
M A S T E R  T H E S I S

# Using Lixto for Abstract Automation of Web Applications

written at

Vienna University of Technology

Computer Science Department

Institute of Information Systems (184)

Database and Artificial Intelligence Group (184/2)

under supervision of

O.Univ.Prof. Dipl.-Ing. Dr.techn. Georg Gottlob

by

Christopher Semturs

Leegasse 10/38

1140 Wien

Vienna, 11. November 2006

| | |
|---|---|
| Date | Signature |

# Chapter 1

# Declaration of Authorship

I certify that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other University.

Vienna, 11. November 2006

| | |
|---|---|
| Date | Signature |

# Chapter 2

# Abstract

Software Test is an important part of the whole Software Process - no matter whether you follow Extreme Programming, Rational Unified Process or any other software process available. This is also well accepted by many companies.
However, within Software Test, most companies rely on a straight-forward approach: Having single technicians doing the test - and trying to do it from a professional point of view. Especially when professional testing meets technical testing (on test automation), the technical tester is forced very often to learn facts on insurance companies, credit risk or protocol testing he probably never wanted to know.
This is the reason why the concept of "Abstract Automation" was introduced, therefore effectively separating technical and professional aspects of Software Testing.
In chapter 5 on page 2, you will be able to learn all about this concept and it's advantages.

On the other side, web-applications gained big attention. In the last few years, more and more high-professional applications have been made available on the Web (e.g. Web-Mail which acts like a "real" application, or even spreadsheets and word processing). A lot of new technologies have been introduced, which are tied together. Chapter 6 on page 29 provides a technical insight on web-technologies in use.

The technical platform in use (e.g. Web, Java, MFC) is not important for Software Testing in General, but it is very important when test automation is desired. The market reacted and provided Tools for Test-Automation especially for Web-Applications - which all focused on the test-environment itself, but not so much on effective "understanding" of the web-page. However, none of them follows the modern approach of Abstract Automation. You can see an overview of the products investigated in chapter 5.5.1.4 on page 20.

Lixto, on the other hand, provides a framework to get a good “understanding” of the web-page and to automatise web interactions - but no real support for test-automation “out-of-the-box”.
Therefore, the main aim of this master thesis is to provide a “Proof of Concept” that Lixto is actually well-usable for Abstract Automation - and to provide a small prototype for this type of functionality.

# Chapter 3

# Acknowledgments

I would like to take this opportunity to express my deepest gratitude to the following people.

I would like to thank my parents - they encouraged me to continue at university and to write the master thesis. Without your unending support, going to university would have been much harder.

I would like to thank my companion in life, Petra. You encouraged me to continue until the end, and were always successful to show me that life is more than just IT and software, especially when I was buried in work too much.

I would also like to thank Roland. He was my mentor at work right from the beginning. Whatever I learned about work, IT or programming languages - I learned it from him. Without his extensive support, life at university would have been much harder, if not impossible in parallel to the work.

I would like to thank Professor Gottlob, who enabled me to include a big part of my academical career (Lixto) and a big part of my business career (Software Test) into one combined master thesis.

I would like to thank Dr. Robert Baumgartner, who supervised me for the whole master thesis and provided me valuable input, yet still giving me a huge flexibility during writing.

# Contents

# List of Figures

# List of Tables

# Listings

# Chapter 4

# Introduction

While studying at the Technical University of Vienna, I had the luck to work at the company Objentis[1], where one focus of the company was Software Testing. While not being active in Software Test on my own, working in the same company gave me valuable insight into software test in general, aspects on test automation and different methods of automation. I was able to take brief looks at "real-life-projects" on test automation at insurance and bank companies.

In parallel, on projects within the courses, I got in touch with Lixto[2], also right from the beginning. Lixto is a product for effective extraction of data from web-applications - which means, not only from static web pages, but also from dynamic web-pages, where the content must be accessed before using different forms, for instance.

Within this master thesis, I combined both ideas into one. While my primary aim is to make more people know on newer test-practices in general, it might also a good start for the Lixto-Product to enhance its focus.

[1] http://www.objentis.com
[2] http://www.lixto.com

# Chapter 5

# General Theory of Testing

## 5.1 Introduction

Within this chapter, the main theories of software testing are going to be explained. Having this background knowledge is very important for understanding the requirements for test-automation, its focus of usage and how it fits into the global concept of test case management.
It is very important to know the basic principles of software test in order to be able to understand software test automation and its requirements.

## 5.2 Test Definition

Software Test itself is defined as written in [Kan88]:

> Software testing is the process used to help identify the correctness, completeness, security and quality of developed computer software.

This means, on Software Testing you verify that all functionalities and features of a system under test are working as designed and promised. Each test-run results into a clear statement of success on this goal.

## 5.3 Testing Terms

For understanding software testing, it is important to get insight to the most commonly used terms and concepts.

In the next sections, the following terms are going to be explained:

- Test Structure Items (see chapter 5.3.1) - this also include the test-case itself
- Flight Scenario Types (see chapter 5.3.2 on page 5) - Flight-Scenarios are the high-level-grouping of testing concepts (whether you test data-consistency under severe conditions, like power-outages, for instance).
- Test Transparency (see chapter 5.3.3 on page 8)
- Test Strategies (see chapter 5.3.4 on page 9)

### 5.3.1 Test Structure

It is very easy to say "test an application". However, just like development has to break its work into several sub-parts, so does the test-team have to do for their testing. For splitting the work, several levels of testing are to be used.
Each application, no matter if it is web or rich-client, no matter if it is user-driven or an automated software, can be splitted into a functional tree - all functions of the application have a dependency to each other. This is an an excerpt of an example functional tree:

- Web Mail Application
  - Security
    - Login behavior
  - Person Record Handling
    - Create New Record
    - Modify existing record
    - delete record
    - search for record
    - browse list

These functionalities and their dependencies can be extracted directly from the specification, or by analysing the implemented application itself.
Once the functional tree is established, it can be mapped into the so-called test case tree.

**Test Case Tree** Representing the functional hierarchy of the application under test. Used for grouping test cases

**Test Case** Single Test Case to be executed. Test Cases are the leafs of the test-case-tree.

The test-case-tree is very similar to the functional tree - for each function, one or more test-cases can be written:

- All Text Cases
  - Security Test Cases
    - Login behavior
      - **Login using correct credentials**
      - **Login using wrong password**
      - **Login using wrong user name**
      - **Login 3-times wrong, then once correct - should still be rejected**
  - Person Record Handling
    - Create New Record
      - **Create using valid data**
      - **Create, but leave out mandatory field "surname"**
      - **Create, but do not save. Verify that entry is not in list**
      - **Create duplicated entry, verify that user gets warning**
    - Modify existing record
      - **Modify Birth date**
      - **Modify surname - verify re-sorting on list**
      - **Don't modify anything, warning should occur**
    - delete record
      - **Delete - verify confirmation comes up**
      - **Delete and deny confirmation - verify entry still there**
    - search for record
      - **Search by Last Name**
      - **Search by Last Name and First Name**
      - **Search by Birthday**
    - browse list
      - **Sort by Last Name**
      - **Sort by First Name**
      - **Sort by Birthday**

Do not forget: Each function must be covered by at least one test case - but it is not limited to one test-case only.

#### 5.3.1.1 Test Case

It is important to note that each test-case should be executable on its own, independent of any other test-cases, and independent of any test tree positioning.
Each test-case should contain the following information elements:

**Pre-Condition** Pre-Conditions which must be fulfilled before the test-case can be executed. Typical pre-conditions include existence of data (e.g. in order to test proper deletion of data records, they must exist previously in order to execute the "delete"-action). Pre-conditions have to be formatted in an executable way. This means, for manual testing they must contain clear instructions, and for test-automation they must be entered in a formal language. Pre-conditions might consist of test-actions (see chapter 5.4.1.3 on page 12). If the pre-condition fails, the test case has to be marked as "inconclusive", as it cannot be executed. The pre-condition itself should be redundant to other test-cases - as for the previous example, this means that there should be a test-case covering the creation of data records.

**Test Actions** Test actions (see chapter 5.4.1.3 on page 12) of the test-case itself are the ones to be executed. If any of the test-actions fail, it must be evaluated manually if the test-case is "failed", or if the automation itself has a problem. In case of manual testing, failing a test-action means failing the test-case.

**Post-Condition** After the test-case is executed, clean-up has to be executed on the application under test. For instance, if a test-case verifies that no two entries for the same social security number must be allowed to enter, the post-condition must clean-up the first entry being created - otherwise the pre-condition would fail on the second run.
It is hard to make the post-condition in a correct way. As a general guideline, the post-condition must ensure that, if the test-case is executed twice or several times, the result shall not vary.

### 5.3.2 Flight Scenario Types

To distinguish the different scenario types of testing, analogies to flight scenarios are widely chosen. They include the following scenarios. A general overview can be seen in table 5.1 on the next page:

- Flight in the nice weather (see chapter 5.3.2.1 on the following page)

| Scenario Type | Professional | Technically | Used for Web-Applications |
|---|---|---|---|
| Flight in the nice weather | Must be correct | Must be correct | Yes |
| Cloudy Flight | Can be wrong | Must be correct | Yes |
| Flight in the bad weather | Can be wrong | Can be wrong | Yes |
| Catastrophic Flight | Can be wrong | Can be wrong | No |

Table 5.1: Comparison of Flight Scenario Types

- Cloudy Flight (see chapter 5.3.2.2)
- Flight in the bad weather (see chapter 5.3.2.3 on the following page)
- Catastrophic Flight (see chapter 5.3.2.4 on the next page)

#### 5.3.2.1 Flight in the nice weather

This is the most basic test scenario to be executed. It validates the correct functionality of the application under test for perfect conditions, which means:

- The user is behaving technically correct and as expected
- All input data is correct from the professional point of view

Examples include the test of a working login providing correct credentials, always using the mouse instead of trying standard keyboard shortcuts.
Passing this test phase is mandatory for having an application released to public testing, it covers the basic functionality which must work. Still, having passed this test can only act as a release milestone for the beta-version, never for a final release.
Nice-Weather-Scenarios are very good scenarios for test automation. These scenarios are the ones which will have to be executed most often, and are often also part of the regression test (see chapter 5.3.4 on page 9) acceptance criteria.

#### 5.3.2.2 Cloudy Flight

Cloudy flight is the next phase, based on nice-weather-scenarios. It aims at variety from the professional point of view. So while, technically, the tester must still act correct, he can en widen his test spectrum by using false data. This might be a set of invalid credentials (for instance wrong password) used for login, or searching for a term which does not exist within the search index.
Having this scenario pass is a mandatory criteria for releasing a product, although not sufficient on it's own.

#### 5.3.2.3 Flight in the bad weather

Bad Weather Flights are finally allowing the input of wrong data, both from the professional point of view, but also technically. Famous examples include invalid characters on searches or using all type of keyboard shortcuts instead of mouse. Especially for web-applications, this might also include testing the behavior under various, non-supported browsers or browser-settings (the user should at least get a qualified error message).
Having this scenario pass means that the application under test can be released to the public, as long as it does not require to fulfill special safety conditions (which is usually not the case for web-based applications).

#### 5.3.2.4 Catastrophic Flight

The last scenario type impacts the behavior of the application, or rather a whole system, under catastrophic scenarios. A "catastrophic scenario" might include:

- Behavior of a server-system under hardware damage
- Data consistency / data recovery on power failures
- Ensure availability in critical systems (like mainframe controlling phone networks)

Testing these types of scenario is usually required for highly critical systems - either in real - time - environment, or on server-side in general. Typical examples include Nuclear Power Plants, or chip functionality within an airplane. We will most likely never touch those test scenarios when testing web applications (as they are end-user specific), therefore they won't be handled in detail here.

#### 5.3.2.5 Summary

The main important test scenarios for web-applications are Nice-Weather-, Cloudy- and Bad-Weather-Scenarios. All of them are candidates for automation, even though most likely the focus on automation will be put on the nice-weather-scenarios, as they will be used very often for regression testing (a repetitive task, see chapter 5.3.4 on page 9).

### 5.3.3 Test Transparency

Depending on your testing needs, different levels of application transparency can be chosen for testing:

- BlackBox Testing (see chapter 5.3.3.1)
- GreyBox Testing (see chapter 5.3.3.2)
- WhiteBox Testing (see chapter 5.3.3.3 on the following page)

#### 5.3.3.1 BlackBox Testing

BlackBox Testing, also known as behavioral testing, emulates the end-user testing. It forces the tester to act on the application right via the user interface, not making any technical bypasses of any kind.
Especially for end - user - applications (the main usage-scenario for web-applications), black box testing is one of the most crucial testing methods to be used. Having black-box-tests passed ensures that the end-user functionality works as desired.
Each Test-case within black-box-testing must concentrate on one specific function only - having different functions mixed in one test case makes tracking of test-results much harder. One could also say that each black box testcase represents a single use-case (in case use-cases are defined).
As for web applications, black box testing implies that the testing is done via the browser.

#### 5.3.3.2 GreyBox Testing

Moving from black-box to grey-box, the tester has to move a little bit into the system. While, on black-box-testing, the UI is the limit, for grey-box, the tester is allowed to dive into the data sources, for instance. Grey-Box-Testing makes sense especially in scenarios where data is hard to emulate via blackbox, or where some verification-methods are simply not accessible.
Typical scenarios include data-verification directly via the database: For instance, the user is allowed to fill in a response-form via the web. Based on the input of this form, a mail has to be sent, and the tester wants to verify this. He needs to enter the grey box in order to verify the content of the receiving mailbox.
On web-tests, Grey-Box-testing is mainly used for data verification, or for generating

direct test data where a web-interface is not accessible (e.g. direct SQL manipulations).

#### 5.3.3.3 WhiteBox Testing

White-Box Testing is quite often mixed up with unit-testing[1]. The white-box-test is allowed to access directly the API by any means.
White-Box-Testing is a hard job for the testing - they have to keep strictly to the defined APIs by development, and they have to follow changes on each release. In addition, the whole application and workflow logic has to be understood and supported. In case of automation errors, a lot of analysis is needed in order to understand if the problem is due to the application under test, or due to the way of automation.
White-Box testing does not pay off on normal regression test and other types of automation. It is mainly used for scenarios where blackbox-testing is too slow - like for instance stress tests.

### 5.3.4 Test Strategies

Within software-testing, a series of specialised terms have been established, in order to represent the different types of testing (also known as test strategies):

**Stress tests** Stress-tests goes beyond the pure functional test. It does not verify correct functionality, but it especially analyses the scaling of the application under stress (e.g. 1.000 users instead of 2). Stress-test often reveal new information and facts because the developers could not emulate the scenarios on their own.
Stress-Test must be automated.

**Tests against specification** Strictly sticking to the specification, the tester sits in front of the application and verifies the behavior. This type of testing not only brings up items implemented in a wrong way (or not at all), but it also shows up the weaknesses in keeping specifications up-to-date.
Usually, no test-automation is applied here.

**Tests of risk** This test-type, also done manually most often, focuses on the highest risk of failure within an application. For instance, in online banking it is not really crucial if the account information being displayed is complete (as long as it

[1]Unit-Tests are developer-specific code-pieces for verifying API-functionality. They are done by development themselves and do not touch the external testing by any means

is a display-error), but it is very important to have a correct verification method on online transactions.

Nowadays, risk-based testing is overlapping to the security-tests, especially on web-applications.

**Tests with random data** On functional tests, it is very important to test using the same test-data (or the same set of test-data), in order to have reproducible results. However, there might also be cases where special data inputs are causing problems which would have never shown up in normal functional testing. This is the part where tests using random data pay off - they run the same functional tests over and over again, and try to determine abnormal behavior (e.g. longer response times, exceptions, ...). Famous examples include random testing of phoning switches (where a rare number of digit combinations caused the switch to crash) or testing of map routing software, where some routes show up as a duration of a few days while, in reality, it should last 10 minutes only.

This type of testing should not be confused with stress-testing, where the same load of data is used in parallel a few hundreds times to simulate stress-scenarios. When testing web-applications, tests using random data are just as important as on normal rich-client-applications. The more user inputs are allowed, the more likely it is that certain input patterns are causing problems on the software.

**Function tests** The "real" testing, or the classic one as some would say. Functional testing focuses on verifying functionality. For each functionality under test, an own test case can be developed, and it can be executed against the application using a specific set of data. Functional tests are designed to be reproducible, and they are a classical candidate for test automation.

**Regression tests** A subset of the function tests is used for the regression test. The so-called "regression test set" must be executed on each delivery from development to test, and it must be passed successfully before any of the other tests starts. If the regression test fails, the delivery is not accepted. Therefore, regression-test-cases must contain all basic functionalities (login, search, add new entry, ...), but it must not fail due to minor problems in the application, which can still be analysed on functional testing.

The regression-test-set is the first set of test cases which are automated.

**Scenario tests** Sometimes, a single functional tests is not sufficient. For instance, on large intranet-applications, a series of actions must work together in order to have the application work. Example: On insurance-companies-intranets, when the sales-person adds a new contract, it must be added to all internal systems,

and a letter of welcome must be sent to the customer. All these steps can be verified on their own, but at the end, the whole scenario must also be tested for working together. This is also known as integration test.

**Security tests** Security becomes more and more a very important topic on web applications (although not only there). Security-Tests focus on the typical weaknesses of applications (input handling, for instance) and emulate the "evil part" in order to break the system. Any flaw showing up is an error.
Security Tests are most often not limited to BlackBox-Testing. In order to find security flaws, a series of greybox and whitebox-testing-methods have to be applied (see chapter 5.3.3 on page 8). For instance, a typical security-test might try to ensure that the credentials of the user are transmitted only once, and that the session of the user cannot be hijacked from another machine just by knowing the cookies (see chapter 6.4.4 on page 44) of the user.

Of course, there's a series of other test strategies being available, which are used in several occasions. However, this is the list of main test strategies one should be aware of.

## 5.4 Structured Test Methodology

In this chapter, we are going to explain and cover basic testing methodologies and tools being used for structured and formalized testing. Testing Terms themselves are covered in chapter 5.3 on page 2). Structured testing also includes test-automation, but not only.

### 5.4.1 Test Case Structure

Each Test-Case being executed covers exactly one functionality (see chapter 5.3.1.1 on page 5). However, in order to be able to execute the test-case, the quality of test-case-description can vary.

#### 5.4.1.1 Descriptive Test-Case

The fastest way to describe a test-case is the descriptive version. To achieve this, the test-case could be formulated in a way as shown in table 5.2 on the next page.

| Action | Content |
|---|---|
| **Pre-Condition** | Make sure that the database is empty, and no data records are available. |
| **TestCase** | Add a new person to the customer list, having the name "Franz Meier". Afterward, search for the same person to contain via the search form. Verify that it shows up as a search result. |
| **Post-Condition** | Delete the person again. |

Table 5.2: Descriptive Test Case

This description might help to test when you know the application - but it is a very basic description and leaves space for interpretation by the test-executor. For instance, as for the search form, there might be several fields to fill in. So, shall he search for last name, first name, both, or something else? And does the person have to contain any additional attributes (e.g. "customer"-flag, in order to show up as search results), or not?

#### 5.4.1.2 Detailed Test-Case

In order to make the test-case more precise, it can be described in a much more detailed way, as you can see in table 5.3 on the following page.

As you can see, this description is far more precise than the previous one. Even if somebody did not yet see the application under test, he can execute the test-case.
This type of test-case description is very powerful for manual testing - especially for regression-test, virtually anybody can execute the tests, and you are no longer limited to the rare resources knowing the application really well. Each test is reproducible, as the instructions are very clear.
However, this type of test cases can not be used for proper test automation, and

#### 5.4.1.3 Formal Test Case using Test-Actions

As for the simple writing of test-cases, we have reached the best formal way being available by plain text. In order to achieve even better test-case content, a new set of abstract description layers is required

- Test Control
- Test Action

| Action | Content |
|---|---|
| **Pre-Condition** | Make sure that the database is empty, and no data records are available. In order to verify this, take a look at the database-table "PersonRecords" using direct SQL-statements - the table must be empty. |
| **TestCase** | Add a new person to the customer list by using the action "add new person" on the main menu. On the form, enter the following information:<br>**First Name** Franz<br>**Last Name** Meier<br>**Customer flag** checked<br>Click on "Save and Close" to store the record. Afterward, go back to the main menu and click on the "Search"-command. Enter "Meier" into the main search field and click on "Search now". Verify that your entry "Franz Meier" now shows up on the search results list. |
| **Post-Condition** | Delete the person again using the action within the menu"Delete". |

Table 5.3: Detailed Test Case

**Test Control** Whenever you test an application having an user-interface, you can split the user-interface into it's components. For instance, taking a look at the example-user-interface in figure 5.1.

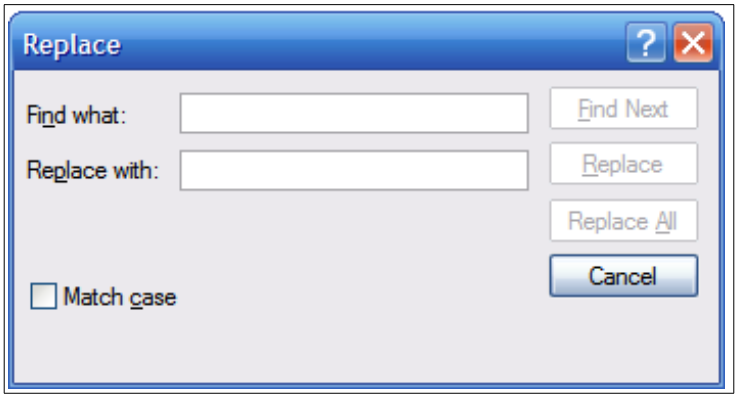


Figure 5.1: Example UI for Test Controls

You can see a variety of test-controls being available:

**txtFind** An input field for keyboard entries.

**txtReplace** An input field for keyboard entries.

**chkMatchCase** A check box. Can be checked or unchecked

**btnFindNext** A button. Can be clicked

**btnReplace** A button. Can be clicked

**btnReplaceAll** A button. Can be clicked

**btnCancel** A button. Can be clicked

All of these controls belong to this simple mask[2].
Now, in order to be able to write completely formalized test-cases, we need to have the application under test, including all masks, formalized. This is an important pre-condition for writing high-quality test-cases. Usually, a dedicated team within the test-department is responsible for providing this data.
Test-Controls are very similar to GUI controls being used on development of rich client applications. Of course, Test-Controls are not limited to rich-client-applications: Everything where you have a UI (also web-applications) consists of a bunch of web-controls.

[2] A mask is another term for windows or dialogs which are to be used while testing

**Test Action** Based on Test-Control, a test-action can be defined. The term "Test-Action" was introduced by Hans Buwalda in [BJPW01] (he named it "Action-Word". A Test-action is a formalised action, acting on a test-control. It has to use one of the provided functionalities (e.g. "click" or "check" or "set text"). All aspects of a test-case, which means pre-condition, test-case and post-condition, can be formulated by using test controls. You can see an example in table 5.4 on the following page.

The formalized test-case is very easy to understand, and it can be executed in manual test by virtually everybody. However, in addition, it can also be used as input for automated testing. This aspect is covered in detail in chapter 5.4.1.3 on page 12.
Note that a test-action is not necessarily executing actions - it might as well extract values from the user-interface (e.g. read value and compare it to expected value).

### 5.4.2 Test Results

Each Test-Case, after execution, must have a test-result. A test-result can be one out of three possibilities, as you can see in table 5.5 on the following page.

**PASS** The Test-case was executed successfully. All pre-conditions, and the test case itself, have been executed, and all actions plus verifications have passed.

**FAIL** While the preconditions passed successfully, during executing of the test actions themselves, the test-case failed. Usually, a FAILED test case results into a reported problem for development to be fixed.

**INCONCLUSIVE** The pre-conditions failed. This means that, due to the conditions, the test case itself could not be executed, therefore no result was produced. "INCONCLUSIVE" could also be named "Blocked".

Please note that "FAILED" does not automatically mean that the application as a problem. There's always a good chance of errors within the test-execution, errors in test automation, or mis-interpretations of desired behavior. Therefore, "FAIL" means something like "For further investigation", and this should be handled within the testing-workflow accordingly.

### 5.4.3 Test Tools

In order to ensure proper test structure, it is mandatory to use a decent set of tools for proper test management. This set of tools includes:

| Action | Content |
|---|---|
| **Pre-Condition** | |
| | **Database->Connect** personRecordTable |
| | **Database->verifyValue** numberAllRecords == 0 |
| **TestCase** | |
| | **btnAddNewPerson->Click** |
| | **waitForNewMaskFinished** |
| | **txtFirstName->setValue** Franz |
| | **txtLasttName->setValue** Meier |
| | **chkCustomerFlag->setChecked** true |
| | **btnSaveAndClose->Click** |
| | **waitForNewMaskFinished** |
| | **btnSearch->Click** |
| | **waitForNewMaskFinished** |
| | **txtMainSearchField->setValue** Meier |
| | **btnSearchNow->Click** |
| | **lstResult->verifyValue** Meier |
| **Post-Condition** | |
| | **btnSearch->Click** |
| | **waitForNewMaskFinished** |
| | **txtMainSearchField->setValue** Meier |
| | **btnSearchNow->Click** |
| | **lstResult->verifyValue** Meier |
| | **lstResult->clickCurrentValue** |
| | **btnDelete->Click** |

Table 5.4: Formalized Test Case

| Test Result | Pre-Condition | Test-Actions |
|---|---|---|
| FAIL | OK | NOT OK |
| INCONCLUSIVE | NOT OK | |
| PASS | OK | OK |

Table 5.5: Summary of Test Result Types

**Test Case Management** A good test case management tool must support the user by storing and managing:

- Test Cases, including the tree structure representing the test case dependencies (see chapter 5.3.1.1 on page 5)
- Test Controls (see chapter 5.4.1.3 on page 14)
- Test Actions (see chapter 5.4.1.3 on page 12)
- Test Results (see chapter 5.4.2 on page 15)

Furthermore, it should support good tracking of changes on the test-cases and other relevant data.

