Functional Verification Test Plan
Version 2
Mar 15, 2014

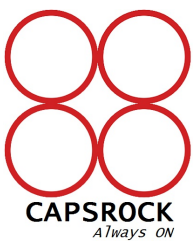
Prepared for

Oracle

Prepared By

Capsrock

Richard Bae
Chris Beichler
Cody Hunter
Taylor Woods



1 Introduction ---- 3
2 Document Control Information ---- 3
2.1 Author / Owner ---- 3
2.2 Approvers ---- 3
2.3 Reviewers ---- 3
2.4 Summary of Changes ---- 4
3 Supporting Documents ---- 4
4 Objective ---- 4
5 Assumptions ---- 5
6 Functional Verification Test ---- 5
6.1 Functional Verification Test Scope ---- 6
6.1.1 Functional Scenario Testing ---- 6
6.1.2 Access Control Testing ---- 7
6.1.3 Large Scale Data Testing ---- 8
6.1.4 Exploratory Testing / Adhoc Testing ---- 9
6.1.5. Sanity Testing ---- 9
6.2 What is NOT within the Functional Verification Test Scope ---- 10
7 Test Methodology ---- 10
7.1 Entry and Exit Criteria ---- 11
7.1.1 Entry Criteria ---- 11
7.1.2 Exit Criteria ---- 11
7.1.3 Exception Handling for Entry / Exit Criteria ---- 11
7.2 Regression Testing ---- 12
8 Environments Supported and Tested ---- 13
8.1 Server Environment ---- 13
8.2 Client Environment ---- 13
9 Dependencies ---- 14
10 Responsibilities ---- 14
11 Risk Management ---- 15
11.1 Risks ---- 15
11.2 Risk Tracking and Management ---- 15
11.3 Deviations ---- 15
12 Test Resources ---- 16
12.1 Hardware ---- 16
12.2 Software ---- 17
12.3 Staffing ---- 17
13 Defect Management ---- 18
13.1 Defect ---- 18
13.2 Defect Severity ---- 18
13.3 Defect Reporting System ---- 19
13.4 Defect Reporting Data ---- 20
13.5 Defect Validation ---- 20
14 Test Results ---- 21

Text

CAPSROCK
Always ON

1 Introduction

This is the Smart Time Entry Geofence system Functional Verification High Level Test Plan prepared by the Capsrock team for Oracle corporation.

2 Document Control Information

2.1 Author / Owner

Richard Bae - Device Interface Designer - Bbae@calpoly.edu
Chris Beichler - Software Designer - cbeichle@calpoly.edu
Cody Hunter - User Interface Developer - Chunte02@calpoly.edu
Taylor Woods - System Architect - twoods01@calpoly.edu

2.2 Approvers

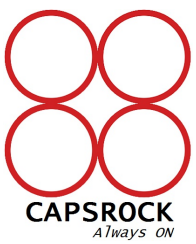
The following people provide the approval of this document:

Name	Function	Approval Date
Oracle	*Client*	

2.3 Reviewers

The following people provide the review comments to this document:

Name	Function	Mandatory/Optional
Chris Lupo	*Advisor*	*Mandatory*



2.4 Summary of Changes

The Author/Owner of this document is authorized to make the following types of changes to the document (identified in the change history) without requiring the document re-approval:

- Editorial, formatting, and spelling
- Clarification
- Document structure

Note: Any other changes may require re-review and re-approval.

To request a change to this document, contact it's Author/Owner.

Changes to this document are summarized chronologically in the following table.

Version Number	Date	Summary of Changes	Editor
1.0	02/05/14	*Document created*	*Richard Bae, Chris Beichler, Cody Hunter, Taylor Woods*

3 Supporting Documents

The following are the supporting documents for this Functional Verification Test plan:

Project charter - Charter

4 Objective

This is the High Level Functional Verification Test Plan for Smart Time Entry System 1.0. The test plan outlines the following aspects of a Functional Verification Test:

- Use case coverage
- Assumptions
- Scope
- Test methodology
- Entry & exit criteria
- Exclusions
- Dependencies & responsibilities
- Schedule
- Risks management
- Status tracking mechanism
- Test resources
- Defect management for functional test activities

These aspects should be reviewed and approved by stakeholders before the beginning of functional verification test execution.

