POS Performance Testing Proof of Concept

31/08/2011

Document Authorisation

| Document Details | |
| --- | --- |
| Document Title | Performance Testing Proof of Concept |
| Version | Draft 1 |
| Date | 31 August 2011 |
| File Location |  |
| Author | Andrew Gray |
| Quality Reviewed | James Carvosso, <date> |

***Printed copies of this document are uncontrolled.***

Revision History

| Date | Author | Version | Revision Notes |
| --- | --- | --- | --- |
| 19/08/2011 | Andrew Gray | 0.1 | Initial Version |
|  |  |  |  |

This document has been prepared by:

Oakton Services Pty Ltd

ABN 31 100 103 268

Level 19, 303 Collins Street

Melbourne, Victoria 3000

Australia

Tel: (03) 9617 0200

Fax: (03) 9621 1951

[1 Introduction 5](#_Toc384158232)

[1.1 Background 5](#_Toc384158233)

[1.2 Purpose 5](#_Toc384158234)

[1.3 Audience 5](#_Toc384158235)

[1.4 Approach 5](#_Toc384158236)

[2 High Level Requirements 6](#_Toc384158237)

[2.1 Automated Testing 6](#_Toc384158238)

[2.2 Visual Presentation 6](#_Toc384158239)

[2.3 Ease of Use 6](#_Toc384158240)

[3 Solution Architecture 7](#_Toc384158241)

[4 Explanation of Solution Architecture 8](#_Toc384158242)

[4.1 Background of POS / BOS Interaction 8](#_Toc384158243)

[4.2 Testing Process 8](#_Toc384158244)

[5 Technical Architecture 9](#_Toc384158245)

[6 Explanation of Technical Architecture 10](#_Toc384158246)

[6.1 Testing Process 10](#_Toc384158247)

[6.2 Software Components used in Solution 11](#_Toc384158248)

[6.3 Development Tools Used 11](#_Toc384158249)

[6.4 Source Code Control 11](#_Toc384158250)

[7 Limitations and Constraints 12](#_Toc384158251)

[7.1 Lack of UI Automation Framework support in POS application 12](#_Toc384158252)

[7.2 Lack of Interbase DB driver 12](#_Toc384158253)

[7.3 Test set tied to given POS setup 12](#_Toc384158254)

[8 Future Opportunities 13](#_Toc384158255)

[8.1 Write more tests to provide coverage 13](#_Toc384158256)

[8.2 Make robotic user smarter 13](#_Toc384158257)

[8.3 Test more than transactions 13](#_Toc384158258)

[8.4 Potential reuse 13](#_Toc384158259)

[8.5 Extend Jenkins NUnit Plugin 13](#_Toc384158260)

[8.6 Require Retalix to support UI Automation Framework 14](#_Toc384158261)

[8.7 Purchase Interbase DB driver 14](#_Toc384158262)

[9 Appendix A – Configuration Settings 15](#_Toc384158263)

[9.1 Remote Services 15](#_Toc384158264)

[9.2 Logger 16](#_Toc384158265)

[9.3 File Monitor 17](#_Toc384158266)

[9.4 POS Tests 17](#_Toc384158267)

[10 Appendix B – Handover File 18](#_Toc384158268)

[10.1 Layout 18](#_Toc384158269)

[10.2 SQL Template Filename 18](#_Toc384158270)

[10.3 SQL File 18](#_Toc384158271)

[10.4 Product Name 18](#_Toc384158272)

[10.5 Example Contents 18](#_Toc384158273)

[11 Appendix C – File Monitor 19](#_Toc384158274)

[11.1 Screenshot 19](#_Toc384158275)

[11.2 Usage Instructions 19](#_Toc384158276)

# Introduction

## Background

The business lacked the capability to validate the performance of a release of the POS received from the vendor (Retalix) before accepting it.

Oakton provided technology know-how to build this capability within the limitations presented by the vendor product in its current state.

## Purpose

To document the outcome of the Performance Testing Proof of Concept

## Audience

The audience of this document are BP IT&S Personnel

## Approach

Undertake work on a Proof of Concept to automate regression testing of the Retalix POS. Functional testing is performed on the latest release from Retalix by the Global Testing team. The focus of this Proof of Concept is testing of the next to last release focusing on the performance of the POS.

# High Level Requirements

## Automated Testing

The testing process needs to be automated.

## Visual Presentation

The test results need to be displayed in a graphical form for management purposes and presentation to external vendors

## Ease of Use

The solution should be easy to use

# Solution Architecture



# Explanation of Solution Architecture

## Background of POS / BOS Interaction

When a transaction is completed on the POS it initially writes the information to a file stored on the local file system. Within seconds this same information is committed to an Interbase 7.5 database that resides on the BOS.

Due to limitations and constraints (see section 4 for more information), it is from this Interbase database that the Performance Tester will assert whether the test passed or failed.

## Testing Process

### Run Test

The first step in the testing process involves the Performance Tester robotically driving the POS as a human user would to complete a transaction. (E.g. Sell a Mars Bar).

