Software Test Plan: **Customisable VNC Viewers**

Brandon Byskov
1068517
`byskovbm@mcmaster.ca`

Xuchao Ding
1233855
`dingx3@mcmaster.ca`

Natalie Perna
1066785
`pernanm@mcmaster.ca`

October 24th, 2014

CS 4ZP6: Capstone Project
Department of Computing and Software
McMaster University

Supervised by:
Dr. Wolfram Kahl
`kahl@mcmaster.ca`

Instructed by:
Dr. Rong Zheng
`rzheng@mcmaster.ca`

Revision History

Revision #	Revision Date	Description of Change	Author
0.1	2014-10-24	First draft.	Natalie Perna

Distribution

Recipient Name	Recipient Organization	Distribution Method
Dr. Rong Zheng	McMaster University	Avenue to Learn Dropbox

Acknowledgments

This document is based upon the IEEE Standard for Software Test Documentation (Std 829-1998).

Contents

1 Introduction

1.1 Document Identifier

This Software Test Plan is to outline the tests procedures to verify conformance to the requirements specification of the software that will be built for the 4th year capstone project by the group consisting of Brandon Byskov, Xuchao Ding, and Natalie Perna. This document is intended to be used by the development team, supervisors, and course instructors.

1.2 Scope

This document will outline the test procedures of the *Customisable VNC Viewer*. The test plan will ensure conformance to the requirements as outlined in the software requirements specification. It will describe the unit tests of the system, such as initializing systems and connections. It will also describe integration tests, such as presisting connections, updating displays, and distubting connections.

1.3 Definitions, acronyms, and abbreviations

In general, the definitions of terms used in this document conform to the definitions provided in the RFB Protocol.[1]

The definitions below are key terms as they are used in this test plan.

1.3.1 Virtual Network Computing (VNC)

Virtual Network Computing (VNC) is a system for graphical desktop sharing.

1.3.2 VNC Server

The VNC server, or RFB server, is the remote endpoint where the applications and graphical desktop environment are running, and changes to the framebuffer originate. It may also be referred to as the host.

1.3.3 VNC Client

The VNC client, or RFB client, is the user's endpoint, where the display is projected (i.e. onto a monitor), and input devices are connected (i.e. keyboard, mouse, etc.). It may also be referred to as the viewer.

1.3.4 RFB Protocol

The remote framebuffer (RFB) Protocol specifies the rules and procedure for remote access to a graphical user interface. RFB is the protocol used in VNC.

1.4 System overview and key features

The *Customisable VNC Viewer* software will allow a user to an run application or desktop on a remote server, while viewing that application or desktop in a local window and interacting using local input devices. Furthermore, the user will be allowed to segment contents of the application window across multiple physical displays. This would allow a user to tile multiple physical displays beside each other and view a software application that can span continuously across all of the displays. The Customisable VNC Viewer will be written in code that can be easily reused for other projects. There is a goal to have this software implemented on McMaster University Computing and Software Department's VidaLab large display.

2 Test Items

The following items are to be tested in the Proof of Concept:

- Connecting a client to a server
- Performing connection handshakes
- Initialising the connection up to, but not including, the first image

3 Features to Be Tested

The following features will be tested in the Proof of Concept:

- Requesting a connection
- Protocol version handshake
- Security handshake
- Challenge-response scheme (password authentication)
- Client initialisation
- Server initialisation
- Requesting display segment

4 Features Not to Be Tested

The following features will not be tested in the Proof of Concept:

- Persisting a connection between client and server post-initialisation
- Updating screen image continuously
- Abrupt client or server disconnection after setup

5 Approach

5.1 Unit Testing

As we will not have the resources to have a VNC server running persistently throughout the development cycle, much of the unit testing will be done manually. That is, when changes are made to the connection module, a VNC client and server will be manually started by the developer in order to test the appropriate module.

5.1.1 Requesting connection

5.1.1.1 Type of testing Manual, dynamic, blackbox

5.1.1.2 Test factors Reliability, methodology, correctness

5.1.1.3 Input The relevant user input to the connection initiation module is, firstly, the server (host) IP address or URL.

5.1.1.4 Output The output of the connection request function will be:

- a boolean value, indicating whether the client was able to reach the server
- an enumerated type, indicating any additional detail regarding the failure, if applicable (specific potential messages to be determined)
- an enumerated type, indicating the challenge-response scheme, if applicable

5.1.1.5 Test cases The following cases must then be tested from the client:

1. Request a connection without specifying a URL (that is, with an empty string)
2. Request a connection with a URL or IP address in an invalid format (i.e. containing special characters, or missing a URL protocol)
3. Request a connection to a valid address, but where there is no VNC server running
4. Request a connection to a valid address, where there is a VNC server running

5.1.1.6 Testing criteria

The following criteria are expected to be met for the above test cases:

1. Output should indicate failure to connect (false), and a message indicating that the cause of the failure was a missing server address
2. Output should indicate failure to connect (false), and a message indicating that the cause of the failure was an invalid server address
3. Output should indicate failure to connect (false), and a message indicating that the endpoint was reached, but a VNC server did not respond
4. Output should indicate successful to connection (true), and a message indicating the challenge-response scheme (i.e. whether a password is required for connection)

5.1.2 Protocol version handshake

Both our client and server will support only RFB protocol version 3.8.