5 Assumptions

The following assumptions are made for the functional verification test of this project:

- All Pebble users on 2.0
- All Pebble testing assumes active connection with Android
- Requires Android 4.2.2 or above.
- User will not work over 12 hours per day
- Users will enter reasonable durations for work/break time (> 1 min)

6 Functional Verification Test

Functional verification testing is a type of black box testing that uses the solution specifications, design document and use case document as a point of comparison to validate that the product is functioning properly. Functional scenario testing includes main flow test, exceptional flow test and exception handling test. FVT also needs to ensure the system is working properly and not leading to an undesired behaviour when an end user uses the features in a way not described in the use cases. FVT testing is not limited to positive flow tests described in use cases.

6.1 Functional Verification Test Scope

The tests are responsible for the validation of functional correctness for the following aspects of the scope:

- Functional scenario test
    - User login
    - Create geo-fence
    - Enter/exit geo-fence
    - Log time
    - View logged data
- Access control test
    - Verify authentication against server
    - Verify new account creation
- Large scale data test
    - Check time efficiency of retrieving data from large databases
- Ad-hoc test
    - Confirm functionality as added in
- Sanity test
    - Record time with other timekeeping device and compare against our time
    - Obtain GPS point using other device and compare to GPS point from our device
    - Verify data sent to server is what is stored in server
- Full regression test
    - Notification on entering/exiting geo-fence
    - Creation of new geo-fences
    - Posting time entry data to server
    - Retrieving time entry data from server

6.1.1 Functional Scenario Testing

Functional scenario testing is a type test that verifies the use cases are functioning as intended. Functional scenario testing is the major part of FVT coverage and it includes main path testing, error handling testing and exceptional flow testing.
Each use case is validated to ensure the functionalities are working as intended. In addition to testing each use cases individually, FVT is also responsible for testing scenarios where multiple use cases are put together to validate functional end to end scenarios.

The following examples are the type of scenarios that will be covered in our tests:

Main Flow:

- User logs into the application - verify that users can log in and application loads correctly.
- User creates a new geo-fence - Ensure that users can create a new geo-fence including its name, location, and radius.
- User enters geo-fence - Verify that an alert is triggered on a device for the proper location.
- User logs time - verify that users can log time and then view that time.

Error Flow:

- User logs in with a bad username/password.
- User enters an invalid location for geo-fence

Functional End to End Flow:

- User logs into the app, creates a geo-fence, enters the geo-fence, logs time, then exits the geo-fence and views their logged time.

6.1.2 Access Control Testing

Access control testing is performed to ensure only the authorized roles can perform actions on authorized business objects.The access control policies need to be tested for positive and negative access control. For positive access control testing, the functional verification test is responsible for ensuring roles can perform actions on all allowed objects. For negative access control testing, the functional verification test is responsible for ensuring roles cannot perform actions on unauthorized objects.
Access control policies need to be tested through user interactions such as logging into the database with his/her own account to ensure users with front-end access are not accessing the administrative tools.

We will be testing access control in the followings manners:

Positive Access Control:

- Registered user can view only his/her timesheet
- User should be able to create a new account with his/her own timesheet and only he/she can access it

Negative Access Control:

- Registered users cannot view another user's timesheet
- Unregistered users does not have access other than login and create account page

Roles	Actions on objects	What to validate
New Time Entry User	*Create a new Time Entry Account*	*Create a new account on the database for the user. User should be able to log in and out of his/her account*
Existing Time Entry User	*Log in* *View their own timesheets*	*Be able to log on to his/her account and be able to read their own database. Should not be able to see other time entries on the database*
Database Admin	*Log in through Django* *View all timesheets*	*Be able to log on to his/her Django admin account and be able to view all time entries by any user*

6.1.3 Large Scale Data Testing

Large scale data function testing verifies that the system will continue to function correctly when a large number of business objects are present. Large scale data response time test should be executed by performance test in conjunction with this functional test. Large scale data boundary is defined with configurable limits for business objects. If the data size boundaries are undefined, large scale data testing with various data sets can help to determine the boundaries.

