file, run squishidl with the *`--impl`* option, and specify both the C++ header using *`-i`*, and the generated header using *`-h`*. For example:

```
squishidl --impl -i myapp.h -h generated.h -o generated.cpp
```

All the other options can be used with both *`--decl`* and *`--impl`*, and they are all optional.

- *`-o <outputfile>`* By default the output is written to *`stdout`*. With this option the output will be written to the specified file.
- *`--dump [<dumpfile>]`* squishidl creates an intermediate format after parsing the C++ header which is then processed by the output generator. For debugging purposes this intermediate data structure can be dumped to `stdout`, or to a file if specified.
- *`-D <macro>`* and *`-U <macro>`* In a similar way to the C++ compiler, macros can be defined and undefined on the command line. The same macro definitions as passed to the C++ compiler when compiling the input file should be passed to squishidl. This option can be specified multiple times.
- *`-I <includepath>`* In a similar way to the C++ compiler, include search paths can be added on the command line. The same include paths as passed to the C++ compiler when compiling the input file should be passed to squishidl. This option can be specified multiple times.
- *`--includePrefix [<prefix>]`* Specifies the prefix which should be used when generating `#include` statements. An empty <prefix> means the same as specifying the current directory.
- *`--strict`* If this option is on, parse errors when parsing the C++ header file will result in fatal errors. Otherwise, parse errors are just interpreted as warnings.
- *`--extraInclude [<include>]`* Specifies an include file which should be included in the generated file via an `#include` statement. This option can be specified multiple times.
- *`--filter [<expr>]`* Specifies to not generate introspection and automation code for a class, global function, class member function or template instantiation matching <expr>. This option can be specified multiple times.
- *`--nocache`* By default squishidl caches the intermediate format and only parses the C++ header file again if its timestamp changed. Use this option to switch off squishidl's caching.

Arguments can also be read from Section 8.2.5 via the *`--commandFile`* option.

8.2.4 Squish IDE

The Squish IDE is a graphical environment to record, develop, edit, debug and run tests. There are a few options supported by this GUI application.

- *`--startclean`* Starts the Squish IDE without reading any settings using default settings.
- *`--testsuite <testsuite>`* Starts the Squish IDE by opening only the test suite <testsuite>.

8.2.5 Command Files

Command line arguments can also be read in from a file. These so-called command files are plain text files that contain the arguments in a single line or spread over multiple lines. Sometimes this may just be a convient way to specify a complex list of arguments. In other cases this approach might be become a necessity because the maximum command line length of the underlying operating system is exceeded.

Command files can contain comments if they are prefixed with a # character as the first character of a line. Arguments containing whitespace character can be surrounded by single or double quotes as can be seen in the following example:

```
# mycmd.txt
-i myheader.h

-I /usr/local/include -I "/opt/My App/include"
```

An invocation using above command file could like this:

```
squishidl --decl --commandFile mycmd.txt
```

Note: as of the time of writing squishidl is the only tool supporting this feature.

8.3 Environment Variables

Squish respects the following environment variables:

- `SQUISH_WRAPPER_PATH` In addition to all AUT paths, *Squish* will also search all paths listed in the variable when trying to locate a wrapper library.
- `SQUISH_LIBQTDIR` If this variable is set, *Squish* will look into the directory specified by this variable when locating the Qt library when starting and hooking into an AUT instead of using the default location.
- `SQUISH_SCRIPT_DIR` In addition to the shared scripts folder of a test suite, *Squish* will search the directories listed in this variable when trying to locate a script file.
- `SQUISH_USER_SETTINGS_DIR` If this variable is set, *Squish* uses that directory to store its user settings (settings of the squishserver and the Squish IDE as well as the descriptor files). The default locations are no longer searched for settings.

Squish sets the following environment variables:

- `SQUISH_TESTSUITE_NAME` This variable is set in the AUT's shell to the name of the currently executed test suite.
- `SQUISH_TESTCASE_NAME` This variable is set in the AUT's shell to the name of the currently executed test case.

Chapter 9

Add-Ons

9.1 Introduction

There exist several add-ons to *Squish* with specialized functionality for certain areas of software testing. The add-ons need to be purchased separately. For details, contact sales@froglogic.com.

This chapter documents all available add-ons.

9.2 Mercury Quality Center™ Integration

With this addon package, *Squish* integrates tightly with any Mercury Quality Center™ server, allowing to export test cases and test suites from the Squish IDE into the Mercury Quality Center™, running *Squish* tests from within Mercury Quality Center™ and inspecting the test results in the browser and more. All that's necessary for this is an extra software package provided by *froglogic* which installs a plugin into your Mercury Quality Center™ server, making it aware of how to invoke *Squish*. No changes to the Squish IDE are necessary.

9.2.1 Integration Features

Here's a quick overview what you can do when integrating Mercury Quality Center™ with *Squish*.

From within Mercury Quality Center™ you can...

- Create new *Squish* test suites and test cases in any of the scripting languages supported by *Squish*.
- Edit test scripts of *Squish* test cases.
- Run *Squish* tests on any host known by Mercury Quality Center™.
- Run arbitrary commands before and after a test case is run.
- Inspect the results of a test run without having to start the Squish IDE.
- Inspect results of any previous *Squish* test runs, all test runs are recorded in the Mercury Quality Center™ database.

From the Squish IDE you can...

- Create test suites directly within the Mercury Quality Center™ without leaving the Squish IDE.
- Open test suites which are stored in the Mercury Quality Center™.
- Edit test scripts, object maps, suite configuration, test data and more of test suites stored in the Mercury Quality Center™ - any changes done are transparently uploaded to the Mercury Quality Center™.
- Maintain multiple test suites which come from different Mercury Quality Center™ servers in the same Squish IDE.

9.2.2 Installing The Integration Plugin

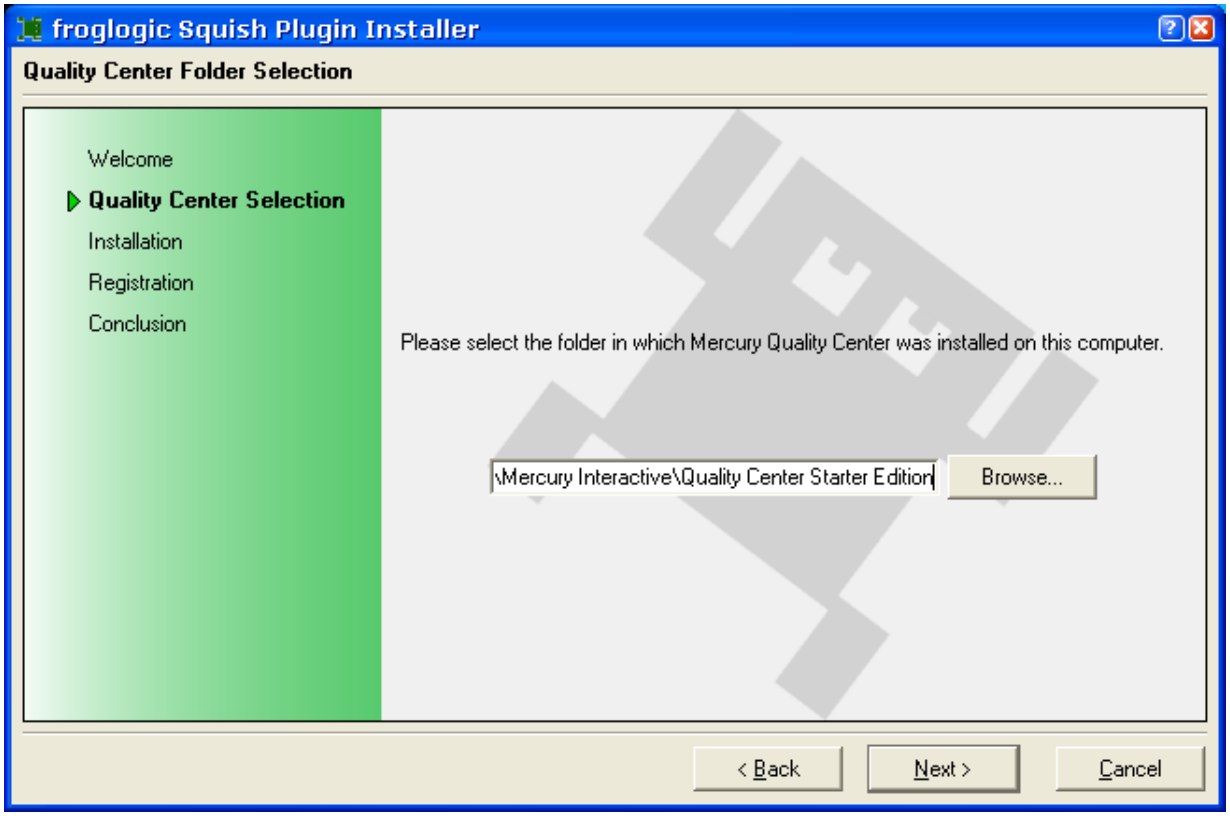
Besides the augmented Squish IDE, the Mercury Quality Center™ integration consists of a special software package which needs to be installed on the computer on which the Mercury Quality Center™ server is running. The installation is done by a mostly automatic installer program and should be finished within five minutes.

Warning
Please note that during the installation of the Mercury Quality Center™ plugin, the Mercury Quality Center™ server needs to be shut down (and then restarted) by the installation program. Hence, users won't be able to connect to the Mercury Quality Center™ for a short period during the installation.

Please note that this installation process only has to be executed once, on the Mercury Quality Center™ server. The client desktops won't need any installation at all.

9.2.2.1 Choosing the Quality Center Folder

After acknowledging the welcome page, the first step in the installation process is to tell the setup program where Mercury Quality Center™ was installed on the computer.



The first page of the wizard used for installing the Mercury Quality Center integration plugin.

Specify the folder in which Mercury Quality Center™ was installed on this computer in the input field, or use the button at the right side of the input field to open a dialog which lets you browse to the directory. The setup program will try to find your Mercury Quality Center™ installation automatically, but depending on your software configuration this might not work in any case. If the input field stays blank, you have to specify the path yourself.

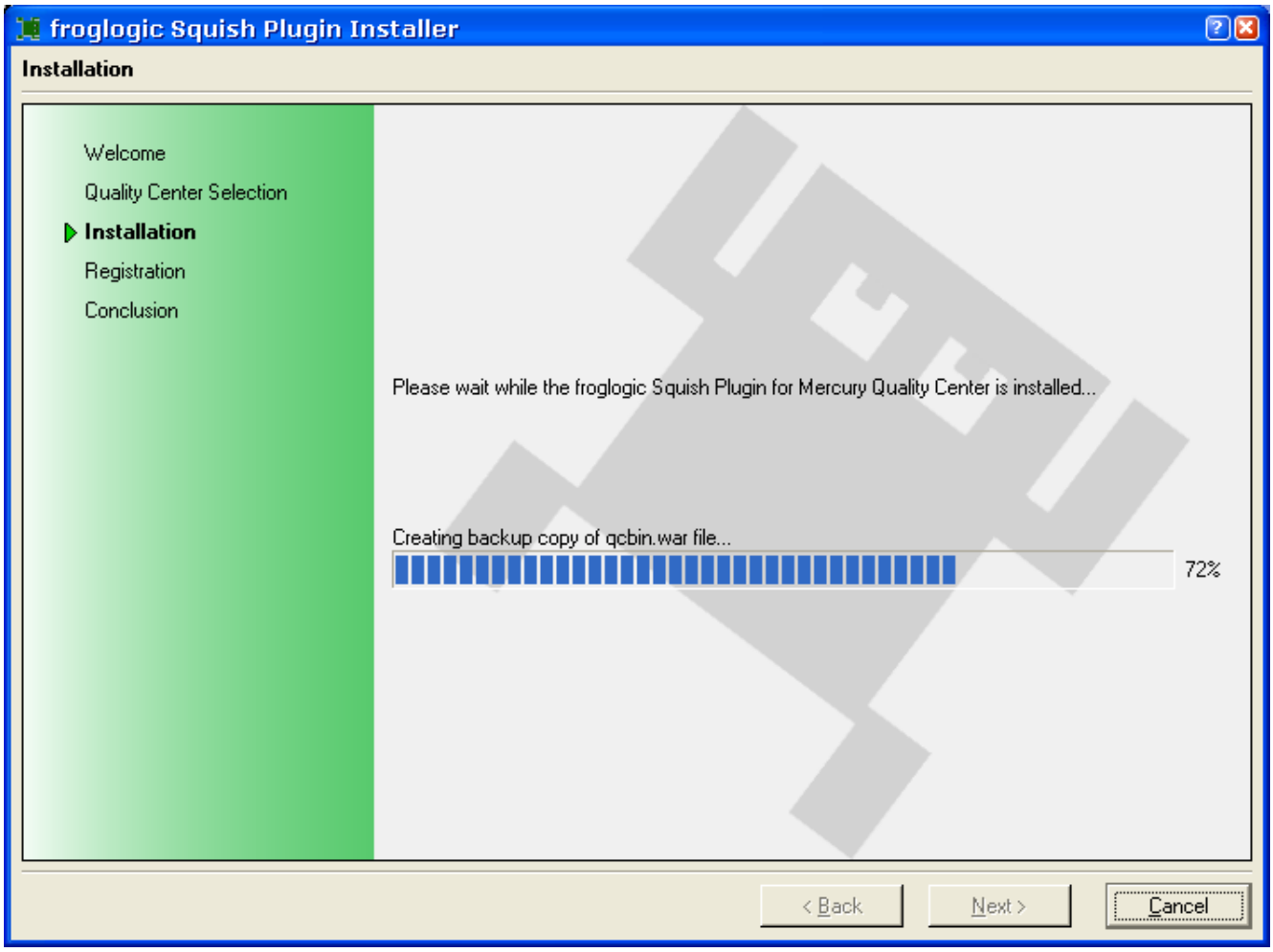
Often, the directory has a name like `C:\ProgramFiles\MercuryInteractive\QualityCenter` or similar.