5.1.2.1 Type of testing

Automated, dynamic, blackbox

5.1.2.2 Test factors

Reliability, access control, methodology

5.1.2.3 Input

The relevant input to the handshake is the *ProtocolVersion* message, which will come from the server, and is represented by a 12 byte string of ASCII characters in the format `"RFB xxx.yyy\n"`, indicating the highest RFB protocol version number supported by the server.[1]

5.1.2.4 Output

The output, and eventual response to the server, should be the protocol version to used, that is, the highest version supported by the client, not exceeding in magnitude the *ProtocolVersion* from the server.

5.1.2.5 Test cases

The following cases must then be tested from the client:

1. Input of invalid *ProtocolVersion* message format
2. Input of *ProtocolVersion* message with unrecognized protocol version (that is, not 3.3, 3.7, or 3.8)[1]
3. Input of RFB protocol version prior to 3.8
4. Input of RFB protocol version 3.8

5.1.2.6 Testing criteria

The following criteria are expected to be met for the above test cases:

1. The client should break the connection to the server by not responding to the handshake

2. The client should break the connection to the server by not responding to the handshake
3. The client should break the connection to the server by not responding to the handshake
4. The client should respond with `"RFB 003.008\n"`

5.1.3 Security handshake

Both our client and server will support only two VNC security types[1]:

- None (1)
- VNC Authentication (2)

5.1.3.1 Type of testing Automated, dynamic, blackbox

5.1.3.2 Test factors Reliability, access control, methodology

5.1.3.3 Input The relevant input to the handshake is the list of supported security types by the server, in the form of:

- an integer, indicating the number of security types supported
- an array of integers, wherein each element indicates a supported security type, as specified in the RFB protocol.[1]

5.1.3.4 Output The output, and eventual response to the server, should be the security type to be used, that is, the highest security supported by both the client and server:

- an integer, indicating the security type to be used[1]

5.1.3.5 Test cases The following cases must then be tested from the client:

1. Number of security types doesn't match size of array
2. No security types are supported, or only Invalid (0)
3. Only security types greater than 2 are supported (i.e. encryption is required)
4. Security type None (1) is supported, but not VNC Authentication (2)
5. Security type VNC Authentication (2) is supported, but not None (1)
6. Both security types None (1) and VNC Authentication (2) are supported

5.1.3.6 Testing criteria The following criteria are expected to be met for the above test cases:

1. The client should break the connection to the server by not responding to the handshake
2. The client should break the connection to the server by not responding to the handshake
3. The client should break the connection to the server by not responding to the handshake
4. The client should respond with 1 for no authentication
5. The client should respond with 2 for VNC authentication
6. The client should respond with 2 for VNC authentication

5.1.4 Challenge-response scheme

5.1.4.1 Type of testing Automated, dynamic, blackbox

5.1.4.2 Test factors Reliability, access control, methodology

5.1.4.3 Input The relevant inputs for a challenge-response are:

- a 16 byte random string challenge, from the server
- a password string, supplied by the user

5.1.4.4 Output The output of the connection request function will be:

- a 16 byte string response, resulting from encrypting the challenge with DES[1]

5.1.4.5 Test cases Specific tests to be determined as edge cases for DES are identified prior to implementation.

5.1.4.6 Testing criteria To be determined prior to implementation of DES.

5.1.5 Client initialisation

5.1.5.1 Type of testing Automated, dynamic, blackbox

5.1.5.2 Test factors Reliability, access control, methodology

5.1.5.3 Input None.

5.1.5.4 Output The output, and eventual *ClientInit* message to the server, should be the shared flag (boolean) bit, indicating whether the server should leave other clients connected (true) or give exclusive access to this client (false).

5.1.5.5 Test cases The function takes no input, so the only test case is calling the function.

5.1.5.6 Testing criteria The output for our client should always be true, that is, other clients should be allowed to connect simultaneously.