The goals of large scale data testing is to ensure the system is functioning correctly and as expected in the following conditions:

- Reach the data size boundary for an object
- Exceed the data boundary value by at least one
- Extremely large data (much higher than typical data set size)

We will be performing large scale data testing in the following ways:

- Large time entry with 50 unique geo-fence locations with 20 clock in and clock out time log.

- When 1000 users are already registered, register a 1001st user.
- Perform a search that results in a very large result set and view the results and refresh the resulting page.
- Switch between different tabs on Android application
- Add a geo-fence location or time to the work/break time entry where the maximum boundary for the number of locations and work/break time is already reached.

6.1.4 Exploratory Testing / Adhoc Testing

Exploratory testing / adhoc testing is a part of functional verification testing performed without planning or documentation. The tests are intended to be run only once and are performed to identify unexpected errors in the project. If an unexpected error is discovered, additional exploratory testing of that area should be performed. Testers are encouraged to achieve the exploratory test objective through various deviated paths.

Currently, the only ad hoc testing expected is:

- Confirming functionality of components, such as creating geo-fences or sending data to the database, as they are added into the project as a whole

6.1.5 Sanity Testing

Sanity testing verifies that the system performs all calculations with an expected, rational result. Sanity test is done to catch any obvious and noticeable errors that are completely out of expected range of results.

Our sanity testing will include:

- Confirming stopwatch functionality is accurate by comparing it to other timekeeping devices
- Verifying GPS data collected with our device matches data collected by other GPS-enabled devices
- Confirming that data sent to the database is actually stored in the database

6.2 What is NOT within the Functional Verification Test Scope

The following are not in the scope of this functional verification test:

- Globalization testing - application expected to run in English only
- Security verification of stored data
- Investigation of storage for continuously growing data set

7 Test Methodology

High level test planning is done prior to formal detailed test planning and execution. As part of the high level test planning following are determined.

FVT Criteria:

- FVT entrance criteria
- FVT exit criteria
- FVT Exception handling for entry and exit criteria

Regression:

- Definition and test case selection guideline
- When to execute

Automation:

- Automation considerations
- Automation tools

A detailed test case document is prepared based on the project specifications and the use case document. The test case document is reviewed and approved as part of the functional test preparation.
Prior to formal entry into the test execution phase, the Lab environment will be setup with appropriate operating systems, protocols, and any other pre-requisite software and hardware.

7.1 Entry and Exit Criteria

7.1.1 Entry Criteria

The development is based on Waterfall development model.

The following is the post development code and unit test entry criteria for FVT.

Functional verification test preparation can begin once:

- Solution specifications are reviewed and approved for the whole project

Functional verification test execution can begin once:

- Functional verification high level test plan (this document) is reviewed and approved

7.1.2 Exit Criteria

The development is based on Waterfall development model.

Functional verification test preparation can exit once:

- High Level Functional Verification Test Plan is reviewed and approved

Functional verification test can exit once:

- All test cases have been attempted and completed
- All defects have been addressed
- Functional verification test exit checklist is completed, reviewed, and approved

7.1.3 Exception Handling for Entry / Exit Criteria

A document is prepared by the test team lead with the exceptions and associated reason, action plan, and risks. The document is then reviewed by the solution owner/manager, development lead, test manager and project manager. The exceptions must be approved before officially claiming functional verification test entry or exit.