### Write Transaction Details to BOS DB

As a consequence of the Performance Tester completing a transaction, the transaction details are written to the BOS DB residing on the BOS server.

### Assert Transaction Details

The Performance Tester retrieves the transaction details from the BOS DB and compares them against what is expected. How it actually does this is expanded on in section 6 below.

# Technical Architecture

# Explanation of Technical Architecture

## Testing Process

### Run Test

The Jenkins Continuous Integration Server is used to provide a runtime environment for the build steps and provide a dashboard environment for the presentation of the test results.

The Jenkins Service must be set to “Interact with the desktop” to interact with the POS.

The BPStore user can be logged in as long as the GSS\CommonScripts\ALL-System-SecurityLockDown-OFF.vbs has been run and the server rebooted. This opens the POS up enough to be able to be usable.

### Assertion Request

Due to the Jenkins Service’s required authentication settings, a handoff is required to another entity (running in a context with more appropriate permissions) to provide authentication services. This other entity is the File Monitor. The File Monitor is a custom built Windows Forms application that runs on the BOS server and interacts with the BOS DB.

The File Monitor and the Jenkins process must interact via the file system of the POS server as the Jenkins Service must necessarily run as the Local System user when interacting with the desktop. This Local System user can only “communicate” via the local file system.

At this juncture it should be noted that during the course of the Proof of Concept project the first choice for the File Monitor was for it to be a Windows Service. This approach had to be rejected as a desktop context is required to interact with the BOS database (see Limitations and Constraints below).

The TS user is required to be the logged in user when executing the File Monitor. The TS user has sufficient administrative rights.

The File Monitor must be running before the Jenkins job is started as it polls and waits for the handover file containing configuration information for the assertion steps that it performs.

The content of this “handover file” essentially amounts to an internal API. Its contents are discussed in Appendix B.

The File Monitor reads the handover file and spawns a batch process, which in turn executes the isql command line tool to get the transaction details from the database.

### Transaction Details

Lastly, the File Monitor cleanses the raw output from Interbase and writes out the results ready for consumption by the Jenkins process.

This file is picked up by the Jenkins process and compared to the expected results. The current code base is able to assert 3 pieces of information regarding the transaction from the database:

1. Quantity of units sold
2. Unit Price of item sold
3. GST applicable to transaction

## Software Components used in Solution

* .NET 3.5 SP1
* Jenkins 1.420
* Various Jenkins Plugins
* NUnit 2.5.10
* White Automation Framework
* Firefox 6.0 (required to configure Jenkins)

## Development Tools Used

* Visual Studio 2008 SP2

## Source Code Control

* VisualSVN Server
* TortoiseSVN

# Limitations and Constraints

## Lack of UI Automation Framework support in POS application

The current release of the POS application does not support the Microsoft UI integration framework. The implication of this limitation is that is not possible to determine programmatically whether each sub-step within a POS was successful before proceeding to the next sub-step. For example:

1. Did clicking on the Operator button during the login process present the pin entry screen?
2. Did completing the login process result in the application main screen being displayed?
3. Did clicking the next button result in the next screen of available items for sale being displayed.
4. Did clicking an item to sell button enable the exact cash button and add the item to the list of items being sold?

This limitation is partially mitigated by adding waits in the process for the POS to respond to the directions initiated by the robotic user.

## Lack of Interbase DB driver

In the absence of an Interbase database driver, the File Monitor process needs to spawn a batch process to launch the isql.exe command line tool to interact with the BOS DB which necessitates the reading and writing of flat text files via the file system. Not only is this approach cumbersome, it is also slow due to the consequent file I/O operations.

The spawning of this process requires a desktop context, which a Windows service does not have – thus precluding a Windows service from the solution’s final form.

## Test set tied to given POS setup

The robotic user is clicking buttons on the POS as a real user would, but unlike a real user it does not currently “look” at the screen to see which buttons are enabled. It blindly clicks a point on the screen that the developer of the test has previously checked to be the correct location on the screen to click. See section 8 below for more on this.

# Future Opportunities

## Write more tests to provide coverage

Due to time constraints the Proof of Concept has successfully built:

1. The supporting infrastructure for performance testing
2. Code for 3 tests (1 of which is a exact cash transaction)

Further work is needed to write more tests to achieve sufficient coverage of the POS

## Make robotic user smarter

The reference data that defines the POS buttons that will be enabled is loaded in to the BOS database when the POS rig is setup. Before being loaded into the BOS database, this data exists in a CSV file. The system could be enhanced to read this CSV file so that it would know how the POS was configured. This would also allow sections of the Testing code to be re-factored and improved.

## Test more than transactions

This solution could be extended to test any actions on the POS that result in data being written to the database or to the file system of either the POS or the BOS server.

## Potential reuse

Due to the modular design of components of the overall solution, there is potential to utilise some of these components in other areas of the business in their own right. .NET and C# integration skills required.

|  |  |
| --- | --- |
| **Component Name** | **Function Description** |
| Windows Service Controller | Ensures a named Windows service on a local or remote machine is started regardless of its current state. |
| File utilities | Specialised classes to read, write, copy, delete and move files. |
| Batch Executor | Executes a batch file. |
| Network Drive Mapper | Maps an available drive letter to specified file path. |

These components could be used as-is or could be extended further if the business so desired.