**Problem Tracking** Problem Tracking applications are the main interface between developers and testers. Testers enter problem reports, and developers fix them. In order to ease the work and management of changes, the problem tracking software shall support:

- Versioning support for tracking changes in code
- Direct or indirect connection to code versioning system - in order to enable a good analysis of side-effects and intensity of code-changes.
- Based on the entered and solved problems, forecasts on test-phase-duration and fixing time are possible and should be available.

**Change Request Management** Some reported problems are too complex to be implemented without side-effects or within a given budget. In order to track also those problems, a change-request-management should be implemented. This covers:

- Direct control on the budget of work-intensive fixes (and risk-based management)
- All other advantages of Problem Tracking also apply here.

This was just a brief overview of tools needed for the proper testing workflow. The test-workflow itself is not covered in this document. However, if you are interested to learn it, you can find further information in [Hut03] and [CJ02].

## 5.5 Test Automation

So far, all theory of testing was independent of the way of test execution. Designing test cases, tracking test results, ... - this is all needed both on manual test execution and

automatic test execution, as soon as the project has reached a decent size[3].
In this chapter, we will cover the specific aspects of automation, on top of the existing testing theory. Please note that only the professional aspects are covered here - which are applicable not only on web-applications, but also MFC-based applications, java-based applications or API-tests. The technical aspects regarding web-applications are covered in chapter 6.5 on page 47.

A brief overview of existing testing tools is provided in chapter 5.5.1.4 on page 20 and chapter 5.5.2.4 on page 26.

**Why Test Automation** Generally said, Test automation pays off as soon as you have to fulfill repetitive tasks - for instance if you have to run a regression test once per week.
You could say this is similar to implementing software: One would never develop a piece of software (not even some Excel-Spreadsheet) for a one-time task. However, depending on the duration of manual execution versus effort of implementation for automatisation, it will pay off to develop the automated approach.

The test automation software itself fulfills two very basic tasks:

- Execution of the test case - including pre-conditions and post-conditions
- Providing the result of the test case

As we learned in chapter 5.4.2 on page 15, the result might be “PASS”, “FAIL” or “INCONCLUSIVE”. Even if it failed, the result has to be verified by a human, as failure can be both due to automation or due to the application under test.

**Test Automation Approaches** Roughly said, there are two different approaches of test automation in use on the field:

- Capture’n’Replay (see chapter 5.5.1 on the next page)
- Abstract Automation (see chapter 5.5.2 on page 25)

[3] “decent size” usually means “more than 2 person months of effort”, so in reality most of the projects

### 5.5.1 Capture'n'Replay

#### 5.5.1.1 Description

Capture'n'Replay is the most basic way of test automation, and it is used in all environments where quick automation success is needed. The basic principle is very simple and can be compared to a complex macro recorder:

**Record** The user records all the actions he does, 1:1

**Understand** The test automation recorder tries to map the actions into the most generic version available. For instance, it will try not to work via XY coordinates on the screen for mouse click events, but rather identify what the user wanted to click, based on the DOM-tree (see chapter 6.3.1 on page 33).

**Replay** Once a good macro is recorded, it can be replayed as often as needed.

#### 5.5.1.2 Advantages

Using the Capture'n'Replay Approach leads to fast results for the first few macros - and it is certainly impressive to show. The approach certainly pays off on a small set of test cases which are used for automation - for instance, if a set of 10 test-cases is used for automated regression test, they are a good candidate for capture'n'replay.

#### 5.5.1.3 Disadvantages

The biggest disadvantage is maintainance. All test scripts run without any problems - as long as the application under test does not change.
Now, assuming that it does change (and this happens quite frequently during non-public testing phases), a big load of test scripts might need to be adapted. Remember, each script runs on its own, and no code-sharing is implemented between the automation scripts. So, for instance, if the login mask of the application changes, it might well be that all pre-conditions of a few docent test-scripts have to be recorded again.
There are workarounds available for this (code-sharing, automation code management, ...) - but all of them result into the maintainance of a structure of test-scripts, redundant to the existing structure of test-cases.

| Feature | Lixto | iOpus iMacro | RadView Suite |
|---|---|---|---|
| **Capture'n'Replay** | ✓ | ✓ | ✓ |
| **Scriptable** | ✓ | ✓ | ✓ |
| **Flexible on DOM-Changes** | ✓ | | |
| **External API available** | | ✓ | |
| **Includes Script-Management** | ✓ | | ✓ |
| **Support Images (OCR)** | | ✓ | |
| **Support Flash and Applets** | | ✓ | |

Table 5.6: Comparison of Capture'n'Replay Tools

#### 5.5.1.4 Products

The Capture'n'Replay-Approach is followed by many products nowadays, especially for web-applications. Table 5.6 contains a brief selection of competitors specialized on web-test-automation. It does not include enterprise-test-automation-tools like Mercury WinRunner.

Below you can find details on the products listed.

**Lixto** This product was developed by Lixto Software GmbH (`http://www.lixto.com`), originally indented to be a semantic data extraction engine (as highlighted in [BCL05]). However, it can also be used for test automation. Lixto supports capturing sequences of user input and replaying them. It uses a compiled-in version of the Firefox web-browsing engine.

Figure 5.2 on the next page shows an example for the scripting capatibilities of Lixto. It uses the homepage of Microsoft Expedia. The action-outline to the lower left corner highlights the actions that have been recorded - which means, all actions that can be replayed.
Lixto uses this methodology internally for getting homepages to their desired state, for extracting data. Due to modern web technologies (like AJAX, see chapter 6.3.5 on page 38), just loading a web-page is no longer sufficient. User-actions are needed to get the data needed.
Lixto and its capabilities will be covered in detail in chapter 7 on page 59.

**iOpus iMacro** was developed by iOpus (`http://www.iopus.com`) as a macro-recorder for easing up repetitive tasks, like e.g. verifying stock trades. However, special editions for software test are available, too.

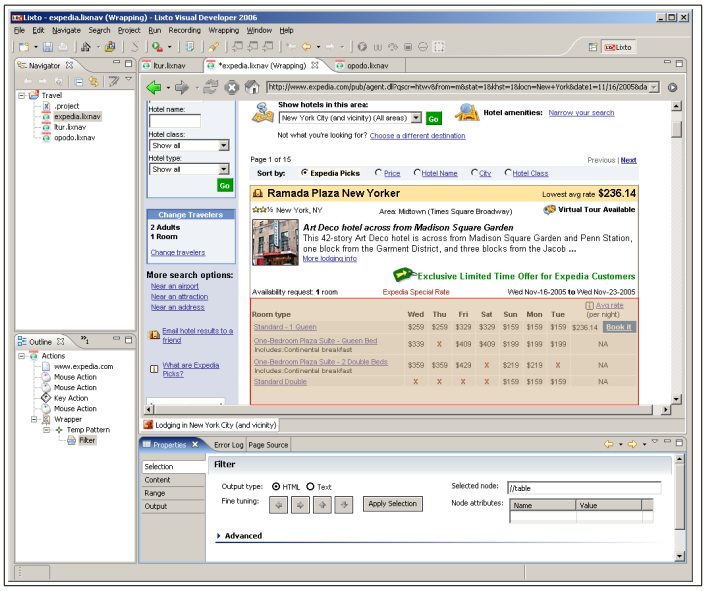

Figure 5.2: Lixto Example using Expedia Homepage

In contrast to Lixto, it does not use a compiled-in browser. Instead, the local installation of Internet-Explorer is used (as embedded ActiveX control), as you can see in figure 5.3.

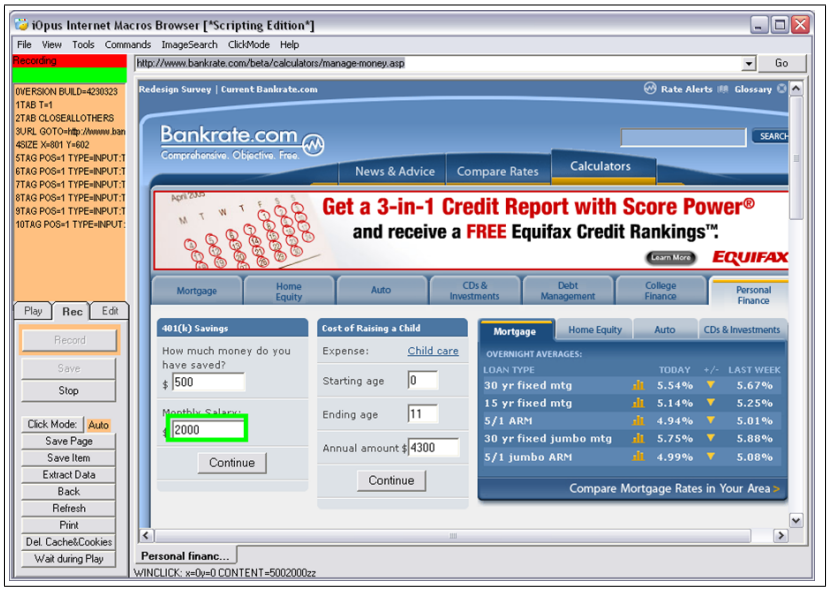

Figure 5.3: iOpus iMacro Example using Bankrate

iMacro, as it focuses on testing, contains some test-specific features in addition to Lixto:

- can be fully automated from an external application via COM (controlled from another PC remotely). This makes it a valuable option for Abstract Automation

(see chapter 5.5.2 on page 25) in addition to Capture'n'Replay.

- Understands Flash-applications and Java-Applets
- can issue verification of images (OCR-Technology)

iOpus iMacro is available for download. For testing purposes, the price is around 500 USD per License. It contains advantages on testing in comparison to Lixto (as Lixto is not designed for software-test-automation currently), but the data extraction engine is not that strong.

The processing engine works via the DOM of the web-page itself (direct adressing). iOpus focused on enhancing the product functionality itself, which includes:

- Providing intensive API-interface for usage via external program - this includes API for .NET, Visual Basic or Visual C++ (Microsoft Technologies).
- It can be run as Windows Service (e.g. for implementing distributed testing scenarios).
- Provide export for the web-page data, for post-processing in your own code (e.g. via CSV).
- Measurement of Web-Page response times

Overall, iOpus iMacro really focused on providing easy-to-use APIs for common functionalities - and to enhance the product to be a good test-automation-tool, too. It comes short in terms of flexible data extraction (compared to Lixto).

**RadView** RadView (`http://www.radview.com`) focused on Web-Testing, but covers not only functional testing, but also load testing and Test Management. The product suite offered by RadView includes:

- WebFT: The Functional Web-Testing Module
- WebLoad: The Load-Testing module (see chapter 6.5.1 on page 47 for technical details).
- WebLoad Analyzer: Based on the WebLoad-logs, this product suite offers detailed analysis and trace-downs of behavior patterns
- TestView: TestView covers the management of test scripts and test data. Features like scheduling for execution are available on this application (for batch-processing and overnight runs), and versioning control for the sources as well. This is the main component of the test process here (as you can see in figure 5.4 on the next page).

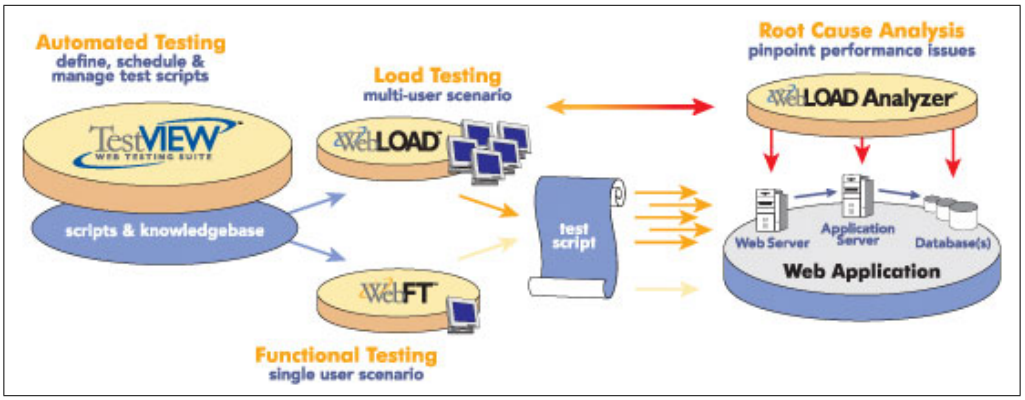


Figure 5.4: RadView Overview of Components

RadView was specifically developed for Web-Testing - and focused on many solutions needed in this area.
RadView WebFT uses an external web-browser - and therefore, supports both FireFox and Internet Explorer. In addition, WebFT on its own supports the usual set of features:

- Complete Capture'n'Replay, using its own scripting language (derived from JavaScript).
- Access of the components directly via the DOM-tree of the web-page

However, WebFT itself does not have any special advantages compared to other Capture and Replay - Tools like iMacro. The main advantage of the RadView-tools is the integration with TestView:

- Changes to the applications (e.g. new form layout) can be managed centrally and are automatically updated to all test scripts. This means that one of the main disadvantages of Capture'n'Replay-Approach is resolved.
- A lot of Test Management Capatibility is delivered "out-of-the-box", like scheduling of execution, notification on test-completion, Test Plan Management, Repository Management for Test - Scripts, ...

Overall, RadView delivers a well-integrated set of components for web test automation, both for load testing and functional testing. Weaknesses of their approach (Capture'n'Replay) are eliminated by their cross-tool-integration (Test-Case-Management). Still, compared to Abstract Automation (see chapter 5.5.2 on page 25), it lacks the possibility to integrate the application professionals on designing test cases, because technical automation and test-case-design are not separated.

#### 5.5.1.5 Example - Lixto

To make the Capture'n'Replay approach better understandable, this chapter will cover a little example approach: We will verify that, when searching for the term "DBAI" in Google[4]), the first match on the result-list will be the Homepage `http://www.dbai.tuwien.ac.at`.
As for Lixto, this means we want to create a macro which:

- Navigates to the webpage `http://www.google.com`
- Enters the Search-Term "DBAI" into the search-field on the search mask (as you can see in figure 5.5)
- Click on the Google-Search-Button
- Verify that the first result links to `http://www.dbai.tuwien.ac.at`


Figure 5.5: Google - Search for Term DBAI

First of all - because Lixto does not support verifications - we have to create a data-model to contain the result we want to match[5]. In our example, the data-set being defined is very simple, as you can see in figure 5.6 on the next page.

Afterward, we have to create the Lixto Wrapper Script which extracts the required information. The details for creation of wrapper scripts, and their meaning, can be read in [Gmb06]. The extracted information has to be compared with our desired result - and our test is completed.
As you can see, testing web-applications using Capture'n'Replay on Lixto is pretty straight-forward and easy to implement.

[4] `http://www.google.com`
[5] More detailed information on this indirect approach can be found in chapter 7.3.1 on page 64

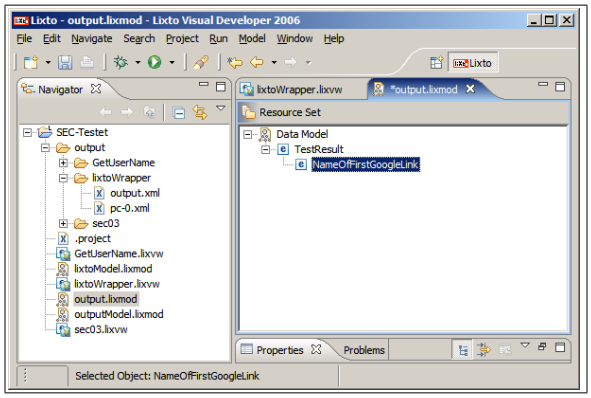


Figure 5.6: Lixto DataModel for Capture'n'Replay

## 5.5.2 Abstract Automation

### 5.5.2.1 Description

Abstract Automation takes advantage, and heavily relies on, a proper test environment being set up - this means, a set of Test Tools (see chapter 5.4.3 on page 15) must be available and in use. Test-Cases must be defined in a formal way, including test actions and test controls (see chapter 5.4.1.3 on page 12).
In order to implement abstract test automation, the abstract concept of test controls (see chapter 5.4.1.3 on page 14) has to be enhanced: For each function of each test-control, abstract code for automation has to be implemented - for instance, code executing the "click"-functionality of a button via the automation software. This abstract work is a once-only task, which has to be done by a dedicated development team within the testing department. It is not even once per application, but once per technology and control, and most often the code consists of no more than 5 to 10 lines.
Based on the meta-code, plus considering the formal notation of test-cases, the script-code for executing test-cases and their pre-conditions can be generated automatically, Note that immediately all test cases can be automated - as long as they are written in a formal way. Therefore, even if the professional department writes the test-cases instead of the technical testing department, all of the test cases are immediately available for automation.
Note that, in order to declare the declare the testcases, the application under test has to be defined for input - similar to the definition of a database-table in software-engineering before the actual data can be written. The declaration often takes some time, and can be done much faster if the test automation software supports the concept of GUI-maps. A GUI-map represents the application under test and it's masks

(e.g. web-page-layouts) in a technical way. Once this is importable, defining the application masks for formal test case declaration is not much work any longer.
The test-robot doing the "real" Test Execution is known in this concept as the "Virtual Test Engine", as explained in [Har00].

#### 5.5.2.2 Advantages

Abstract Automation pays off as soon as change is needed. For instance, if the login mask changes, this is at most a one-time-change to the defined mask within the abstract test case definitions - nothing more. Immediately, all scripts should be executable without any big problems.
Due to the test process in use in general, test automation can advance without using valuable technical resources on rare skills. Once the basic framework is established, even the professional department (not knowing anything about automation scripts in general) can create test cases ready for automation, they just have to stick to the agreed formal rules. This is a very powerful rule - based on a little extra-effort at the beginning, the series of automated test cases can grow and grow, without having to double any work.

#### 5.5.2.3 Disadvantages

The biggest disadvantage of the approach is the high effort needed before the first automation takes place. For companies being not so familiar to testing, the initial effort might be overwhelming: They need not only skills on test automation scripts themselves, but also qualified test management, tools for managing test cases, stick to formal test case notations and much more. This is too much for starting testing in general, the approach is mainly useful for companies who already have some testing effort and want to make testing more efficient.
Secondly, a set of skills on the test automation product and its script-language is needed within the test department.

#### 5.5.2.4 Products

Abstract Automation is not followed widely on the market, since the idea is still pretty new. Therefore, companies have been forced to develop their own internal tools, like e.g. Generali Vienna. However, there are also products on the market available for

licensing supporting this approach, like e.g. QAS.TCS. Each product consists of two components: Test-case-management, and test-case-automation. The common interface is the exported list formal test cases, and an importable list of formal test results.

**Generali ABCD** was developed for the software test of company-internal products of Generali Vienna and CCE. As you can see in `http://www.objectarchitects.de/tu2002/`, the application was in use since more than 5 years.

Test case creation and management takes place within the ABCD-application, as you

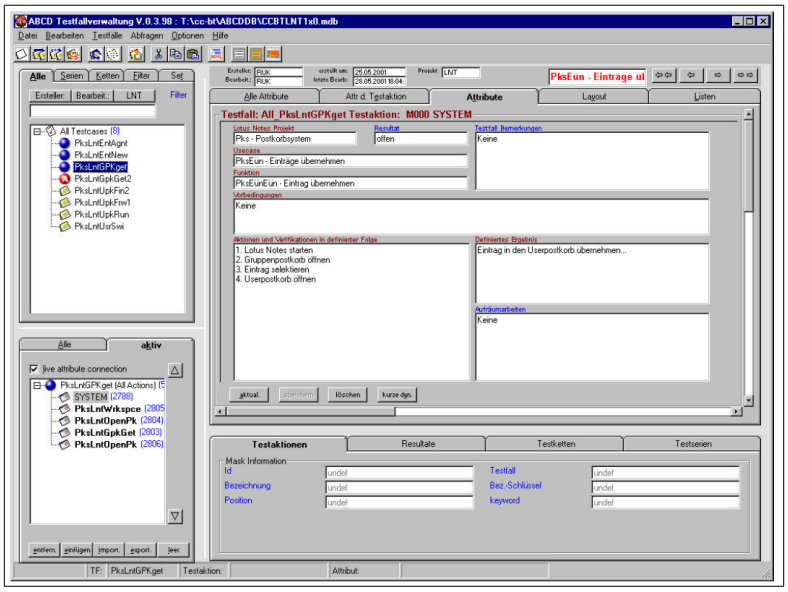


Figure 5.7: Generali ABCD Application Screenshot

can see it in figure 5.7. The test case definition is done based on defined test actions, test controls and test masks. Based on these definitions, the test-cases can be exported in a formal way - in ABCD, the exported format is called “tcxml”. This exported file can be either rendered to a manual tester for execution, or passed on to a library for test automation.

This is where the second part of the program becomes important - the automation script Library. For test automation, Mercury WinRunner is used. The self-implemented software-module converting txXML to WinRunner-code is called WIS (WinRunner Interpreter Suite). [Lut03]. WIS, as implemented by Generali, is capable of supporting several test platforms, including Web, Windows-Rich-Client or SAP - also in combination.

**Objentis QAS.TCS** is one of the first products on the market supporting abstract test automation. You can define all test objects, including test control, directly within QAS.TCS (see figure 5.8).

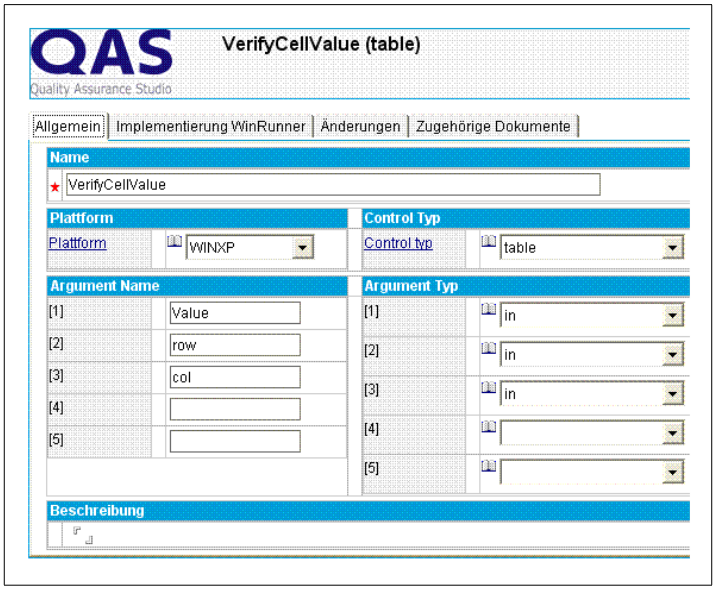


Figure 5.8: Objentis QAS.TCS Screenshot - Test-Control

QAS.TCS itself fully supports manual testing - which means, presenting the abstract test-cases in a readable way, and let the user enter test-results back to the database. In addition, tight integration to problem-tracking (QAS.PTAR) is supported. In terms of automation, QAS.TCS supports export of test-cases, using an XML-format. And it supports import of test-results, again based on XML.
The automation itself is available within the add-on product QAS.TCS/AE (AE stands for "Automation Edition") - which is using Mercury WinRunner for Test Automation. Note that TCS/AE is not necessarily limited to Mercury WinRunner - this is just the pre-delivered set of code. The customer can decide to use his own automation tool, like e.g. iOpus iMacro, to fulfill the automation. Mercury WinRunner is supported as default, because most companies already own the licenses, and it supports a series of testing platforms.

#### 5.5.2.5 Example - Lixto

An example implementation for Abstract Test Automation, using Lixto for the automation part, is covered in chapter 7 on page 59.
The example also includes some use-cases and a sample-application.

# Chapter 6

# Testing Web Applications

## 6.1 Introduction

This chapter will cover all the technical aspects of testing web-applications. This will cover not only some basic guidelines on web-technology itself, but also a brief set of examples on how to access it automatically, and how test-software can take use of this. This chapter will not cover low-level description of the TCP- or the IP-Protocol, and knowledge of basic networking terms is required. If you need documentation and technical details to fresh up your knowledge regarding basic networking, I recommend to take a look at the book *TCP/IP Network Administration* [Hun02].

## 6.2 Web-Browsing - What's inside

Browsing web-pages involves a lot of technologies to work transparently in the background. While the end-user just types in an URL and waits for the page to show up, the computer in fact has a lot to do. Each of these aspects is re-explained within the sub-chapters: All information in this chapter is extracted from *TCP/IP Network Administration* [Hun02].

1. Resolve the URL typed in on the DNS to match an IP-Address
2. Connect to the web-server on this IP-address. Depending on your protocol, this might be port 80 (http) or port 443 (https)
3. Issue a HTTP GET-Request on this server
4. Receive the HTTP-response

5. Extract the HTML of this HTTP-Response
6. Web-Browser creates a layout to be displayed, according to HTML-content
7. Display the web-site to the user

In this example, I will assume the user typed in the URL `http://www.dbai.tuwien.ac.at/education`.

### 6.2.1 Resolve URL in DNS

Each PC has some network-configurations applied, one of them being the address of the DNS-Server ("DNS" is the abbreviation for "Domain Name System")
As the user enters a Web-Address (e.g. http://www.dbai.tuwien.ac.at), the primary action will be to resolve "www.dbai.tuwien.ac.at" to it's IP-Address. To do this, the PC establishes a Connection to it's local DNS-Server and queries the Address

```
Default Server:  vnsc-pri.sys.gtei.net
Address:  4.2.2.1

> set type=a
> www.dbai.tuwien.ac.at
Server:  vnsc-pri.sys.gtei.net
Address:  4.2.2.1

Non-authoritative answer:
Name:    www.dbai.tuwien.ac.at
Address:  128.131.111.3
```

By issuing this simple query, the address is resolved.
Of course, the DNS-System itself covers much more aspects, which are not needed for this thesis. For instance, the DNS is responsible for directing mail clients to proper servers for mail-delivery, and it also plays an important role in fighting SPAM. Further details on the Domain Name System can be found in RFC 1034 [Moc87a] and RFC 1035 [Moc87b].