7.2 Regression Testing

The purpose of a regression test is to ensure that as code is checked in and defects are fixed, existing code is not regressed.

The regression test buckets are grouped into 2 categories:

1) Sanity bucket (approximately 10% of the test cases)

The goal of this test bucket is to verify that the major functionalities are working correctly. A small subset of test cases is selected to ensure the critical paths are tested quickly. These include:

- Login, start work time, switch tabs, wait 5 seconds, switch back to time entry. Timer should have elapsed ~5 seconds. Stop time.
- Login, switch to Timesheet tab. View Today's date. Verify information is accurate.

Anytime a change is made to the Time Entry or Time Sheet Tabs, these tests should be run.

2) Full regression bucket (approximately 30% of the test cases)

The goal of this bucket is to verify the majority of functional flows are working correctly. The regression bucket selection also depends on the areas of code changes made and the importance of functionality. These Include the Previous Tests plus:

- Login, switch to Timesheet, go to a date with no available data.
- Login, switch to Activity, add activity.
- Login, start time on the pebble. switch to break time on the pebble. stop time on the pebble.
- Login, switch to Activity, start time on the pebble. wait 5 seconds. switch to Time Entry. Timer shoulder have elapsed ~5 seconds. Switch to the Activity. Stop time on the pebble. wait 5 seconds. Switch to Time Entry. Timer should be at 0, ready to start work time.

Any major changes to the android app or pebble should warrant these tests.

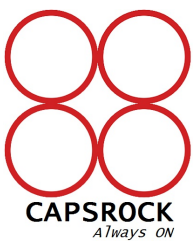
8 Environments Supported and Tested

8.1 Server Environment

Software	Version Supported	Remarks
Django	1.4	
SQLite Database	3	
CentOS	5.10	

8.2 Client Environment

Software	Version Supported	Version Tested	Percentage Covered	Remarks
IE	IE 8 on Windows 7	IE 8 on Windows 7 and Windows XP	100%	
	IE 7 on Windows XP and Windows Vista	Not Tested	0%	
	IE 6 on Windows 2000, 2003	IE 6 on Windows 2000	Full Regression	
Firefox	Firefox 3.0, 3.5	Firefox 3.5	100%	
Safari	Safari 4.0	Safari 4.0	Full Regression	



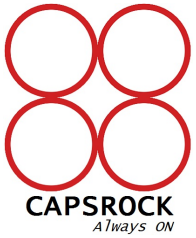
9 Dependencies

Following dependencies must be met on time for successful FVT.

Dependency	By (date)	By Whom	Risk if Not Met
Unit test completion	March 7, 2014	All developers	Unit test defects may be caught by FVT and takes more turn around time to fix.
Large scale data Test data	Phase 1 March 7, 2014	All developers	Large scale test data must be available for large scale data testing
Quick turn around on blocking defects	Fixes within 24 hours	Defect owning developer	Blocking defects delay the FVT execution

10 Responsibilities

Role	Responsibilities	Remarks
Project Manager	Keep the project on track in regards to the Gantt chart. Prioritize tasks for each member, including himself. Assist with the tasks of the other developers	Taylor Woods
User Interface Developer	Develop the UI of each device. Thoroughly test cases where user input could break or disrupt the process flow.	Cody Hunter
Device Integration Developer	Develop the functions to	Richard Bae



	allow the devices to communicate with each other. This includes Pebble to Android and Android to Server communication.	
Software Developer	Develop data structures and functions that assist the UI and Device Integration Developers as needed. Install and maintain needed software.	Chris Beichler

11 Risk Management

11.1 Risks

The following risks have been identified in this project:

- Device support - We will only be able to verify our application against a small number of all possible android devices. Any user with a Pebble not using 2.0 firmware will not be able to run our application.

11.2 Risk Tracking and Management

Risks are brought up as discovered during group meetings and tracked until their resolution. General issues involve basic Android functionality, and data transportation from Android to the web server.

