



# Horus

## IMSETY

### Integration Test Plan

Version 0.3 21st June 2007

<b>Project team:</b>	Jeroen Keiren	0569081
	Frank Koenders	0575629
	Thijs Nugteren	0574426
	Joeri de Ruiter	0578312
	Stijn Stiefelhagen	0579816
	Carst Tankink	0569954
	Pim Vullers	0575766
	Freek van Walderveen	0566348
<b>Project manager:</b>	Egbert Teeselink	
<b>Senior management:</b>	L. Somers	TU/e (HG 7.83)
	M. v.d. Brand	TU/e (HG 7.44)
<b>Adviser:</b>	R.J. Bril	TU/e (HG 5.09)
<b>Customer:</b>	E. v. Breukelen	ISIS

Computer Science, Eindhoven University of Technology, Eindhoven

### **Abstract**

This document provides the main guidance for the Integration Tests (IT) during the Detailed Design (DD) phase of the IMSETY project. It describes the environment needed to perform the IT. When this environment is set up, all test cases must be executed according to their corresponding test procedures. After a test has been performed a report needs to be written.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Purpose . . . . .	1
1.2	Overview . . . . .	1
1.3	List of definitions . . . . .	1
1.4	List of references . . . . .	1
<b>2</b>	<b>Test plan</b>	<b>2</b>
2.1	Test items . . . . .	2
2.2	Features to be tested . . . . .	2
2.3	Test deliverables . . . . .	2
2.4	Testing tasks . . . . .	2
2.5	Environmental needs . . . . .	3
2.6	Test case pass/fail criteria . . . . .	3
<b>3</b>	<b>Test case specifications</b>	<b>4</b>
3.1	Common . . . . .	4
3.2	Server . . . . .	4
3.3	Client . . . . .	5
<b>4</b>	<b>Test procedures</b>	<b>6</b>
4.1	Client . . . . .	6
4.2	Server . . . . .	6
<b>5</b>	<b>Test reports</b>	<b>8</b>

# Document status sheet

<b>Document title</b>	Integration Test Plan
<b>Document identifier</b>	IMSETY/doc/ITP
<b>Author(s)</b>	Jeroen Keiren, Pim Vullers
<b>Version</b>	0.3
<b>Document status</b>	Internally approved

<b>Version</b>	<b>Date</b>	<b>Author(s)</b>	<b>Summary</b>
0.1 (Revision 1421)	30-05-2007	Jeroen Keiren, Pim Vullers	First version up for review
0.2 (Revision 1510)	04-06-2007	Jeroen Keiren, Pim Vullers	Fixed review remarks
0.3 (Revision 1991)	21-06-2007	Jeroen Keiren, Pim Vullers	Updated test reports chapter

# Document change report

<b>Document title</b>	Integration Test Plan
<b>Document identifier</b>	IMSETY/doc/ITP
<b>Date of changes</b>	21-06-2007

<b>Section number</b>	<b>Reason for change</b>
5	Updated text

# Chapter 1

## Introduction

### 1.1 Purpose

This document describes the plan for integration testing the developed software package against the architectural design as defined in the ADD [2]. The purpose of this integration test is to make sure that IMSETY complies with the design as described in the ADD [2]. These tests should be executed during the Detailed Design (DD) phase of the IMSETY project as described in the ESA software engineering standard [1].

### 1.2 Overview

Chapter 2 mentions the items to be tested together with the general criteria for the IT. A specification for each test case is given in chapter 3. The procedures for these test cases are explained in chapter 4. In chapter 5 the reports for all test cases are presented.

### 1.3 List of definitions

<b>AD</b>	Architectural Design
<b>ADD</b>	Architectural Design Document
<b>DD</b>	Detailed Design
<b>ESA</b>	European Space Agency
<b>IT</b>	Integration Test
<b>ITP</b>	Integration Test Plan
<b>SVVP</b>	Software Verification and Validation Plan

### 1.4 List of references

- [1] European Space Agency. ESA Software Engineering Standards. Technical report, ESA, February 1991. 1
- [2] Horus. Architectural Design Document, May 2007. 1, 2, 3
- [3] Horus. Integration Test Plan, June 2007. 2
- [4] Horus. Software Verification and Validation Plan, May 2007. 2
- [5] Eric Sommerlade, Michael Feathers, Jerome Lacoste, Baptiste Lepilleur, Bastiaan Bakker, and Steve Robbins. CppUnit Documentation. URL <http://cppunit.sourceforge.net/doc/lastest/>. 6

# Chapter 2

## Test plan

### 2.1 Test items

The software to be tested is IMSETY. Information about the architectural design of this system can be found in the ADD [2].

### 2.2 Features to be tested

IMSETY must meet the design as stated in the ADD [2]. The integration between each defined component should be subject to a test.

### 2.3 Test deliverables

Before the testing starts the following items must be delivered:

- SVVP [4]
- ADD [2]
- ITP [3], chapters 1, 2, 3 and 4
- IT input data
- IMSETY software package

After completing the testing the following items must be delivered:

- IT report (ITP chapter 5)
- IT output data
- Problem reports (if necessary)

### 2.4 Testing tasks

Before any testing in the IT phase can take place, the following tasks need to be done:

- Designing the integration tests.
- Tracing all test cases to components.
- All integrations of components need to be covered by test cases.