### 6.2.2 Connect to Web-Server

Once the IP-Address of the destination server is resolved, the client has to connect to the web-server in order request the data.
Note that, on TCP/IP IPv4 networks, multiple ports (up to 65.536) are available for the

server to listen on, and the client has to know the right one. As long as not specified different by the end-user, the client will choose the default port - which is Port 80 for http [FGM+99], and Port 443 for https [Res00].

### 6.2.3 Issue HTTP GET Request

Once the connection is established, the client issues a so-called "HTTP GET Request" to the server. It sends the following commands to the server (plain text):

```
1 GET /education/ HTTP/1.0
2 Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/
      vnd.ms-excel, application/vnd.ms-powerpoint, application/msword,
      application/x-shockwave-flash, */*
3 Accept-Language: de-at
4 User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET
      CLR 1.1.4322)
5 Host: www.dbai.tuwien.ac.at
6 Connection: Keep-Alive
```

In addition, the client might send a big load of additional information as additional headers, telling the server about it's capatibilities. This include the identification of the web-browser (also called "user-agent"), viewer capatibilities (image formats) and user-defined preferred language.

### 6.2.4 Receive HTTP Response

The server, on it's side again, has to answer to the client request:

```
1 HTTP/1.1 200 OK
2 Date: Mon, 27 Mar 2006 00:08:45 GMT
3 Server: Apache/1.3.28 (Unix) tomcat/1.0 PHP/4.3.3
4 Last-Modified: Wed, 08 Mar 2006 16:18:01 GMT
5 ETag: "17581-2173-440f03b9"
6 Accept-Ranges: bytes
7 Content-Length: 8563
8 Keep-Alive: timeout=15, max=99
9 Connection: Keep-Alive
10 Content-Type: text/html
11
12 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
13 <HTML>
```

```
<HEAD>
<TITLE>DBAI: Lehrveranstaltungen</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000" LINK="#3333FF" VLINK="#336633">
[.....]
```

The response of the server contains first of all the so-called HTTP Status Code. "200" means "OK, here you get the data". Other famous status codes are 404 (page not found) and 500 (server processing error).
After some statistical information, which is most relevant for caching- and connection-keep-alive-methodologies, the content of the response itself is sent. In our example, this is plain HTML-code (indicated by the Content-Type-information).
Note that, when you load a web-page as an end-user, typically you don't load only HTML-page. In addition, all the images have to be fetched. This is handled in separate HTTP-request, so in fact, when loading a single web-page, multiple requests are being done.

### 6.2.5 Extract HTML-Code

Everything starting after the first empty line is considered as the content itself - in our example, the HTML-content. This information is passed on to the web-browser.

### 6.2.6 Create Layout based on HTML

HTML (HyperText Markup Language) itself is a straight-forward layout language for web-pages [RHJ99].
Based on the HTML-code, the web-browser can issue further HTTP-requests (e.g. for images) and start rendering the page for display on the screen. The output can be seen in figure 6.1 on the following page.

### 6.2.7 Display Web-Site to end-user

Finally, when everything is completed, the web-site is displayed to the user. By clicking on one of the hyperlinks, the same process (using a different page, of course) will start again.


Figure 6.1: Example Display of Web-Page

## 6.3 Technologies in use - Client-Side

In the beginnings of the internet, the only task for the web-browser was to fetch web-pages and images, and display them appropriately.
Nowadays, however, the client fulfills much more complex tasks - which are sometimes as complex as rich-client-applications. This chapter will explain some of the technologies in use, and how they work. Knowing and supporting these technologies is very important for any test-automation-software to work properly.

### 6.3.1 HTML - HyperText Markup Language

HTML [RHJ99] is a markup language, used for setting the layout to be displayed to the user. In HTML itself are no dynamic aspects for the client, it's nothing more than formatting and content.
The markup elements can be grouped into the following variants:

**Structural Markup** Structural Markup is used for describing the structure of the document - like for instance headlines:

```
1    <h2>Primary Headline</h2>
```

**Presentational Markup** Presentational Markup is used for setting the layout for specific text elements. It does not give any machine-extractable information.

```
1    <b>This Text should be bold</b>
```

Presentational Markup is not used on modern websites - instead, layout using CSS is applied.

**Hypertext Markup** Hypertext Markup is used for setting references to other HTML-pages. They are the fundamental part of web-pages.

```
1    <a href=\enquote{http://www.dbai.tuwien.ac.at}>DBAI Homepage</a>
```

HTML-Pages can usually be parsed not only as a text-file, but also in a tree-style structure, based on the markup.
An example web-page can be seen in figure 6.1 on the previous page. The DOM-tree belonging to this web-page can be seen in figure 6.2.

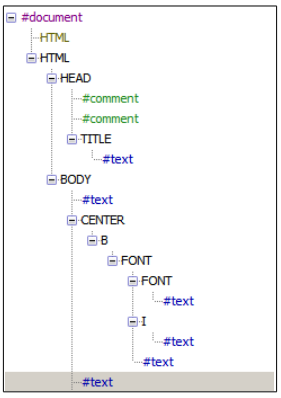


Figure 6.2: Example Display of Web-Page - DOM

Each component on the web-page (no matter whether it is a label, an input box or a button) can be identified uniquely via it's X-Path. This is the most common used way for test-automation to identify components. However, when a web-page changes, it is not really flexible anymore (e.g. on dynamic advertisements).

### 6.3.2 JavaScript

JavaScript, introduced in 1995, was the first possibility to actually execute code on the client-side. This code is to be executed by the web-browser.
Most common usage scenarios for this language include:

- Opening other web-pages in a new window
- Input Validation of Forms
- Visual Effects, e.g. mouse-over changes on images

Theoretically, JavaScript does not need to be considered by test automation. Every web-page must be working correct even if JavaScript is disabled, because the user might also disable JavaScript. Therefore, even if JavaScript is used for validation of

form input, the server must do the same validation, too.
Practically, however, many Web-Pages (especially on the intranet) rely on proper execution of JavaScript. Therefore, test automation software must support JavaScript and handle it appropriately.

Listing 6.1: Example JavaScript Form Validation

```
<html>
<head>
<title>Verify Input</title>
<script type="text/javascript">
function chkFormular () {
  if (document.Formular.User.value == "") {
    alert("Please enter your name");
    document.Formular.User.focus();
    return false;
  }
}
</script>
</head>
<body>

<h1>Form</h1>

<form name="Form" action="formInput.pl"
  method="post" onsubmit="return chkFormular()">
<pre>
Name:     <input type="text" size="40" name="User">
Formular: <input type="submit" value="Send">
</pre>
</form>

</body>
</html>
```

In this example form, the user can enter any input for the field “user”. However, already on the client-side, verification occurs that the content of the field is not empty.

### 6.3.3 Microsoft VB-Script

Microsoft VB-Script is the main scripting language for Windows-Installations. It is very similar to JavaScript.
Because it does not have any real advantages in favor to JavaScript. In addition, only

Microsoft Internet Explorer in Windows-Platforms supports executing those scripts. While VB-Script never came to popularity on the internet, it is in use on some intranet sites.

### 6.3.4 Dynamic HTML

Dynamic HTML is not a technology on it's own. Rather than that, it is a set of JavaScript functions and libraries which can be used on Websites, to enhance the user experience. The following example page is taken from [Wik06]:

Listing 6.2: Example Dynamic HTML Code

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>

<head>
<title>Test</title>
<style type="text/css">
   h2 {background-color: lightblue; width: 100%}
   a {font-size: larger; background-color: goldenrod}
   a:hover {background-color: gold}
   #example1 {display: none; margin: 3%; padding: 4%; background-color:
       limegreen}
  </style>
<script type="text/javascript">
  <!--
   function changeDisplayState (id) {
    e=document.getElementById(id);
    if (e.style.display == 'none' || e.style.display =="") {
     e.style.display = 'block';
     showhide.innerHTML = 'Hide example';
    } else {
     e.style.display = 'none';
     showhide.innerHTML = 'Show example';
    }
   }
  //-->
  </script>
</head>

<body>

<h2>How to use a DOM function</h2>
```

```
<p><a id="showhide" href="javascript:changeDisplayState('example1')">Show
    example</a></p>
<div id="example1">
  This is the example. (Additional information, which is only displayed on
      request)
  ..............</div>
<p>The general text continues ....</p>
<p></p>

</body>

</html>
```

In this example, by default you will see the webpage as you can see in figure 6.3.

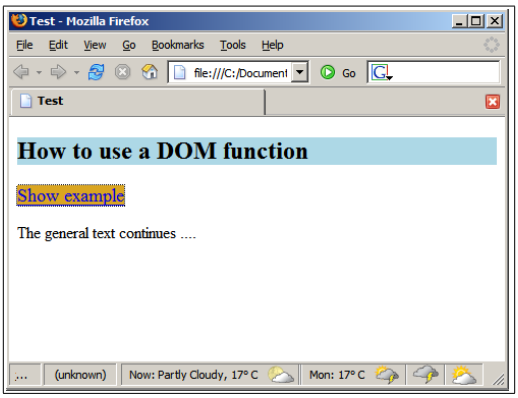


Figure 6.3: Display of example DHTML-Page before clicking

However, when the user clicks on the link, the appearance will change, as you can see in figure 6.4. Note that no server-communication occurred here, this happens completely within the browser.

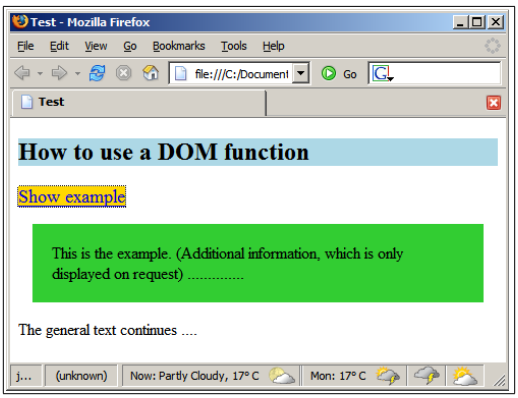


Figure 6.4: Display of example DHTML-Page after clicking

This is a very important aspect for automating web-tools. Some functionalities needed for automatisation, and some content of the webpage, might become available only after another action was taken[1]. This means, the automation software must be capable of understanding changing web-pages, and it must be able to emulate those changes if not provided by an external application. However, it is once more important to note that no server-communication takes place.

### 6.3.5 Ajax

Ajax, abbreviation for "Asynchronous JavaScript and XML", introduced a new concept to web-applications: The background-fetching of data on web-pages. This means, starting with Ajax, the web-page can dynamically update itself on the client.
Before this technology was introduced, all content of web-pages was served completely within the web-page. This means, whenever the user wanted to see new data (e.g. search-results for his searching-term), the request was sent to the server. And the server provided the answer by providing a complete new HTML-page containing the result data (and also all layout). Technically from the browser point of view, there was no connecting point between the search-form and the answer-page, except that one followed after the other. All application logic is covered by the server. See figure 6.5 for an overview.

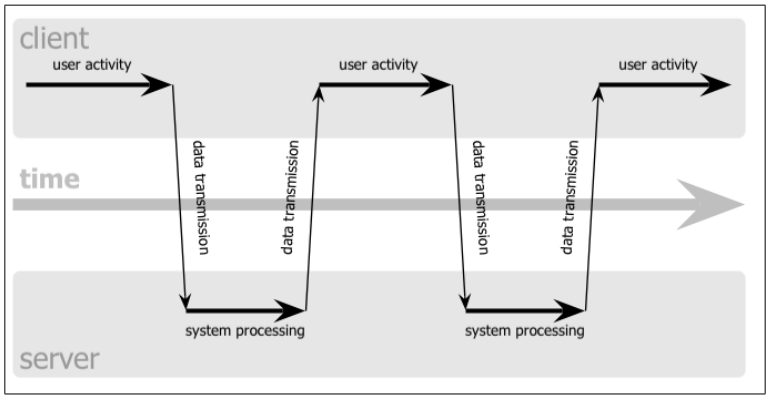


Figure 6.5: Classic Web Application Model

Starting with Ajax, the application-logic moves to the web-browser itself - acting as a rich-client. Figure 6.6 on the following page covers the concept. The browser UI receives a web-page once - containing not only HTML, but also a load of JavaScript-code (which contains the AJAX-Functionality). Now, imagine that the user enters a

[1]The same challenge applies also for web engine search robots

search-term on some form and starts the query. Instead of requesting a new answer-page, the so-called Ajax-Engine processes the logic of the form, and sends a technical data request to the server (using XML-Format). The server itself answers this technical request - by providing the data only (again in XML-Format). But the server does not provide any layout information. All information regarding layout exists already at the client-side, within the Ajax-Engine.
The Ajax-Engine, when receiving the data, processes it and renders it appropriately to the user - who can see the search-result, just as if they were provided by the web-server, only faster.

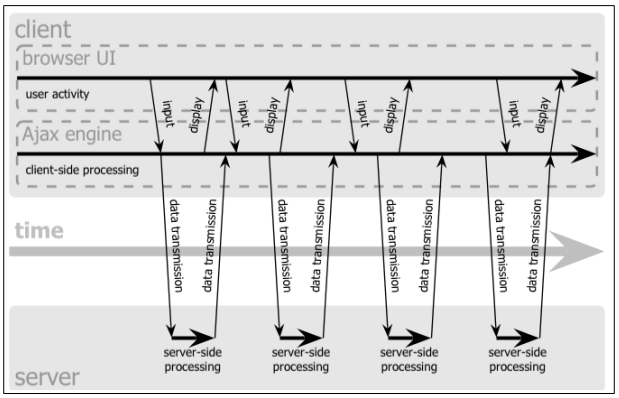


Figure 6.6: AJAX Web Application Model

Technically, AJAX is not really a new technology. It is a summary term for combined usage of existing technologies, like HTML, JavaScript and XML-Objects available in JavaScript. There's no real standard behind it, and the behaviour of course depends heavily on the application logic within the Ajax-Layer itself.
From the automation point of view, Ajax covers big challenges. Just as for JavaScript-functions in general, the whole functionality of the Ajax-Layer must either be emulated, or it must be available by using a web-browser for automation itself. Ajax and similar technologies are becoming more and more popular these days, therefore supporting the client-side application layer is a must for all automation tools.

Listing 6.3: Example Ajax Code

```
<script type="text/javascript" language="javascript">
    function makeRequest(url) {
        var http_request = false;
        if (window.XMLHttpRequest) { // Mozilla, Safari,...
            http_request = new XMLHttpRequest();
            if (http_request.overrideMimeType) {
                http_request.overrideMimeType('text/xml');
```

```
8                 // See note below about this line
9             }
10        } else if (window.ActiveXObject) { // IE
11            try {
12                http_request = new ActiveXObject("Msxml2.XMLHTTP");
13            } catch (e) {
14                try {
15                    http_request = new ActiveXObject("Microsoft.XMLHTTP");
16                } catch (e) {}
17            }
18        }
19        if (!http_request) {
20            alert('Giving up :( Cannot create an XMLHTTP instance');
21            return false;
22        }
23        http_request.onreadystatechange = function() { alertContents(
              http_request); };
24        http_request.open('GET', url, true);
25        http_request.send(null);
26    }
27
28    function alertContents(http_request) {
29        if (http_request.readyState == 4) {
30            if (http_request.status == 200) {
31                alert(http_request.responseText);
32            } else {
33                alert('There was a problem with the request.');
34            }
35        }
36    }
37 </script>
38 <span style="cursor: pointer; text-decoration: underline" onclick="
      makeRequest('test.html')">
39        Make a request
40 </span>
```

In this example, the user initially just sees the text "Make A Request". When he clicks on the link, the client issues a GET-request for the file "test.html" and display its content within a dialog box. New content is loaded from the server to the client.

### 6.3.6 Comet

Similar to Ajax, Comet is not really describing a new technology - it is a new concept introduced.
Ajax was assuming that all action is triggered by the client - this means, client requesting search results, or client entering new information. On Comet, however, an event-driven approach is chosen.

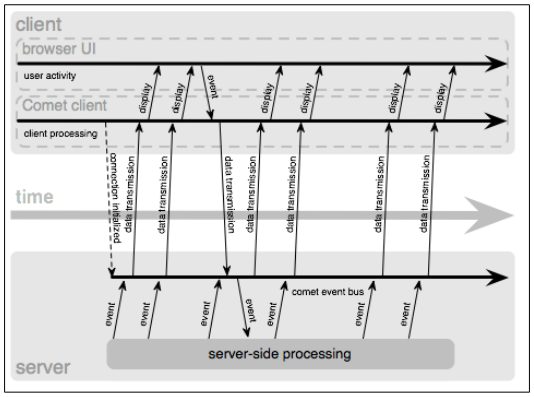


Figure 6.7: Comet Web Application Model

As you can see in figure 6.7, Comet is completely event-driven by the server. The client establishes once a http-connection to the server, and it keeps this connection alive. There's no longer the need for the client to poll for new information - the server can simply push it. The most famous example for this technology is the integration of Instant-Messaging into Web-Applications. Other examples include monitoring systems and multi-user-applications in general.

### 6.3.7 Java Applets

Java Applets have been introduced 1995, to enable rich web applications.
Basically, a Java-Applet is an embedded application within a web-page. This application itself is implemented in Java and is executed by a virtual machine within the web-browser. From the web-browser point of view, the applet is a completely self-contained application, which gets some space provided on the screen (a rectangular) and acts on its own.
Within the HTML-code itself you can only find a reference to the applet binary to be downloaded:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>This page contains a Java Applet</title>
</head>
<body>

<h1>Try it out</h1>

<p>
  <object classid="java:MyJavaApplet.class" codetype="application/java-vm"
          width="600" height="400"></object>
</p>
</body>
</html>
```

From the browsers point of view, it just has to render a headline, reserve an area of 600x400 pixels, and invoke the Java Virtual Machine to populate this area. The browser does not know what will be displayed within this area. It might be a form, a game, or whatever.
Normally, Java Applets are allowed only to interact with the web-page itself, the user and the server where the web-page came from. However, especially in intranet-applications, further access is granted to the applets, like local database-connections for instance.
Embedded Applications like Java Applets are a high challenge for all test automations. Even the browser itself does not have any concept of the Java-Applet and its content, therefore it's even harder for external application (test automation software) to get an understanding of the content.

### 6.3.8 Microsoft ActiveX

Similar to Java Applets, Microsoft developed a counterpart technology for these functionalities. It is called "ActiveX" (also known as "COM") and provides similar functionality as Java Applets. They are executable within Microsoft Internet Explorer only and are used mainly on intranet websites.
Same challenges as for Java Applets apply here.

### 6.3.9 MacroMedia Flash

MacroMedia Flash, the third of the embedded application types beside of Java Applets and Microsoft ActiveX, is the most popular variant of embedded applications on the Internet. As for the browser, the same rule applies as for ActiveX or Applets: The complete control on a given area is handed over to the embedded application itself (FlashPlayer).

## 6.4 Technologies in use - Server-Side

Web-Servers are fulfilling a lot of tasks - and as the web evolves, the complexity of the tasks grows, too.
This chapter will explain the server-technologies being important to know about in regard to test-automation. Section by section, different technologies will be revealed. Knowing the server-side is not a real need for testing web applications. However, it is important to understand the different concepts being available.

### 6.4.1 Serving Static Content

The most basic functionality of any Web-Server is to serve static content - which consists of text-content (html-files, JavaScript-Files) and binary content (e.g. images).
The content is available on the web-server as static files, and there's nothing special about serving them. Of course, the server has to be capable of the HTTP-protocol [FGM+99] to server the content.

### 6.4.2 Dynamic Content

Things are getting more interesting when the content is dynamic - for instance search-results of a web search engine. The main concept behind dynamic content:

1. The HTTP-request is forwarded to an application, which is executed within the context of the web-server. This application can be implemented both in script-languages (PHP) or full object-oriented languages (Java).
2. The program itself evaluates the passed content, and returns information back to the client, via the http-response. The response can be content of any type, including binary content and text content.

3. The browser requesting the page cannot distinguish whether it is getting dynamic or static content, the process is transparent to the browser.

All Application Logic of any web-application is covered on the server-side in this scenario.
Nowadays, most web-pages are dynamic.

### 6.4.3 Serving XML-Requests

In addition to HTTP-requests for content, more and more web-servers and their applications are working based on XML-Data-Requests. This means, the whole application logic is sent once (in a static context) to the browser, and the application, being executed in the context of the browser, is only requesting specific data from the server. This concept is also known as Ajax, explained in chapter 6.3.5 on page 38.

### 6.4.4 Client Tracking - HTTP Cookie

A Web-Server is serving several clients at the same time - could be up to thousands. As long as the server is handling anonymous data only (e.g. news-sites), this is not a problem at all.
However, many web-pages require user-specific behaviour. This includes exclusive access rights, maybe editing possibilities, webmail access and similar topics. Still, the web-server must distinguish all the different user connections, in order not to provide the mail of person A to person B, connected at the same time.
HTTP Cookies [KM00] are used for user tracking. The main principle is very easy:

1. The Web-Server sends cookie-information on the first page being requested (the first page is served to an anonymous user)
2. The browser has to send this cookie together with each request to the server, to identify itself.

A HTTP-cookie normally contains no specific information, it is just a random, unique string of characters, just enough to identify the user for the web-server.

Listing 6.4: Example HTTP Cookie Content

```
1 PREF
2 ID=fbd6ab51e763e007:TM=1143521536:LM=1158643389:S=DeECkzlp4Lqt-ncs
3 google.at/
4 1456
```

```
2611289336
38641634
375942112
```

Now, even if the user has to log in with credentials, the web-server can still associate the credentials to the cookie internally, and therefore know which user is requesting which page. The user-specific content itself is handled within dynamic content web-pages, and most programming languages for dynamic content support appropriate user-handling.

Client-Tracking is very important to be aware of when automating test of web-applications. Just as if you test rich-client-applications, each test case must act within it's context. For instance, if you test a web-application for sending mails via a web-mail-interface, your automated test-case must be preceded by another test case doing the login. Therefore, all test automation software must support dependencies between test-cases, or applying an order of execution. In addition, before executing any tests, it is important to verify that the test environment is clear - which means, if browser-automation is used, all cookies must be cleared before executing the test case.

### 6.4.5 Client Tracking - Basic Authentication

HTTP Basic Authentication [FHBH+99] is the original specified way of user-tracking. Since HTTP itself is a stateless protocol, the web-server cannot keep context of the users. While this is solved by using HTTP Cookies, the HTTP-protocol itself also provides a way to handle this.
The main idea is that certain areas of a web-server (e.g. a directory) can be marked as "private" and only accessible using a certain set of credentials. Whenever a user wants to access this area, the browser will try an anonymous request first:

Listing 6.5: HTTP-Request (anonymous)

```
GET /confidentialArea/ HTTP/1.0
Host: mySecretServer
```

The server, knowing that the file should not be available to the public, answers to the browser that credentials are needed:

Listing 6.6: HTTP 401 Authentication Request

```
HTTP/1.0 401 Unauthorised
Server: MyWebServer/1.0
```

```
Date: Mon, 26 Mar 2006 12:13:37 GMT
WWW-Authenticate: Basic realm="My Personal Realm"
Content-Type: text/html
Content-Length: 311

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">
<HTML>
  <HEAD>
    <TITLE>Error</TITLE>
    <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO
        -8859-1">
  </HEAD>
  <BODY><H1>401 Unauthorised.</H1></BODY>
</HTML>
```

The browser understands this answer (HTTP Code 401). Therefore, the user is prompted for entering his credentials within an own dialog box, as you can see in figure 6.8. The user enters his credentials, and using the input of the user, the client

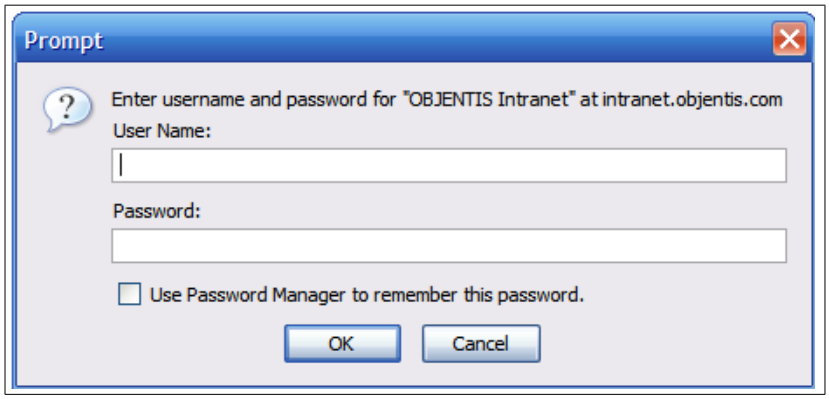


Figure 6.8: HTTP Authentication Dialog

tries again to request the same page - this time providing credentials:

Listing 6.7: HTTP-Request (with credentials)

```
GET /confidentialArea/ HTTP/1.0
Host: mySecretServer
Authorization: MyAuthentication QWfjkdlsjlfjeVF65fd4HQ==
```

The same authorization is now included with every request on this directory and all subdirectories. This method of authentication is stateless - which means that, technically, it is possible to enter the web-application everywhere using this authentication.

However, it can be expected that some entry points (e.g. main page) are expected by the web-application. Therefore, the same pre-conditions for testing apply as in chapter 6.4.4 on page 44.

## 6.5 Test Automation Methodologies

In this chapter, approaches on test automation for web-applications are covered. The chapter covers the technicals aspects of test-automation, not any environmental or professional aspects on testing. The professional aspects are covered in chapter 5.5 on page 17. There are three approaches for automated test of web-applications:

- Direct emulation of HTTP-requests (see chapter 6.5.1)
- Automation indirect via the web-browser (see chapter 6.5.2 on page 51)
- Automation using an embedded web-browser (see chapter 6.5.3 on page 57)

All of these test-approaches are different in their way of accessing or emulating the web-application behaviour. However, all of them have in common that you cannot have one global solution for testing web-applications. Each web-application is implemented and behaving differently, and the automation implementation must be done separated for each web-application. This can be compared to making automated test of windows-applications: You cannot automate notepad and calculator using the same test-script. They are both windows-applications, but that's their only common point, everything else is completely different, like layout or functionality. In the same way, two web-applications (e.g. google search and google mail) are totally different applications.
The main aim of the automation is to behave as close as possible to real-user-behaviour. Of course, automation software must rely on internal APIs to work correctly. For instance, no test automation can move the mouse physically - it can only trigger OS-internal APIs. All automation approaches make some assumptions on proper replacements.