11.3 Deviations

Deviations from this function test plan will be possible through the use of the Deviation Request Template below. These requests will be submitted to Taylor Woods for review and approval.

Deviation Request Template

Deviation Request:

What is deviated:

The reason for the deviation:

Risk Assessment on the deviation:

Justification for the deviation:

Deviation Reviewers:

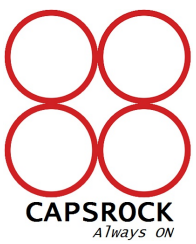
Deviation Approver:

12 Test Resources

12.1 Hardware

The following machines were used for testing:

Hardware	Located	Hardware Specification	Used For	Contact Person
Laptop	Mobile	2.2 Ghz Dual Core CPU, 500GB hard drive, 4 GB RAM	Android App Testing	Cody Hunter
Laptop	Mobile	2.4 Ghz Dual Core CPU, 500 GB hard drive, 4 GB RAM	Server	Taylor Woods
Samsung Galaxy S4	Mobile	1900MHz Quad Core, 16 GB hard drive, 2 GB RAM	Android App Testing	Richard Bae
Pebble Smart Watch	Mobile	Cortex M3 CPU, 1024Kb flash memory, 128Kb RAM	Pebble App Testing	Richard Bae



12.2 Software

Software	Version	License Type	Where to download	Contact Person
Eclipse	4.2.1	Free	link	Cody Hunter
Cloud Pebble	NA	Free	NA	Richard Bae
Django	1.6	Free	link	Taylor Woods

12.3 Staffing

Name	Phone (External / Internal)	E-mail	Location	Title
Taylor Woods	831-430-6305	*twoods01@calpoly.edu*	San Luis Obispo	System Architect
Richard Bae	707-239-0901	*Bbae@calpoly.edu*	San Luis Obispo	Device Interface Designer
Cody hunter	714-290-2094	*chunte02@calpoly.edu*	San Luis Obispo	User Interface Developer
Chris Beichler	408-806-9768	*cbeichle@calpoly.edu*	San Luis Obispo	Software Designer

13 Defect Management

13.1 Defect

A defect occurs when one or more of the following three rules are true:

- The software behaves opposite to the description in the product specification.
- The software does something that the product specification does not mention.
- The software is difficult to understand, hard to use, slow, or, in the software tester's opinion, will be viewed by the end user as incorrect.

13.2 Defect Severity

Defects are classified by severity. Defect severity levels can only be changed by the defect's originator or by concurring with the originator or test lead. Severity levels are defined as:

SEVERITY 1 - A severity 1 defect involves a data integrity failure and/or jeopardizes test conduct. A failure, which gates the testing of an entire project, may also be considered severity one. Schedules are impacted. A problem fix and problem retest is required before any test activity can effectively proceed. The defect must be resolved within 48 hours and a fixed test driver must be delivered to the test team as soon as possible after the defect is fixed; so that testing can resume. When the problem is fixed, the test execution record may not be closed as complete until a supported build containing the fix has been delivered to the test team.

SEVERITY 2 - A severity 2 defect represents a significant functional defect, which may invalidate continued testing in the functional area. The defect may mask the further detection of problems in the functional area affected. A problem fix and retest is required before test activity can proceed in the functional area. The next scheduled build (no more than 10 days later) must resolve the defect. When the problem is fixed, the test execution record may not be closed as complete until a supported build containing the fix has been delivered to the test team.

SEVERITY 3 - A severity 3 defect constitutes a functional problem that indicates no interaction with other functions. This is a singular rather than nested failure. It does not constitute a test or retest exposure for other test activity. Test activity can proceed while this defect is being resolved. The defect turnaround time is 20 days.

SEVERITY 4: A severity 4 defect is very low impact and may be easily found, recreated and retested. This type of defect may include low impact, but not easily re-creatable, problems. It does not constitute a test or retest exposure for other test activity. Test activity can proceed while this defect is being resolved.