As soon as you have specified the correct directory, the Next button which commences the installation of the plugin becomes available. Just click it to start the installation.

Warning
Even though the installation program takes great care to not to cause and stale files in case an error occurs during the installation, it cannot be guaranteed that i.e. a power outage during the installation won't cause any data corruptions. Hence, it is suggested that you first try the installation process on a spare server.

9.2.2.2 Monitoring the Installation Progress

After specifying the folder in which Mercury Quality Center™ was installed, the setup program begins with deploying all files which are required to access *Squish* from within the Mercury Quality Center™. This process should take roughly five minutes.

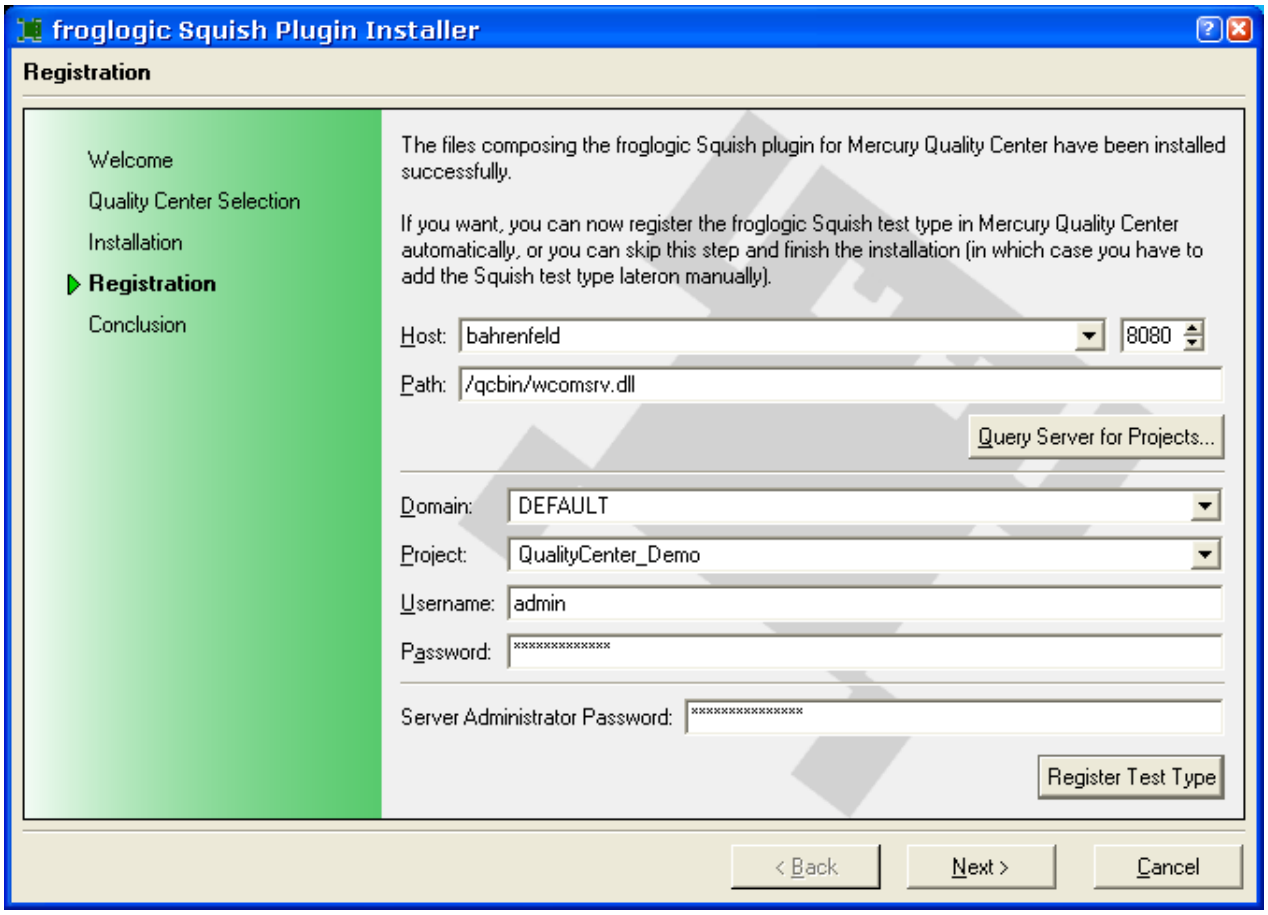


A picture of the Mercury Quality Center plugin installation wizard deploying the files to the server.

In case anything goes wrong during the installation, a message box will be shown giving detailed information about what happened. Any stale files installed up to that point will be removed.

9.2.2.3 Post-Installation Registration

After installing the files composing the Mercury Quality Center™ integration plugin, the setup program can augment the Mercury Quality Center™ server's configuration file automatically so that it becomes aware of the new plugin files. However, administrator login credentials are required for this. In case you have access to that login data, you can specify it on this screen to let the setup program do the work.



A picture of the Mercury Quality Center plugin installation wizard's configuration page.

You can also skip this step by pressing the Next button at the bottom of the setup program window. However, you'll have to do the required configuration by hand then before you're able to use the *Squish* integration from within Mercury Quality Center™.

9.2.2.4 Manual Plugin Registration

In some situations it may be necessary to register the *Squish* plugin in the Mercury Quality Center™ server manually. This registration however requires having the proper login credentials to your Mercury Quality Center™ server (to the Project Customization interface, in particular). Please check back with your system administrator to find out whether you can do this yourself.

For registering the *Squish* test type, open the Project Customization interface (available through the Customize link at the left hand side of the Mercury Quality Center™ login screen). Then, select Customize Project Entities from the links at the left side. In the dialog that shows up, navigate through the tree view of items by selecting TEST, System Fields and then Type.

The right side of the dialog will be populated with controls now. One of them is a button labelled Goto List which you need to click to open the list of test types. In the list dialog, click on the button labelled New Item and enter the string SQUISH-TEST exactly like that (all uppercase, a dash separating the two words) and press OK.

Close the list dialog by clicking on Close, close the 'Customize Project Entities' dialog by clicking on the button labelled OK, then log out of the Project Customization interface by selecint Logout in the upper-right corner of the screen.

9.2.3 Configuration of the Plugin

After the plugin has been installed and registered, you will need to setup the initial configuration. You might also want to adjust and augment the configuration lateron. How this is accomplished will be discussed in this section.

The configuration of the *Squish* integration plugin consists of two settings keys which are stored in the 'Site Configuration' of your Mercury Quality Center™ server.

SQUISH_SCRIPTING_LANGUAGE Indicates the scripting language to be used when creating new *Squish* test cases within the Mercury Quality Center™. Can be 'Python', 'Tcl' or 'JavaScript'.

SQUISH_PATHS This string in the format *hostname=path, hostname=path* tells the *Squish* plugin, where the *Squish* installation can be found on each of the hosts on which you intend to run the tests. *hostname* can be the name of a host (or an IP address of one) on which a test should be run. *path* stands for the path on which *Squish* is installed on the given host.

An example entry which describes that on the host 'TREBULUS' *Squish* is installed in `C:\Tools\Squish` and on the host with the IP address 192.168.42.3 *Squish* is installed in `E:\Deploy\Tools\Squish-Stable` looks like this:

```
TREBULUS=C:\Tools\Squish,192.168.42.3=E:\Deploy\Tools\Squish-Stable
```

You need to configure these entries before being able to execute *Squish* runs in the Site Configuration. In case they don't exist yet, you might have to add them. To do so, open the Site Administrator interface of Mercury Quality Center™, then click on the Site Config tab at the top of the screen to move to the site configuration screen. Here you can add and edit any keys.

9.2.4 Using the Integration From Mercury Quality Center™

After the integration plugin has been installed and configured correctly, you can work on *Squish* tests from within Mercury Quality Center™ fairly easily. All changes you do to test cases from within the Mercury Quality Center™ can be seen from the Squish IDE and vice versa.

9.2.4.1 Creating New Test Cases

Creating new *Squish* test cases within the Mercury Quality Center™ works virtually the same as creating any other kind of test. In the Test Plan module, create a new test folder called 'froglogic Squish Tests'. In this folder, you can then create as many test folders (preferably you start the names of the folders which are going to contain test cases with the 'suite_' prefix, since the Squish IDE expects that) as you want.

Note
As described further down, only those test suites which are in the 'froglogic Squish Tests' folder are visible from the Squish IDE. You can create *Squish* tests in another Mercury Quality Center™ folder as well and execute them. They won't be visible from the Squish IDE though.

Here's a quick rundown on how to create a new *Squish* test in Mercury Quality Center™ which will be visible from the Squish IDE.

1. Log into the Mercury Quality Center™ project to which you want to add a test.
2. Click on the button labelled Test Plan at the left side of the screen to move ot the Test Plan module.
3. In the tree view of the Test Plan module screen, open the froglogic Squish Tests folder. If it doesn't exist yet, create it using the New Folder button above the tree view.
4. Right-click on the froglogic Squish Tests folder item to open the context menu. In the context menu, select New Test....
5. In the dialog that shows up, select 'SQUISH-TEST' from the combo box to indicate that you want to create a *Squish* test. Choose any name for the test you want, then press OK.
6. In the Required Fields that now shows up, select any values which are appropriate for your organizational testing discipline. For this sample, choose any values you want (you can always change them lateron), then press OK.

Congratulations, you just created a *Squish* test!

Since *Squish* tests always have at least one test script attached to them, you could now switch to the Test Script tab of the newly created test case to see the simple (mostly empty) test script.

9.2.4.2 Running *Squish* Tests

Executing this *Squish* test works like executing any other test which is stored in a Mercury Quality Center™ server: move to the Test Plan module by clicking on Test Plan on the left side of the screen, then drag the tests you want to execute into the exeuction grid. When you have assembled all tests for a test run, press the Run button to run the test.

Executing a *Squish* test within Mercury Quality Center™ will cause the integration plugin to invoke squishrunner, much like the Squish IDE does internally. You might want to pass special arguments to squishrunner when executing the test though (for instance, you probably want to specify any additional wrapper libraries you want to use, and you will want to specify an application which to test).

Note
A comprehensive reference of the command line options for squishrunner is available in the command line reference section for Section 8.2.1.

You can also specify arbitrary commands to be executed before and/or after squishrunner is invoked.

To configure a *Squish* test run, right-click on the test in the test execution grid of the Test Plan, then select Test Run Properties... in the context menu to open the dialog which presents all kinds of execution parameters. The *Squish*-specific settings are accessible by clicking on Configuration on the left side of the dialog, then clicking on the Automatic tab.

9.2.5 Using the Integration from the Squish IDE

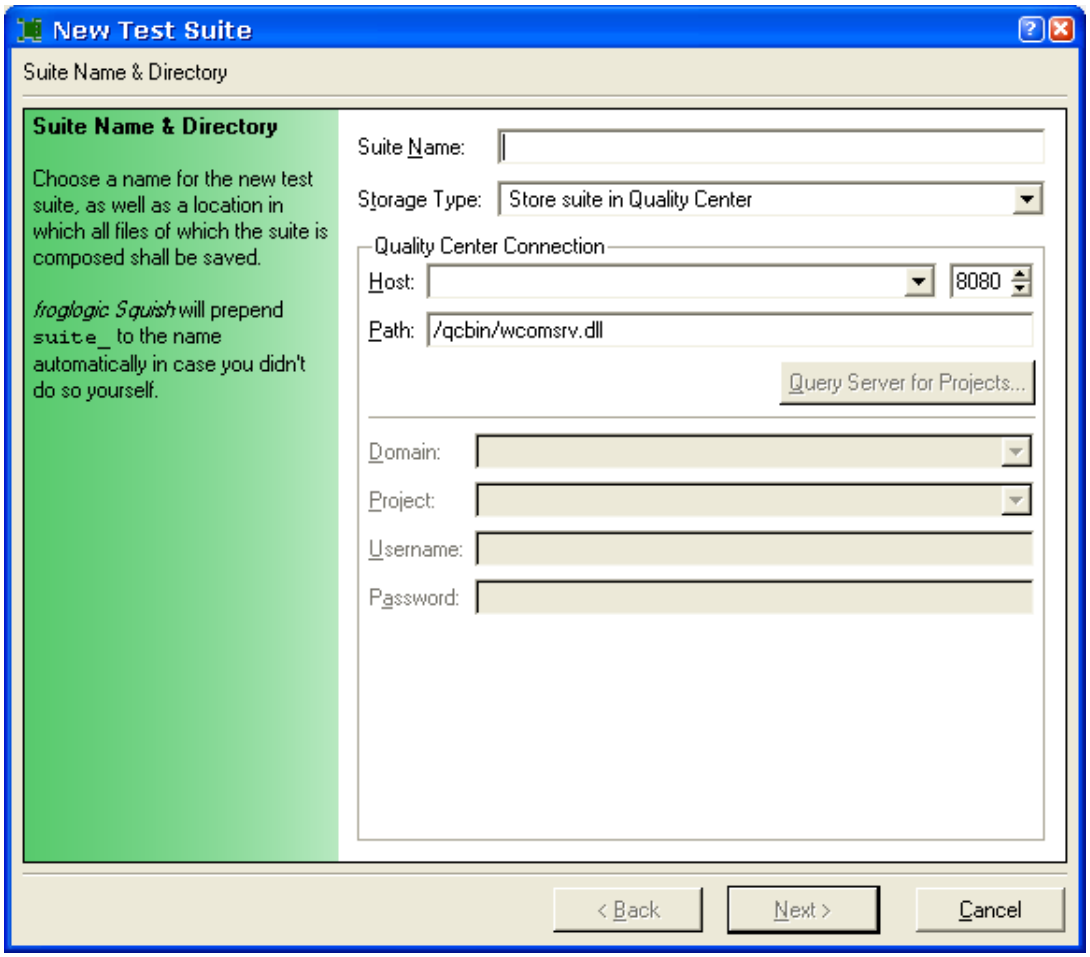
When using an Squish IDE with a license key which allows accessing Mercury Quality Center™ servers, a set of new features is available which makes working with tests in Mercury Quality Center™ servers much easier. Among other things you can

- Create test suites directly in the Mercury Quality Center™.
- Import test suites from the Mercury Quality Center™ to the Squish IDE.
- Export test suites from the Squish IDE to Mercury Quality Center™.