### 6.5.1 Direct emulation of HTTP-Requests

A scenario being closest to Whitebox-test (see chapter 5.3.3.3 on page 9) is the direct HTTP-Test.
Instead of using any html rendering engine, the http-request-approach directly uses

HTTP-requests (chapter 6.2.3 on page 31) [FGM+99] and parses information out of the response.
Most known HTTP-request-testers are Mercury LoadRunner or WebInject[2].

#### 6.5.1.1 Technical Example

As an example, the query-scenario on Google will be emulated using HTTP-Tests. Now, the normal functional test would cover the following steps:

1. Fetch the homepage at `http://www.google.com`.
2. populate the search-term field, and press the "submit"-button
3. wait for the answer-page to appear in the browser, extract information from html-page

On HTTP-request-test, which is used for load-testing scenarios mainly, the functional scenario ("enter information into field") is only emulated and therefore cannot be tested anyway. Still, the first two steps must be executed, in order to get a appropriate session cookie (see chapter 6.4.4 on page 44). The test procedures starts by the basic HTTP-Request.

Listing 6.8: HTTP-Get-Request for Google Homepage

```
1 GET / HTTP/1.0
2 Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/
      vnd.ms-excel, application/vnd.ms-powerpoint, application/msword,
      application/x-shockwave-flash, */*
3 Accept-Language: de-at
4 User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET
      CLR 1.1.4322)
5 Host: www.google.com
6 Connection: Keep-Alive
```

The server answers by providing the HTTP-response, containing also the HTML-code of the homepage:

Listing 6.9: HTTP Response for http://www.google.com

```
1 HTTP/1.0 200 OK
2 Cache-Control: private
3 Content-Type: text/html
```

[2] `http://www.webinject.org`

```
4 Set-Cookie: PREF=ID=642d9e1e2bf7691c:TM=1143612348:LM=1143612348:S=r92IQ4-
      ZTcy5Bi5D; expires=Sun, 17-Jan-2038 19:14:07 GMT; path=/; domain=.
      google.com
5 Server: GWS/2.1
6 Date: Wed, 29 Mar 2006 06:05:48 GMT
7 Connection: Close
8
9 <html><head><meta http-equiv="content-type" content="text/html; charset=UTF
      -8"><title>Google</title><style><!--
10 body,td,a,p,.h{font-family:arial,sans-serif;}
11 .h{font-size: 20px;}
12 [......]
```

Note that the response contains a lot of valuable information for HTTP-based test:

**Caching-Information** Important for server-load-tests

**Cookie-Information** It is very important to track the cookie being assigned to the client. Based on the "setcookie" information, the HTTP Load Tester must send the cookie on all further requests, just like the web-browser would do.

**Connection Close** Indicator that the HTTP-connection is closed now. This is an important figure for sizing (under which circumstances are connections closed by the web-server, and does this fit to the expected behaviour).

**HTML-Code** The HTML-Code of the web-page itself, for verification of correct output

Based on the output, the http-tester can now issue the next GET-request, for executing the query itself. Issuing the search-query implicitly covers the UI-functionalities of entering the search-term and pressing the search-button.

Listing 6.10: HTTP-Get-Request for Google Search

```
1 GET /search?hl=de&q=DBAI+TU-Wien&lr= HTTP/1.0
2 Accept: */*
3 Referer: http://www.google.com
4 Accept-Language: de-at
5 User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET
      CLR 1.1.4322)
6 Host: www.google.com
7 Connection: Keep-Alive
8 Cookie: PREF=ID=642d9e1e2bf7691c:TM=1143612348:LM=1143612348:S=r92IQ4-ZTcy5
      Bi5D
```

Note that the cookie is sent on this request now.
Again, the http-response can be received and evaluated - not only for performance

measures (caching, response time, ...), but also for the content itself. As the HTTP-tester receives exactly the same information as the browser does, it can also verify in the same way if a specific information is contained in the output (in this example: A specific test result). This type of parsing is often needed to extract a piece of information from a response, to use it within one of the next requests again.

#### 6.5.1.2 Advantages

Direct HTTP-Request testing is a big advantage when you want to perform stress tests (see chapter 5.3.4 on page 9). Instead of acting as a single user, you can easily enwiden your test scenario to act as hundreds or thousands of different users at the same time, only limited by test machine resources.
Because you are acting directly on the API to the server (the HTTP-requests), it is possible all types of needed information, like number of transferred bytes, average response time, peak times, caching-behaviour (number of cache-hits vs. miss), and several other aspects. It is a powerful tool to stress-test you application and collect a bunch of statistics to optimize your behaviour. Most often, this type of testing is used before deployment of an application, to get a qualified estimation on the expected performance under real load.

#### 6.5.1.3 Disadvantages

Using the whitebox-test demands a high level of technical skills in web technologies. In addition you will need a good and detailed documentation of the communication between client and server in order to work out test cases on this level. The main problem here is that development becomes more and more abstract (most developers implementing Ajax-technologies probably don't know what's happening in the background, they rely on libraries instead), therefore documentation on the protocol content won't be available. This makes reverse engineering a necessity.
As soon as application logic moves to the web-browser (like in Ajax, see chapter 6.3.5 on page 38), the whole logic must be emulated by the test itself, it cannot rely on any web-browser or another component to do this, as they are not part of the test-scenario. Even protocol-techniques like Cookie tracking (see chapter 6.4.4 on page 44) must be emulated. Furthermore, the functionality on the web-client itself cannot be tested.
In addition, just as for all other non-blackbox-tests, the test-team is going to have a hard time to keep up with internal API-changes.

#### 6.5.1.4 Summary Statement

In most cases, it is not practical to cover a variety of use-cases in this type of testing. HTTP-testing is a powerful tool for stress-testing, where you will choose the 3-5 most important use cases. You have high efforts to get it up and running, and you will have troubles following API-changes - but once the test-environment for a few use cases is up and running, having the stress-test-capatibility certainly pays off.
Still, HTTP-request-testing cannot be compared to web-page-based testing. They serve different purposes and should not be used for each other's purpose.

### 6.5.2 Automation indirect via the web-browser

One of the most common approaches in test automation of web-application is by using the web-browser itself, just as the user does. Many test automation tools, like Mercury Winrunner (see figure 6.9) or iOpus iMacro (see chapter 5.5.1.4 on page 20), are going for this approach. All existing solutions are focusing on windows-based automation, most often using Microsoft Internet Explorer. It is very important to note that two

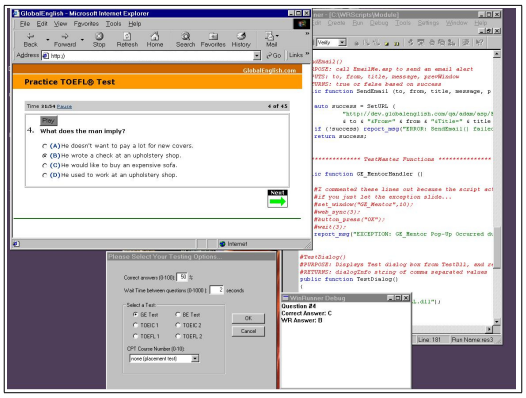


Figure 6.9: Mercury WinRunner Example Screenshot

totally different concepts of technology are used in this approach:

- APIs of the Operating System itself, in order to get a grip on the browser window.
- APIs of the browser, to automate the browser itself

If an unknown browser is used and, therefore, the API not known to the test automation software, it can still act on the web-page using XY-coordinates. This variant of test automation is not covered in this document, as it is very unreliable and should be avoided in all cases.

From the automation software point of view, there's nothing special about the web-browser in comparison to other applications (rich-client-automation), except that the content of the web-page itself is accessed in a browser-native way.

#### 6.5.2.1 Technical Example

In order to understand this type of automation better, within this document, a complete example will be provided, based on a self-implemented test automation software for Internet Explorer on Microsoft Windows.
The implementation consists, similar to the 2 used APIs, of two parts:

- One part, implemented in C++, to get the appropriate handle of the browser window
- One part, implemented in a VB-Script, to act on the browser window

The simple test script will load up the web-page `http://www.google.com`, enter a search term and provide the URL of the search results page.

**Getting the Window-Handle** Each component within Windows has a so-called "window handle". This Window-Handle can be used to uniquely identify and access the component. For Instance, one loaded instance of Microsoft Internet Explorer contains the tree of components you can see in figure 6.10[3]. In this tree, you can see the handle of the main component, but also the sub-components of the window, like the toolbar or the menubar. For getting the window handle, native Windows-APIs have

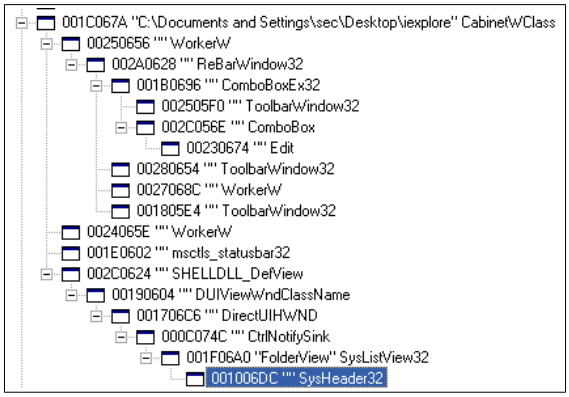


Figure 6.10: Microsoft Spy++ Output on Internet Explorer handle

[3]You can always get this by using the tool Microsoft Spy++, provided within the Microsoft Visual Studio.

to be used.

Listing 6.11: Example C++ Code for finding a window-handle

```
IDispatch* CGuiTestDlgAutoProxy::FindWindow(LPCTSTR szName, LPCTSTR szClass
    )
{
  AFX_MANAGE_STATE(AfxGetAppModuleState());

  if( strcmp( szName, "" ) == 0 )
  {
    szName = 0;
  }
  if( strcmp( szClass, "" ) == 0 )
  {
    szClass = 0;
  }
  if( CWnd *pWnd = CWnd::FindWindow( szClass, szName ) )
  {
    pWnd->BringWindowToTop();
    pWnd->SetActiveWindow();

    Window *pWindow = new Window;
    pWindow->InternalAddRef();
    pWindow->SetWnd( pWnd->m_hWnd );
    IDispatch *pIDispatch = 0;
    pWindow->InternalQueryInterface( &IID_IDispatch, (void**)&pIDispatch );
    pWindow->InternalRelease();

    return pIDispatch;
  }
  return 0;
}
```

The unique identifiers for finding a window handle an be either a name (the title of the window), or the so-called class of the window. The “class” means the type of application that launched the window - whether it is Word, Internet Explorer, calculator or something different. If we are searching for any instance of the Microsoft Internet Explorer, we would just leave the name of the window empty, and provide the class “IEFrame” to be searched for. In order to test this application, you have to embed the function provided above into a COM-Object-DLL, so that we can access the function via VB-Script.

**Act on the Browser Window** The following VB-Script-Code uses the function to get a handle on the Explorer-Window.

Listing 6.12: VB-Script for automating Internet Explorer

```
1  Set aut = WScript.CreateObject("Name-of-my-COM-Server")
2
3  hWndIE = aut.FindWindow("", "IEFrame")
4  If hWndIE = 0 Then
5    WScript.Echo "Cannot find Internet Explorer window!"
6    WScript.Quit
7  End If
8
9  Set ie = aut.FindInternetExplorer(hWndIE)
10 ie.Navigate2("http://www.google.com")
11 Do While ie.busy
12   WScript.Sleep 100
13 Loop
14
15 Set editBox = ie.Document.all.item("q")
16 editBox.value = "DBAI TU-Wien"
17 WScript.Sleep 1000
18 Set button = ie.Document.all.item("btnI")
19 button.click
20 WScript.Sleep 1000
21 WScript.Echo "URL: " + ie.Document.url
22
23 WScript.Quit
```

This example is very important to understand - it highlights all technical concepts needed for automating basic HTML pages.

- In the first few lines, we are getting the handle on any open instance of the Internet Explorer (and bail out in case no one can be found)
- Starting on line 10, the script acts directly on functions provided by the Internet Explorer itself. The code is absolutely specific to the browser being used. In this example, line 10 tells Explorer to switch to the webpage `http://www.google.com`, and waits within a loop until the page is fully loaded.
- Starting in line 15, the script acts on the "document"-property. This property, and all properties and functions contained within, are absolutely specific to the web-page loaded, and are in no way generalised or standardised. All functions represent the DOM-tree (see chapter 6.3.1 on page 33) of the web-page itself. In

this example, the query-item of the web-page is loaded by name and populated with some text. In addition, the button is pressed to trigger an action.

- After one second of waiting, the current URL is extracted and presented to the user

This is a very basic example of the functionality. In reality, the first 10 lines will always be the same. However, starting on line 15 (where the "document"-property is accessed the first time), all content is specific to the web-application in-test, and complexity of the automation script will of course depend on the complexity of the tested application itself. In real testing, you will never write this code manually. Rather than that, it is either a pre-recorded macro (see chapter 5.5.1 on page 19) or managed via test cases and automation script templates (see chapter 5.5.2 on page 25).

#### 6.5.2.2 Advantages

Automating indirect via the web-browser is a well-established and broadly used way of testing web-applications. After all, it comes nearest to the end-user test: It uses the web-browser just as the user does. But instead of clicking and typing in the components, the automation software uses the internal APIs of the browser to fulfill those actions (just like JavaScript does) - which behaves identical in almost all cases.
Due to this advantage (acting like the end-user), all web-browser-specific behaviors are automatically covered by the application. This includes easy parts (JavaScript-field-validation), but also complex concepts like Ajax (see chapter 6.3.5). All application logic covered by the web-browser should still work seamlessly for the automation software. Since the automation software is acting outside of the browser, it does not have any problems on external applications being involved to the testing. One typical example is automating the test of report-webpages, where you can download the generated report. Now, when downloading, the automation software could automatically open the report within a spreadsheet application to verify the provided numbers and ensure compatibility. Cross-application test is a powerful strength of these types of applications.
A special challenge for test-automation is the support of embedded applications - like Java Applets (see chapter 6.3.7 on page 41) or MacroMedia Flash (see chapter 6.3.9 on page 43). Test automation software using the indirect approach can be equipped with a list of plugins to support additional technologies. Due to the encapsulation approach (Web-Browser being just another window), it is easy to automate testing of embedded

applications (Java-Applet being part of the web-page). Of course, again, the content of the automation is completely up to the web-application itself.

#### 6.5.2.3 Disadvantages

The automation software gets exactly the information the user gets - and this is just enough as long as blackbox-testing (see chapter 5.3.3.1 on page 8) is done. However, as soon as you need more advanced testing methods, you cannot get it. One example might be to ensure that all passwords entered on any web-page are transmitted in https-mode only, and that this rule is never broken. This type of test is not possible on indirect browser automation.
Since the browser is acting like the user, it might also get in conflict with the user. Assume that the user wants to surf the web while the automated test is in progress[4]- this might affect test-results, as the explorer-instances might share the same cookies, same cache content and other resources.

Without taking explicit care, it is not ensured that the test-environment is "clean". Ideally, each test-run must start under the same conditions - which means, same cache-content, same cookies available, same authentication information, and several other subtle aspects. As the automation uses the installed instance of Internet-Explorer, the settings are not reset by default. It is a mandatory task for the test management to make ensure that the test-environment is stable all the time, and in this case, explicit measures (e.g. manual reset of all settings) must be taken.

#### 6.5.2.4 Summary Statement

Automation via indirect web-browser automation contains a lot of advantages, and it comes closest to the end-user experience. However, it absolutely needs a dedicated machine for its acting (just as an end-user does), and understanding the different layers of automatisation - Windows, Internet Explorer, Web-Application - might become tricky for the test automation developer.

[4]In theory, a machine under test must never be used for anything else than the testing, because each action can influence test-results somehow. But practically, this will happen frequently.

### 6.5.3 Automation using an embedded web-browser

In comparison to the indirect approach (see chapter 6.5.2 on page 51), the embedded web-browser approach works pretty similar: It uses web-browsing technology to emulate 100 percent the client-side. However, instead of using some browser external, the browser functionality is compiled in - and therefore, more direct accessible. For instance, the Gecko-Browser-Engine[5], in use in Firefox-Browser, is available as source and therefore can be compiled in. In addition, modifications are possible wherever needed. Lixto (see chapter 5.5.1.4 on page 20) uses this approach, for instance.

#### 6.5.3.1 Advantages

Due to the direct-API-access to all web-browsing components, embedded browser tests can provide more valuable information. While the indirect test focuses on blackbox-testing only, embedded browser can provide backend-information in addition - e.g. HTTP protocol analysis, or cache-usage. It can be some sort of hybrid mixture of indirect-web-browser-test and direct emulation of HTTP-Requests (see chapter 6.5.1 on page 47). The current existing solutions don't support this backend-information, but it can be made available if needed.
In addition, it is resistant to any environmental changes - for instance, it is very easy for the embedded-browser-approach to reset all cookies and cache-related information on start of a test, as it does not need to touch the environment (like the installed Microsoft-Internet-Explorer, for instance). Furthermore, it does not conflict with any other windows currently open, and multiple instances of the test can run on the same machine at the same time - as no common components are shared.

#### 6.5.3.2 Disadvantages

Due to the compiled-in-version of the browser, the automated test might show up differences in application-behaviour, compared to behaviour in normal browsers. First of all, because not all rendering engines are available in open source - Microsoft Internet Explorer, for instance, is not. And while it can be used as an embedded control, it will not provide the benefits mentioned before due to the tight API. In addition, even when the same browser engine is used, the browser itself might still behave differently for whatever reasons. And those differences, if they show up, might become a hard obstacle on test automation.

[5] `http://www.mozilla.org/newlayout/`

#### 6.5.3.3 Summary Statement

The embedded approach combines the advantages of both indirect testing and HTTP-based testing, and this is a great advantage. However, when compatibility issues come up, they will be very hard to bypass due to the all-in-one compilation.

# Chapter 7

# Lixto for Abstract Test Automation

Chapters 5 on page 2 and chapter 6 on page 29 covered aspects both for Software Test in General, but also the technical aspects of web-applications, especially in respect to test automation methodologies (see chapter 6.5 on page 47).
Now, the main aim is to provide a prototype, combining the following functionalities:

- Use Lixto as Test-Automation Engine
- Take Advantage of the abstract automation concept (see chapter 5.5.2 on page 25)

For this prototype, actual code has to be written. All Source-Code is printed in chapter A on page 75.
Overall, three pieces of software are needed here:

- Example Web Application: Some example web application. This enables everybody to reproduce the implemented software without dependency on external web pages. See chapter 7.1 on the following page for more information on this.
- Test Case Management: The concept of abstract test automation relies on proper test-case-creation and management in a formal way. No full solution is provided within here (this is up to commercial products), but enough for providing the main feature, the generator.
- Generator: The Generator converts the abstract formal test case definitions into a Lixto-Script for execution. See chapter 7.3 on page 63 for details.

## 7.1 Example Web-Application

For Testing the Web Automation, a simple Test Application is provided as Java Servlets (see sources in chapter A.1 on page 75). The simple Web-Application just provides the following services:

- The purpose of the web-application is to provide a simple user-account management
- At the beginning, the user must log in, using hardcoded credentials
- Show all person records currently available to the user - within a HTML-table, where the user can see both first name and last name of all records (see figure 7.2 on the next page)
- You can click on one item to edit the entry, or delete the data entry. When editing, all fields are shown and default values are provided (see figure 7.1).
- In addition, the user can add new entries by using the appropriate action button.
- When a new record is created, it must be usable for login as credentials.
- When a password is changed, it must come into effect immediate as soon as a new login is attempted (by returning to the start-page).

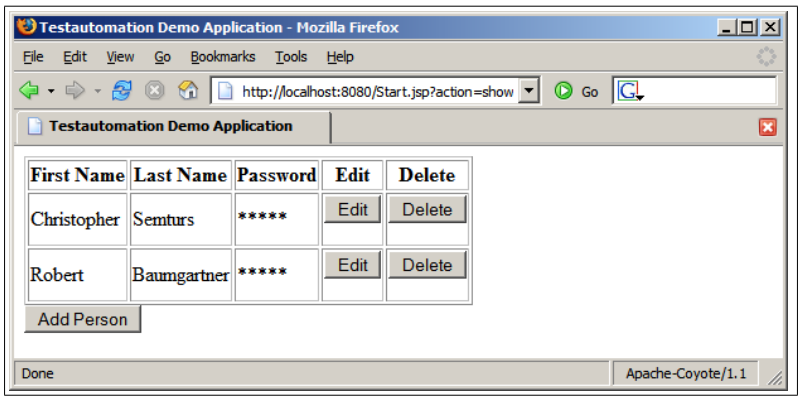


Figure 7.1: Example Web Application - Data List Overview

The Web-Application is kept simple by intention. Lixto itself has a powerful pattern recognition engine, but we don't need to take advantage of it in order to provide the concept of abstract test automation. Furthermore, by using the internal test-application, we can be sure that the example will still work one year later.

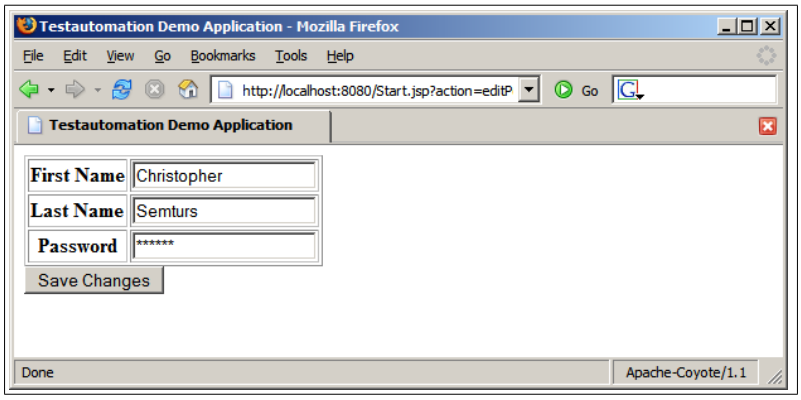


Figure 7.2: Example Web Application - Edit Single Entry

## 7.2 Test Case Management

As for Test Case Management, we don't want to rely on commercial products (like QAS.TCS, see chapter 5.5.2.4 on page 27 for more details) here, because they are not available for the public without licensing. Therefore, a simple test case management is provided, without any user-interface (files only).
For both Test Cases and Mask descriptions, XML-files and appropriate grammars are provided (see chapter A.2 on page 83).

### 7.2.1 Application Definition

Of course, in order to apply formal test automation, we need to actually define the application under test - which means, make the input screen parameterizable. Practically, as describe in chapter 7.1 on the previous page, this means three masks are important to be covered by our automation. Therefore, for each of the three masks, an appropriate XML-description is provided (for the XML-content itself, take a look at chapter A.3 on page 85)

- The Login Mask
- The list view for seeing the persons currently available in the database
- The "Edit"-Mask for Persons - which is used both for creation and modification

As an example, this is an excerpt of the description - covering the login-mask.

Listing 7.1: Example Web-Application Mask Descriptions

```
3  <Mask id="Login">
4    <control>
```

```
5       <type>field</type>
6       <name>fieldUsername</name>
7       <identifiertype>xpath</identifiertype>
8       <identifier>/html/body/form/table/tbody/tr[1]/td/input</identifier>
9     </control>
10    <control>
11      <type>field</type>
12      <name>fieldPassword</name>
13      <identifiertype>xpath</identifiertype>
14      <identifier>/html/body/form/table/tbody/tr[2]/td/input</identifier>
15    </control>
16    <control>
17      <type>button</type>
18      <name>LoginButton</name>
19      <identifiertype>xpath</identifiertype>
20      <identifier>/html/body/form/table/tbody/tr[3]/td/input</identifier>
21    </control>
22  </Mask>
```

The XML-descriptions themselves are easy to understand, both for humans and software. The fields are all identified using their XPATH - which is sufficient for our example. For more complex applications demand will arise to take advantage of Lixto's full capatibilities (for identifying fields in a more robust way).

### 7.2.2 Test Cases

The test cases themselves are based on the defined application - which might, each test-case must take reference to one or more masks defined.
The test cases are defined in a formal way using test-actions (see chapter 5.4.1.3 on page 12 for more Details), using XML-Format.

We will define one big test-case, verifying 4 functionalities[1]:

1. Verify Successful Login by using the hardcoded credentials
2. Verify that the first entry in the person list is "Christopher Semturs"
3. Edit the person by changing the Last Name to "Abcde"
4. Go Back to the Login Page
5. Verify that Login works now by providing "Abcde" as last name instead of "Semturs"

[1]Normally, you would need a list of test-cases for each, using appropriate dependencies

When using the formalized way (using XML according to our grammar), this test-case can be defined by a list of single test actions - as you can see in the following listing, where we covered the actions on the first login mask.

Listing 7.2: Test actions for our Example-Test-Case - Excerpt

```
<Mask id="Login">
  <!-- Enter username and password -->
  <Testaction>
    <Control>fieldUsername</Control>
    <Action>setValue</Action>
    <Parameter>Semturs</Parameter>
  </Testaction>
  <Testaction>
    <Control>fieldPassword</Control>
    <Action>setValue</Action>
    <Parameter>Pass01</Parameter>
  </Testaction>
  <!-- press login button -->
  <Testaction>
    <Control>LoginButton</Control>
    <Action>push</Action>
  </Testaction>
</Mask>
```

For the complete listing of all test actions, take a look at chapter A.4 on page 88.