13.3 Defect Reporting System

Defects are opened depending on where the defect occurs:

- Android defects are opened using either Eclipse running the Android emulator or the Android device itself.
- Pebble defects are opened using the Pebble smartwatch.
- Database defects are opened using any computer on the same network as the computer hosting the database. In this case, we will be using Taylor's laptop.

Defect access:

- Eclipse can be downloaded online for free from http://www.eclipse.org/downloads/
- Any Android device can be used for testing. Full Android list can be found at http://www.android.com/devices/
- Any Pebble smartwatch running v2.0 can be used for testing. Full Pebble list can be found at https://getpebble.com/
- Any computer on the same network as the database's host can be used for testing. A web browser is needed to view the output of the database.

Following is the defect template:

- Brief summary of the problem - REQUIRED
- Procedure to reproduce problem - REQUIRED
- Observed Results - REQUIRED
- Expected Results - REQUIRED

PD checklist:

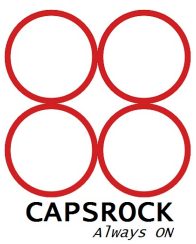
- Screen capture
    - Machine Information
- Machine (Hostname, IP), if applicable
- Build Date for Driver

13.4 Defect Reporting Data

All defect tracking data should be stored in the Capsrock Google Drive folder.

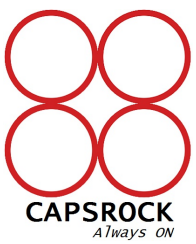
13.5 Defect Validation

When a defect is fixed, the build team notifies the tester regarding the defect fixture via an automated mail. The tester then installs the build and follows the steps that produced the steps to ensure the defect is fixed.

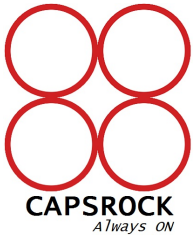


14 Test Results

Test Category	Test Description	Test Result
Functionality Tests	User can log into application	Functionality not implemented
	User creates a new geo-fence	Pass (created)
	User enters a geo-fence	Pass (Android/Pebble Notifications)
	User can switch to Time Sheet Tab	Pass
	User can switch to Activity Tab	Pass
	User can bring up 'Add Activity' Dialog	Pass
	User can add repeating Activity	Pass
	User cannot add repeating Activity for past days	Pass
	User can make radius between 5m and 3000m	Pass
	User can delete Activity	Pass
	User can change view date on Activity Tab	Pass
	User can start logging time from Pebble	Pass
	User can start logging time from inside app	Pass
	User can start logging time from pop-up notification	Pass
	User can change work mode from Pebble	Pass



	User can change work mode from inside app	Pass
	User can stop time from Pebble	Pass
	User can stop time from inside app	Pass
	Data gets sent to server on mode change/stop	Pass
	Data pulled from server when switching to the Time Sheet tab	Pass
	Data pulled from server when switching the date in the Time Sheet tab	Pass
	User can put Android App in the background and run	Pass
	User can exit Pebble app and return	Pass
Error Handling Tests	User logs in with bad password/username	Functionality not implemented
	User tries to create an overlapping geo-fence	Pass (not allowed)
	User tries to add an Activity in the past	Pass (not allowed)
	User tries to view future Time Sheets	Pass (not allowed)
	User leaves ‘name’ and ‘address’ fields blank on ‘Add Activity’ Dialog	Pass (not allowed)
	User tries to start logging in ‘break’ mode on Pebble	Pass (not allowed)
	User enters bad address	Google Services defines behavior
Large Scale Data tests	Large set of time entries pulled from server	Pass
	Large amount of users in database	Functionality not implemented



Sanity tests	Stopwatch functionality on Android accurate (tested against 3rd party stopwatch)	Pass
	Stopwatch functionality on Pebble accurate (tested against 3rd party stopwatch	Pass
	GPS data is accurate	Our GPS accuracy is dependent on Android GPS accuracy
	Data sent from app is received by server	Pass