## Extend Jenkins NUnit Plugin

NUnit tracks the time it takes for a test to run, the Jenkins NUnit plugin parses the NUnit output and displays this information graphically.

The intended usage scenario for this Proof of Concept is as follows:

1. Take a baseline for the current release under test.
2. Ask the business to specify a threshold above this baseline across the board (say, 5%) that is still acceptable performance
3. Monitor future test runs in light of this acceptable threshold

The Jenkins NUnit plugin does not currently support the concept of reporting a failure if a test takes longer than a given threshold.

It may be possible to enhance or extend the NUnit plugin to have this capability. With this added functionality it would be easier for the business to see tests that failed to run within the threshold.

## Require Retalix to support UI Automation Framework

BP should make it a requirement that the POS application support the Microsoft UI Automation Framework

## Purchase Interbase DB driver

BP does not currently have an Interbase OLE DB driver.

An OLE DB driver would enable the accessing of the BOS DB programmatically and would expose the results as record sets. The current scripts used by the IT&S POS deployment team may be able to be streamlined by using such a driver.

# Appendix A – Configuration Settings

## Remote Services

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Description** | **Example Value** |
| SQLUser | String | Username used to authenticate against the BOS DB | SYSDBA |
| SQLPassword | String | Password used to authenticate against the BOS DB | \*\*\*\*\* |
| RemotePOS | String | Name of POS associated with this BOS | AU20200135P01-G |
| AdminUser | String | Username used to launch batch process | TS |
| AdminPassword | String | Password used to launch batch process | \*\*\*\*\* |
| NormalisedStateTries | Long | Not currently used | 5000 |
| DriveLetter | String | Drive letter to use to map a drive (not currently used by solution) | K |
| SyBaseServiceName | String | Name of Interbase 7.5 Windows Service. | IBS\_gds\_db |
| RemoteFSPath | String | File system path via which the POS and BOS “communicate”. First directory must be a shared folder. | GSS\AutomatedTesting |
| NotificationFile | String | The name of the file to handover control to the File Monitor. | Handball.txt |
| LocalOnly | Bool | Indicates the location of the Interbase service in relation to the RemoteServices.dll. | True |
| TransactionAssertionTemplateFile | String | Name of transaction assertion SQL template. | AssertTrans\_TEMPLATE.sql |
| TransactionAssertionConcreteFile | String | Name of parameterised transaction assertion SQL file | AssertTrans.sql |
| RawActualsFile | String | Name of file written out containing raw output from Interbase. | ActualsData.txt |
| BoomerangFile | String | Name of file written out that contains cleansed output from Interbase. | CleansedActualsData.txt |

## Logger

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Description** | **Example Value** |
| ApplicationName | String | Name given to the source of event written by this application to the Windows Event Log. | AsserterService |

## File Monitor

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Description** | **Example Value** |
| RemotePOS | String | Name of POS associated with this BOS | AU20200135P01-G |
| RemoteFSPath | String | File system path via which the POS and BOS “communicate”. First directory must be a shared folder. | GSS\AutomatedTesting |
| NotificationFile | String | The name of the file to handover control to the File Monitor. | Handball.txt |
| BoomerangFile | String | Name of file written out that contains cleansed output from Interbase. | CleansedActualsData.txt |

## POS Tests

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Description** | **Example Value** |
| POSApplicationPath | String | Path on POS Server to the POS executable. | C:\Positive\Exe\Positive32.exe |
| ApplicationLaunchWaitMS | Int | Number of milliseconds to wait for the POS application to launch at the start of each test case. | 80000 |

# Appendix B – Handover File

## Layout

The handover file contains three lines. The layout is as follows:

*<SQL Template Filename>*

*<SQL File>*

*<Product Name>*

## SQL Template Filename

This is the name of the file that is the template for asserting transactions. It is packaged with the File Monitor installer package.

## SQL File

This is the name of the file that is created after applying the Product Name, as a parameter to the SQL Template file.

## Product Name

This is the name of the product that was purchased on the POS. This name must match the name of the product as defined in the BOS database, otherwise the transaction record will not be found and the test will ultimately fail.

## Example Contents

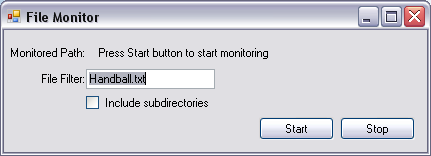
AssertTrans\_TEMPLATE.sql

AssertTrans.sql

Chup Chups 13G

# Appendix C – File Monitor

## Screenshot



## Usage Instructions

1. Specify the name of the file to monitor for. Wildcards (\*) can be used. Recommended to leave unchanged as Handball.txt.
2. Indicate via the checkbox whether subdirectories are to be included. Recommended OFF.
3. Press the Start button to start monitoring on the specified File Filter. Monitored Path is set automatically by the configuration settings RemotePOS and RemoteFSPath.

Logs are written to the Windows Event log.

If you wish to stop monitoring, press the Stop button.