
# CSIR - Distributed Application Manager Test Document

101 Solutions

October 24, 2013

Version 0.3


| | |
|---|---|
| Francois Germishuizen | 11093618 |
| Jaco Swanepoel | 11016354 |
| Henko van Koesveld | 11009315 |

## Change Log

| Date | Version | Description | Done by |
|---|---|---|---|
| 12 Sept | Version 0.1 | Document Created | Henko |
| 13 Sept | Version 0.2 | Merged old test docs into one | Jaco |
| 24 Oct | Version 0.3 | Updated | Jaco |

# Contents

# 1 Overview

## 1.1 Background

The CSIR is actively developing a distributed simulation framework that ties in with various other real systems and is used to exchange information between them. The client has a number of configurations of this system depending on the requirements of the client which can involve various external applications as well.

One of the issues the client has is to quickly distribute the latest build or configuration files of their software over various computers that are needed for an experiment. In some cases the same computers may be used for other experiments which mean each of the computers may need to have various builds and configuration options.

Another issue they experience is the running, stopping and restarting of the complete simulation. During a simulation it may be determined that certain configuration options may need to be changed and distributed to the affected machines, in which case either all or some components will need to be restarted which can become tedious and time consuming.

## 1.2 Business opportunity

The goal of our project is to develop an application which is able to maintain various build versions of the simulation framework and distribute these builds to certain designated machines that may be required for an experiment. The application will monitor system statistics of the various machines attached to an experiment and will have the ability to execute applications on those machines which will have different configuration options.

The application will consist of a master and slave component where the master is used to control the distribution of slaves. From the master one will be able to start an experiment which will run the relevant applications on all the necessary machines.

# 2 Unit Testing

This document provides an outline of the approach followed when doing unit testing. Different units will be tested and mock testing will be used in order to display that the classes behave as they should.

We are making use of QT unit tests which allow for cross platform unit testing. This allows us to test the applicaiton for both Windows and Linux.

Unit testing is used in order to test across levels of granularity. This is done in order to test whether the individual modules or units behave as expected.

## 2.1 Contract testing

Within unit testing that are done there are testing done to determine if contracts are met.

- If preconditions are met
  - Service is provided
  - All post conditions hold true after service was provided
- Any other units not directly tested will be mocked in order to simulate the unit requesting the service

## 2.2 Testing the JSON class

### 2.2.1 Why test it

Within both the AppMan and AppManClient applications there is a class JSON which converts any JSON value into QVariantMap and QVariant maps in order to make use of JSON for communication.

### 2.2.2 Methodology

There are two areas with regards to JSON that are tested. Firstly the JSON class which parses the JSON string, and secondly the functions generating the JSON strings in order to communicate. Both of these classes and functions are mirrored on both the AppMan and AppManClient applications.

- JSON
  - AppMan - JSON class is tested in order to see if it generates the required classes from a JSON string

  - AppManClient - JSON class is tested in order to see if it generates the required classes from a JSON string

- JSON communication
  - AppMan - JSON generating functions which will generate the JSON strings and prepend and append the required strings
  - AppManClient - JSON generating functions which will generate the JSON strings and prepend and append the required strings

# 3 Integration Testing

This section is a stub and will continuously be improved

## 3.1 Integration Testing AppMan

### 3.1.1 Communication

### 3.1.2 Management Class

- Signal Testing : Signals are tested in order to determine whether they exibit their required behavior when the management class should emit them.
  - A signal emitted when a new machine connects successfully
  - A signal emitted if a machine updates its md5sum value in order to determine which builds are not synched yet

## 3.2 Integration Testing AppManClient

### 3.2.1 Communication

### 3.2.2 Management Class

- Signal Testing : Signals are tested in order to determine whether they exibit their required behavior when the management class should emit them.

# 4 Non-functional Testing

This section is a stub and will continuously be improved

- We have conducted minor scalability tests in the Informatorium labs with 6(5 slaves, 1 master) pc's so far
- We will be testing on various versions of Windows and Linux at home and in the labs
- Our usability testers found some features hard to use, we updated the GUI accordingly and made a user manual for this reason.