9.2.5.1 Creating a Test Suite in the Mercury Quality Center™

Creating a new test suite in a Mercury Quality Center™ server works very much like creating a normal test suite locally, only one step in the creation process differs from the normal workflow.

In the Squish IDE, select File → New Test Suite to open the wizard for creating new test suites. The very first step of this wizard is what differs between creating a normal test suite and a test suite in a Mercury Quality Center™ repository.



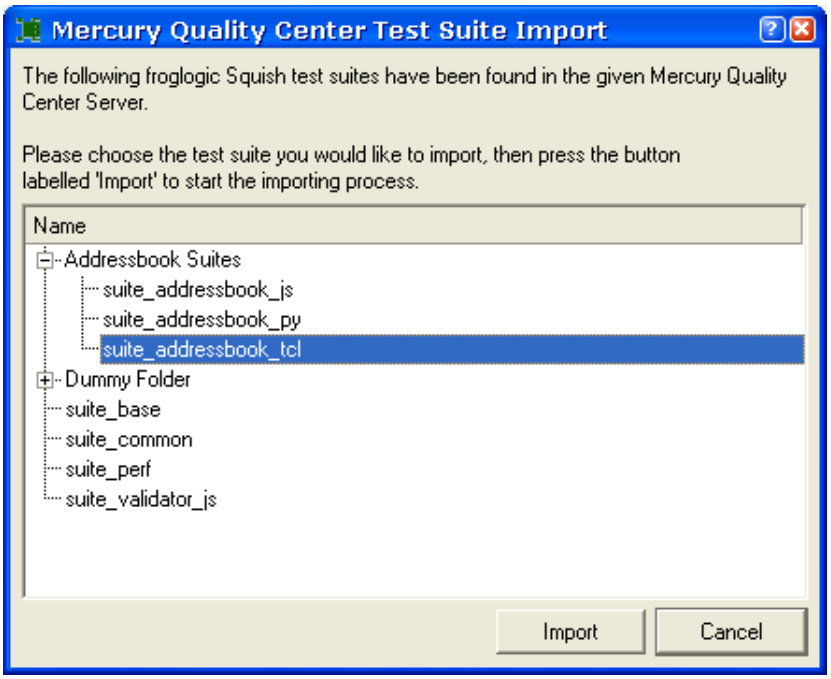
A picture of the wizard for creating new tests suites.

In this very first page, right below the input field for the test suite name, a combo box lets you choose whether the test suite should be stored locally, or whether it should be created within a given Mercury Quality Center™ repository. If you decide to create the suite in a repository, then you will need to enter all required login credentials so that the data can be uploaded. All other steps of the wizard are the very same.

The login credentials used when creating the suite will be stored in the suite configuration file. Whenever you change anything in the test suite (for instance, when adding entries to the object map or when adjusting one of the test scripts), the suite will be uploaded in the background to the server so that it becomes less likely for the two copies to go out of synch.

9.2.5.2 Importing a Test Suite from Mercury Quality Center™

For importing a test suite from Mercury Quality Center™ into the Squish IDE, select File → Open Test Suite from Quality Center... from the menu bar. A dialog will show up which asks you to enter the name of the Mercury Quality Center™ server you want to connect to as well as the login credentials for the particular project.



A picture of the dialog for importing test suites from the Mercury Quality Center into the Squish IDE.

A dialog like the one shown above will show up, showing all folders and test suites the Squish IDE could find in the given project of the server. Simply select one of the suites and press the button labelled Import to import that suite into the Squish IDE.

Note
The import dialog will not show all available test suites in the Mercury Quality Center™ database but only those which are stored in a folder called 'froglogic Squish Tests'!

The login credentials used when importing the suite will be stored in the suite configuration file. Whenever you change anything in the test suite (for instance, when adding entries to the object map or when adjusting one of the test scripts), the suite will be uploaded in the background to the server so that it becomes less likely for the two copies to go out of synch.

9.2.5.3 Exporting a Test Suite to Mercury Quality Center™

For exporting a test suite which you have opened in the Squish IDE, simply select Test Suite → Export To Quality Center from the menu bar. A dialog will be shown asking you for the server to which the suite should be exported. You will also need to give the information necessary for logging into a project on the server.

After having entered the required information, press the button labelled Connect to Server to connect to the specified project and commence the upload. The dialog will disappear, and the upload of all data belonging to the test suite commences in the background You can see some progress information in the status bar at the bottom of the Squish IDE window.

The login credentials used when exporting the suite will be stored in the suite configuration file. Whenever you change anything in the test suite (for instance, when adding entries to the object map or when adjusting one of the test scripts), the suite will be uploaded in the background to the server so that it becomes less likely for the two copies to go out of synch.

9.3 TPTP integration

This Eclipse plugin allows the user to take existing *Squish* tests and add them as regular tests into the Test and Performance Tools Platform (TPTP) framework, allowing execution, monitoring, log analysing, charting and reporting amongst others.

9.3.1 Obtaining the plugin

You can obtain the TPTP plugin from the following URL:

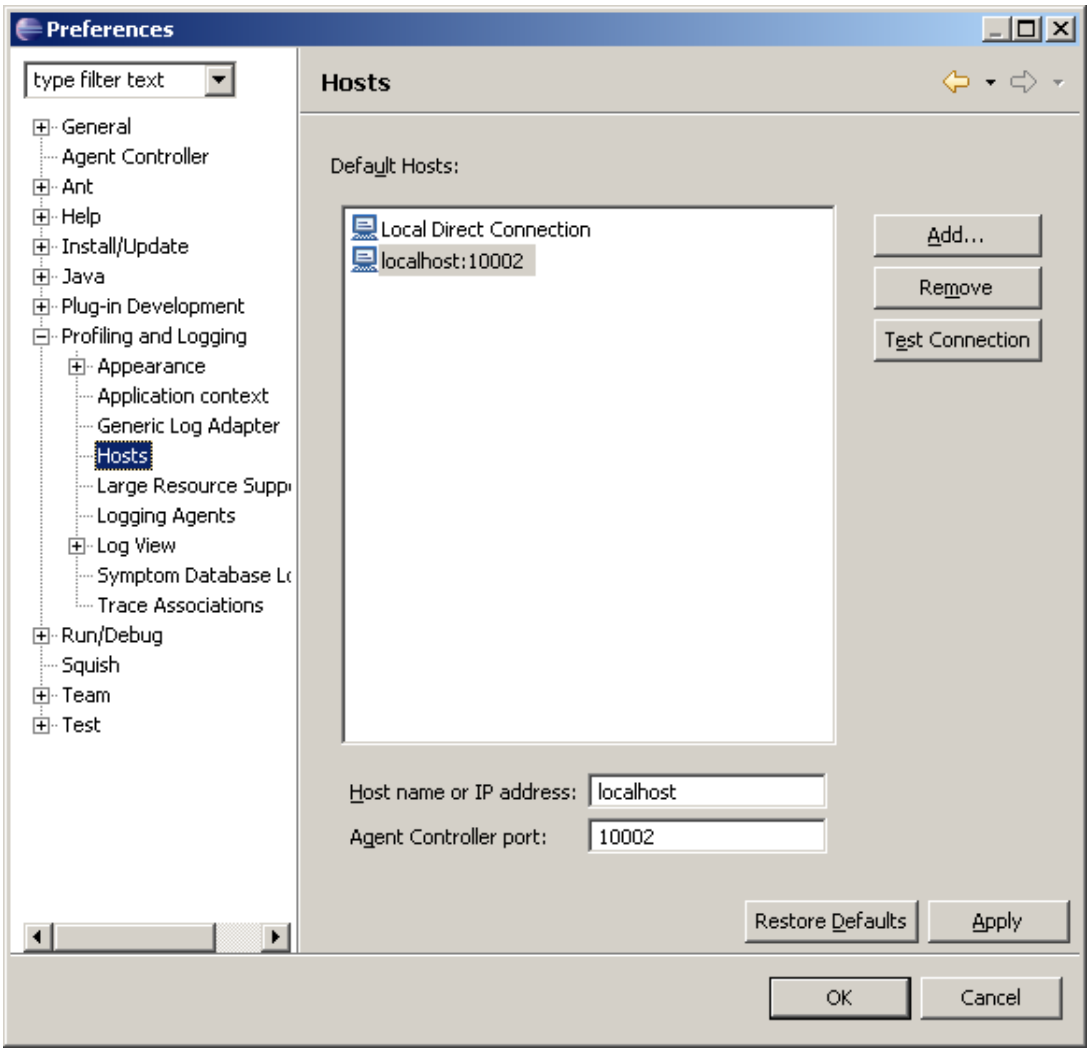
www.froglogic.com/download/com.froglogic.squish.plugin_latest.zip

This zip file contains the latest distribution of the TPTP plugin at the time of download. Unzip the file to get the jar file which is named in the standard Eclipse naming scheme, namely com.froglogic.squish.plugin_x.y.z.jar, where x,y,z are the major, minor and service version numbers. Below we will keep using this name.

9.3.2 Installing the plugin

The following steps are needed for installing the *Squish* plugin.

- The `com.froglogic.squish.plugin_x.y.z.jar` file needs to be copied into the eclipse plugins directory.
- The *Squish* plugin can be used with either the integrated or the remote agent controller. To determine which one you are using, you can go to Window → Preferences → Profiling and Logging → Hosts. The screenshot below shows how the dialog looks:



Screenshot of the Hosts dialog.

When the Local Direct Connection entry is selected, you are using the Integrated Agent Controller. When another entry is selected, the Remote Agent Controller is in use. You can verify the selection works by using Test Connection.

Note: the Remote Agent Controller is standalone and not contained in the TPTP plugins. Follow the instructions here(section Agent Controller) if you want to obtain and install it.

- If you are using the Integrated Agent Controller, its environment is in `eclipse\plugins\org.eclipse.hyades.execution_4.?.?.?\iac-runtime\`. Add this in the AgentControllerEnvironment section of the `config/serviceconfi xml` file of the Integrated Agent Controller:

  <Variable name="CLASSPATH" position="append" value="%PLUGINS_HOME%\com.froglogic.squish.plugin_x.y.z.jar"/>

  Copy the `com.froglogic.squish.plugin_x.y.z.jar` in the `extensions` dir of Integrated Agent Controller.

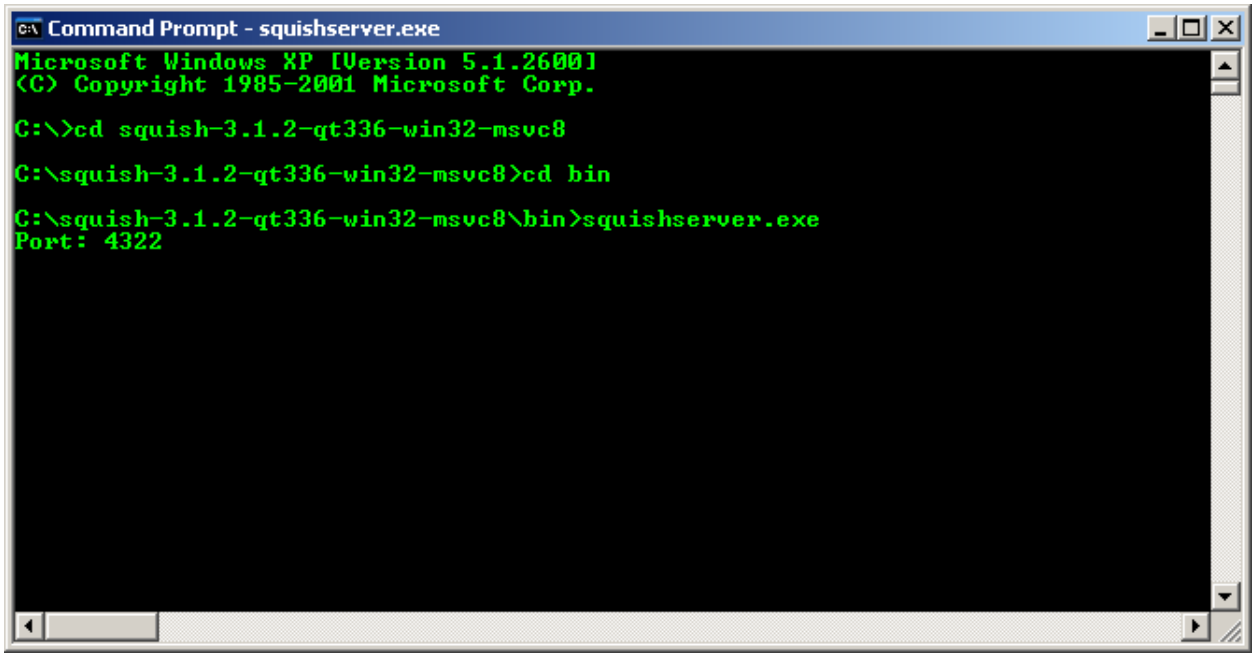
- If you are using the Remote Agent Controller, add this in the AgentControllerEnvironment section of the `plugins/serviceconfi xml` file of the Remote Agent Controller:

  <Variable name="CLASSPATH" position="append" value="%PLUGINS_HOME%\com.froglogic.squish.plugin_x.y.z.jar"/>

  Copy the `com.froglogic.squish.plugin_x.y.z.jar` in the `plugins` dir of Remote Agent Controller.