5.1.6 Server initialisation

5.1.6.1 Type of testing Manual, dynamic, blackbox

5.1.6.2 Test factors Reliability, access control, methodology

5.1.6.3 Input The *ClientInit* message, as outlined above.

5.1.6.4 Output The output, and eventual *ServerInit* message to the client, includes[1]:

- an integer framebuffer width
- an integer framebuffer height
- the pixel format to be used
- a string name for the desktop server machine

5.1.6.5 Test cases The function takes no input, so the only test case is calling the function.

5.1.6.6 Testing criteria It should be manually checked that the framebuffer height and width match the server display resolution. The pixel format will always be the same for our server, and must be determined before implementation. It should also be manually checked that the desktop name matching the host name.

5.1.7 Requesting display segment

5.1.7.1 Type of testing Automated, dynamic, blackbox

5.1.7.2 Test factors Reliability, access control, methodology

5.1.7.3 Input The relevant input for the client to request a segment of the server's display buffer is two sets of two positive integers corrosponding to the two sets of disllay coordinates consiting each of a (x, y) coordinate pair.

5.1.7.4 Output The output of the client segment request will be a flag (boolean) bit, indicating whether the server accepted to return the given display segment (true), or the server rejected the segment request (false).

5.1.7.5 Test cases The following cases must then be tested from the client:

1. Input where any of the four integers is negative
2. Input where the x and y value of the second coordinate pair are less than the x and y value of the first coordinate pair
3. input where the x and y value of the second coordinate pair are equal to the x and y value of the first coordinate pair (single pixel segment)
4. input where the x and y value of the second coordinate pair are less than the x and y value of the first coordinate pair

5.1.7.6 Testing criteria The following criteria are expected to be met for the above test cases:

1. The bloolean flag should be false
2. The bloolean flag should be false
3. The bloolean flag should be true
4. The bloolean flag should be true

5.2 Integration/System Testing

The following tests will be performed manually until an automated process is developed with the implementation

5.2.1 Persisting a connection between client and server post-initialisation

5.2.1.1 Set up Through the graphical user interface of the client software, a connection will be established to the server and authenticated by the server.

5.2.1.2 Test event The client will disable network communication with the server. The client will will then re-enable network communication with the server.

5.2.1.3 Expected response The client should respond by first, displaying a graphical message to the user that the client has lost its connection to the server, and then, after the connection has been re-established, removing the message and resuming the connection.

5.2.2 Updating screen image continuously

5.2.2.1 Set up Through the graphical user interface of the client software, a connection will be established to the server and authenticated by the server.

5.2.2.2 Test event The client will then throught the graphical user interface request a display segment corresponding to the entire server display.

5.2.2.3 Expected response The client should respond with a graphical windowed display of the server display buffer. If there is no correct visualization of the server display, then this test has failed.

5.2.3 Abrupt client or server disconnection after setup

5.2.3.1 Set up Through the graphical user interface of the client software, a connection will be established to the server and authenticated by the server.

5.2.3.2 Test event The client will disable network communication with the server.

5.2.3.3 Expected response The client should display a graphical message to the user that the client has lost its connection to the server. If no message is displayed to the user, then this test has failed.

6 Pass/Fail Criteria

The pass/fail criteria for each test case is outlined in the Testing criteria sections.

7 Testing Process

All automated tests will be written and executed as HUnit tests in Haskell.

All manual tests will require launching a VNC client and VNC server, on separate machines (where the VNC server may optionally be a virtual machine). The manual tests will then be performed according to the Test case instructions outlined above.

8 Environmental Requirements

8.1 Hardware

- Physical Linux client machine with keyboard and mouse input, and monitor display (more specific system requirements to be determined)
- Physical or virtual Linux server machine (more specific system requirements to be determined)

8.2 Software

- GHC for compiling software applications
- HUnit, for automatable unit tests

9 Change Management Procedures

Using Test Driven Development, developers will add new features to the codebase by first writing a test case. That test case should fail before the feature is complete, and pass once it is finished. Tests, and therefore features, to be added must first be submitted for approval by proposing an addition to this Software Test Plan, via the shared repository.

Wherever possible, these unit tests should be automated, but due to the nature of our applications, and their sometimes complicated user input, test will frequently be manual. Therefore, for manual tests, one or more test cases should be outlined alongside the code, with instructions to execute the tests for other developers, and expected results for each test case.

Before changes are pushed to the shared repository, the developer must verify that all existing, and newly added tests are passing, that is, the new feature is complete and correct, and did not cause regressions elsewhere. That includes both automated and manual tests, whether seemingly relevant to the new feature or not.

10 References

[1] T. Richardson and J. Levine. (2011, Mar.) The remote framebuffer protocol. RealVNC Ltd. [Online]. Available: http://tools.ietf.org/html/rfc6143

11 Plan Approvals

Approver	**Role**	**Date**	**Signature**
Dr. Wolfram Kahl	Project Supervisor		
Brandon Byskov	Team Member		
Xuchao Ding	Team Member		
Natalie Perna	Team Member		