## 7.3 Abstract Automation Generator

The generator combines the concept of formal test case management and test automation using Lixto: Based on the formal test case definitions, Lixto-code is created for execution.
Using normal Java-classes, both the application-definition-XML and the test case-XML is read in. Afterward, using hardcoded function-call-mappings, the appropriate lixto-script is created:

Listing 7.3: Automation Generator - Excerpt of Class ScriptGenerator

```
  public void process() throws Exception {
      final ApplicationUnderTest aut = getAut(applicationDefinitionXml);
      final TestCase tc = getTestCase(testcaseXml, aut);
      // now get the lixto-script
      LixtoScript script = new LixtoScript(tc);
```

```
42          // get the XML-documents for their output
43          // Lixto Model
44          final File outputLixtoModelFile = new File(outputDirectory,
                FILENAME_MODEL);
45          final Document outputLixtoModel = script.getLixtoModel();
46          writeDocumentToFile(outputLixtoModel, outputLixtoModelFile);
47          // Lixto Wrapper
48          final File outputLixtoWrapperFile = new File(outputDirectory,
                FILENAME_WRAPPER);
49          final Document outputLixtoWrapper = script.getLixtoWrapper();
50          writeDocumentToFile(outputLixtoWrapper, outputLixtoWrapperFile);
51          // desired output as reference
52          final File outputLixtoDesiredResultFile = new File(outputDirectory,
                 FILENAME_DESIREDRESULT);
53          final Document outputLixtoDesiredResult = script.getDesiredResult()
                ;
54          writeDocumentToFile(outputLixtoDesiredResult,
                outputLixtoDesiredResultFile);
55      }
```

The complete sources of all classes and packages in use can be reviewed in chapter A.5 on page 90. They are pretty straight forward.

### 7.3.1 Lixto generated Code

Lixto does not support any concept of content verifications (e.g. "Verify if field 1 has the content ABCD"), as we saw already in chapter 5.5.1.5 on page 24. However, verifications are a very crucial aspect of automated software test.
Therefore, we will follow a slightly different approach on the generated XML-files: The XML-files being generated are representing the Lixto-Model, the Lixto-Wrapper and an expected output. The Wrapper, once executed, produces an output fitting to the Lixto Data Model (which is explained in [BCL05]). And the output must be verified manually[2] to fit the expected output - which means that all verifications within the testcase must be done via the output model. For each testcase, a different subdirectory is populated - which represents the different Lixto-Projects to be loaded.
Note that the model-approach is only needed if actual verifications happen. A testcase might as well just execute some commands on the web-pages, without verifying results (verification could happen at the backend, for instance).

[2]This could also be automated, of course. But since Lixto itself does not support batch-mode with external error-handling right now, we are not automating this step.

For example, as for the test-action where the login-button is pressed, the following Lixto-code is generated[3]:

Listing 7.4: Example Excerpt from generated Lixto Wrapper Code

```
180           <keyflags />
181         </events>
182       </actions>
183       <actions xsi:type="actions:Comment" label="Mask=PersonList/Control=
              PersonLastNameLabel01/Testaction=verifyValue/LixtoActionId=
              TestActionForLixto4" />
184       <actions xsi:type="actions:MouseAction">
185         <events target="/html/body/table/tbody/tr[2]/td[2]">
186           <keyflags />
187         </events>
188         <events target="/html/body/table/tbody/tr[2]/td[2]" type="mouseup">
189           <keyflags />
190         </events>
191       </actions>
```

For verifications, as mentioned above, we cannot do direct verification within Lixto. Therefore, we extract all values to be verified.
Based on all testactions, the ones which are verifying are extracted, and a data-model is created for their return value (comments are created for each action, even if it is not verifying, you can ignore them):

Listing 7.5: Lixto Data Model created based on verification actions

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <model:OutputModel xmlns:model="model" xmlns:navigation="navigation"
       xmlns:wrapper="wrapper" xmlns:definitions="definitions" xmlns:actions="
       actions" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.
       org/2001/XMLSchema-instance" xmi:version="2.0">
3    <root name="TestcaseResult">
4      <!--Mask: Login-->
5      <!--Control: fieldUsername-->
6      <!--Testaction: setValue-->
7      <!--LixtoActionId: TestActionForLixto1-->
8      <!--Mask: Login-->
9      <!--Control: fieldPassword-->
10     <!--Testaction: setValue-->
11     <!--LixtoActionId: TestActionForLixto2-->
12     <!--Mask: Login-->
```

[3]There might be more efficient ways for the Lixto-Script. However, this implementation fulfills our purpose just fine.

```
13    <!--Control: LoginButton-->
14    <!--Testaction: push-->
15    <!--LixtoActionId: TestActionForLixto3-->
16    <!--Mask: PersonList-->
17    <!--Control: PersonLastNameLabel01-->
18    <!--Testaction: verifyValue-->
19    <!--LixtoActionId: TestActionForLixto4-->
20    <children name="TestActionForLixto4" xsdSimpleType="string" minOccurs="
          1" maxOccurs="1" />
```

The actions for extracting the value from the web-page into the data-model is also generated. Note that the script contains Lixto-internal comments. They are displayed during execution of the Wrapper-script and will help the end-user identify which testcase and which specific test-action are currently being executed:

Listing 7.6: Example Code from generated Lixto Wrapper for verifications

```
183     <actions xsi:type="actions:Comment" label="Mask=PersonList/Control=
            PersonLastNameLabel01/Testaction=verifyValue/LixtoActionId=
            TestActionForLixto4" />
184     <actions xsi:type="actions:MouseAction">
185       <events target="/html/body/table/tbody/tr[2]/td[2]">
186         <keyflags />
187       </events>
188       <events target="/html/body/table/tbody/tr[2]/td[2]" type="mouseup">
189         <keyflags />
190       </events>
191     </actions>
192     <actions xsi:type="actions:WrapAction">
193       <root xsi:type="wrapper:SimpleModelPattern">
194         <outputSchemaNode xsi:type="model:OutputElement" href="lixtoModel
                .lixmod#//@root/@children.0" />
195         <filters occurenceType="SINGLE">
196           <extractor xsi:type="definitions:PathDefinition" pathType="
                  XPATH" target="/html/body/table/tbody/tr[2]/td[2]">
197             <input xsi:type="definitions:PatternBinding" pattern="//
                    @pageClasses.0/@actions/@actions.10/@root" />
198             <attributes name="type" value="text" />
199             <attributes name="name" value="PersonLastNameLabel01" />
200             <attributes name="text" value="" nodetype="INTERNAL" />
201           </extractor>
202           <range toDirection="BEGIN" />
203         </filters>
204       </root>
205     </actions>
```

```
206       <actions xsi:type="actions:Comment" label="Mask=PersonList/Control=
              buttonEditPerson01/Testaction=push/LixtoActionId=
              TestActionForLixto5" />
207       <actions xsi:type="actions:MouseAction">
208         <events target="/html/body/table/tbody/tr[2]/td[4]/form/input[3]">
209           <keyflags />
210         </events>
211         <events target="/html/body/table/tbody/tr[2]/td[4]/form/input[3]"
                type="mouseup">
212           <keyflags />
213         </events>
214       </actions>
```

In addition, a small XML-file representing the expected results is generated. It could be parsed by a program. But in our case, we have to read it manually.

Listing 7.7: Expected Results for Data Extraction for Sample Testcase

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <results xmlns:navigation="navigation" xmlns:wrapper="wrapper" xmlns:model=
       "model" xmlns:definitions="definitions" xmlns:actions="actions"
       xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/
       XMLSchema-instance">
3    <!--Mask: Login-->
4    <!--Control: fieldUsername-->
5    <!--Testaction: setValue-->
6    <!--LixtoActionId: TestActionForLixto1-->
7    <!--Mask: Login-->
8    <!--Control: fieldPassword-->
9    <!--Testaction: setValue-->
10   <!--LixtoActionId: TestActionForLixto2-->
11   <!--Mask: Login-->
12   <!--Control: LoginButton-->
13   <!--Testaction: push-->
14   <!--LixtoActionId: TestActionForLixto3-->
15   <!--Mask: PersonList-->
16   <!--Control: PersonLastNameLabel01-->
17   <!--Testaction: verifyValue-->
18   <!--LixtoActionId: TestActionForLixto4-->
19   <ExpectedResult lixtoModelName="TestActionForLixto4" expectedResult="
         Semturs" />
```

The complete output of the program can be reviewed in chapter A.6 on page 113.

## 7.4 Using Lixto itself

The script-file created by the generator can be loaded in Lixto, to be executed by the Lixto Engine, as you can see in figure 7.3:

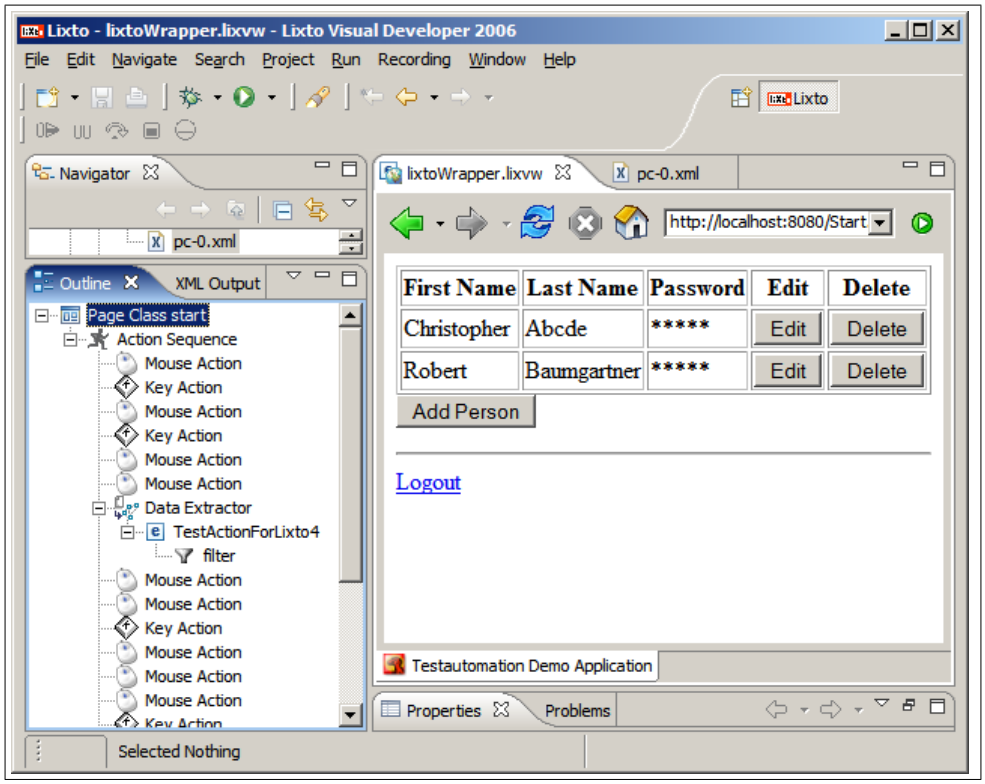


Figure 7.3: Lixto Execution on example Test-case

If the Lixto-run fails during runtime, there must be an error somewhere either within the application under test, or the test-script itself.
If it succeeds, the last step is to verify the output of the Lixto-extraction with our desired results. The output for our sample test-case can be found in the listing below. The expected result is shown in chapter A.6.3 on page 129.

Listing 7.8: Real Results for Data Extraction for Sample Testcase

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <lixto:documents xmlns:lixto="http://www.lixto.com/navigation">
3    <lixto:document lixto:uri="http://localhost:8080/Start.jsp"/>
4    <lixto:document lixto:uri="http://localhost:8080/Start.jsp">
5       <TestActionForLixto4>Semturs</TestActionForLixto4>
6    </lixto:document>
7    <lixto:document lixto:uri="http://localhost:8080/Start.jsp?action=
         editPerson&amp;PersonLastName=Semturs"/>
8    <lixto:document lixto:uri="http://localhost:8080/Start.jsp"/>
```

```
9     <lixto:document lixto:uri="http://localhost:8080/Start.jsp"/>
10    <lixto:document lixto:uri="http://localhost:8080/Start.jsp">
11       <TestActionForLixto12>Abcde</TestActionForLixto12>
12    </lixto:document>
13 </lixto:documents>
```

As you can see, they matched - our testcase was executed successfully.

Note that we used only a very limited set of Lixto action types in our test environment:

- MouseClick-Events
- KeyPress-Events
- DataExtraction-Actions, using Simple-Extraction based on xpath only.

Depending on the application under test, you might need much more complex constructs - like mouse-movements or wait-commands. If you want to take advantage of new Lixto-constructs, you would have to apply new patterns to our abstract test case description language, too. However, this is not the focus of this master thesis. The advanced capatibilities of Lixto for navigation and extraction are well documented and certainly ease your work on complex test applications.

## 7.5 Possible Product Enhancements

In this chapter, possible additional tools within the Lixto Product Series are explained, which would support on Lixto usage for abstract automation:

- Test Case Modeling Tool (see chapter 7.5.1)
- GUI-Map Importer (see chapter 7.5.2 on the next page)

### 7.5.1 Test Case Modeling Tool

In our example the test cases are defined manually - using an XML Editor.
Of course, manual editing a XML is not user friendly for professional testers, it is rather technical. And even if - using an advanced editor - the XML can be edited in an easy way, it is still hard to model the test cases without actually seeing the real application to “play around”. Clearly, we need a better way to write test cases.

Once we have the test mask definitions, there are two approaches to write test cases:

1. Write them directly based on the test mask definitions, without having the application under test. In order to make this easily doable for the test case writer, we need to provide a "real" UI to the user - which means, the fields and buttons of the application must really be displayed as fields and buttons, not as text instructions. The user must be able to interact with the UI (e.g. input values, enter expected values for verification).
   This scenario is usually used if the tested application is not available for testing (e.g. early within the development process).
2. Interact with the real tested application. This is just like current recording of scripts within Lixto. However, based on the GUI-Map (see chapter 7.5.2), Lixto VD could automatically map the recorded input into real test cases. Additional actions like e.g. verification need to be made configurable via Lixto VD, and a tighter integration with test case management would be required for this.

While the first option is a completely new application (most likely not web-based at all), the second option is rather a big enhancement for Lixto VD itself.

### 7.5.2 GUI-Map Importer

As for external additional tools, a big topic would be the introduction of a so-called GUI-Map Importer. When you record once a wrapper-script within Lixto Visual Developer, you most likely apply very advanced data extraction rules. It would be great to bring them in full power to your test automation, without having to fizzle around with the generated Lixto-Wrapper-XML-file right within your text editor.

Therefore, the perfect approach would be to record the first few test-cases using Capture and Replay approach - and to implement a tool which converts the created wrapper-script into testmask and testcontrol-definitions. This does not only take away the biggest disadvantage on abstract automation, but also bring Lixto's advanced features on wrapper creation and extraction functionalities into the picture. As for the end user, he just has to navigate the tested application and specify meaningful names for the masks (e.g. "LoginMask", "PersonList" or "PersonEdit") and items (e.g. "Field-LastName").

As for effective Test Case Modeling, we need in addition layout information for the application under test - which is painful to define manually. The perfect approach is having Lixto extract this information automatically, as much as it can theoretically. At least the rough table structure of all fields, labels and buttons could be extracted

and stored automatically within the test mask representation in XML. This would be a big advantage compared to other competitors on the market, as they extract only technical information on their implementations of GUI-Map Importers, but no layout information.

# Chapter 8

# Conclusion

## 8.1 Summary

Within this master thesis, the basic concept of Abstract Automation (also known as Keyword-Driven-Testing) is explained. It picks up the reader at the very basics of testing, like testing terms, and compares the two main approaches for automated software test: Capture and Replay approach, and abstract automation - including a small introduction to some products currently available on the market.
In parallel, a brief introduction to web-applications - from a technical point of view - is provided. Understanding the technical aspects of web-applications helps to get the full picture on the special aspects of automated software-test on web-applications.

## 8.2 Contribution of Knowledge

Lixto is currently used for regular extraction of information from various web-sites, fitting them into a defined data format (XML-based) and make this information therefore easily accessible, even cross heterogeneous web-pages.
Being used for software-test is a new aspect for Lixto. While Lixto itself was not modified for this master thesis, I provide - as a result of this work - a small test-case-management-tool which enables effective use of Lixto for automated test for web-applications. This framework is immediately available and can be used on test projects.
In addition, an analysis of possible next work items is highlighted, as well as a general analysis of Lixto compared to other tools on web test automation.

## 8.3 Future Research

The test case management implementation available as for now can be classified as a "proof of concept", or as a feasibility study.
It might be already enough for testing web-applications in some projects. But if Lixto shall become a competitive player on the market of software test automation, some key features might become required to be researched and implemented[1]:

**API for Batch-Mode** Currently, executing one test-case still involves a series of manual steps: The Lixto-Files (Model, Wrapper) have to be created. Afterward, they have to be imported into a Lixto-Project for execution. The Wrapper-script has to be started manually. And finally, the output has to be compared manually to the expected output (for the verification actions - see chapter 7.3.1 on page 64). In the ideal case, however, none of this steps should be required.
Theoretically, this goal can be achieved: Lixto has an API to start a wrapper-script automatically on a dedicated URL (via Command-Line). And an automatic verification/comparison of the output-XML can be implemented, too. However, in case of a problem on one specific testcase, Lixto will just stop working (and present the location of the problem of the user). For batch-mode operations - which means e.g. executing 100 test-cases during night - the execution should not stop with a user-prompt, but rather still finish the wrapper-script and write appropriate log-output to a given output-file for later analysis.

**Asserts** Currently, the verification of content is done indirectly via extraction to a data model. This is working, but not easy to understand and error-prone. I suggest to introduce a new event-type "assert" to the Wrapper Script, in combination with data-extractors - similar to assert-commands in C, C++ or Java. On asserts, I can define a statement like "If the extracted value does not match a specific string, bail out". This will make automated test immediately understandable and much easier. In addition, it might help on improving quality and debug-possibility when creating extraction-scripts for web-page extraction. Also, some sort of debug-log-output, and a simple debug.print-statement in wrapper-code, could help on possible problems.

**Verification on Non-Existence** While existence itself can be verified easily in Lixto[2], there's no easy way[3] to verify that a specific control does actually not exist on the

[1] None of these features is critical. Lixto, as it is right now, is already capable enough to be used for complete automation coverage. However, the additional features will ease the work a lot.
[2] My favorite approach is to let a mouseevent click on the appropriate xpath-control - it must not fail
[3] Technically, an evaluation for not existingis possible via XPath-2 (see `http://www.w3.org/TR/`

current webpage - e.g. verification that certain action-links should not be available if the web-user does not have enough privileges. This could be combined with the assert-statements.

**Embedded Object Support** A topic which is true both for web-test as well as data extraction: Non-existant support for embedded objects - like Java Applets, ActiveX-controls or Flash-Layouts - might become a showstopper on some webpages. However, this is a very generic topic and not necessarily associated to web-test itself.

These are the extensions needed within the Lixto-Engine itself in order to improve on web-test-automation.
Further (bigger) enhancements are highlighted in chapter 7.5 on page 69, but can be classified as additional products within the Lixto-Suite, not real requirements for Lixto VD itself. From my point of view, I recommend to implement the GUI-Map Importer (see chapter 7.5.2 on page 70) and in parallel also a test case modeling tool based on test mask definitions (see chapter 7.5.1 on page 69). I would not go for the integrated test case modeling based on the real application, as this is really a big load of work. If the GUI-Map Importer is done well, there will be no other need for any integrational test case modeling.

Personally, I enjoyed to implement the prototype for abstract test case management using Lixto. It was a very satisfying feeling to see it up and running, and to be able to add additional and more complex testcases without any problems. It is always nice to see when a concept comes to life.

`xpath20` for details), but a native method on Lixto directly eases both implementation and understanding.

# Appendix A

# Sources

## A.1 Example Web Application

The example Web-Application consists of 3 source-files:

- Main JSP-File "start.jsp" (see chapter A.1.1)
- Main Data-Manager "DataManager.java" (see chapter A.1.2 on page 80)
- Class representing one person "Person.java" (see chapter A.1.3 on page 82)

### A.1.1 Start.jsp

Listing A.1: Example Web-Application File "Start.jsp"

```
1  <%@ page import="com.lixto.abstractautomation.examplewebapp.DataManager,
2                   com.lixto.abstractautomation.examplewebapp.Person"%>
3   <%--
4    Main Start File
5    The Web-App is kept simple by intention.
6    It does not even contain proper session management, for instance, and
         should never be used for real-life JSP-implementations
7  --%>
8  <%@ page contentType="text/html;charset=UTF-8" language="java" %>
9  <html>
10     <head>
11         <title>Testautomation Demo Application</title>
12     </head>
13     <body bgcolor="white">
14
15         <% boolean justShowPersonList = false; %>
```

```
16          <% String additionalMessage = null; %>
17          <% if (request.getParameter("action") == null) { %>
18              <%-- Login Mask --%>
19              <h1>Please login by providing your credentials</h1>
20              <form method="post" action="Start.jsp">
21                  <input type="hidden" name="action" value="login">
22                  <table>
23                      <tr>
24                          <th>Last Name</th>
25                          <td><input type="text" name="lastname"></td>
26                      </tr>
27                      <tr>
28                          <th>Password</th>
29                          <td><input type="password" name="password"></td>
30                      </tr>
31                      <tr>
32                          <td colspan="2">
33                              <input type="submit">
34                          </td>
35                      </tr>
36                  </table>
37              </form>
38          <% } else if (request.getParameter("action").equals("login")) { %>
39              <%-- Login action - verify credentials --%>
40              <%-- URL is enough for entering the following webpages. There's
                     no session management or whatever implemented here
41                  Therefore, as soon is this should be used for anything more
                         serious than the simple test-applicatino, be sure to
                        find appropriate examples forehand --%>
42              <%
43                  DataManager dataManager = DataManager.getInstance();
44                  Person myPerson = dataManager.getPersonByLastName(request.
                        getParameter("lastname"));
45                  if (myPerson == null) { %>
46                      <h1>User does not exist!</h1>
47                  <% } else if (myPerson.getPassword().equals(request.
                        getParameter("password"))== false) { %>
48                      <h1>Wrong password!</h1>
49                  <% } else { %>
50                      <!-- Login successful -->
51                      <% justShowPersonList = true; %>
52                  <% } %>
53          <% } else if (request.getParameter("action").equals("showList")) {
                 %>
54              <%
```

```
55                  justShowPersonList = true;
56                  additionalMessage = "Login successful";
57              %>
58          <% } else if ((request.getParameter("action").equals("addPerson"))
                || (request.getParameter("action").equals("editPerson"))) { %>
59              <%-- Add person and edit person have almost the same behaviour
                    - only default values differ --%>
60              <%
61                  String previousPersonLastName = "";
62                  String defaultFirstName = "";
63                  String defaultLastName = "";
64                  String defaultPassword = "";
65                  if (request.getParameter("PersonLastName") != null) {
66                      // this is an edit request
67                      previousPersonLastName = request.getParameter("
                            PersonLastName");
68                      Person currPerson = DataManager.getInstance().
                            getPersonByLastName(previousPersonLastName);
69                      defaultFirstName = currPerson.getFirstName();
70                      defaultLastName = currPerson.getLastName();
71                      defaultPassword = currPerson.getPassword();
72                  }
73              %>
74              <%-- now we have all our default values - make input form --%>
75              <form method="post" action="Start.jsp">
76                  <input type="hidden" name="action" value="savePersonChanges
                        ">
77                  <input type="hidden" name="previousPersonLastName" value
                        ="<%= previousPersonLastName %>">
78
79                  <table border="1">
80                      <tr>
81                          <th>First Name</th>
82                          <td><input type="text" name="defaultFirstName"
                                value="<%= defaultFirstName %>"></td>
83                      </tr>
84                      <tr>
85                          <th>Last Name</th>
86                          <td><input type="text" name="defaultLastName" value
                                ="<%= defaultLastName %>"></td>
87                      </tr>
88                      <tr>
89                          <th>Password</th>
90                          <td><input type="password" name="defaultPassword"
                                value="<%= defaultPassword %>"></td>
```

```
91                      </tr>
92                  </table>
93                  <input type="submit" value="Save Changes">
94              </form>
95          <% } else if (request.getParameter("action").equals("
                savePersonChanges")) { %>
96              <%-- change person or add person - depending on parameters --%>
97              <%
98                  String firstName = request.getParameter("defaultFirstName")
                        ;
99                  String lastName = request.getParameter("defaultLastName");
100                 String password = request.getParameter("defaultPassword");
101                 // create new person object by default
102                 Person person = new Person();
103                 boolean isNewPerson = true;
104                 if (request.getParameter("previousPersonLastName") != null)
                        {
105                     // we are editing a current person instead
106                     isNewPerson = false;
107                     person = DataManager.getInstance().getPersonByLastName(
                            request.getParameter("previousPersonLastName"));
108                 }
109                 // now set new values
110                 person.setFirstName(firstName);
111                 person.setLastName(lastName);
112                 person.setPassword(password);
113                 // save changes
114                 if (isNewPerson) {
115                     DataManager.getInstance().addPerson(person);
116                 } else {
117                     // nothing further needed - person object was already
                            modified
118                 }
119             %>
120             <%
121                 justShowPersonList = true;
122                 additionalMessage = "Changes saved";
123             %>
124         <% } else if (request.getParameter("action").equals("deletePerson")
                ) { %>
125             <%-- delete person --%>
126             <%
127                 Person deletionPerson = DataManager.getInstance().
                        getPersonByLastName(request.getParameter("
                        PersonLastName"));
```

```
128                 DataManager.getInstance().removePerson(deletionPerson);
129             %>
130             <%
131                 justShowPersonList = true;
132                 additionalMessage = "Person deleted";
133             %>
134         <% } else { %>
135             <h1>Unknown request type occured!!!!</h1>
136         <% } %>
137
138         <% if (justShowPersonList == true) { %>
139              <!-- we show the person list "as is"            -->
140              <% if (additionalMessage != null) { %>
141                 <h1><%= additionalMessage %></h1>
142              <% } %>
143
144             <%-- show list of persons currently available --%>
145             <table border="1">
146                 <tr>
147                     <th>First Name</th>
148                     <th>Last Name</th>
149                     <th>Password</th>
150                     <th>Edit</th>
151                     <th>Delete</th>
152                 </tr>
153                 <% for (Person currPerson : DataManager.getInstance().
                        getPersonList()) { %>
154                 <tr>
155                     <td><%= currPerson.getFirstName() %></td>
156                     <td><%= currPerson.getLastName() %></td>
157                     <td>*****</td>
158                     <td>
159                         <form method="get" action="Start.jsp">
160                             <input type="hidden" name="action" value="
                                    editPerson">
161                             <input type="hidden" name="PersonLastName"
                                    value="<%= currPerson.getLastName()
                                    %>">
162                             <input type="submit" value="Edit">
163                         </form>
164                     </td>
165                     <td>
166                         <form method="get" action="Start.jsp">
167                             <input type="hidden" name="action" value="
                                    deletePerson">
```