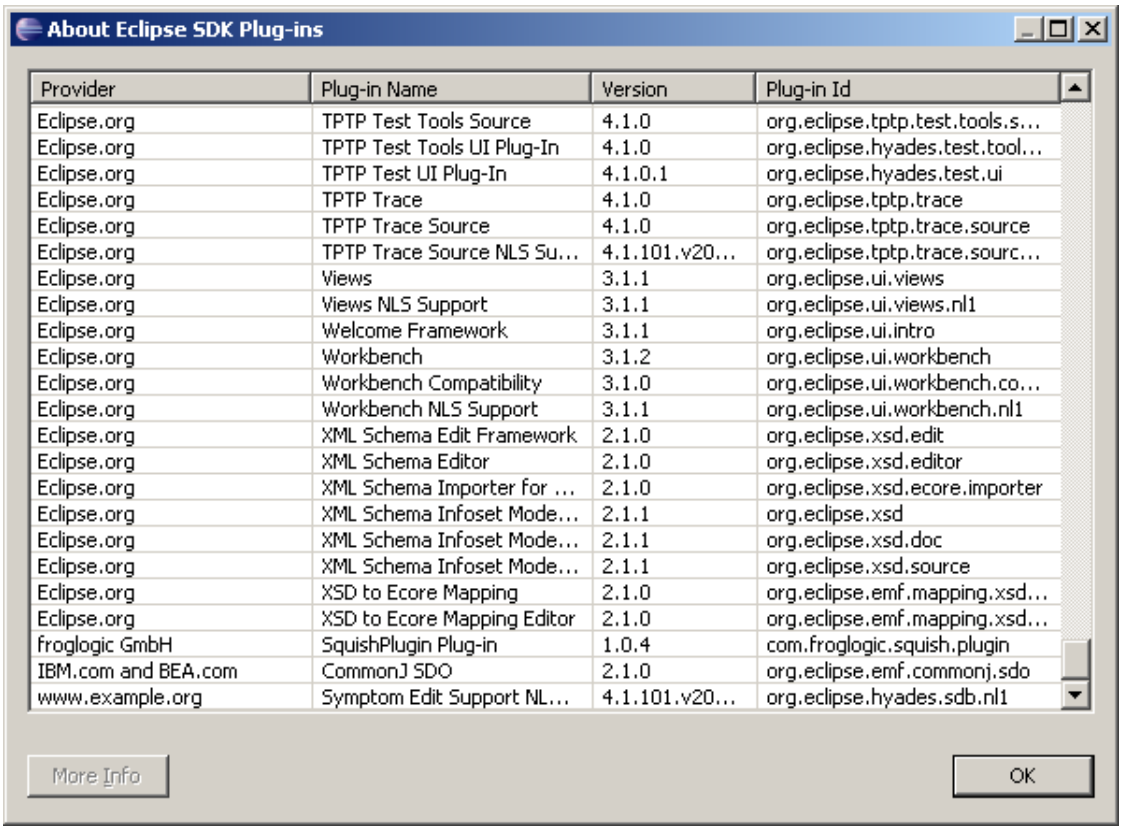
  After this the Remote Agent Controller has to be restarted.

- Make sure the application `squishserver.exe` is running. Start up a shell, go to the *Squish*`bin` directory and run `squishserver.exe`.



Starting squishserver app from a shell.

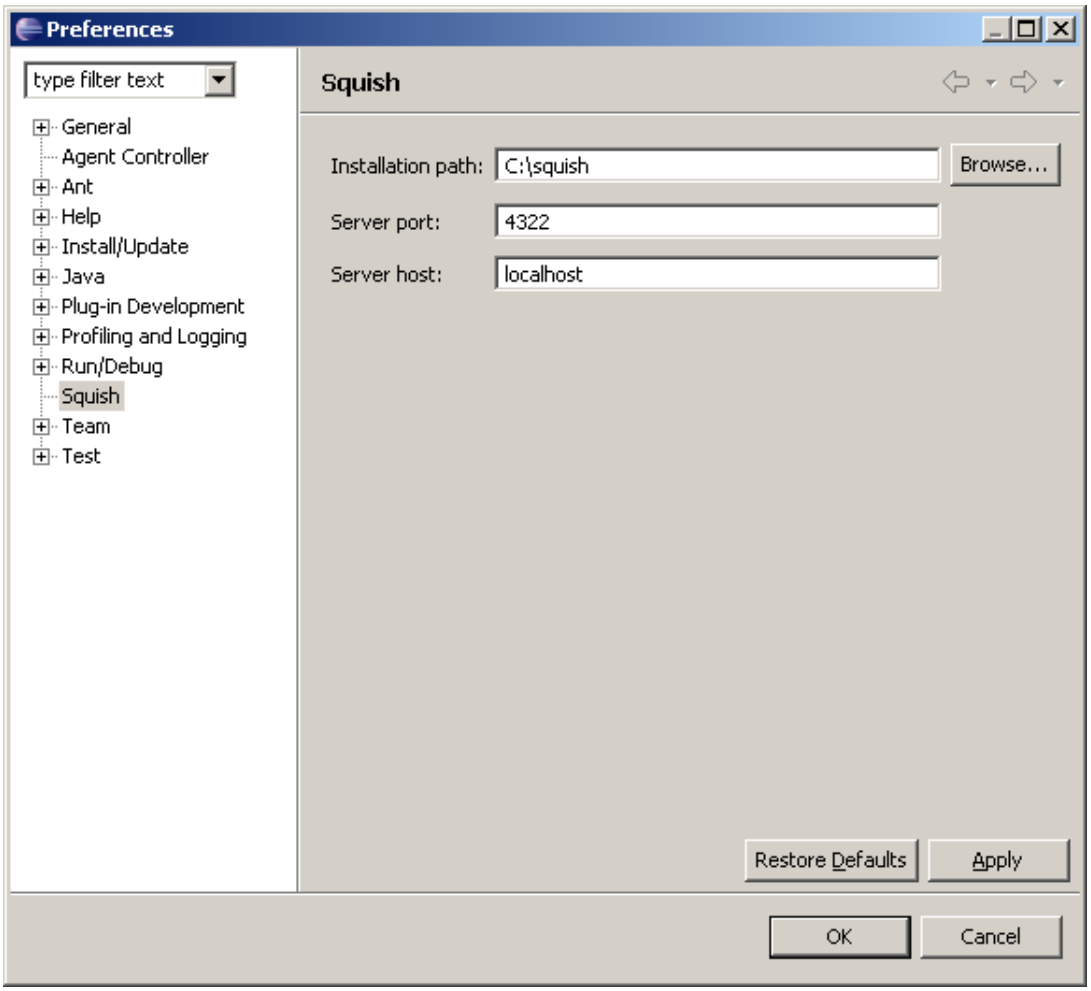
One way to check whether the plugin is detected is by looking at the plugin list in Eclipse using Window → About Eclipse → Plug-in details. SquishPlugin should be listed there, like in this screenshot:



The Squish plugin in the plugin list.

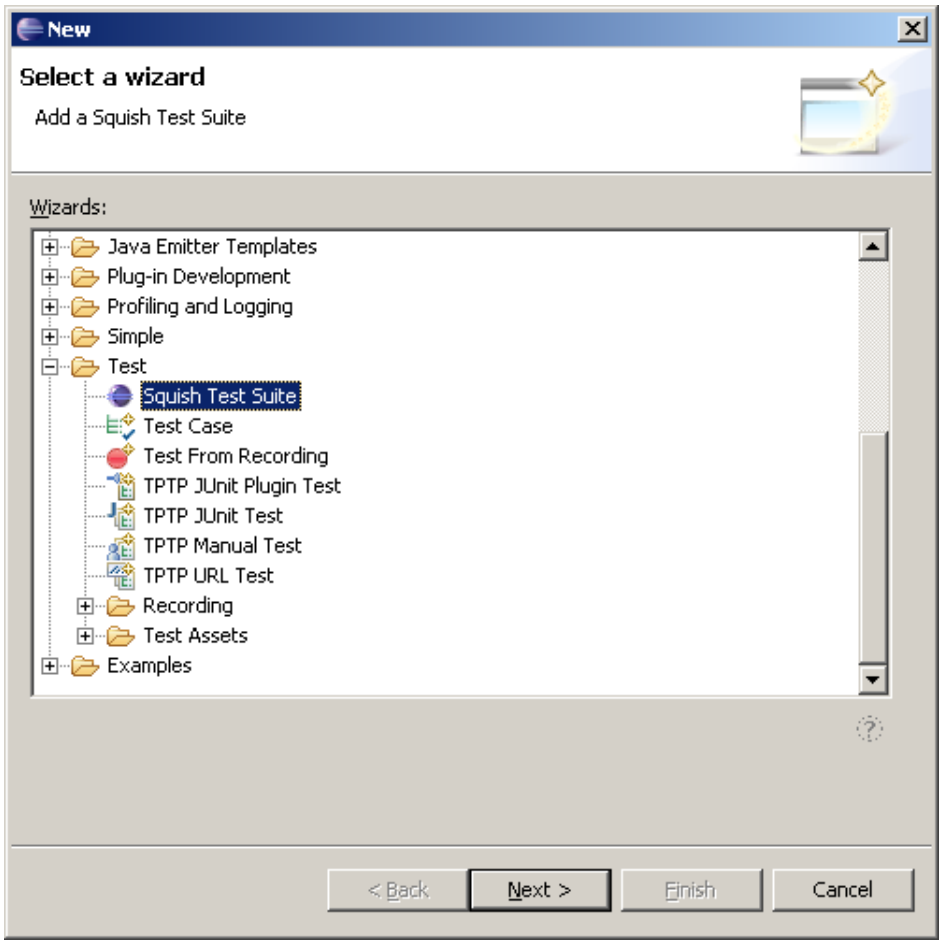
9.3.3 Working with the plugin

A first thing to make sure is that the plugin knows where *Squish* is located on the operating system. To do so select Window → Preferences → Squish. On this page use the Browse button to select the *Squish* installation path. Here you can also set the hostname of the server and the port it is listening on.



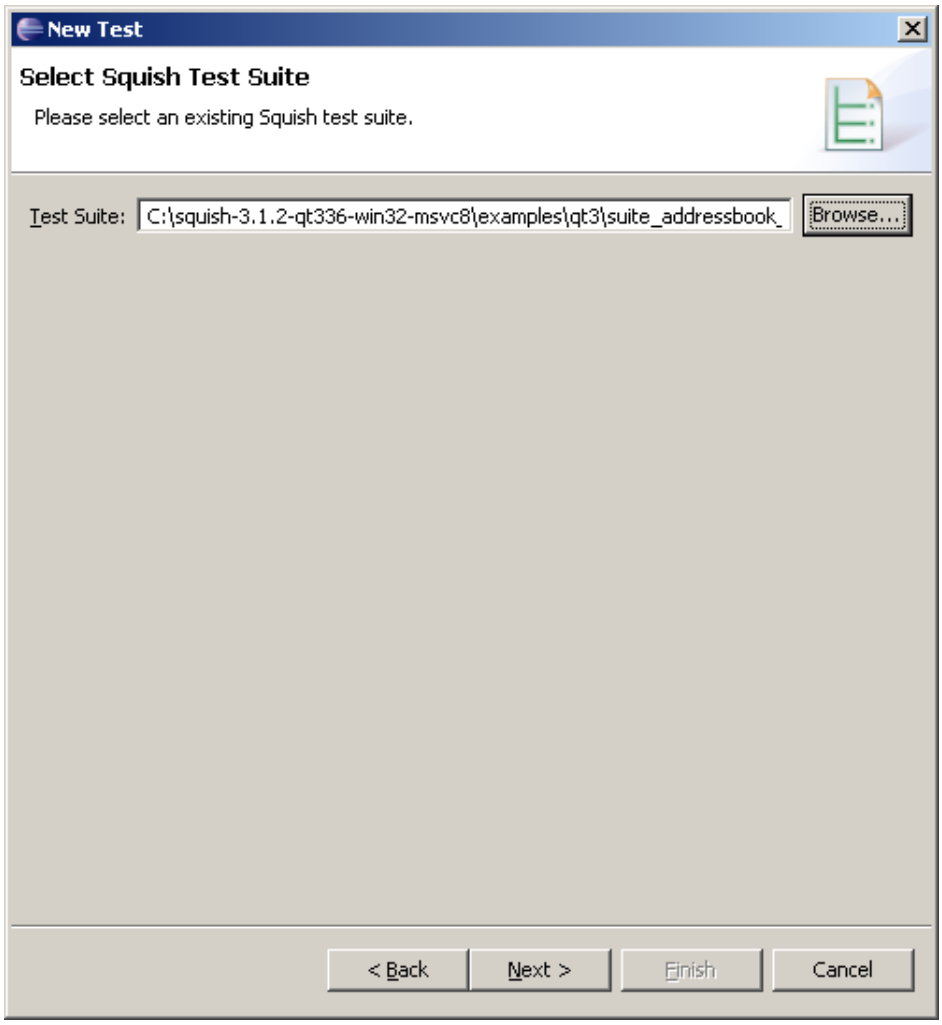
The Squish preferences page.