- Creation of IT input data.
- Ensuring that all environmental needs are satisfied for the IT.

When these tasks have been done an IT can be performed according to the procedures described  
50 in chapter 4.

## 2.5 Environmental needs

To be able to perform the IT the following resources are needed:

For the server:

- A computer running Linux with CppUnit, XML-RPC and MySQL++ installed.
- 55 • An MCC (-stub) available local or remote to the above described machine.
- An MCS (-stub) available local or remote to the above described machine.

For the client:

- A computer running Windows 2000/XP/Vista, Linux or OpenSolaris.
- Qt4 available on the local machine.

60 See also the constraints described in the ADD [2] in chapter 7.

## 2.6 Test case pass/fail criteria

Every test should describe the criteria that should be met to pass a specific test. An overall IT pass can only be achieved when all tests described in chapter 3 have been performed and passed.

## Chapter 3

# 65 Test case specifications

For the test case specifications see the respective test case specifications in the automated test framework. For the client this can be found in `src/client/test`, for the common files (mostly the administration packages) this is `src/common/test`, for the server the test cases can be found in `src/server/test`.

70 All tests can be found in these folders in the files `testname_itest.{h, cpp}`, where `testname` is the name found in the **Test name** column in the tables in the following sections.

We will however describe the various integration tests with respect to what integrations need to be carried out in the following sections. This testing will be done cumulative. This means that if two components pass an integration test they can be integrated together with other components.

75 This implies that the test should be performed in the given order within each part, and that the Common part must precede the others.

### 3.1 Common

Test name	Integrates
administration	Administration, Account, Experiment, Data, Command, Payload, Log, Satellite

### 3.2 Server

Test name	Integrates
mcc	MCC controller, MCC connectivity
mcs	MCS controller, MCS connectivity
admin_db	Administration, Database abstraction
bookings	Bookings, Booking
ext_queue	External queue, Upload
int_queue_upl	Internal queue, Upload
int_queue_downl	Internal queue, Download
80 scheduler	Scheduler, Internal Queue, External Queue
qm_scheduler	Queue manager, Scheduler
qm_admin_bookings	Queue manager, Administration, Bookings
qm_mcc	Queue manager, MCC controller
qm_mcs	Queue manager, MCS controller
qm_mcc_mcs	Queue manager, MCC controller, MCS controller
controller_client	Controller, Client connectivity
controller_admin_qm	Controller, Administration, Queue manager

### 3.3 Client

<b>Test name</b>	<b>Integrates</b>
admin_server	Administration, Server connectivity
observation_intervention	Observation & Intervention
experiment_oi_sched	Experiment, Observation & Intervention, Schedule
experiment_server	Experiment, Server connectivity
gui_admin_experiment	Graphical User Interface, Administration, Experiment

## Chapter 4

# Test procedures

85 In order to be able to automate the integration testing of IMSETY the authors of the integrated components are also responsible for writing the specific integration test belonging to the components. The integration tests need to be placed in the directories described further on in this chapter (either in the Client or the Server section). Note that testing components from the `src/common` directory will be carried out conform the test framework of the server.

90 By having the authors of the components also write the test belonging to the components, it should never occur that a component goes into the source code repository untested. This means that the author is also responsible for carrying out the tests before committing code.

Carrying out tests is different for the client and the server because at the client side we also need to be able to test the graphical user interface (GUI). Therefore we will also describe the test  
95 procedures for the client and the server separately.

### 4.1 Client

On the client side, for each test that needs to be run there is a `.pro` file in the `src/client/test/` directory, along with a `.cpp` file which implements a specific test. These tests cannot be easily aggregated in a single test suite because of shortcomings of the Qt test framework (if we would  
100 aggregate the tests in one test suite, a crash of one test would lead to the rest of the suite not being executed). In order to overcome this we have added a shell script which runs all individual tests, as the tests won't run from within the build system.

Note that all test names (denoted with *testname*) should be *name\_itest*, as this is needed for running the batch of tests.

105 Individual client tests can be run by going to the `src/client/test/` directory and there executing:

```
# scons testname  
# ./testname
```

A batch with all tests can be run by executing `src/client/test/integrationtest.sh`.

### 110 4.2 Server

Carrying out unit tests for the server is a lot simpler than for the client. For carrying out server integration tests we use the CppUnit [5] test framework. Whenever we mention the server, this also applies to the common files, substituting `server` by `common`.

For each integration test that needs to be run against the server, there is a `name_itest.{h, cpp}` file in the `src/server/test` directory.  
115

Full server tests can be run by going to the `src/server` or the `src/server/test` directory and running:

```
# scons integrationtest
```

When only a single test needs to be run this can be done by going to the `src/server/test`  
120 directory and running:

```
# scons name_itest
```

## Chapter 5

# Test reports

When a test report for an integration test is desired, for all components a test report can be generated from the test run. In case of the server and common files a test report is always automatically generated respectively in the `src/server/test/report` or the `src/common/test/report` folder. For the client a test report is always written to the screen. This can be overcome by redirecting test output to `src/client/test/report/testname_report.txt` by using the `--report` option, like `src/client/test/integrationtest.sh --report`.

Due to lack of time during the coding phase of the project no integration tests have been automated. Integration tests have been executed manually in order to verify the integration of subsystems. However this has been done in an ad-hoc manner, such that no test reports have been written.