```
168                                 <input type="hidden" name="PersonLastName"
                                        value="<%= currPerson.getLastName()
                                        %>">
169                                 <input type="submit" value="Delete">
170                             </form>
171                         </td>
172                     </tr>
173                 <% } %>
174             </table>
175             <%-- add form for adding persones --%>
176             <form method="get" action="Start.jsp">
177                 <input type="hidden" name="action" value="addPerson">
178                 <input type="submit" value="Add Person">
179             </form>
180 
181         <% } %>
182 
183         <!-- we always provide a logout-link at the end -->
184         <hr>
185         <a href="Start.jsp">Logout</a>
186     </body>
187 </html>
```

### A.1.2 DataManager.java

Listing A.2: Example Web-Application File "DataManager.java"

```
1  package com.lixto.abstractautomation.examplewebapp;
2 
3  import java.util.ArrayList;
4  import java.util.List;
5 
6  /**
7   * Data-Manager. Straight forward class managing the "person" elements.
        Static implementation, no database, ... - just the things needed for
        making a demo on test-automation
8   * We don't take any care on performance here - e.g. lists are iterated
        when searching for common terms, instead of building hash-maps.
9   * This should never be used as an example for coding itself.
10  */
11 public class DataManager {
12     // singleton
13     private static DataManager _dataManager;
```

```
14
15      private final List<Person> personList = new ArrayList<Person>();
16
17      private DataManager() {
18          // we create some dummy data hardcoded, for having the demo data as
                needed
19          Person person = new Person();
20          person.setFirstName("Christopher");
21          person.setLastName("Semturs");
22          person.setPassword("Pass01");
23          personList.add(person);
24          person = new Person();
25          person.setFirstName("Robert");
26          person.setLastName("Baumgartner");
27          person.setPassword("Pass02");
28          personList.add(person);
29          // done
30      }
31
32      /**
33       * singleton pattern
34       */
35      public final static DataManager getInstance() {
36          if (_dataManager == null) {
37              _dataManager = new DataManager();
38          }
39          return _dataManager;
40      }
41
42      public Person getPersonByLastName(String lastName) {
43          for (Person person : personList) {
44              if (person.getLastName().equals(lastName)) {
45                  return person;
46              }
47          }
48          // nothing found
49          return null;
50      }
51
52      public List<Person> getPersonList() {
53          return personList;
54      }
55
56      public void removePerson(Person deletionPerson) {
57          this.personList.remove(deletionPerson);
```

```
58      }
59
60      public void addPerson(Person person) {
61          this.personList.add(person);
62      }
63  }
```

### A.1.3 Person.java

Listing A.3: Example Web-Application File "Person.java"

```
1  package com.lixto.abstractautomation.examplewebapp;
2
3  /**
4   * Class representing one person
5   */
6  public class Person {
7      private String firstName;
8      private String lastName;
9      private String password;
10
11     public String getFirstName() {
12         return firstName;
13     }
14
15     public void setFirstName(String firstName) {
16         this.firstName = firstName;
17     }
18
19     public String getLastName() {
20         return lastName;
21     }
22
23     public void setLastName(String lastName) {
24         this.lastName = lastName;
25     }
26
27     public String getPassword() {
28         return password;
29     }
30
31     public void setPassword(String password) {
32         this.password = password;
```

```
33     }
34 }
```

## A.2 XML-Grammars for Abstract Test Automation

### A.2.1 Mask-Description

Listing A.4: XSD for Mask Descriptions

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="
       qualified">
3    <xs:element name="ApplicationUnderTest">
4      <xs:complexType>
5        <xs:sequence>
6          <xs:element ref="Mask" maxOccurs="unbounded"/>
7        </xs:sequence>
8        <xs:attribute name="name" type="xs:string" use="required"/>
9      </xs:complexType>
10   </xs:element>
11   <xs:element name="Mask">
12     <xs:complexType>
13       <xs:sequence>
14         <xs:element ref="control" maxOccurs="unbounded"/>
15       </xs:sequence>
16       <xs:attribute name="id" type="xs:string" use="required"/>
17     </xs:complexType>
18   </xs:element>
19   <xs:element name="control">
20     <xs:complexType>
21       <xs:sequence>
22         <xs:element ref="type"/>
23         <xs:element ref="name"/>
24         <xs:element ref="identifiertype"/>
25         <xs:element ref="identifier"/>
26       </xs:sequence>
27     </xs:complexType>
28   </xs:element>
29   <xs:element name="identifier" type="xs:string">
30   </xs:element>
31   <xs:element name="identifiertype">
32     <xs:simpleType>
```

```
      <xs:restriction base="xs:string">
        <xs:enumeration value="xpath"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="name" type="xs:string">
  </xs:element>
  <xs:element name="type">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="button"/>
        <xs:enumeration value="field"/>
        <xs:enumeration value="label"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
</xs:schema>
```

## A.2.2 Test-cases including Test-Actions

Listing A.5: XSD for Test Cases and Test-Actions

```
<?xml version="1.0" encoding="UTF-8"?>
<!--W3C Schema generated by XMLSPY v2004 rel. 3 U (http://www.xmlspy.com)-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="Action">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="push"/>
        <xs:enumeration value="setValue"/>
        <xs:enumeration value="verifyValue"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="Control" type="xs:string">
  </xs:element>
  <xs:element name="Mask">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Testaction" maxOccurs="unbounded"/>
      </xs:sequence>
```

```
      <xs:attribute name="id" type="xs:string" use="required">
      </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name="Parameter" type="xs:string">
  </xs:element>
  <xs:element name="Testaction">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Control"/>
        <xs:element ref="Action"/>
        <xs:element ref="Parameter" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Testcase">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Mask" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

## A.3 Mask Description for Example Web-Application

Listing A.6: Example Web-Application Mask Descriptions

```
<?xml version="1.0" encoding="UTF-8"?>
<ApplicationUnderTest name="LixtoExampleWebApp" xmlns:xsi="http://www.w3.
    org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="
    ApplicationDefinition.xsd">
  <Mask id="Login">
    <control>
      <type>field</type>
      <name>fieldUsername</name>
      <identifiertype>xpath</identifiertype>
      <identifier>/html/body/form/table/tbody/tr[1]/td/input</identifier>
    </control>
    <control>
      <type>field</type>
      <name>fieldPassword</name>
      <identifiertype>xpath</identifiertype>
```

```
14      <identifier>/html/body/form/table/tbody/tr[2]/td/input</identifier>
15    </control>
16    <control>
17      <type>button</type>
18      <name>LoginButton</name>
19      <identifiertype>xpath</identifiertype>
20      <identifier>/html/body/form/table/tbody/tr[3]/td/input</identifier>
21    </control>
22  </Mask>
23  <Mask id="PersonList">
24    <!-- we handle the first two lines of the table, this is sufficient for
          our testcases -->
25    <control>
26      <type>label</type>
27      <name>PersonFirstNameLabel01</name>
28      <identifiertype>xpath</identifiertype>
29      <identifier>/html/body/table/tbody/tr[2]/td[1]</identifier>
30    </control>
31    <control>
32      <type>label</type>
33      <name>PersonLastNameLabel01</name>
34      <identifiertype>xpath</identifiertype>
35      <identifier>/html/body/table/tbody/tr[2]/td[2]</identifier>
36    </control>
37    <control>
38      <type>button</type>
39      <name>buttonEditPerson01</name>
40      <identifiertype>xpath</identifiertype>
41      <identifier>/html/body/table/tbody/tr[2]/td[4]/form/input[3]</
            identifier>
42    </control>
43    <control>
44      <type>button</type>
45      <name>buttonDeletePerson01</name>
46      <identifiertype>xpath</identifiertype>
47      <identifier>/html/body/table/tbody/tr[2]/td[5]/form/input[3]</
            identifier>
48    </control>
49    <!-- second row of data -->
50    <control>
51      <type>label</type>
52      <name>PersonFirstNameLabel02</name>
53      <identifiertype>xpath</identifiertype>
54      <identifier>/html/body/table/tbody/tr[3]/td[1]</identifier>
55    </control>
```

```
56      <control>
57        <type>label</type>
58        <name>PersonLastNameLabel02</name>
59        <identifiertype>xpath</identifiertype>
60        <identifier>/html/body/table/tbody/tr[3]/td[2]</identifier>
61      </control>
62      <control>
63        <type>button</type>
64        <name>buttonEditPerson02</name>
65        <identifiertype>xpath</identifiertype>
66        <identifier>/html/body/table/tbody/tr[3]/td[4]/form/input[3]</
              identifier>
67      </control>
68      <control>
69        <type>button</type>
70        <name>buttonDeletePerson02</name>
71        <identifiertype>xpath</identifiertype>
72        <identifier>/html/body/table/tbody/tr[3]/td[5]/form/input[3]</
              identifier>
73      </control>
74      <!-- Logout link -->
75      <control>
76        <type>button</type>
77        <name>buttonLogout</name>
78        <identifiertype>xpath</identifiertype>
79        <identifier>/html/body/a</identifier>
80      </control>
81
82
83    </Mask>
84    <Mask id="PersonEdit">
85      <control>
86        <type>field</type>
87        <name>fieldFirstName</name>
88        <identifiertype>xpath</identifiertype>
89        <identifier>/html/body/form/table/tbody/tr[1]/td/input</identifier>
90      </control>
91      <control>
92        <type>field</type>
93        <name>fieldLastName</name>
94        <identifiertype>xpath</identifiertype>
95        <identifier>/html/body/form/table/tbody/tr[2]/td/input</identifier>
96      </control>
97      <control>
98        <type>field</type>
```

```
99      <name>fieldPassword</name>
100     <identifiertype>xpath</identifiertype>
101     <identifier>/html/body/form/table/tbody/tr[3]/td/input</identifier>
102   </control>
103   <control>
104     <type>button</type>
105     <name>buttonSave</name>
106     <identifiertype>xpath</identifiertype>
107     <identifier>/html/body/form/input[3]</identifier>
108   </control>
109 </Mask>
110 </ApplicationUnderTest>
```

## A.4 Formal Test-Case Example Web-Application

Listing A.7: Testactions for our Example-Test-Case

```
1  ï¿£<?xml version="1.0" encoding="UTF-8"?>
2  <Testcase xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="C:\DEV-Dir\Private\Latex-Document\
       Diplomarbeit\listing\TestcaseTestActions.xsd">
3    <Mask id="Login">
4      <!-- Enter username and password -->
5      <Testaction>
6        <Control>fieldUsername</Control>
7        <Action>setValue</Action>
8        <Parameter>Semturs</Parameter>
9      </Testaction>
10     <Testaction>
11       <Control>fieldPassword</Control>
12       <Action>setValue</Action>
13       <Parameter>Pass01</Parameter>
14     </Testaction>
15     <!-- press login button -->
16     <Testaction>
17       <Control>LoginButton</Control>
18       <Action>push</Action>
19     </Testaction>
20   </Mask>
21   <Mask id="PersonList">
22     <!-- first of all, verify that the first entry in the list is actually
           the value "Semturs" -->
23     <Testaction>
```

```
24      <Control>PersonLastNameLabel01</Control>
25      <Action>verifyValue</Action>
26      <Parameter>Semturs</Parameter>
27    </Testaction>
28    <!-- now click the button on the same line -->
29    <Testaction>
30      <Control>buttonEditPerson01</Control>
31      <Action>push</Action>
32    </Testaction>
33  </Mask>
34  <!-- edit the single person -->
35  <Mask id="PersonEdit">
36    <Testaction>
37      <Control>fieldLastName</Control>
38      <Action>setValue</Action>
39      <Parameter>Abcde</Parameter>
40    </Testaction>
41    <!-- now click the button to commit changes -->
42    <Testaction>
43      <Control>buttonSave</Control>
44      <Action>push</Action>
45    </Testaction>
46  </Mask>
47  <!-- relogin - logout first of all -->
48  <Mask id="PersonList">
49    <!-- click on logout link -->
50    <Testaction>
51      <Control>buttonLogout</Control>
52      <Action>push</Action>
53    </Testaction>
54  </Mask>
55  <Mask id="Login">
56    <!-- Enter username and password -->
57    <Testaction>
58      <Control>fieldUsername</Control>
59      <Action>setValue</Action>
60      <Parameter>Abcde</Parameter>
61    </Testaction>
62    <Testaction>
63      <Control>fieldPassword</Control>
64      <Action>setValue</Action>
65      <Parameter>Pass01</Parameter>
66    </Testaction>
67    <!-- press login button -->
68    <Testaction>
```

```
69        <Control>LoginButton</Control>
70        <Action>push</Action>
71      </Testaction>
72    </Mask>
73    <!-- verify that we still got the list -->
74    <Mask id="PersonList">
75      <!--Verify that we have the changed value now-->
76      <Testaction>
77        <Control>PersonLastNameLabel01</Control>
78        <Action>verifyValue</Action>
79        <Parameter>Abcde</Parameter>
80      </Testaction>
81    </Mask>
82  </Testcase>
```

## A.5 Automation Generator

### A.5.1 Class aut.ApplicationUnderTest

Listing A.8: Automation Generator - Class aut.ApplicationUnderTest

```
1  package com.lixto.abstractautomation.scriptgenerator.aut;
2
3  import java.util.HashMap;
4
5  /**
6   * Application under test
7   */
8  public class ApplicationUnderTest extends AutElement {
9      private final HashMap<String, AutMask> maskList = new HashMap<String,
           AutMask>();
10
11     public ApplicationUnderTest(String id) {
12         super(id);
13     }
14
15     public void addMask(AutMask mask) {
16         this.maskList.put(mask.getId(), mask);
17     }
18
19     public final AutMask getMask(String id) {
20         return this.maskList.get(id);
```

```
21      }
22
23  }
```

## A.5.2 Class aut.AutControl

Listing A.9: Automation Generator - Class aut.AutControl

```
1  package com.lixto.abstractautomation.scriptgenerator.aut;
2
3  import com.lixto.abstractautomation.scriptgenerator.controltypes.
       ControlType;
4
5  import java.util.logging.Logger;
6
7  /**
8   * control-definition for the mask in application under test
9   */
10 public class AutControl extends AutElement {
11     private static final Logger logger = Logger.getLogger(AutControl.class.
           getName());
12     private final String identifier;
13     private final ControlType controlType;
14     private final AutMask mask;
15
16     public AutControl(AutMask mask, String type, String name, String
           identifierType, String identifier) {
17         super(name);
18         // identifiertype must be xpath in our version
19         if (identifierType.equals("xpath")==false) {
20             logger.severe("Unknown identifier type: '" + identifierType + "
                   '");
21         }
22         this.identifier=identifier;
23         this.controlType = ControlType.getControlType(type);
24         this.mask=mask;
25     }
26
27     public String getIdentifier() {
28         return identifier;
29     }
30
31     public ControlType getControlType() {
```

```
32          return controlType;
33      }
34
35      public AutMask getMask() {
36          return mask;
37      }
38  }
```

### A.5.3 Class aut.AutElement

Listing A.10: Automation Generator - Class aut.AutElement

```
1  package com.lixto.abstractautomation.scriptgenerator.aut;
2
3  /**
4   * common abstract class for AUT-definitions
5   */
6  public abstract class AutElement {
7      private final String id;
8
9      public AutElement(String id) {
10         this.id = id;
11     }
12
13     public final String getId() {
14         return id;
15     }
16
17 }
```

### A.5.4 Class aut.AutMask

Listing A.11: Automation Generator - Class aut.AutMask

```
1  package com.lixto.abstractautomation.scriptgenerator.aut;
2
3  import java.util.HashMap;
4
5  /**
6   * mask-definition for application under test
7   */
8  public class AutMask extends AutElement {
```

```
9     private final HashMap<String, AutControl> controlList = new HashMap<
          String, AutControl>();
10
11    public AutMask(String id) {
12        super(id);
13    }
14
15    public void addControl(AutControl control) {
16        this.controlList.put(control.getId(), control);
17    }
18
19    public final AutControl getControl(String id) {
20        return this.controlList.get(id);
21    }
22 }
```

## A.5.5 Class controltypes.ControlType

Listing A.12: Automation Generator - Class controltypes.ControlType

```
1  package com.lixto.abstractautomation.scriptgenerator.controltypes;
2
3  import java.util.logging.Logger;
4
5  /**
6   * common abstract class for control types
7   */
8  public abstract class ControlType {
9     private static final Logger logger = Logger.getLogger(ControlType.class
          .getName());
10
11    private static final ControlType ctrlField = new ControlTypeField();
12    private static final ControlType ctrlButton = new ControlTypeButton();
13    private static final ControlType ctrlLabel = new ControlTypeLabel();
14
15    public static ControlType getControlType(String typeName) {
16        if (typeName.equals("field")) {
17            return ctrlField;
18        }
19        if (typeName.equals("button")) {
20            return ctrlButton;
21        }
22        if (typeName.equals("label")) {
```

```
23              return ctrlLabel;
24          }
25          // unknown control type
26          logger.severe("Unknown Control Type: '" + typeName + "'");
27          return null;
28      }
29
30  }
```

### A.5.6 Class controltypes.ControlTypeButton

Listing A.13: Automation Generator - Class controltypes.ControlTypeButton

```
1  package com.lixto.abstractautomation.scriptgenerator.controltypes;
2
3  /**
4   * single control Type
5   */
6  class ControlTypeButton extends ControlType {
7  }
```

### A.5.7 Class controltypes.ControlTypeField

Listing A.14: Automation Generator - Class controltypes.ControlTypeField

```
1  package com.lixto.abstractautomation.scriptgenerator.controltypes;
2
3  /**
4   * single control Type
5   */
6  class ControlTypeField extends ControlType {
7  }
```

### A.5.8 Class controltypes.ControlTypeLabel

Listing A.15: Automation Generator - Class controltypes.ControlTypeLabel

```
1  package com.lixto.abstractautomation.scriptgenerator.controltypes;
2
3  /**
4   * single control Type
```

```
 */
class ControlTypeLabel extends ControlType {
}
```

### A.5.9 Class lixto.LixtoAction

Listing A.16: Automation Generator - Class lixto.LixtoAction

```
package com.lixto.abstractautomation.scriptgenerator.lixto;

import com.lixto.abstractautomation.scriptgenerator.testcase.TestAction;
import org.jdom.Element;

import java.util.List;

/**
 * LixtoAction-class. Used to wrap a testaction and create the lixto-script
 */
abstract class LixtoAction {
    // unique number generator
    private static int UNIQUENUMBERCOUNTER = 0;

    private final TestAction testAction;
    private final int myUniqueNumber;


    protected LixtoAction(TestAction testAction) {
        this.testAction = testAction;
        synchronized (this) {
            myUniqueNumber = ++UNIQUENUMBERCOUNTER;
        }
    }

    /**
     * this is the main method to get lixto-actions. Just return the
         correct object, that's it.
     *
     * @param testAction the testaction
     */
    public final static LixtoAction getLixtoAction(TestAction testAction) {
        switch (testAction.getTestActionType()) {
            case push:
                return new LixtoActionPush(testAction);
```

```
35              case setValue:
36                  return new LixtoActionSetValue(testAction);
37              case verifyValue:
38                  return new LixtoActionVerifyValue(testAction);
39              default:
40                  throw new RuntimeException("Unhandled test action type,
                        implementation fault");
41          }
42      }
43
44      public final TestAction getTestAction() {
45          return testAction;
46      }
47
48      public final String getUniqueId() {
49          return "TestActionForLixto" + Integer.toString(myUniqueNumber);
50      }
51
52      public abstract List<Element> getLixtoXmlList(int
            numberOfActionsBeforeThis, int numberOfVerificationsSoFar);
53
54      /**
55       * this is one of the central elements for all actions - marking the
             correct UI-element by clicking it once.
56       *
57       * @return the list of xml-commandos needed for lixto wrapper
58       */
59      protected final Element getLixtoXmlForMarkingElement() {
60          Element actions = new Element("actions");
61          actions.setAttribute("type", "actions:MouseAction", XmlNamespaces.
                NS_XSI);
62          actions.addContent(getEventsTargetElement(getTestAction().
                getControl().getIdentifier()));
63          actions.addContent(getEventsTargetElement(getTestAction().
                getControl().getIdentifier(), "mouseup"));
64          // completed
65          return actions;
66      }
67
68      private final static Element getEventsTargetElement(String targetName)
            {
69          Element rValue = new Element("events");
70          rValue.setAttribute("target", targetName);
71          rValue.addContent(new Element("keyflags"));
72          return rValue;
```

```
73      }
74
75      private final static Element getEventsTargetElement(String targetName,
            String typeName) {
76          Element rValue = getEventsTargetElement(targetName);
77          rValue.setAttribute("type", typeName);
78          return rValue;
79      }
80  }
```

## A.5.10 Class lixto.LixtoActionPush

Listing A.17: Automation Generator - Class lixto.LixtoActionPush

```
1  package com.lixto.abstractautomation.scriptgenerator.lixto;
2
3  import com.lixto.abstractautomation.scriptgenerator.testcase.TestAction;
4  import org.jdom.Element;
5
6  import java.util.ArrayList;
7  import java.util.List;
8
9  /**
10  * Action-wrapper for pushing buttons (mouseaction)
11  */
12 public class LixtoActionPush extends LixtoAction {
13
14     LixtoActionPush(TestAction testAction) {
15         super(testAction);
16     }
17
18     public List<Element> getLixtoXmlList(int numberOfActionsBeforeThis, int
           numberOfVerificationsSoFar) {
19         final List<Element> rValue = new ArrayList<Element>();
20         rValue.add(getLixtoXmlForMarkingElement());
21         // done - we only mark it
22         return rValue;
23     }
24 }
```

## A.5.11 Class lixto.LixtoActionSetValue

Listing A.18: Automation Generator - Class lixto.LixtoActionSetValue

```
1  package com.lixto.abstractautomation.scriptgenerator.lixto;
2
3  import com.lixto.abstractautomation.scriptgenerator.testcase.TestAction;
4  import org.jdom.Element;
5
6  import java.util.ArrayList;
7  import java.util.List;
8
9  /**
10  * Action-wrapper for setting values
11  */
12 public class LixtoActionSetValue extends LixtoAction {
13
14     LixtoActionSetValue(TestAction testAction) {
15         super(testAction);
16     }
17
18     public List<Element> getLixtoXmlList(int numberOfActionsBeforeThis, int
           numberOfVerificationsSoFar) {
19         final List<Element> rValue = new ArrayList<Element>();
20         rValue.add(getLixtoXmlForMarkingElement());
21         // now set the value
22         // the value is set character by character, by using the charcodes
23         Element actionsElement = new Element("actions");
24         actionsElement.setAttribute("type", "actions:KeyAction",
               XmlNamespaces.NS_XSI);
25         // first of all, the content of the current field must be deleted (
               we replace content, never append)
26         // this is a static procedure
27         actionsElement.addContent(getFieldDeletionEvents(getTestAction().
               getControl().getIdentifier()));
28         // now go through the string, character by character
29         String newValue = getTestAction().getParameterString();
30         char[] singleCharacters = newValue.toCharArray();
31         for (char singleChar : singleCharacters) {
32             // based on reverse engineering of created event-structures, 3
                   steps are needed for key-pressing:
33             // keyCode - keypress - keyUp
34             // therefore, all 3 elements are created for each character
35             Element eventsElement = getBasicEventsElement(getTestAction().
                   getControl().getIdentifier(), Character.isUpperCase(
                   singleChar), false);
```

```
36              eventsElement.setAttribute("keyCode", Integer.toString(
                    singleChar));
37              actionsElement.addContent(eventsElement);
38
39              eventsElement = getBasicEventsElement(getTestAction().
                    getControl().getIdentifier(), Character.isUpperCase(
                    singleChar), false);
40              eventsElement.setAttribute("type", "keypress");
41              eventsElement.setAttribute("charCode", Integer.toString(
                    singleChar));
42              actionsElement.addContent(eventsElement);
43
44              eventsElement = getBasicEventsElement(getTestAction().
                    getControl().getIdentifier(), Character.isUpperCase(
                    singleChar), false);
45              eventsElement.setAttribute("keyCode", Integer.toString(
                    singleChar));
46              eventsElement.setAttribute("type", "keyup");
47              actionsElement.addContent(eventsElement);
48          }
49          // we are complete
50          rValue.add(actionsElement);
51
52          // done - we only mark it
53          return rValue;
54      }
55
56      private static final List<Element> getFieldDeletionEvents(String xpath)
            {
57          final List<Element> rValue = new ArrayList<Element>();
58
59          Element eventsElement = getBasicEventsElement(xpath, false, true);
60          eventsElement.setAttribute("keyCode", Integer.toString(17));
61          rValue.add(eventsElement);
62
63          eventsElement = getBasicEventsElement(xpath, false, true);
64          eventsElement.setAttribute("keyCode", Integer.toString(65));
65          rValue.add(eventsElement);
66
67          eventsElement = getBasicEventsElement(xpath, false, true);
68          eventsElement.setAttribute("type", "keypress");
69          eventsElement.setAttribute("charCode", Integer.toString(97));
70          rValue.add(eventsElement);
71
72          eventsElement = getBasicEventsElement(xpath, false, true);
```