Now we are ready to import existing *Squish* tests. Make sure there is an existing project in the workbench where the test cases can be stored. Now choose File → New → Other → Test → Squish Test Suite and click Next. The selection page looks like this:



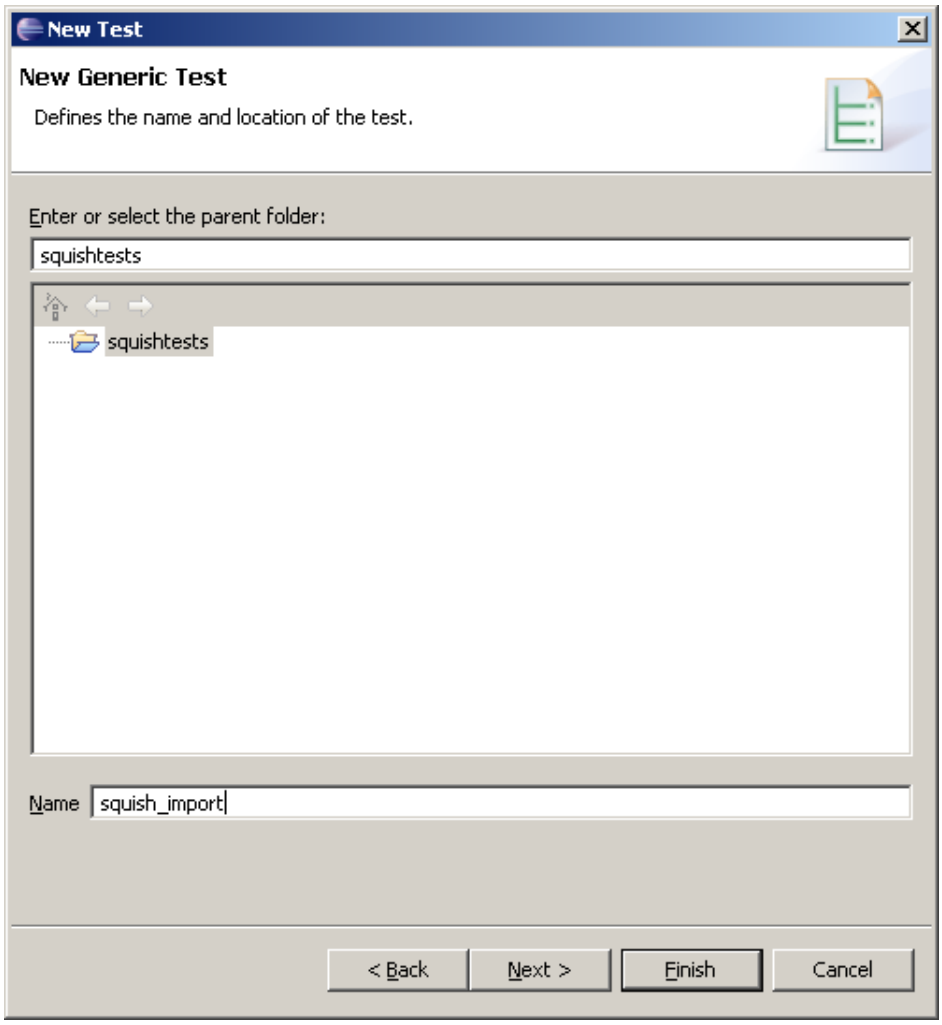
The wizard selection page with the Squish wizard highlighted.

On the first page we can choose the *Squish* suite.conf file to import. Use the Browse file dialog to choose the testsuite you want to import, then click Next. This screenshot shows how this page looks:



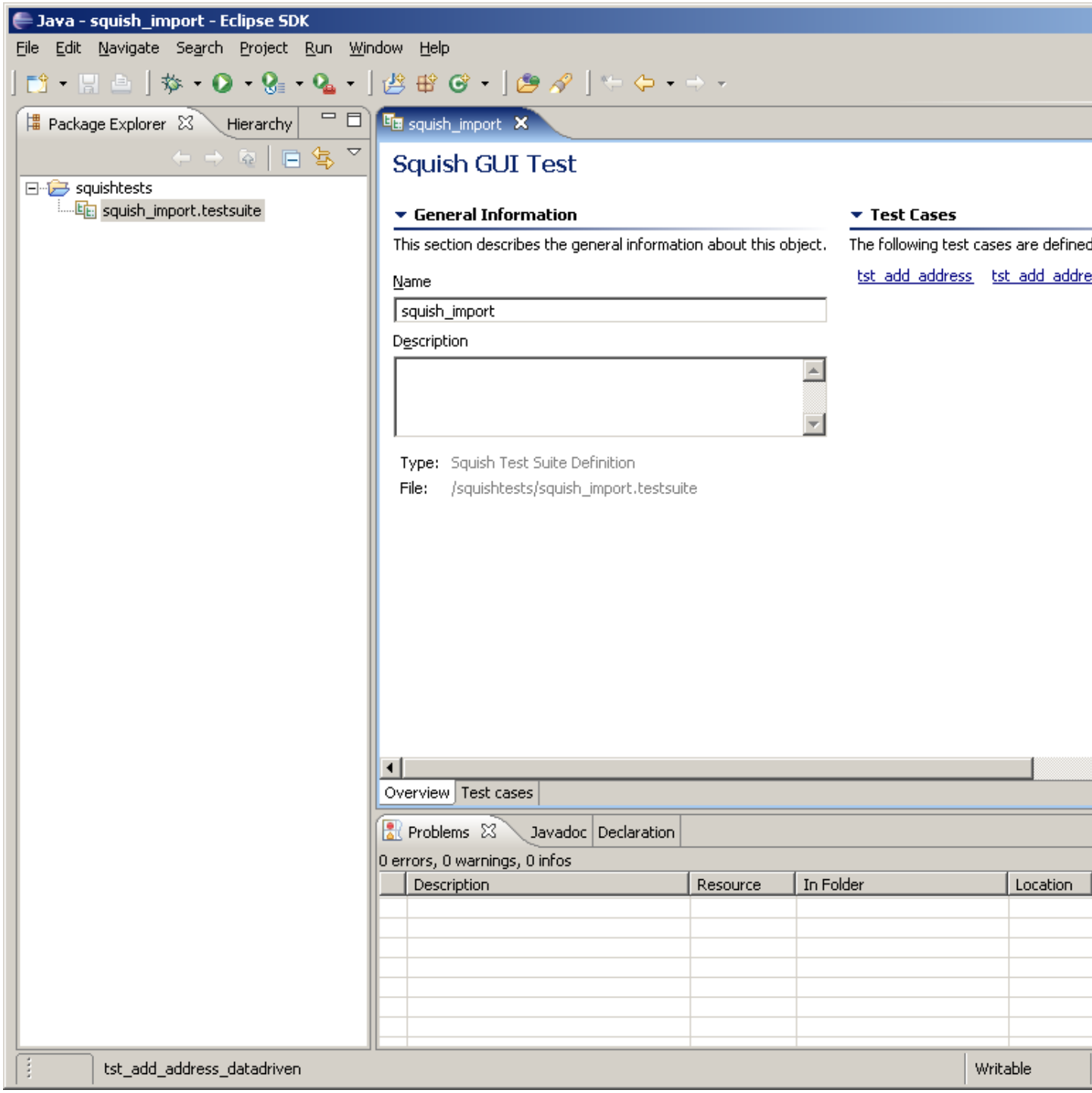
The first page of the wizard used for importing a Squish test.

On this last page you need to select the Eclipse project to import the test cases into and give a name to the imported suite (this can be the same name as the original squish testsuite). Again, below is a screenshot showing how the final page in the wizard looks:



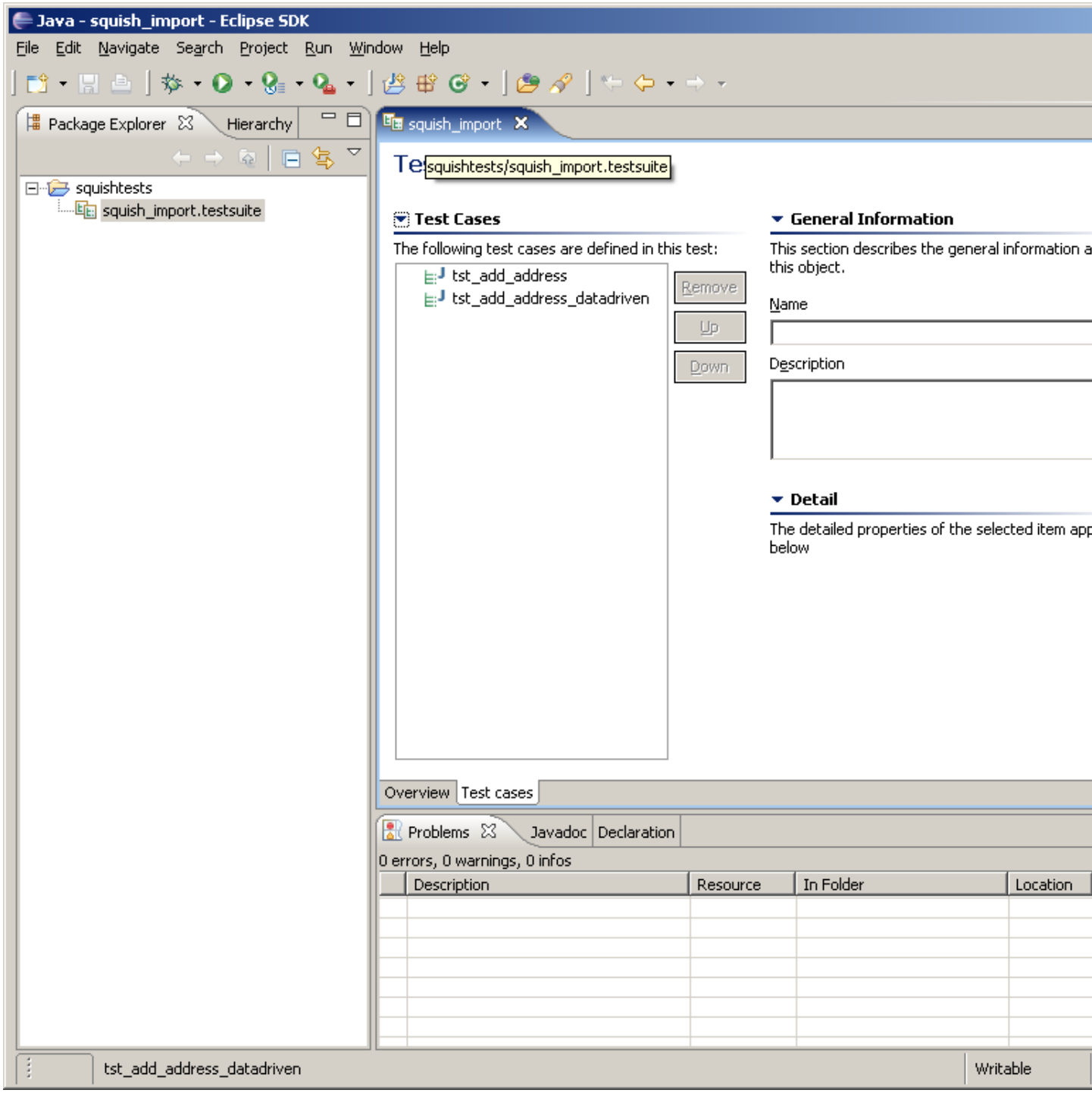
The last page of the wizard used for importing a Squish test.

Click on Finish to start importing the tests. After importing, in the workbench in the chosen project there should be a new `.testsuite` file, with its first tab page showing general information about the test. It should look like the screenshot below.



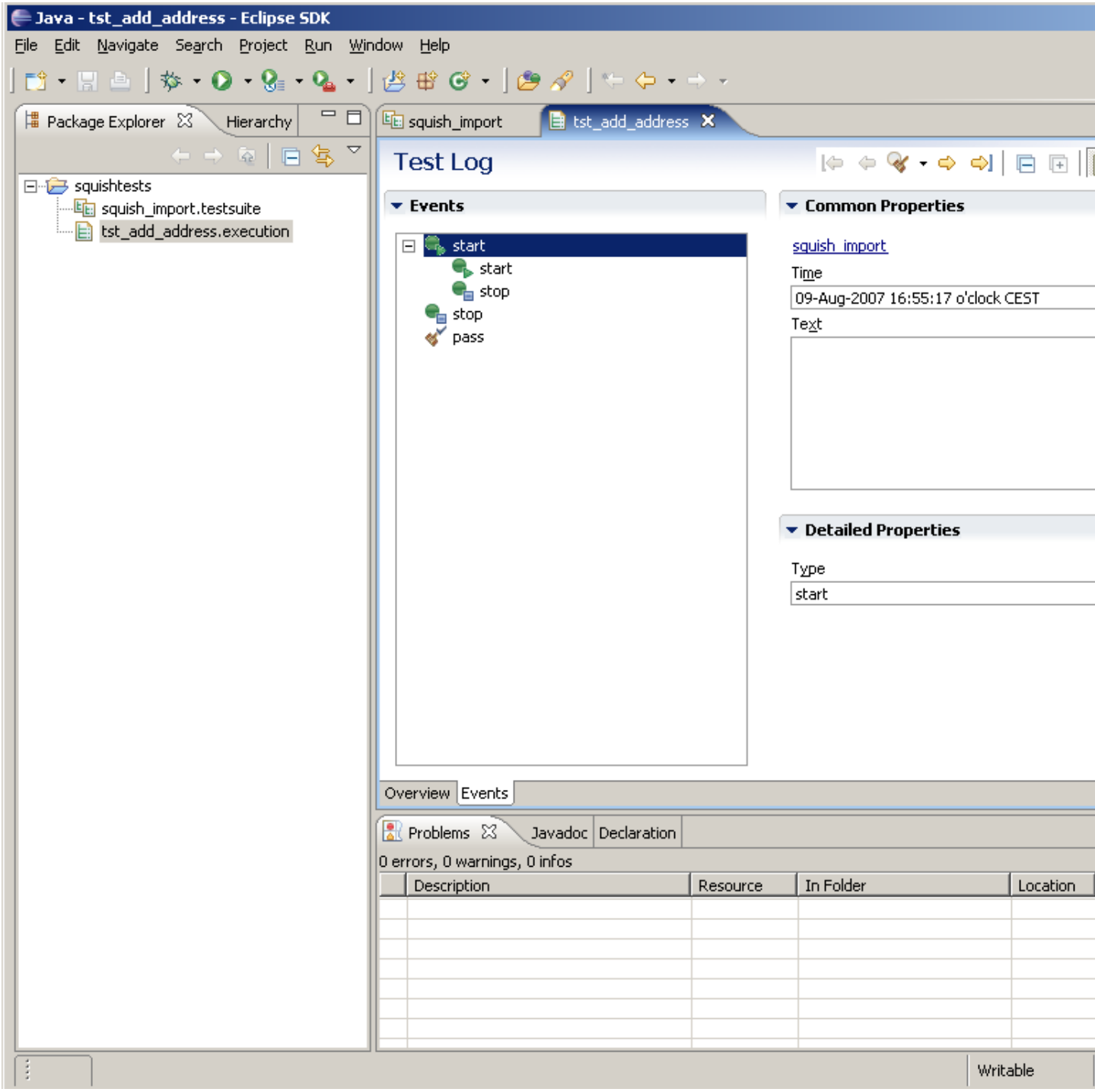
The workbench after importing a Squish testsuite.

When you click on the second tab, it shows the test cases in the testsuite. Select a test case and press the play button in the top-right of the tab.



The second tab page showing the available test cases.

When the test has run it should have produced an execution file detailing the test run. Click on the `.execution` file to examine the test run. Execution files have the same names as the test case name, but will include a timestamp in the name to make them unique.



Showing the execution file after the test has run.

9.4 Ant integration

Apache Ant is a software tool for automating software build processes. It is similar to make but is written in the Java language, requires the Java platform, therefore is cross-platform, and is best suited to building Java projects. More information can be found here.

Using the *Squish* plugin for Ant described in this chapter, it is possible to run *Squish* tests from an Ant build file.

9.4.1 Obtaining the plugin

You can obtain the Ant plugin from the following URL:

www.froglogic.com/download/squish-ant-plugin_latest.jar

9.4.2 Installing the plugin

There are multiple ways to install the *Squish* plugin:

- Copy the jar to ANT_HOME/lib. This often requires Administrator rights.
- Copy the jar to USER_HOME/.ant/lib.
- When calling ant use the -lib option to specify the *Squish* plugin.

Details for installing external Ant libraries are described here.

9.4.3 Example use of plugin

Here is a typical build.xml file:

```
<project name="MyProject" basedir=".">
    <description>
        simple example build file
    </description>
  <!-- set global properties for this build -->
  <property name="src" location="src"/>
  <property name="build" location="build"/>
  <property name="dist"  location="dist"/>

  <target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="${build}"/>
  </target>

  <target name="compile" depends="init"
        description="compile the source " >
    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <target name="dist" depends="compile"
        description="generate the distribution" >
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}/lib"/>

    <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
    <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
  </target>

  <target name="clean"
        description="clean up" >
    <!-- Delete the ${build} and ${dist} directory trees -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
```

```
</project>
```

To use the plugin functionality the *Squish* namespace has to be mentioned:

```
<project name="MyProject" basedir="." xmlns:squish="antlib:com.froglogic.squish.ant">
    <description>
        simple example build file
    </description>
    ...
```

In this case it may make sense to run the *Squish* test as a separate target. The following will accomplish this:

```
    ...
  <target name="test" depends="compile"
    description="run the tests">
    <squish:runtest suite="C:\squish\examples\qt3\suite_addressbook_js" />
  </target>
    ...
```

Finally, running a *Squish* test from the command line gives the following output for the above example when using target test:

```
$ ant test
Buildfile: build.xml

init:

test:
[squish:runtest] Running testcase: add_address_datadriven took 47.0s
[squish:runtest] Running testcase: add_address took 16.0s
[squish:runtest] Tests run : 2, Failures : 0, Errors : 0, Fatals : 0

BUILD SUCCESSFUL
Total time: 1 minute 3 seconds
```

9.4.4 Xml reference

This section gives an overview of the tags that can be used after installing the plugin.

9.4.4.1 squish:config

The config tag can be used as a convenient way to set the path to the *Squish* installation that should be used to run the tests. The table below shows the attributes that can be used:

attribute	value
path	The absolute path to the root directory of *Squish*.
host	The hostname where squishserver is running.
port	The port number where squishserver is listening on.

Table 9.1: Config tag

Here is an example of using the config tag in an Ant build file:

```
<project name="MyProject" basedir="." xmlns:squish="antlib:com.froglogic.squish.ant">
<squish:config path="C:\squish"/>
    <target>
    ...
```

When the config tag is used, like in the example, at the start of the document the path that is set will be used for all *Squish* tests in the targets. When used inside a target container, the path will only be used for that container.

Note It is also possible to set the *Squish* path in the runtest tag, but if there are many *Squish* tests to be run and they all use the same *Squish* installation, the config tag is more convenient.

9.4.4.2 squish:runtest

The runtest tag can be used to run a *Squish* testcase or suite. The table below shows the attributes that can be used:

attribute	value	required
suite	The absolute path to the *Squish* suite to run.	Yes
testcase	The one testcase from the suite to run.	No
host	The hostname of the machine where squishserver is running.	No
port	The port number where squishserver is listening on.	No

Table 9.2: runtest tag

Note The attributes host and port are not marked as required since they can be set using the config tag too.

Here is an example of using the runtest element in an Ant build file:

```
<project name="MyProject" basedir="." xmlns:squish="antlib:com.froglogic.squish.ant">
    ...
  <target name="test" depends="compile"
    <description> run the tests </description>
    <squish:runtest suite="C:\squish\examples\qt3\suite_addressbook_js"
     testcase="add_address" />
  </target>
    ...
```

For the runtest tag to work, the suite attribute has to indicate a valid *Squish* suite. Also it must know the absolute path to the *Squish* installation to use for running tests, either through the path attribute or squish:config element. The ant process will wait for the *Squish* test to be completed, and the results are reported back on standard output (maybe screenshot/show output?).

9.5 CruiseControl integration

CruiseControl is a framework for a continuous build process. It includes, but is not limited to, plugins for email notification, Ant, and various source control tools. A web interface is provided to view the details of the current and previous builds. More information can be found here.

Using the CruiseControl plugin described in this chapter, it is possible to run squish tests from a Cruise Control continuous build.

9.5.1 Obtaining the plugin

You can obtain the CruiseControl plugin from the following URL: www.froglogic.com/download/squish-cc-plugin_latest.jar

9.5.2 Installing the plugin

To install the plugin simply place the jar file in CC_HOME\lib.

9.5.3 Example use of plugin

Below is the example used in the CruiseControl distribution:

```
<cruisecontrol>
    <project name="connectfour">

        <listeners>
            <currentbuildstatuslistener file="logs/${project.name}/status.txt"/>
        </listeners>

        <bootstrappers>
            <antbootstrapper anthome="%%APACHE_ANT%%" buildfile="projects/${project.name}/ ↩
                build.xml" target="clean" />
        </bootstrappers>

        <modificationset quietperiod="30">
            <!-- touch any file in connectfour project to trigger a build -->
            <filesystem folder="projects/${project.name}"/>
        </modificationset>

        <schedule interval="300">
            <ant anthome="%%APACHE_ANT%%" buildfile="projects/${project.name}/build.xml"/>
        </schedule>

        <log>
            <merge dir="projects/${project.name}/target/test-results"/>
        </log>

        <publishers>
            <onsuccess>
                <artifactspublisher dest="artifacts/${project.name}" file="projects/${ ↩
                    project.name}/target/${project.name}.jar"/>
            </onsuccess>
        </publishers>

    </project>
</cruisecontrol>
```

The *Squish* plugin must be declared before using the functionality from it. The best option is to declare it as the first child of the project. Taking the above example, here is how that should look:

```
<?xml version="1.0" encoding="UTF-8"?>
<cruisecontrol>

  <project name="connectfour">
  <plugin name="squishtest"
      path="C:\squish"classname="com.froglogic.squish.cc.builders.SquishTestBuilder"/>
        ....
```

To run a *Squish* test, the following change has to be made to the schedule section:

```
        ....
        <schedule interval="300">
  <composite>
```

```
                <ant anthome="%%APACHE_ANT%%" buildfile="projects/${project.name}/build.xml ↩
                    "/>
<squishtest suite="C:\squish\examples\qt3\suite_addressbook_js"
    testcase="add_address"/></composite>
      </schedule>
      ....
```

The composite is needed since we have multiple tasks in this case, i.e. the ant build file is executed and a *Squish* test is run. Note that it is also possible to do the squishtest first or to use squishtest more than once in order to run multiple testsuites.

9.5.4 Xml reference

This section gives an overview of the tags that can be used after installing the plugin.

9.5.4.1 squishtest

The squishtest tag can be used to run a *Squish* testcase or suite. The table below shows the attributes that can be used:

attribute	value	required
suite	The absolute path to the *Squish* suite to run.	Yes
testcase	The one testcase from the suite to run.	No
path	The absolute path to the root directory of *Squish*.	No
host	The hostname of the machine where squishserver is running.	No
port	The port number where squishserver is listening on.	No

Table 9.3: squishtest tag

Note When the attributes mentioned in the table are set in the *Squish* plugin binding element, they act as default values. So in the plugin binding listed in the previous section, path is set to a default of "C:\squish", and the same can be done for the other attributes, for example to set the default host and port. This means the attributes are inherited from the plugin element and can be overridden again in the squishtest element.

9.6 Maven integration

Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

More information can be found here.

Using the Maven plugin described in this chapter, it is possible to run squish tests during any Maven phase.

9.6.1 Obtaining the plugin

You can obtain the Maven plugin from the following URL: www.froglogic.com/download/squish-maven-plugin_latest.zip

9.6.2 Installing the plugin

To install the plugin first unzip the package. Then run the install-squish-plugin.bat for the windows version and install-squish-plugin.sh for linux.

9.6.3 Example use of plugin

To use the plugin from your POM file add this to the plugins section:

```
<project>
   ....
    <plugins>
       ....
       <!-- Make the Squish plugin known to Maven and run one testcase -->
       <plugin>
         <groupId>com.froglogic.squish.maven</groupId>
         <artifactId>squish-plugin</artifactId>
         <executions>
           ....
           <!-- This test uses local Squish server -->
           <execution>
             <id>test1</id>
             <phase>test</phase>
             <goals>
               <goal>run-test</goal>
             </goals>
             <configuration>
               <path>C:\squish</path>
               <testsuite>C:\squish\examples\qt4\suite_addressbook_js</testsuite>
               <testcase>add_address</testcase>
             </configuration>
           </execution>
           ....
         </executions>
       <plugin>
       ....
    </plugins>
   ....
</project>
```

This example runs the testcase*add_address* from the testsuite *suite_addressbook_js* during the test phase. Note that the test can be run during any phase by changing the phase tag. To do multiple test runs simply add execution sections to executions.

9.6.4 Xml reference

This section gives an overview of the goal and configuration that can be used after installing the plugin.

9.6.4.1 runtest

The runtest goal should be used to run a *Squish* testcase or suite. The table below shows the tags that can be used within the configuration section:

Note When specifying a testcase the tst_ prefix must not be used. For example when *Squish* IDE shows tst_add_address, add_address should be specified.

---

attribute	value	required
testsuite	The absolute path to the *Squish* suite to run.	Yes
testcase	The one testcase from the suite to run.	No
path	The absolute path to the root directory of *Squish*.	Yes
host	The hostname of the machine where squishserver is running.	No
port	The port number where squishserver is listening on.	No

Table 9.4: runtest goal

Chapter 10

Frequently Asked Questions

1. *Why does text entered with type() end up garbled in a Qt application? Why does a clickButton() call have no effect?*

   The Qt 4.3 series has a bug that mixes up the order of posted events in the internal queue. In some seldom cases this breaks text input or mouse click sequences. This problem will be solved in Qt 4.4 and possibly also in Qt 4.3.4. A source code patch is available on request.