```
73          eventsElement.setAttribute("keyCode", Integer.toString(65));
74          eventsElement.setAttribute("type", "keyup");
75          rValue.add(eventsElement);
76
77          eventsElement = getBasicEventsElement(xpath, false, true);
78          eventsElement.setAttribute("keyCode", Integer.toString(17));
79          eventsElement.setAttribute("type", "keyup");
80          rValue.add(eventsElement);
81          // done
82          return rValue;
83      }
84
85      private final static Element getBasicEventsElement(String xpath,
            boolean isUpperCase, boolean isCtrlKey) {
86          Element eventsElement = new Element("events");
87          eventsElement.setAttribute("focus", xpath);
88          Element keyFlagsElements = new Element("keyflags");
89          if (isUpperCase) {
90              keyFlagsElements.setAttribute("shift", "true");
91          } else if (isCtrlKey) {
92              keyFlagsElements.setAttribute("ctrl", "true");
93          }
94          eventsElement.addContent(keyFlagsElements);
95          return eventsElement;
96      }
97  }
```

## A.5.12 Class lixto.LixtoActionVerifyValue

Listing A.19: Automation Generator - Class lixto.LixtoActionVerifyValue

```
1  package com.lixto.abstractautomation.scriptgenerator.lixto;
2
3  import com.lixto.abstractautomation.scriptgenerator.ScriptGenerator;
4  import com.lixto.abstractautomation.scriptgenerator.testcase.TestAction;
5  import org.jdom.Element;
6
7  import java.util.ArrayList;
8  import java.util.List;
9
10 /**
11  * Action-wrapper for setting values
12  */
```

```
13 public class LixtoActionVerifyValue extends LixtoAction {
14
15     LixtoActionVerifyValue(TestAction testAction) {
16         super(testAction);
17     }
18
19     public List<Element> getLixtoXmlList(int numberOfActionsBeforeThis, int
           numberOfVerificationsSoFar) {
20         final List<Element> rValue = new ArrayList<Element>();
21         rValue.add(getLixtoXmlForMarkingElement());
22         // It is marked - now extract information
23         // note: marking is done primary for verifying that the content
               actually exists
24         Element actionsElements = new Element("actions");
25         actionsElements.setAttribute("type", "actions:WrapAction",
               XmlNamespaces.NS_XSI);
26         Element rootElement = new Element("root");
27         actionsElements.addContent(rootElement);
28         rootElement.setAttribute("type", "wrapper:SimpleModelPattern",
               XmlNamespaces.NS_XSI);
29         Element outputSchemaNodeElement = new Element("outputSchemaNode");
30         rootElement.addContent(outputSchemaNodeElement);
31         outputSchemaNodeElement.setAttribute("type", "model:OutputElement",
               XmlNamespaces.NS_XSI);
32         outputSchemaNodeElement.setAttribute("href", ScriptGenerator.
               FILENAME_MODEL + "#//@root/@children." + Integer.toString(
               numberOfVerificationsSoFar));
33         Element filtersElement = new Element("filters");
34         rootElement.addContent(filtersElement);
35         filtersElement.setAttribute("occurenceType", "SINGLE");
36         Element extractorElement = new Element("extractor");
37         filtersElement.addContent(extractorElement);
38         extractorElement.setAttribute("type", "definitions:PathDefinition",
               XmlNamespaces.NS_XSI);
39         extractorElement.setAttribute("pathType", "XPATH");
40         extractorElement.setAttribute("target", getTestAction().getControl
               ().getIdentifier());
41         Element inputElement = new Element("input");
42         extractorElement.addContent(inputElement);
43         inputElement.setAttribute("type", "definitions:PatternBinding",
               XmlNamespaces.NS_XSI);
44         inputElement.setAttribute("pattern", "//@pageClasses.0/@actions/
               @actions." + Integer.toString(numberOfActionsBeforeThis +
               rValue.size()) + "/@root");
45         Element attributesElement = new Element("attributes");
```

```
46          extractorElement.addContent(attributesElement);
47          attributesElement.setAttribute("name", "type");
48          attributesElement.setAttribute("value", "text");
49          attributesElement = new Element("attributes");
50          extractorElement.addContent(attributesElement);
51          attributesElement.setAttribute("name", "name");
52          attributesElement.setAttribute("value", getTestAction().getControl
                ().getId());
53          attributesElement = new Element("attributes");
54          extractorElement.addContent(attributesElement);
55          attributesElement.setAttribute("name", "text");
56          attributesElement.setAttribute("value", "");
57          attributesElement.setAttribute("nodetype", "INTERNAL");
58
59          Element rangeElement = new Element("range");
60          filtersElement.addContent(rangeElement);
61          rangeElement.setAttribute("toDirection", "BEGIN");
62
63          rValue.add(actionsElements);
64          // we are complete
65          return rValue;
66      }
67  }
```

## A.5.13 Class lixto.LixtoScript

Listing A.20: Automation Generator - Class lixto.LixtoScript

```
1  package com.lixto.abstractautomation.scriptgenerator.lixto;
2
3  import com.lixto.abstractautomation.scriptgenerator.ScriptGenerator;
4  import com.lixto.abstractautomation.scriptgenerator.testcase.TestAction;
5  import com.lixto.abstractautomation.scriptgenerator.testcase.TestCase;
6  import org.jdom.Comment;
7  import org.jdom.Document;
8  import org.jdom.Element;
9
10 import java.util.ArrayList;
11 import java.util.List;
12
13 /**
14  * Main Script-file
15  */
```

```
16 public class LixtoScript {
17     private final List<LixtoAction> actionList = new ArrayList<LixtoAction
           >();
18 
19 
20     public LixtoScript(final TestCase testCase) {
21         // we are immediately wrapping the testcase into lixtoactions
22         for (TestAction action : testCase.getActionList()) {
23             LixtoAction lixtoAction = LixtoAction.getLixtoAction(action);
24             this.actionList.add(lixtoAction);
25         }
26         // done
27     }
28 
29     /**
30      * returns an XML-representation of the desired result
31      */
32     public Document getDesiredResult() {
33         // the procedure is very similar to the lixto-model - except that
               we need only the results now
34         Element xmlRootElement = new Element("results");
35         XmlNamespaces.addNamespacesToElement(xmlRootElement);
36         for (LixtoAction action : actionList) {
37             // first of all, we add comments - just for debugging purposes
38             xmlRootElement.addContent(getTestActionComment(action));
39             // now check if the element is of type "verify"
40             if (action.getTestAction().getTestActionType() == TestAction.
                   TestActionType.verifyValue) {
41                 Element resultNode = new Element("ExpectedResult");
42                 resultNode.setAttribute("lixtoModelName", action.
                       getUniqueId());
43                 resultNode.setAttribute("expectedResult", action.
                       getTestAction().getParameterString());
44                 xmlRootElement.addContent(resultNode);
45             }
46         }
47         return new Document(xmlRootElement);
48     }
49 
50     /**
51      * returns an XML-representation of the lixto model file
52      */
53     public Document getLixtoModel() {
54         // the lixto model is pretty straight-forward. It goes through all
               test-actions and identifies the ones which are of type
```

```
            verification.
55          // all of those - in the right order - are simply representing one
                single element in the output XML.
56          Element xmlRootElement = new Element("OutputModel", XmlNamespaces.
                NS_MODEL);
57          XmlNamespaces.addNamespacesToElement(xmlRootElement);
58          xmlRootElement.setAttribute("version", "2.0", XmlNamespaces.NS_XMI)
                ;
59          Element rootElement = new Element("root");
60          rootElement.setAttribute("name", "TestcaseResult");
61          xmlRootElement.addContent(rootElement);
62          for (LixtoAction action : actionList) {
63              // first of all, we add comments - just for debugging purposes
64              rootElement.addContent(getTestActionComment(action));
65              // now check if the element is of type "verify"
66              if (action.getTestAction().getTestActionType() == TestAction.
                    TestActionType.verifyValue) {
67                  // it is verification! add output node
68                  Element content = new Element("children");
69                  content.setAttribute("name", action.getUniqueId());
70                  content.setAttribute("xsdSimpleType", "string");
71                  content.setAttribute("minOccurs", "1");
72                  content.setAttribute("maxOccurs", "1");
73                  rootElement.addContent(content);
74              }
75          }
76          // document is done
77          return new Document(xmlRootElement);
78      }
79
80      /**
81       * returns an XML-representation of the lixto wrapper file
82       */
83      public Document getLixtoWrapper() {
84          // the lixto wrapper has to iterate all actions and create the
                wrapper-script
85          Element rootElement = new Element("Navigation", XmlNamespaces.
                NS_NAVIGATION);
86          XmlNamespaces.addNamespacesToElement(rootElement);
87          rootElement.setAttribute("version", "2.0", XmlNamespaces.NS_XMI);
88          Element pageClassElement = new Element("pageClasses");
89          pageClassElement.setAttribute("name", "start");
90          rootElement.addContent(pageClassElement);
91          Element actionsElement = new Element("actions");
92          pageClassElement.addContent(actionsElement);
```

```
93          int numberOfVerifications = 0;
94          for (LixtoAction action : actionList) {
95              // now, for each action, get the list of lixto action elements
                    and just append them to the list. That's it, nothing else
96              actionsElement.addContent(getTestActionCommentForWrapper(action
                    ));
97              actionsElement.addContent(action.getLixtoXmlList(actionsElement
                    .getChildren().size(), numberOfVerifications));
98              // increment
99              if (action instanceof LixtoActionVerifyValue) {
100                 numberOfVerifications++;
101             }
102         }
103         // now, in addition, add reference to model
104         Element modelElement = new Element("outputmodel");
105         modelElement.setAttribute("href", ScriptGenerator.FILENAME_MODEL +
                "#/");
106         rootElement.addContent(modelElement);
107         // done
108         return new Document(rootElement);
109     }
110
111     private final static List<Comment> getTestActionComment(LixtoAction
            action) {
112         // create the action-node here
113         List<Comment> rValue = new ArrayList<Comment>();
114         rValue.add(new Comment("Mask: " + action.getTestAction().getControl
                ().getMask().getId()));
115         rValue.add(new Comment("Control: " + action.getTestAction().
                getControl().getId()));
116         rValue.add(new Comment("Testaction: " + action.getTestAction().
                getTestActionType().toString()));
117         rValue.add(new Comment("LixtoActionId: " + action.getUniqueId()));
118         // also add the
119         return rValue;
120     }
121
122     private final static Element getTestActionCommentForWrapper(LixtoAction
            action) {
123         // create the action-node here
124         String commentString = "Mask=" + action.getTestAction().getControl
                ().getMask().getId();
125         commentString += "/Control=" + action.getTestAction().getControl().
                getId();
```

```
126         commentString += "/Testaction=" + action.getTestAction().
                getTestActionType().toString();
127         commentString += "/LixtoActionId=" + action.getUniqueId();
128         Element actionsElement = new Element("actions");
129         actionsElement.setAttribute("type", "actions:Comment",
                XmlNamespaces.NS_XSI);
130         actionsElement.setAttribute("label", commentString);
131         return actionsElement;
132     }
133
134 }
```

### A.5.14 Class testcase.TestAction

Listing A.21: Automation Generator - Class testcase.TestAction

```
1  package com.lixto.abstractautomation.scriptgenerator.testcase;
2
3  import com.lixto.abstractautomation.scriptgenerator.aut.AutControl;
4
5  /**
6   * class representing one testaction
7   */
8  public class TestAction {
9      private final AutControl control;
10     private final TestActionType testActionType;
11     private final String parameterString;
12
13     public enum TestActionType {
14         invalid("invalid"),
15         setValue("setValue"),
16         verifyValue("verifyValue"),
17         push("push");
18
19         private final String stringValue;
20
21         private TestActionType(String stringValue) {
22             this.stringValue = stringValue;
23         }
24
25         final boolean isStringValue(String inputValue) {
26             return (inputValue.equals(this.stringValue));
27         }
```

```
28
29      }
30
31      public TestAction(AutControl control, String actionString, String
            parameterString) {
32          this.control = control;
33          this.parameterString = parameterString;
34          // set action type
35          TestActionType tmpVar = TestActionType.invalid;
36          for (TestActionType actionType : TestActionType.values()) {
37              if (actionType.isStringValue(actionString)) {
38                  tmpVar = actionType;
39                  break;
40              }
41          }
42          this.testActionType = tmpVar;
43      }
44
45      public TestActionType getTestActionType() {
46          return testActionType;
47      }
48
49      public String getParameterString() {
50          return parameterString;
51      }
52
53      public AutControl getControl() {
54          return control;
55      }
56  }
```

## A.5.15 Class testcase.TestCase

Listing A.22: Automation Generator - Class testcase.TestCase

```
1  package com.lixto.abstractautomation.scriptgenerator.testcase;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  /**
7   * Test case definition
8   */
```

```
9  public class TestCase {
10     private final List<TestAction> actionList = new ArrayList<TestAction>()
           ;
11
12     public void addTestAction(TestAction action) {
13         this.actionList.add(action);
14     }
15
16     public List<TestAction> getActionList() {
17         return actionList;
18     }
19 }
```

### A.5.16 Class ScriptGenerator

Listing A.23: Automation Generator - Class ScriptGenerator

```
1  package com.lixto.abstractautomation.scriptgenerator;
2
3  import com.lixto.abstractautomation.scriptgenerator.aut.
       ApplicationUnderTest;
4  import com.lixto.abstractautomation.scriptgenerator.aut.AutControl;
5  import com.lixto.abstractautomation.scriptgenerator.aut.AutMask;
6  import com.lixto.abstractautomation.scriptgenerator.lixto.LixtoScript;
7  import com.lixto.abstractautomation.scriptgenerator.testcase.TestAction;
8  import com.lixto.abstractautomation.scriptgenerator.testcase.TestCase;
9  import org.jdom.Document;
10 import org.jdom.Element;
11 import org.jdom.input.SAXBuilder;
12 import org.jdom.output.Format;
13 import org.jdom.output.XMLOutputter;
14
15 import java.io.File;
16 import java.io.FileOutputStream;
17 import java.io.IOException;
18 import java.util.List;
19
20 /**
21  * Script-Generator itself
22  */
23 public class ScriptGenerator {
24     private final File testcaseXml;
25     private final File applicationDefinitionXml;
```

```
26      private final File outputDirectory;
27      public static final String FILENAME_MODEL = "lixtoModel.lixmod";
28      private static final String FILENAME_WRAPPER = "lixtoWrapper.lixvw";
29      private static final String FILENAME_DESIREDRESULT = "
            desiredTestcaseResult.xml";
30
31      ScriptGenerator(File testcaseXml, File applicationDefinitionXml, File
            outputFile) {
32          this.testcaseXml = testcaseXml;
33          this.applicationDefinitionXml = applicationDefinitionXml;
34          this.outputDirectory = outputFile;
35      }
36
37      public void process() throws Exception {
38          final ApplicationUnderTest aut = getAut(applicationDefinitionXml);
39          final TestCase tc = getTestCase(testcaseXml, aut);
40          // now get the lixto-script
41          LixtoScript script = new LixtoScript(tc);
42          // get the XML-documents for their output
43          // Lixto Model
44          final File outputLixtoModelFile = new File(outputDirectory,
                FILENAME_MODEL);
45          final Document outputLixtoModel = script.getLixtoModel();
46          writeDocumentToFile(outputLixtoModel, outputLixtoModelFile);
47          // Lixto Wrapper
48          final File outputLixtoWrapperFile = new File(outputDirectory,
                FILENAME_WRAPPER);
49          final Document outputLixtoWrapper = script.getLixtoWrapper();
50          writeDocumentToFile(outputLixtoWrapper, outputLixtoWrapperFile);
51          // desired output as reference
52          final File outputLixtoDesiredResultFile = new File(outputDirectory,
                 FILENAME_DESIREDRESULT);
53          final Document outputLixtoDesiredResult = script.getDesiredResult()
                ;
54          writeDocumentToFile(outputLixtoDesiredResult,
                outputLixtoDesiredResultFile);
55      }
56
57      /**
58       * write an xml-document into a file
59       */
60      private final void writeDocumentToFile(Document xmlDocument, File
            outputFile) throws IOException {
61          // output doc
```

```
62          XMLOutputter outputter = new XMLOutputter(Format.getPrettyFormat())
                ;
63          outputter.output(xmlDocument, new FileOutputStream(outputFile));
64      }
65
66      private static ApplicationUnderTest getAut(File
            applicationDefinitionXml) throws Exception {
67          // process the input XML-files
68          SAXBuilder builder = new SAXBuilder();
69          Document xmlDoc = builder.build(applicationDefinitionXml);
70          // first of all, the application-definition has to be mapped into
                java objects.
71          Element autElement = xmlDoc.getRootElement();
72          ApplicationUnderTest rValue = new ApplicationUnderTest(autElement.
                getAttributeValue("name"));
73          // iterate all masks
74          for (Element maskElement : (List<Element>) autElement.getChildren()
                ) {
75              AutMask mask = new AutMask(maskElement.getAttributeValue("id"))
                    ;
76              rValue.addMask(mask);
77              // iterate all controls on the mask
78              for (Element controlElement : (List<Element>) maskElement.
                    getChildren()) {
79                  String type = getSubElementContent(controlElement, "type");
80                  String name = getSubElementContent(controlElement, "name");
81                  String identifierType = getSubElementContent(controlElement
                        , "identifiertype");
82                  String identifier = getSubElementContent(controlElement, "
                        identifier");
83                  AutControl control = new AutControl(mask, type, name,
                        identifierType, identifier);
84                  mask.addControl(control);
85              }
86          }
87          // done
88          return rValue;
89      }
90
91      private static TestCase getTestCase(File testCaseXml,
            ApplicationUnderTest aut) throws Exception {
92          // process the input XML-files
93          SAXBuilder builder = new SAXBuilder();
94          Document xmlDoc = builder.build(testCaseXml);
95          // first of all, the test-case has to be mapped into java objects
```

```
96          Element tcElement = xmlDoc.getRootElement();
97          TestCase rValue = new TestCase();
98          // a testcase has just a series of testactions - we "flatten" the
                mask-structure
99          for (Element maskElement : (List<Element>) tcElement.getChildren())
                 {
100             AutMask autMask = aut.getMask(maskElement.getAttributeValue("id
                    "));
101             for (Element actionElement : (List<Element>) maskElement.
                    getChildren()) {
102                 AutControl control = autMask.getControl(
                        getSubElementContent(actionElement, "Control"));
103                 String actionString = getSubElementContent(actionElement, "
                        Action");
104                 String parameterString = getSubElementContent(actionElement
                        , "Parameter");
105                 TestAction action = new TestAction(control, actionString,
                        parameterString);
106                 rValue.addTestAction(action);
107             }
108         }
109         return rValue;
110     }
111
112
113     /**
114      * get the text-content of the sub-element
115      */
116     private static final String getSubElementContent(Element parentElement,
             String subElementName) {
117         Element subElement = parentElement.getChild(subElementName);
118         if (subElement == null) {
119             // element does not exist
120             return null;
121         }
122         return subElement.getTextNormalize();
123     }
124 }
```

## A.5.17 Class StartClass

Listing A.24: Automation Generator - Class StartClass

```
1  package com.lixto.abstractautomation.scriptgenerator;
2
3  import java.io.File;
4  import java.util.logging.Logger;
5
6  /**
7   * start class - containing MAIN-method
8   */
9  public class StartClass {
10     private static final Logger logger = Logger.getLogger(StartClass.class.
           getName());
11
12     /**
13      * first argument must be the testcase to be executed. Second argument
            must be the application-definition. Third one is the output file
14      *
15      * @param args the command line arguments
16      */
17     public static void main(String[] args) {
18         if (args.length != 3) {
19             logger.severe("Invalid number of parameters");
20             System.exit(-1);
21         }
22
23         // get the files
24         final File testcaseXml = getFile(args[0], true);
25         final File applicationDefinitionXml = getFile(args[1], true);
26         final File outputDirectory = getFile(args[2], false);
27         outputDirectory.mkdirs();
28
29         // start conversion
30         ScriptGenerator generator = new ScriptGenerator(testcaseXml,
               applicationDefinitionXml, outputDirectory);
31         try {
32             generator.process();
33         } catch (Exception e) {
34             logger.severe(e.getClass() + ": " + e.getMessage());
35             e.printStackTrace();
36         }
37         // done
38     }
39
40     private static File getFile(String filename, boolean mustExist) {
41         File rValue = new File(filename);
42         if (rValue.exists() != mustExist) {
```