2. *How do I Build Squish and Qt on IRIX with gcc*

   To build Squish, Qt must be linked as a shared library. When compiling Qt with gcc in IRIX, it is built as a static library by default. To build Qt as a shared library, specify the `-shared` option when running Qt's configure script.The reason, why static is the default build is because Qt doesn't link as a shared library on IRIX when being built with gcc. To fix this problem, edit the file `src/Makefile` after you have run configure. Add the switch `-Wl,-LD_LAYOUT:lgot_buffer=100` to the Makefile's `LFLAGS` variable. The line should then look similar to the following one:

   ```
   LFLAGS = -Wl,-LD_LAYOUT:lgot_buffer=100 -shared -Wl,-soname,libqt-mt.so.3 -Wl,-rpath,/ ↩
       usr/people/reggie/qt32-gcc/lib
   ```

   After this just compile Qt as usual (by running make). Then compile Squish as documented in this manual.

   Note
   This fix has been submitted to Trolltech support and has been incorporated into Qt releases as of version 3.3.0.

3. *How do I Build Squish with Python Support on MinGW*

   Some Python binary distributions already contain a import library for Python that can be used with MinGW. In such a case you don't have to do anything.If you don't have it, you have to generate it yourself. For this you need the pexports and dlltool tools that are available for MinGW. In the following example we assume that Python is installed in `C:\Python23` and the Python DLL is under `C:\WINNT\system32\python23.dll`. To generate a import library for Python do:

   ```
   cd C:\Python23\libs
   pexports C:\WINNT\system32\python23.dll > python23.def
   dlltool --dllname python23.dll --def python23.def --output-lib libpython23.a
   ```

   For more information see the MinGW Wiki on Python extensions (just the Basic Setup section).

4. *I have problems building* Squish *with a single-threaded Qt 3 library on Windows*

   As mentioned in Section 3.2.7 in the chapter Chapter 3, most parts of Squish can be built with a single-threaded Qt library. On Windows there is one problem with this: The single-threaded Qt library is linked against the static `LIBC.LIB` instead of the shared `MSVCRT.LIB`. This leads to problems with memory management because of different heaps when using multiple DLLs linked against Qt, and this problem affects *Squish*.To work around the problem, open the file `%QTDIR%\mkspecs\win32-msvc\qmake.conf` (or `%QTDIR%\mkspecs\win32-icc\qmake.conf`, if you use Intel C++) and change the lines

```
QMAKE_CFLAGS_RELEASE  = -O1
QMAKE_CFLAGS_DEBUG  = -Z7
```

to

```
QMAKE_CFLAGS_RELEASE  = -O1 -MD
QMAKE_CFLAGS_DEBUG  = -Z7 -MDd
```

After this you must regenerate Qt's Makefile and recompile the Qt library:

```
cd %QTDIR%\src
qmake qt.pro
nmake clean
nmake
```

5. *Why doesn't* Squish *record mouse move events*

   Squish only records mouse move events when a mouse button is pressed (even if Qt's mouse tracking is enabled). This is because in most cases mouse move events that take place when no mouse button is pressed can be ignored since they don't need to be replayed for the widgets to function properly. Furthermore, recording these events would produce a lot of script code which would not affect the application, but would make the script harder to read and maintain.For those situations where it is necessary to record all the mouse move events because they are essential for the widget to work correctly, you can change Squish's behavior. Let's assume that we have a custom widget of type `CanvasView` for which we want to record all mouse move events. We can tell Squish to switch on mouse tracking for widgets of this type by creating and registering a suitable init file. For example, we could create a file called `myinit.tcl` that contained this line:

   ```
   setMouseTracking CanvasView true
   ```

   Then we can tell Squish to execute this file at startup:

   ```
   squishrunner --config addInitScript Qt <absolute_path>/myinit.tcl
   ```

   After this when recording a test case, mouse move events on `CanvasView` widgets will be recorded even if no mouse button is pressed.

6. *Why do my widgets always get names like MyWidget34 and not more descriptive names*

   Squish tries to create the most descriptive names for GUI objects. By default is uses the QObject name. If this property isn't unique, other possibly unique properties are examined (window caption, button text, etc.) before falling back to <ObjectType><num>.One solution (and probably the best!) to get better names in generated scripts is to give all the widgets descriptive, short and unique QObject names in the AUT's source code.If the custom widgets have other possibly unique properties, like a label, etc., *Squish* can be told to try using these properties. For example, suppose that we have a custom widget called `CanvasView` which has a unique `label` property. To tell *Squish* about this, create a file `myinit.tcl` that contains the following line:

   ```
   setUniqueProperty CanvasView label
   ```

   Then tell *Squish* to interpret this file at startup:

   ```
   squishrunner --config addInitScript Qt <absolute_path>/myinit.tcl
   ```

   After this, *Squish* will attempt to use the `CanvasView's label` property for identification if its value is unique.

7. *Why does comparing unicode strings containing e.g. German umlauts fail*

   When comparing a string from the AUT (e.g. a QString such as returned from QLineEdit::text()) with an expected native script string containing 8-bit characters, you get an error although the strings match.The problem here is that *Squish* assumes a UTF8 encoding while your application uses a different default Qt encoding, e.g. Latin1. Now when *Squish* tries to convert the QString to a const char* so it can be compared against the string from the script, this default encoding is used instead of UTF8. In a future version of Squish this problem will be solved, for now the following workarounds are possible:

- Use use UTF8 as default encoding in your application by calling

```
QTextCodec::setCodecForCStrings( new QUtf8Codec );
```

This will result in Qt using UTF8 as encoding when e.g. converting a QString from or to a const char*. Note that this will change the AUT's behavior.

- In the test script explicitly convert the string to a QString, e.g.:

```
test.compare(object.text, QString.fromUtf8("üBcher")
test.compare(object.text, QString.fromUtf8("üBcher");
test compare [invoke $object text] [invoke QString fromUtf8 "üBcher"]
```

- You explicitly convert the QString coming from the AUT to UTF8, e.g.

```
test.compare(object.text.utf8(), "üBcher")
test.compare(object.text.utf8(), "üBcher");
test compare [invoke [invoke $object text] utf8] "üBcher"
```

8. *Why aren't mouse clicks blocked by modal dialogs on Windows when replaying a test*

   On Windows test scripts successfully deliver mouse clicks to widgets which are blocked by a modal dialog, although a user never could perform such a click. To allow *Squish* detecting such a modality to block a click and add an error to the test log, on Windows a private symbol of the Qt library needs to be exported. To do this, open `src\kernel\qapplication.cpp` and replace the line

   ```
   bool qt_tryModalHelper( QWidget *widget, QWidget **rettop );
   ```

   by

   ```
   Q_EXPORT bool qt_tryModalHelper( QWidget *widget, QWidget **rettop );
   ```

   Then rebuild Qt and re-run `configure` and `build` in *Squish*. After this clicks to blocked widgets will be caught by *Squish* resulting, as expected, in an error.On Unix and Mac this problem doesn't exist since this symbol is always exported.

9. *Why does* Squish *fail to hook into my AUT on AIX*

10. *Why does squishrunner fail to start with the error* `cannot restore segment prot after reloc: Permission denied`*?*

    Usually, *Squish* hooks into the AUT without requiring any special preparations. However, in some cases such as on AIX this is not possible due to technical limitations. See the chapter [?] in the Chapter 7 for a solution.

    Due to the usage of 3rd party code and imperfect compilers *Squish* contains libraries with so called "text relocations". By themselves these are harmless but a "hardened" Linux distribution with SELinux (Security Enhanced Linux) installed may nevertheless disallow such a property. We'll investigate solving this problem. Until then, one workaround is to change the *`SELINUX=enforcing`* entry in `/etc/selinux/config` to *`SELINUX=permissive`* and reboot the system. A more fine-grained solution is also possible using the **chcon** command. Please contact support if you are interested in going this route.

11. *Why does* Squish *not properly replay paintings in my painting application*

    Squish always compresses a mouse press events followed by mouse move events, while the button is pressed, into a mouseDrag(). A mouseDrag() is much more robust against changes in the application then recording the individual mouse events. However during the replay of the mouseDrag() there is only one mouse move event synthesized, starting from the point where the mouse button was pressed to the point where the mouse button was released.To enable recording of applications that depend on all of the individual mouse move events being recorded there's a way to disable the creation of the mouseDrag() function for a specific widget type. See `setRecordMouseDrag` for the details on how to do that.

Part I

Java™ Extension API Reference

2008froglogic GmbH

April 10, 2009

Abstract

This reference documents the *Squish* extension API for testing custom Java™ components.

Chapter 11

Package com.froglogic.squish.extension

11.1 Interface Inspectable

Name

Interface Inspectable – This interface needs to be implemented for custom Java support in Squish.

Synopsis

```
 public interface com.froglogic.squish.extension.Inspectable {

// Public Methods  public boolean findObjects(ObjectQuery query,
                              int n,
                              java.util.AbstractList matches);
  public boolean findObjects(ObjectQuery query,
                              Object obj,
                              java.util.AbstractList matches);
  public Object getChildAt(Point pos);
  public Object[] getChildren();
  public Rect getGlobalBounds();
  public String getName();
  public Object getObject();
  public Object getParent();
  public Object getPropertyValue(String prop);
  public boolean isObjectReady();
  public Point mapFromGlobal(Point pos);
  public Point mapToGlobal(Point pos);
}
```

Description

This interface needs to be implemented for custom Java support in Squish. When fully implemented it ensures that the custom classes/widgets are shown correctly in the spy, will show a correct highlight rectangle when picking, and can be used to record/playback events on. An Inspectable interface will have to be implemented for each custom class that needs to be known to Squish. These classes can either be stored in the AUT jar or in an external jar. The classes can look like this for an imaginary canvas example:

```
class MyCanvasItemExtension extends InspectableAdapter
{
```

```
  MyCanvasItem itm;
  public Rect getGlobalBounds()
  {
   ....
}
class MyCanvasExtension extends InspectableAdapter
{
  MyCanvas can;
  public Rect getGlobalBounds()
  {
   ....
}
```

After defining such classes for every custom Object that needs to be exposed, these need to be registered into Squish. The way to do it is to create a factory that returns the right Inspectable for the given object:

```
class MyCanvasFactory implements InspectableFactory
{
    public Inspectable query( Object object )
    {
        if ( object instanceof MyCanvas ) {
            return new MyCanvasExtension( object );
        } else {
     ....
    }
```

Now we have the factory we need an init hook which registers the factory into the registry:

```
     public static void init( InspectableRegistry registry )
     {
         registry.register( new MyCanvasFactory() );
     }
 }
```

Also, in the manifest file for the jar where the above extensions are stored, there needs to be an extra entry so that Squish can find the init method:

```
Extension: MyCanvasFactory
```

Finally, in order to let Squish know the wrapped custom Java classes better, it is encouraged to let Squish know the new wrapped classes better through the .ini file. See Wrapping custom Java classes in the user guide. The same .ini file should be used to let Squish know the location of the extension jar file generated above. An entry can look like this: JavaExtensionDir="/home/squish/extensions/", "/home/squish/mycanvas"

See Also , ,

Methods

findObjects(ObjectQuery, int, AbstractList)

publicboolean`findObjects`ObjectQuery*`query`*int*`n`*java.util.AbstractList*`matches`* PARAMETERS

`query` ObjectQuery object

`n` no more then n objects need to be found

`matches` list that matches the query in order of found

return true if n objects are in the list

Find and add objects that matches query in a list. The search should end if list contains n objects. This function is used for finding the n-th reoccurrence of an object.

IMPORTANT: This function is *subject to change* at any time. Use the InspectableAdapter default implementation, unless there are good reasons not to.

findObjects(ObjectQuery, Object, AbstractList)

publicboolean`findObjects`ObjectQuery*`query`*Object*`obj`*java.util.AbstractList*`matches`* PARAMETERS

`query` ObjectQuery object

`obj` stop searching up until obj is found

`matches` objects that matches the query in order of found

return true if obj is in the list

Find and add objects that matches query in a list. The search should end after obj is found. This function is used for finding the occurrence property of an object.

IMPORTANT: This function is *subject to change* at any time. Use the InspectableAdapter default implementation, unless there are good reasons not to.

getChildAt(Point)

publicObject`getChildAt`Point*`pos`* PARAMETERS

`pos` the point to use for hit-testing

return the object that is hit

Return the child hit at position pos. Containers should return the exact child that is hit if it supports recording at that level of detail (for instance canvasses). The hit child can also be an indirect child, for instance a child of a child of the container. The coordinates are relative to the wrapped object.

getChildren()

publicObject[]`getChildren` PARAMETERS

return array of all children

Return all children of the wrapped object.

getGlobalBounds()

publicRect`getGlobalBounds` PARAMETERS

return the bounds as a Rect

Gets the bounds of this component in the form of a Rect object. The bounds specify this component's width, height, and location relative to its container, all in the container's coordinate system.

getName()

publicStringgetName PARAMETERS

return the name

Return the name of this object.

getObject()

publicObjectgetObject PARAMETERS

return the object

Return the object that this Inspectable inspects.

getParent()

publicObjectgetParent PARAMETERS

return the parent

Return the parent of this object.

getPropertyValue(String)

publicObjectgetPropertyValueString*prop* PARAMETERS

prop property name

return value of this property or null

Return the value of a string property. This function is used for creating an object name for the object map. In the descriptors xml file, any property name "abc" from getter functions "getAbc()" may be added if the object class is known to Squish. For object classes not known to Squish, or if there is not a getter function, this function should return the value instead. Properties marked in the descriptor xml as object, must have a matching Inspectable.

isObjectReady()

publicbooleanisObjectReady Returns true when the wrapped object is ready to receive user input like text for a text field, or button clicks for a button, false otherwise.

mapFromGlobal(Point)

publicPointmapFromGlobalPoint*pos* PARAMETERS

pos the global coordinate

return the converted point

Converts the global/screen coordinates x,y into coordinates local to the wrapped object and returns the result as a Point.

mapToGlobal(Point)

publicPoint`mapToGlobal`Point*`pos`* PARAMETERS

`pos` the local coordinate

return the converted point

Converts the local coordinates x,y into absolute screen coordinates and returns the result as a Point.