```
43              if (mustExist) {
44                  logger.severe("File does not exist: '" + filename + "'");
45              } else {
46                  logger.severe("File does already exist: '" + filename + "'"
                        );
47              }
48              System.exit(-1);
49          }
50          return rValue;
51      }
52
53  }
```

## A.6 Output of Lixto Source Generator

This chapter contains all sources which are created by the Automation Generator, to be executed within Lixto

### A.6.1 Lixto Wrapper

Listing A.25: Output for Testcase-Example - Lixto Wrapper

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <navigation:Navigation xmlns:navigation="navigation" xmlns:wrapper="wrapper
       " xmlns:model="model" xmlns:definitions="definitions" xmlns:actions="
       actions" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.
       org/2001/XMLSchema-instance" xmi:version="2.0">
3    <pageClasses name="start">
4      <actions>
5        <actions xsi:type="actions:Comment" label="Mask=Login/Control=
             fieldUsername/Testaction=setValue/LixtoActionId=
             TestActionForLixto1" />
6        <actions xsi:type="actions:MouseAction">
7          <events target="/html/body/form/table/tbody/tr[1]/td/input">
8            <keyflags />
9          </events>
10         <events target="/html/body/form/table/tbody/tr[1]/td/input" type="
               mouseup">
11           <keyflags />
12         </events>
13       </actions>
```

```
14      <actions xsi:type="actions:KeyAction">
15        <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
              "17">
16          <keyflags ctrl="true" />
17        </events>
18        <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
              "65">
19          <keyflags ctrl="true" />
20        </events>
21        <events focus="/html/body/form/table/tbody/tr[1]/td/input" type="
              keypress" charCode="97">
22          <keyflags ctrl="true" />
23        </events>
24        <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
              "65" type="keyup">
25          <keyflags ctrl="true" />
26        </events>
27        <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
              "17" type="keyup">
28          <keyflags ctrl="true" />
29        </events>
30        <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
              "83">
31          <keyflags shift="true" />
32        </events>
33        <events focus="/html/body/form/table/tbody/tr[1]/td/input" type="
              keypress" charCode="83">
34          <keyflags shift="true" />
35        </events>
36        <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
              "83" type="keyup">
37          <keyflags shift="true" />
38        </events>
39        <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
              "101">
40          <keyflags />
41        </events>
42        <events focus="/html/body/form/table/tbody/tr[1]/td/input" type="
              keypress" charCode="101">
43          <keyflags />
44        </events>
45        <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
              "101" type="keyup">
46          <keyflags />
47        </events>
```

```
48        <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
              "109">
49          <keyflags />
50        </events>
51        <events focus="/html/body/form/table/tbody/tr[1]/td/input" type="
              keypress" charCode="109">
52          <keyflags />
53        </events>
54        <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
              "109" type="keyup">
55          <keyflags />
56        </events>
57        <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
              "116">
58          <keyflags />
59        </events>
60        <events focus="/html/body/form/table/tbody/tr[1]/td/input" type="
              keypress" charCode="116">
61          <keyflags />
62        </events>
63        <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
              "116" type="keyup">
64          <keyflags />
65        </events>
66        <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
              "117">
67          <keyflags />
68        </events>
69        <events focus="/html/body/form/table/tbody/tr[1]/td/input" type="
              keypress" charCode="117">
70          <keyflags />
71        </events>
72        <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
              "117" type="keyup">
73          <keyflags />
74        </events>
75        <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
              "114">
76          <keyflags />
77        </events>
78        <events focus="/html/body/form/table/tbody/tr[1]/td/input" type="
              keypress" charCode="114">
79          <keyflags />
80        </events>
```

```
81        <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
              "114" type="keyup">
82          <keyflags />
83        </events>
84        <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
              "115">
85          <keyflags />
86        </events>
87        <events focus="/html/body/form/table/tbody/tr[1]/td/input" type="
              keypress" charCode="115">
88          <keyflags />
89        </events>
90        <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
              "115" type="keyup">
91          <keyflags />
92        </events>
93      </actions>
94      <actions xsi:type="actions:Comment" label="Mask=Login/Control=
            fieldPassword/Testaction=setValue/LixtoActionId=
            TestActionForLixto2" />
95      <actions xsi:type="actions:MouseAction">
96        <events target="/html/body/form/table/tbody/tr[2]/td/input">
97          <keyflags />
98        </events>
99        <events target="/html/body/form/table/tbody/tr[2]/td/input" type="
              mouseup">
100         <keyflags />
101       </events>
102     </actions>
103     <actions xsi:type="actions:KeyAction">
104       <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
              "17">
105         <keyflags ctrl="true" />
106       </events>
107       <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
              "65">
108         <keyflags ctrl="true" />
109       </events>
110       <events focus="/html/body/form/table/tbody/tr[2]/td/input" type="
              keypress" charCode="97">
111         <keyflags ctrl="true" />
112       </events>
113       <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
              "65" type="keyup">
114         <keyflags ctrl="true" />
```

```
115         </events>
116         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "17" type="keyup">
117           <keyflags ctrl="true" />
118         </events>
119         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "80">
120           <keyflags shift="true" />
121         </events>
122         <events focus="/html/body/form/table/tbody/tr[2]/td/input" type="
                keypress" charCode="80">
123           <keyflags shift="true" />
124         </events>
125         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "80" type="keyup">
126           <keyflags shift="true" />
127         </events>
128         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "97">
129           <keyflags />
130         </events>
131         <events focus="/html/body/form/table/tbody/tr[2]/td/input" type="
                keypress" charCode="97">
132           <keyflags />
133         </events>
134         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "97" type="keyup">
135           <keyflags />
136         </events>
137         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "115">
138           <keyflags />
139         </events>
140         <events focus="/html/body/form/table/tbody/tr[2]/td/input" type="
                keypress" charCode="115">
141           <keyflags />
142         </events>
143         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "115" type="keyup">
144           <keyflags />
145         </events>
146         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "115">
147           <keyflags />
148         </events>
```

```
149         <events focus="/html/body/form/table/tbody/tr[2]/td/input" type="
                keypress" charCode="115">
150           <keyflags />
151         </events>
152         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "115" type="keyup">
153           <keyflags />
154         </events>
155         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "48">
156           <keyflags />
157         </events>
158         <events focus="/html/body/form/table/tbody/tr[2]/td/input" type="
                keypress" charCode="48">
159           <keyflags />
160         </events>
161         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "48" type="keyup">
162           <keyflags />
163         </events>
164         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "49">
165           <keyflags />
166         </events>
167         <events focus="/html/body/form/table/tbody/tr[2]/td/input" type="
                keypress" charCode="49">
168           <keyflags />
169         </events>
170         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "49" type="keyup">
171           <keyflags />
172         </events>
173       </actions>
174       <actions xsi:type="actions:Comment" label="Mask=Login/Control=
              LoginButton/Testaction=push/LixtoActionId=TestActionForLixto3" />
175       <actions xsi:type="actions:MouseAction">
176         <events target="/html/body/form/table/tbody/tr[3]/td/input">
177           <keyflags />
178         </events>
179         <events target="/html/body/form/table/tbody/tr[3]/td/input" type="
                mouseup">
180           <keyflags />
181         </events>
182       </actions>
```

```
183       <actions xsi:type="actions:Comment" label="Mask=PersonList/Control=
              PersonLastNameLabel01/Testaction=verifyValue/LixtoActionId=
              TestActionForLixto4" />
184       <actions xsi:type="actions:MouseAction">
185         <events target="/html/body/table/tbody/tr[2]/td[2]">
186           <keyflags />
187         </events>
188         <events target="/html/body/table/tbody/tr[2]/td[2]" type="mouseup">
189           <keyflags />
190         </events>
191       </actions>
192       <actions xsi:type="actions:WrapAction">
193         <root xsi:type="wrapper:SimpleModelPattern">
194           <outputSchemaNode xsi:type="model:OutputElement" href="lixtoModel
                  .lixmod#//@root/@children.0" />
195           <filters occurenceType="SINGLE">
196             <extractor xsi:type="definitions:PathDefinition" pathType="
                    XPATH" target="/html/body/table/tbody/tr[2]/td[2]">
197               <input xsi:type="definitions:PatternBinding" pattern="//
                      @pageClasses.0/@actions/@actions.10/@root" />
198               <attributes name="type" value="text" />
199               <attributes name="name" value="PersonLastNameLabel01" />
200               <attributes name="text" value="" nodetype="INTERNAL" />
201             </extractor>
202             <range toDirection="BEGIN" />
203           </filters>
204         </root>
205       </actions>
206       <actions xsi:type="actions:Comment" label="Mask=PersonList/Control=
              buttonEditPerson01/Testaction=push/LixtoActionId=
              TestActionForLixto5" />
207       <actions xsi:type="actions:MouseAction">
208         <events target="/html/body/table/tbody/tr[2]/td[4]/form/input[3]">
209           <keyflags />
210         </events>
211         <events target="/html/body/table/tbody/tr[2]/td[4]/form/input[3]"
                type="mouseup">
212           <keyflags />
213         </events>
214       </actions>
215       <actions xsi:type="actions:Comment" label="Mask=PersonEdit/Control=
              fieldLastName/Testaction=setValue/LixtoActionId=
              TestActionForLixto6" />
216       <actions xsi:type="actions:MouseAction">
217         <events target="/html/body/form/table/tbody/tr[2]/td/input">
```

```
218           <keyflags />
219         </events>
220         <events target="/html/body/form/table/tbody/tr[2]/td/input" type="
                mouseup">
221           <keyflags />
222         </events>
223       </actions>
224       <actions xsi:type="actions:KeyAction">
225         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "17">
226           <keyflags ctrl="true" />
227         </events>
228         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "65">
229           <keyflags ctrl="true" />
230         </events>
231         <events focus="/html/body/form/table/tbody/tr[2]/td/input" type="
                keypress" charCode="97">
232           <keyflags ctrl="true" />
233         </events>
234         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "65" type="keyup">
235           <keyflags ctrl="true" />
236         </events>
237         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "17" type="keyup">
238           <keyflags ctrl="true" />
239         </events>
240         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "65">
241           <keyflags shift="true" />
242         </events>
243         <events focus="/html/body/form/table/tbody/tr[2]/td/input" type="
                keypress" charCode="65">
244           <keyflags shift="true" />
245         </events>
246         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "65" type="keyup">
247           <keyflags shift="true" />
248         </events>
249         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "98">
250           <keyflags />
251         </events>
```

```
252         <events focus="/html/body/form/table/tbody/tr[2]/td/input" type="
                keypress" charCode="98">
253           <keyflags />
254         </events>
255         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "98" type="keyup">
256           <keyflags />
257         </events>
258         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "99">
259           <keyflags />
260         </events>
261         <events focus="/html/body/form/table/tbody/tr[2]/td/input" type="
                keypress" charCode="99">
262           <keyflags />
263         </events>
264         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "99" type="keyup">
265           <keyflags />
266         </events>
267         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "100">
268           <keyflags />
269         </events>
270         <events focus="/html/body/form/table/tbody/tr[2]/td/input" type="
                keypress" charCode="100">
271           <keyflags />
272         </events>
273         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "100" type="keyup">
274           <keyflags />
275         </events>
276         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "101">
277           <keyflags />
278         </events>
279         <events focus="/html/body/form/table/tbody/tr[2]/td/input" type="
                keypress" charCode="101">
280           <keyflags />
281         </events>
282         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "101" type="keyup">
283           <keyflags />
284         </events>
285       </actions>
```

```
286       <actions xsi:type="actions:Comment" label="Mask=PersonEdit/Control=
              buttonSave/Testaction=push/LixtoActionId=TestActionForLixto7" />
287       <actions xsi:type="actions:MouseAction">
288         <events target="/html/body/form/input[3]">
289           <keyflags />
290         </events>
291         <events target="/html/body/form/input[3]" type="mouseup">
292           <keyflags />
293         </events>
294       </actions>
295       <actions xsi:type="actions:Comment" label="Mask=PersonList/Control=
              buttonLogout/Testaction=push/LixtoActionId=TestActionForLixto8" /
              >
296       <actions xsi:type="actions:MouseAction">
297         <events target="/html/body/a">
298           <keyflags />
299         </events>
300         <events target="/html/body/a" type="mouseup">
301           <keyflags />
302         </events>
303       </actions>
304       <actions xsi:type="actions:Comment" label="Mask=Login/Control=
              fieldUsername/Testaction=setValue/LixtoActionId=
              TestActionForLixto9" />
305       <actions xsi:type="actions:MouseAction">
306         <events target="/html/body/form/table/tbody/tr[1]/td/input">
307           <keyflags />
308         </events>
309         <events target="/html/body/form/table/tbody/tr[1]/td/input" type="
                mouseup">
310           <keyflags />
311         </events>
312       </actions>
313       <actions xsi:type="actions:KeyAction">
314         <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
                "17">
315           <keyflags ctrl="true" />
316         </events>
317         <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
                "65">
318           <keyflags ctrl="true" />
319         </events>
320         <events focus="/html/body/form/table/tbody/tr[1]/td/input" type="
                keypress" charCode="97">
321           <keyflags ctrl="true" />
```

```
322         </events>
323         <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
                "65" type="keyup">
324           <keyflags ctrl="true" />
325         </events>
326         <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
                "17" type="keyup">
327           <keyflags ctrl="true" />
328         </events>
329         <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
                "65">
330           <keyflags shift="true" />
331         </events>
332         <events focus="/html/body/form/table/tbody/tr[1]/td/input" type="
                keypress" charCode="65">
333           <keyflags shift="true" />
334         </events>
335         <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
                "65" type="keyup">
336           <keyflags shift="true" />
337         </events>
338         <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
                "98">
339           <keyflags />
340         </events>
341         <events focus="/html/body/form/table/tbody/tr[1]/td/input" type="
                keypress" charCode="98">
342           <keyflags />
343         </events>
344         <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
                "98" type="keyup">
345           <keyflags />
346         </events>
347         <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
                "99">
348           <keyflags />
349         </events>
350         <events focus="/html/body/form/table/tbody/tr[1]/td/input" type="
                keypress" charCode="99">
351           <keyflags />
352         </events>
353         <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
                "99" type="keyup">
354           <keyflags />
355         </events>
```

```
356         <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
                "100">
357           <keyflags />
358         </events>
359         <events focus="/html/body/form/table/tbody/tr[1]/td/input" type="
                keypress" charCode="100">
360           <keyflags />
361         </events>
362         <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
                "100" type="keyup">
363           <keyflags />
364         </events>
365         <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
                "101">
366           <keyflags />
367         </events>
368         <events focus="/html/body/form/table/tbody/tr[1]/td/input" type="
                keypress" charCode="101">
369           <keyflags />
370         </events>
371         <events focus="/html/body/form/table/tbody/tr[1]/td/input" keyCode=
                "101" type="keyup">
372           <keyflags />
373         </events>
374       </actions>
375       <actions xsi:type="actions:Comment" label="Mask=Login/Control=
              fieldPassword/Testaction=setValue/LixtoActionId=
              TestActionForLixto10" />
376       <actions xsi:type="actions:MouseAction">
377         <events target="/html/body/form/table/tbody/tr[2]/td/input">
378           <keyflags />
379         </events>
380         <events target="/html/body/form/table/tbody/tr[2]/td/input" type="
                mouseup">
381           <keyflags />
382         </events>
383       </actions>
384       <actions xsi:type="actions:KeyAction">
385         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "17">
386           <keyflags ctrl="true" />
387         </events>
388         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "65">
389           <keyflags ctrl="true" />
```

```
390         </events>
391         <events focus="/html/body/form/table/tbody/tr[2]/td/input" type="
                keypress" charCode="97">
392           <keyflags ctrl="true" />
393         </events>
394         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "65" type="keyup">
395           <keyflags ctrl="true" />
396         </events>
397         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "17" type="keyup">
398           <keyflags ctrl="true" />
399         </events>
400         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "80">
401           <keyflags shift="true" />
402         </events>
403         <events focus="/html/body/form/table/tbody/tr[2]/td/input" type="
                keypress" charCode="80">
404           <keyflags shift="true" />
405         </events>
406         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "80" type="keyup">
407           <keyflags shift="true" />
408         </events>
409         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "97">
410           <keyflags />
411         </events>
412         <events focus="/html/body/form/table/tbody/tr[2]/td/input" type="
                keypress" charCode="97">
413           <keyflags />
414         </events>
415         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "97" type="keyup">
416           <keyflags />
417         </events>
418         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "115">
419           <keyflags />
420         </events>
421         <events focus="/html/body/form/table/tbody/tr[2]/td/input" type="
                keypress" charCode="115">
422           <keyflags />
423         </events>
```

```
424         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "115" type="keyup">
425           <keyflags />
426         </events>
427         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "115">
428           <keyflags />
429         </events>
430         <events focus="/html/body/form/table/tbody/tr[2]/td/input" type="
                keypress" charCode="115">
431           <keyflags />
432         </events>
433         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "115" type="keyup">
434           <keyflags />
435         </events>
436         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "48">
437           <keyflags />
438         </events>
439         <events focus="/html/body/form/table/tbody/tr[2]/td/input" type="
                keypress" charCode="48">
440           <keyflags />
441         </events>
442         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "48" type="keyup">
443           <keyflags />
444         </events>
445         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "49">
446           <keyflags />
447         </events>
448         <events focus="/html/body/form/table/tbody/tr[2]/td/input" type="
                keypress" charCode="49">
449           <keyflags />
450         </events>
451         <events focus="/html/body/form/table/tbody/tr[2]/td/input" keyCode=
                "49" type="keyup">
452           <keyflags />
453         </events>
454       </actions>
455       <actions xsi:type="actions:Comment" label="Mask=Login/Control=
              LoginButton/Testaction=push/LixtoActionId=TestActionForLixto11" /
              >
456       <actions xsi:type="actions:MouseAction">
```

```
457         <events target="/html/body/form/table/tbody/tr[3]/td/input">
458           <keyflags />
459         </events>
460         <events target="/html/body/form/table/tbody/tr[3]/td/input" type="
                mouseup">
461           <keyflags />
462         </events>
463       </actions>
464       <actions xsi:type="actions:Comment" label="Mask=PersonList/Control=
              PersonLastNameLabel01/Testaction=verifyValue/LixtoActionId=
              TestActionForLixto12" />
465       <actions xsi:type="actions:MouseAction">
466         <events target="/html/body/table/tbody/tr[2]/td[2]">
467           <keyflags />
468         </events>
469         <events target="/html/body/table/tbody/tr[2]/td[2]" type="mouseup">
470           <keyflags />
471         </events>
472       </actions>
473       <actions xsi:type="actions:WrapAction">
474         <root xsi:type="wrapper:SimpleModelPattern">
475           <outputSchemaNode xsi:type="model:OutputElement" href="lixtoModel
                  .lixmod#//@root/@children.1" />
476           <filters occurenceType="SINGLE">
477             <extractor xsi:type="definitions:PathDefinition" pathType="
                    XPATH" target="/html/body/table/tbody/tr[2]/td[2]">
478               <input xsi:type="definitions:PatternBinding" pattern="//
                      @pageClasses.0/@actions/@actions.30/@root" />
479               <attributes name="type" value="text" />
480               <attributes name="name" value="PersonLastNameLabel01" />
481               <attributes name="text" value="" nodetype="INTERNAL" />
482             </extractor>
483             <range toDirection="BEGIN" />
484           </filters>
485         </root>
486       </actions>
487     </actions>
488   </pageClasses>
489   <outputmodel href="lixtoModel.lixmod#/" />
490 </navigation:Navigation>
```

### A.6.2 Lixto Data Model

Listing A.26: Output for Testcase-Example - Lixto Data Model

```
<?xml version="1.0" encoding="UTF-8"?>
<model:OutputModel xmlns:model="model" xmlns:navigation="navigation"
    xmlns:wrapper="wrapper" xmlns:definitions="definitions" xmlns:actions="
    actions" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.
    org/2001/XMLSchema-instance" xmi:version="2.0">
  <root name="TestcaseResult">
    <!--Mask: Login-->
    <!--Control: fieldUsername-->
    <!--Testaction: setValue-->
    <!--LixtoActionId: TestActionForLixto1-->
    <!--Mask: Login-->
    <!--Control: fieldPassword-->
    <!--Testaction: setValue-->
    <!--LixtoActionId: TestActionForLixto2-->
    <!--Mask: Login-->
    <!--Control: LoginButton-->
    <!--Testaction: push-->
    <!--LixtoActionId: TestActionForLixto3-->
    <!--Mask: PersonList-->
    <!--Control: PersonLastNameLabel01-->
    <!--Testaction: verifyValue-->
    <!--LixtoActionId: TestActionForLixto4-->
    <children name="TestActionForLixto4" xsdSimpleType="string" minOccurs="
        1" maxOccurs="1" />
    <!--Mask: PersonList-->
    <!--Control: buttonEditPerson01-->
    <!--Testaction: push-->
    <!--LixtoActionId: TestActionForLixto5-->
    <!--Mask: PersonEdit-->
    <!--Control: fieldLastName-->
    <!--Testaction: setValue-->
    <!--LixtoActionId: TestActionForLixto6-->
    <!--Mask: PersonEdit-->
    <!--Control: buttonSave-->
    <!--Testaction: push-->
    <!--LixtoActionId: TestActionForLixto7-->
    <!--Mask: PersonList-->
    <!--Control: buttonLogout-->
    <!--Testaction: push-->
    <!--LixtoActionId: TestActionForLixto8-->
    <!--Mask: Login-->
    <!--Control: fieldUsername-->
    <!--Testaction: setValue-->
```

```
40      <!--LixtoActionId: TestActionForLixto9-->
41      <!--Mask: Login-->
42      <!--Control: fieldPassword-->
43      <!--Testaction: setValue-->
44      <!--LixtoActionId: TestActionForLixto10-->
45      <!--Mask: Login-->
46      <!--Control: LoginButton-->
47      <!--Testaction: push-->
48      <!--LixtoActionId: TestActionForLixto11-->
49      <!--Mask: PersonList-->
50      <!--Control: PersonLastNameLabel01-->
51      <!--Testaction: verifyValue-->
52      <!--LixtoActionId: TestActionForLixto12-->
53      <children name="TestActionForLixto12" xsdSimpleType="string" minOccurs=
            "1" maxOccurs="1" />
54    </root>
55  </model:OutputModel>
```

## A.6.3 Expected Results

Listing A.27: Output for Testcase-Example - Expected Result

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <results xmlns:navigation="navigation" xmlns:wrapper="wrapper" xmlns:model=
       "model" xmlns:definitions="definitions" xmlns:actions="actions"
       xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/
       XMLSchema-instance">
3    <!--Mask: Login-->
4    <!--Control: fieldUsername-->
5    <!--Testaction: setValue-->
6    <!--LixtoActionId: TestActionForLixto1-->
7    <!--Mask: Login-->
8    <!--Control: fieldPassword-->
9    <!--Testaction: setValue-->
10   <!--LixtoActionId: TestActionForLixto2-->
11   <!--Mask: Login-->
12   <!--Control: LoginButton-->
13   <!--Testaction: push-->
14   <!--LixtoActionId: TestActionForLixto3-->
15   <!--Mask: PersonList-->
16   <!--Control: PersonLastNameLabel01-->
17   <!--Testaction: verifyValue-->
18   <!--LixtoActionId: TestActionForLixto4-->
```

```
19   <ExpectedResult lixtoModelName="TestActionForLixto4" expectedResult="
         Semturs" />
20   <!--Mask: PersonList-->
21   <!--Control: buttonEditPerson01-->
22   <!--Testaction: push-->
23   <!--LixtoActionId: TestActionForLixto5-->
24   <!--Mask: PersonEdit-->
25   <!--Control: fieldLastName-->
26   <!--Testaction: setValue-->
27   <!--LixtoActionId: TestActionForLixto6-->
28   <!--Mask: PersonEdit-->
29   <!--Control: buttonSave-->
30   <!--Testaction: push-->
31   <!--LixtoActionId: TestActionForLixto7-->
32   <!--Mask: PersonList-->
33   <!--Control: buttonLogout-->
34   <!--Testaction: push-->
35   <!--LixtoActionId: TestActionForLixto8-->
36   <!--Mask: Login-->
37   <!--Control: fieldUsername-->
38   <!--Testaction: setValue-->
39   <!--LixtoActionId: TestActionForLixto9-->
40   <!--Mask: Login-->
41   <!--Control: fieldPassword-->
42   <!--Testaction: setValue-->
43   <!--LixtoActionId: TestActionForLixto10-->
44   <!--Mask: Login-->
45   <!--Control: LoginButton-->
46   <!--Testaction: push-->
47   <!--LixtoActionId: TestActionForLixto11-->
48   <!--Mask: PersonList-->
49   <!--Control: PersonLastNameLabel01-->
50   <!--Testaction: verifyValue-->
51   <!--LixtoActionId: TestActionForLixto12-->
52   <ExpectedResult lixtoModelName="TestActionForLixto12" expectedResult="
         Abcde" />
53 </results>
```

### A.6.4 Real Result by Lixto

Listing A.28: Output for Testcase-Example - Real Result

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

```
<lixto:documents xmlns:lixto="http://www.lixto.com/navigation">
   <lixto:document lixto:uri="http://localhost:8080/Start.jsp"/>
   <lixto:document lixto:uri="http://localhost:8080/Start.jsp">
      <TestActionForLixto4>Semturs</TestActionForLixto4>
   </lixto:document>
   <lixto:document lixto:uri="http://localhost:8080/Start.jsp?action=
       editPerson&amp;PersonLastName=Semturs"/>
   <lixto:document lixto:uri="http://localhost:8080/Start.jsp"/>
   <lixto:document lixto:uri="http://localhost:8080/Start.jsp"/>
   <lixto:document lixto:uri="http://localhost:8080/Start.jsp">
      <TestActionForLixto12>Abcde</TestActionForLixto12>
   </lixto:document>
</lixto:documents>
```

# Appendix B

# Bibliography


[BCL05] Robert Baumgartner, Michal Ceresna, and Gerald Ledermüller. Deep web navigation in web data extraction. In *CIMCA '05: Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce Vol-2 (CIMCA-IAWTIC'06)*, pages 698–703, Washington, DC, USA, 2005. IEEE Computer Society. 20, 64

[BJPW01] Hans Buwalda, Dennis Janssen, Iris Pinkster, and Paul Watters. *Integrated Test Design and Automation: Using the Testframe Method*. Addison-Wesley Professional, 2001. 15

[CJ02] Rick D. Craig and Stefan P. Jaskiel. *Systematic Software Testing*. Artech House Computer Library, 2002. 17

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFC 2817. 31, 43, 48

[FHBH+99] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617 (Draft Standard), June 1999. 45

[Gmb06] Lixto Software GmbH. *Lixto Visual Developer 2006 User Manual*. Lixto Software GmbH, 2006. 24

[Har00] Hans Hartmann. Archimedes - a pattern used in system testing. USA-PLoP, 2000. 26

[Hun02] Craig Hunt. *TCP/IP Network Administration*. O'Reilly & Associates, Inc., 103a Morris Street, Sebastopol, CA 95472, USA, Tel: +1 707 829 0515, and

90 Sherman Street, Cambridge, MA 02140, USA, Tel: +1 617 354 5800, third edition, 2002. 29

[Hut03] Marnie L. Hutcheson. *Software Testing Fundamentals : Methods and Metrics*. Wiley, 2003. 17

[Kan88] Cem Kaner. *Testing Computer Software*. TAB Books, Blue Ridge Summit, PA, USA, 1988. 2

[KM00] D. Kristol and L. Montulli. HTTP State Management Mechanism. RFC 2965 (Proposed Standard), October 2000. 44

[Lut03] Jürgen Lutz. Softwaretest. `http://www.objectarchitects.de/tu2002/slides2003/(99)Softwaretest.pdf`, 2003. 27

[Moc87a] P.V. Mockapetris. Domain names - concepts and facilities. RFC 1034 (Standard), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035. 30

[Moc87b] P.V. Mockapetris. Domain names - implementation and specification. RFC 1035 (Standard), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 2137, 2845, 3425, 3658, 4035, 4033. 30

[Res00] E. Rescorla. HTTP Over TLS. RFC 2818 (Informational), May 2000. 31

[RHJ99] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.01 Specification, 24. December 1999. 32, 33

[Wik06] Wikipedia. Dynamic html. `http://en.wikipedia.org/w/index.php?title=Dynamic_HTML&oldid=45153387`, 2006. 36

# Glossary

| Notation | Description |
|---|---|
| Abstract Automation | Methodology for Test Automation. Based on an abstract model of the application under test, tests can be specified and are executed based on an once implemented automation. |
| Ajax | Concept for asynchronous communication to the server via JavaScript. Allows transport of data only, without sending layout information again, and without blocking the user. |
| API | Application Programming Interface. Internal technical interfaces for communication between components. |
| Basic Authentication | Authentication Scheme used in HTTP Protocol. Provides user name and password on each request |
| Beta-Release | Pre-Release of an application to an exclusive range of persons. Used for passing the initial testing, to en widen the user spectrum and client variety. |
| Capture'n'Replay | Methodology for Test Automation. Captures user behavior and replays it, similar to the usage of macro recording. |
| Comet | Event-triggered implementation of Ajax |
| DNS | Domain Name System. Decentralised database for resolving internet-addresses to their numeric representation. |
| DOM | Document Object Model. A tree-style representation of the html-document. |

| Notation | Description |
|---|---|
| Dynamic HTML | Summary term for all client-side manipulations of the user-interface, without backend interaction. |
| Fail Result | Possible Result of one execution of a test case. Test case failed, application has a problem. |
| GUI-Map | Feature provided by many test automation products. Enables a technical export of the masks of the application under test. |
| HTML | Hyper Text Markup Language. Descriptive language for describing document structure and layout. |
| HTTP | Protocol on the Internet, used for sending and receiving HTML pages or similar web content. |
| HTTP Cookie | Unique identifier for a user connection. Sent by the web-browser on each request. |
| Inconclusive Result | Possible Result of one execution of a test case. Test case could not be executed. |
| Instant Messaging | Online and live communication between persons. Also known as chat, instant messaging also provides the online status of a list of other persons |
| IP-Address | Unique Identifier for a server or client within the internet. |
| IP-Protocol | Main Protocol in use on the Internet. |
| Java | Programming Language for Object-Oriented Programming. |
| Java Applet | Embedded Application within a webpage, implemented in Java. |
| JavaScript | Script Language. Allows modifications to a web page offline, and background-communication to the server. |
| JVM | Java Virtual Machine. Invokes Java Applets within the context of the web-browser. |

| Notation | Description |
|---|---|
| MacroMedia Flash | Embedded Application within a web page, implemented in MacroMedia Flash Format. Very popular for advertisement and other animated, interactive areas of interest |
| Mercury WinRunner | Test Automation Software, developed by Mercury Software. |
| MFC | Microsoft Foundation Classes. Commonly used as a description for native Windows-Based applications |
| Microsoft ActiveX | Embedded Application within a web page, implemented in an ActiveX compatible programming language, for instance Visual Basic or Visual C++. |
| Microsoft Windows | Operating System Family by Microsoft. |
| Pass Result | Possible Result of one execution of a test case. Test case executed successfully. |
| PHP | Script Language, executed within the context of a web-server. It is used for serving dynamic webpages |
| QAS | Quality Assurance Studio. Software suite for managing software test. |
| QAS.TCS | Test Case Studio. Supports the concept of Abstract Automation. |
| SQL | Structured Query Language. Used for defining queries and data manipulations on relational structured databases. |
| Test Result | Result of the execution of a test case. |
| URL | Uniform Resource Locator. Standardized Format for referring a resource on the internet (e.g. an image) |

| Notation | Description |
|---|---|
| X-Path | Unique key to identify a component within a web-page, based on the DOM of the web page |
| XML-Request | Pure data request from Browser to Web-Server. Often used together with AJAX-Technologies. |

# Index