11.2 Class InspectableAdapter

Name

Class InspectableAdapter – Adapter class for writing extensions.

Synopsis

```
 public abstract class com.froglogic.squish.extension.InspectableAdapterimplements com.fro

// Public Constructors  public InspectableAdapter();
// Public Methods  public boolean findObjects(ObjectQuery query,
                             int n,
                             java.util.AbstractList list);
  public boolean findObjects(ObjectQuery query,
                             Object obj,
                             java.util.AbstractList list);
  public Object getChildAt(Point pos);
  public Object[] getChildren();
  public Rect getGlobalBounds();
  public String getName();
  public Object getPropertyValue(String prop);
  public boolean isObjectReady();
  public Point mapFromGlobal(Point pos);
  public Point mapToGlobal(Point pos);
// Protected Methods  protected boolean findObjects(ObjectQuery query,
                               int n,
                               Object obj,
                               java.util.AbstractList list);
}
```

Description

Adapter class for writing extensions. You can use this class when one or more of the default implementations presented here for the Inspectable methods make sense for your extension. You must to reimplement `Inspectable.getObject()`

Methods

findObjects(ObjectQuery, int, AbstractList)

publicboolean`findObjects`ObjectQuery*`query`*int*`n`*java.util.AbstractList*`list`* PARAMETERS

`query` ObjectQuery object

n no more then n objects need to be found

matches list that matches the query in order of found

return true if n objects are in the list

Description copied from interface: <link linkend="method-com.froglogic.squish.extension.inspectable.findobjects-com.froglogi int-java.util.abstractlist">findObjects</link>

Find and add objects that matches query in a list. The search should end if list contains n objects. This function is used for finding the n-th reoccurrence of an object.

IMPORTANT: This function is *subject to change* at any time. Use the InspectableAdapter default implementation, unless there are good reasons not to.

findObjects(ObjectQuery, int, Object, AbstractList)

protectedboolean`findObjects`ObjectQuery*query*int*n*Object*obj*java.util.AbstractList*list* Depth first implementation for findObjects.

Squish calls the function when searching for an object, its container or window object. Therefore the properties 'container', 'window' and 'occurrence' should not be used when searching for objects. The 'type' property is for performance reasons taken out of the `ObjectQuery.match` function and should be checked in this function instead.

findObjects(ObjectQuery, Object, AbstractList)

publicboolean`findObjects`ObjectQuery*query*Object*obj*java.util.AbstractList*list* PARAMETERS

query ObjectQuery object

obj stop searching up until obj is found

matches objects that matches the query in order of found

return true if obj is in the list

Description copied from interface: <link linkend="method-com.froglogic.squish.extension.inspectable.findobjects-com.froglogi java.lang.object-java.util.abstractlist">findObjects</link>

Find and add objects that matches query in a list. The search should end after obj is found. This function is used for finding the occurrence property of an object.

IMPORTANT: This function is *subject to change* at any time. Use the InspectableAdapter default implementation, unless there are good reasons not to.

getChildAt(Point)

publicObject`getChildAt`Point*pos* PARAMETERS

pos the point to use for hit-testing

return the object that is hit

Description copied from interface: <link linkend="method-com.froglogic.squish.extension.inspectable.getchildat-com.froglogic

Return the child hit at position pos. Containers should return the exact child that is hit if it supports recording at that level of detail (for instance canvasses). The hit child can also be an indirect child, for instance a child of a child of the container. The coordinates are relative to the wrapped object.

getChildren()

publicObject[]`getChildren` PARAMETERS

return array of all children

Description copied from interface: <link linkend="method-com.froglogic.squish.extension.inspectable.getchildren">getChildren

Return all children of the wrapped object.

getGlobalBounds()

publicRect`getGlobalBounds` PARAMETERS

return the bounds as a Rect

Description copied from interface: <link linkend="method-com.froglogic.squish.extension.inspectable.getglobalbounds">getGl

Gets the bounds of this component in the form of a Rect object. The bounds specify this component's width, height, and location relative to its container, all in the container's coordinate system.

getName()

publicString`getName` PARAMETERS

return the name

Description copied from interface: <link linkend="method-com.froglogic.squish.extension.inspectable.getname">getName</lin

Return the name of this object.

getPropertyValue(String)

publicObject`getPropertyValue`String*`prop`* PARAMETERS

`prop` property name

return value of this property or null

Description copied from interface: <link linkend="method-com.froglogic.squish.extension.inspectable.getpropertyvalue-java.lang.string">getPropertyValue</link>

Return the value of a string property. This function is used for creating an object name for the object map. In the descriptors xml file, any property name "abc" from getter functions "getAbc()" may be added if the object class is known to Squish. For object classes not known to Squish, or if there is not a getter function, this function should return the value instead. Properties marked in the descriptor xml as object, must have a matching Inspectable.

isObjectReady()

publicboolean`isObjectReady`

Description copied from interface: <link linkend="method-com.froglogic.squish.extension.inspectable.isobjectready">isObject

Returns true when the wrapped object is ready to receive user input like text for a text field, or button clicks for a button, false otherwise.

mapFromGlobal(Point)

publicPoint`mapFromGlobal`Point*pos* PARAMETERS

pos the global coordinate

return the converted point

Description copied from interface: <link linkend="method-com.froglogic.squish.extension.inspectable.mapfromglobal-com.froglogic.squish.extension.point">mapFromGlobal</link>

Converts the global/screen coordinates x,y into coordinates local to the wrapped object and returns the result as a Point.

mapToGlobal(Point)

publicPoint`mapToGlobal`Point*pos* PARAMETERS

pos the local coordinate

return the converted point

Description copied from interface: <link linkend="method-com.froglogic.squish.extension.inspectable.maptoglobal-com.froglo

Converts the local coordinates x,y into absolute screen coordinates and returns the result as a Point.

11.3 Interface InspectableFactory

Name

Interface InspectableFactory – A Factory that delivers an Inspectable for custom Objects that it manages.

Synopsis

```
 public interface com.froglogic.squish.extension.InspectableFactory {

// Public Static Fields  public static final int MAX_PRIORITY = 10;
  public static final int MIN_PRIORITY = 1;
  public static final int NORM_PRIORITY = 5;
// Public Methods  public int priority();
  public Inspectable query(Object object);
}
```

Description

A Factory that delivers an Inspectable for custom Objects that it manages.

Fields

MAX_PRIORITY

publicstaticfinalint`MAX_PRIORITY`10 The maximum priority that an InspectableFactory can have.

MIN_PRIORITY

publicstaticfinalint`MIN_PRIORITY`1 The minimum priority that an InspectableFactory can have.

NORM_PRIORITY

publicstaticfinalint`NORM_PRIORITY`5 A good default priority for an InspectableFactory.

Methods

priority()

publicint`priority` PARAMETERS

return a value indicating the priority of this factory.

The priority of this factory. Possible values should lie between MIN_PRIORITY (1) and MAX_PRIORITY(10). The priority NORM_PRIORITY is a good default value to use.

query(Object)

publicInspectable`query`Object*`object`* PARAMETERS

`object` the Object to get an Inspectable for.

return an Inspectable that handles the queried object, otherwise null.

Query this factory for an Inspectable that can handle Object object.

11.4 Class InspectableRegistry

Name

Class InspectableRegistry – This class allows registration of factories of extensions that implement Inspectable.

Synopsis

```
 public abstract class com.froglogic.squish.extension.InspectableRegistry {

// Public Constructors  public InspectableRegistry();
// Public Static Methods  public static final InspectableRegistry getRegistry();
  public static final void setRegistry(InspectableRegistry r);
// Public Methods  public abstract Inspectable getInspectable(Object obj);
  public abstract void register(InspectableFactory factory);
}
```

Description

This class allows registration of factories of extensions that implement Inspectable. It is meant to be implemented inside Squish, and used as part of the extension API in init.

See Also ,

Methods

getInspectable(Object)

publicabstractInspectable`getInspectable`Object*`obj`* PARAMETERS

`obj` the object

return An Inspectable for this object obj, or null if non found

Get Inspectable that for an object.

getRegistry()

publicstaticfinalInspectableRegistry`getRegistry` PARAMETERS

return The global registry

Get global registry.

register(InspectableFactory)

publicabstractvoid`register`InspectableFactory*`factory`* PARAMETERS

`factory` the factory to register

Register the factory in the registry.

setRegistry(InspectableRegistry)

publicstaticfinalvoidsetRegistryInspectableRegistry*r* Set global registry.

11.5 Interface ObjectQuery

Name

Interface ObjectQuery –

Synopsis

```
 public interface com.froglogic.squish.extension.ObjectQuery {

// Public Methods  public int getIntProperty(String property);
  public Object getObjectProperty(String property);
  public String getStringProperty(String property);
  public boolean hasProperty(String property);
  public boolean match(Object obj);
  public boolean matchProperty(String property,
                               String value);
}
```

Methods

getIntProperty(String)

publicintgetIntPropertyString*property* PARAMETERS

return int value of property

Return property value as int.

getObjectProperty(String)

publicObjectgetObjectPropertyString*property* PARAMETERS

return resolved object from property value or null if not found

Return property value as object. The value of this property must be a multi-property string or a symbolic name.

getStringProperty(String)

publicStringgetStringPropertyString*property* PARAMETERS

return string value of property

Return property value as string.

hasProperty(String)

publicbooleanhasPropertyString*property* PARAMETERS

return if real name has this property

Return if the real name contains a property

match(Object)

publicbooleanmatchObject*obj* PARAMETERS

return true if the match succeeds

Return if an object matches the query.

matchProperty(String, String)

publicbooleanmatchPropertyString*property*String*value* PARAMETERS

return true if the match succeeds

Return if a property matches a value.

11.6 Class Point

Name

Class Point – Toolkit independent Point class.

Synopsis

```
 public class com.froglogic.squish.extension.Point {

// Public Fields  public int x ;
  public int y ;
// Public Constructors  public Point();
  public Point(int x,
               int y);
// Public Methods  public boolean equals(Object obj);
  public String toString();
}
```

Description

Toolkit independent Point class.

Constructors

Point()

publicPoint Construct a point at origin (0, 0).

Point(int, int)

publicPointintxinty PARAMETERS

x x position of the point

y y position of the point

Construct a point at location x, y.

11.7 Class Rect

Name

Class Rect – Toolkit independent Rectangle class.

Synopsis

```
 public class com.froglogic.squish.extension.Rect {

// Public Fields  public int height ;
  public int width ;
  public int x ;
  public int y ;
// Public Constructors  public Rect();
  public Rect(int x,
              int y,
              int width,
              int height);
// Public Methods  public boolean contains(Point p);
  public boolean equals(Object obj);
  public String toString();
}
```

Description

Toolkit independent Rectangle class.

Constructors

Rect()

publicRect Construct a rectangle with zero width and height at left-top position (0, 0).

Rect(int, int, int, int)

publicRectintxintyintwidthintheight PARAMETERS

x x position of the rectangle

y y position of the rectangle

width the width of the rectangle

height the width of the rectangle

Construct a rectangle at left-top position x, y, with given width and height.

Chapter 12

Constant field values

Appendix A

Acknowledgments

- This software ships with a copy of the KDE JavaScript engine (KJS) which is copyrighted by the authors of the K Desktop Environment and Apple Computer, Inc.
- To provide regular expression support on all platforms the JavaScript engine includes the "Perl-Compatible Regular Expressions" library written by Philip Hazel ph10@cam.ac.uk (© 1997-2003 University of Cambridge).
- Support for the test data import from Excel spreadsheets is provided by the portable spreadsheet library Swinder (© 2003-2006 Ariya Hidayat ariya@kde.org) which is part of the KOffice application suite.
- Testing Qt applications on Mac OS X uses portions of the mach_star project (© 2003-2005 Jonathan 'Wolf' Rentzsch).
- The XML module available to JavaScript users, as well as the squishidl tool, is based on tinyxml (© 2000-2006 Lee Thomason leethomason@mindspring.com and others) which is subject to the following license (the 'zlib license'):

```
This software is provided 'as-is', without any express or implied
warranty. In no event will the authors be held liable for any
damages arising from the use of this software.

Permission is granted to anyone to use this software for any
purpose, including commercial applications, and to alter it and
redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must
not claim that you wrote the original software. If you use this
software in a product, an acknowledgment in the product documentation
would be appreciated but is not required.

2. Altered source versions must be plainly marked as such, and
must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any source
distribution.
```

Appendix B

Index