Introduction to the ISA² Test Bed

Interoperability and conformance testing

ISA²

Europe Direct is a service to help you find answers to your questions about the European Union.

Freephone number (*):

00 800 6 7 8 9 10 11

(*) The information given is free, as are most calls (though some operators, phone boxes or hotels may charge you).

Cover picture: © Fotolia/lisaalisa_ill

More information on the European Union is available on the Internet (http://europa.eu).

Luxembourg: Publications Office of the European Union, 2017

ISBN 978-92-79-63502-1 (print)
ISBN 978-92-79-63501-4 (web)
doi:10.2799/726677 (print)
doi:10.2799/838544 (web)

© European Union, 2017

Reuse is authorised, provided the source is acknowledged.

Printed in Belgium

PRINTED ON ELEMENTAL CHLORINE-FREE BLEACHED PAPER (ECF)

DISCLAIMERS

The views expressed in this document are purely those of the authors and may not, in any circumstances, be interpreted as stating an official position of the European Commission.

The European Commission does not guarantee the accuracy of the information included in this handbook, nor does it accept any responsibility for any use thereof.

Reference herein to any specific products, specifications, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favouring by the European Commission.

All care has been taken by the author to ensure that s/he has obtained, where necessary, permission to use any parts of manuscripts including illustrations, maps, and graphs, on which intellectual property rights already exist from the titular holder(s) of such rights or from her/his or their legal representative.

This document has been carefully compiled by Trasys International, but no representation is made or warranty given (either express or implied) as to the completeness or accuracy of the information it contains. Trasys International is not liable for the information in this presentation or any decision or consequence based on the use of it. Trasys International will not be liable for any damages arising from the use of the information contained in this handbook. The information contained in this handbook is of a general nature and is solely for guidance on matters of general interest. This handbook is not a substitute for professional advice on any particular matter. No reader should act on the basis of any matter contained in this publication without considering appropriate professional advice.

This handbook has been drafted under the interoperability test-bed action, which is supported by the European Commission's ISA² programme. ISA² is a EUR 131 million programme supporting the modernisation of public administrations in Europe through the development of eGovernment solutions. More than 20 solutions are already available, with more to come soon. All solutions are open source and available free of charge to any interested pubic administration in Europe.

More information at ec.europa.eu/isa2

CONTENTS

1 **INTRODUCTION** 5

1.1 What is the purpose of this document? 5
1.2 Who is this document meant for? 5

2 **WHY AN INTEROPERABILITY TEST BED?** 7

2.1 The business need 7
2.2 The ISA² test bed service 7
2.3 How do I use the test bed? 9
2.4 Who is using it? 9
2.5 The GITB test bed software 10
2.6 Where to find the test bed and extra information? 10

3 **INSTALLING THE GITB TEST BED** 13

3.1 Step 1 – Install Docker 14
3.2 Step 2 – Install GITB test bed 14
3.3 Useful commands to manage your GITB test bed installation 15

4 **GITB TEST BED DEMONSTRATION** **19**

4.1 Demonstration test cases 19
4.2 Installing the test cases 21
4.3 Running the test cases 25

5 **GLOSSARY** **31**

6 **ANNEXES** **35**

Annex I. Risk statement 35
Annex II. Troubleshooting common GITB test bed issues 36

1 Introduction

1.1 What is the purpose of this document?

The purpose of this document is to introduce the reader to ISA²'s interoperability test bed service and the software that powers it. Test centre providers can also follow the instructions provided in this document to setup a local a test bed instance, for production or evaluation purposes, and get hands-on experience with it through provided demo scenarios.

In short you would want to read this document to:

- Find out what an interoperability test bed is, why you need it, and what ISA²'s interoperability test bed service is.
- Install the test bed software in your local environment.
- Better understand what a test bed can do through demo scenarios.

1.2 Who is this document meant for?

This document is meant primarily for two types of readers:

- **Organisation decision makers** that are faced with the need to provide interoperability testing services to their user community, which may consider a test bed as a means to achieve this.
- **Technical staff** that is tasked with installing a local test bed instance for evaluation purposes or for production use.

The document is split into two high level parts, a business-level introduction followed by a more technical installation guide. If you are an organisation decision maker and find that the first part applies to your needs the next Step would be to hand the remaining, more technical part to your staff for further evaluation. Having said this however, even for the technical sections, the reader is not required to have advanced technical expertise to follow the listed installation steps. What is expected is familiarity with issuing commands on a command console along with a basic understanding of terms such as "host" and "port".

2 Why an Interoperability Test Bed?

The purpose of this section is to explain what an interoperability test bed is and the needs that it fulfils. It also aims in acquainting you with ISA²'s central and reusable interoperability test bed service and the GITB test bed software that realises it but that can also be locally installed in your organisation.

2.1 The business need

European and national public administrations have developed a multitude of IT systems which are faced with the increasingly common requirement of working together. The goal is to enable cross-border administrative cooperation that ultimately results in empowering citizens and businesses with services that are not restricted to geographic boundaries. This cooperation represents however a challenging task, considering the range of different technologies used to implement such systems but also, at higher levels, the diverse organisational structures they support and the varying semantics that may be applied to the exchanged information.

The European Interoperability Framework (EIF[1]) defines four levels of interoperability – legal, organisational, semantic and technical – that should be taken into account when designing interoperable IT solutions to deliver digital public services. Establishing interoperability is a process that requires alignment on all these levels and one that poses a significant challenge on testing to ensure the required alignment.

This testing challenge is in fact twofold, on the one hand on the system under development that needs to run interoperability tests as early as possible in its elaboration phase, and on the other hand on service providers or peer systems that need a means to expose IT services for testing without impacting their production system operations.

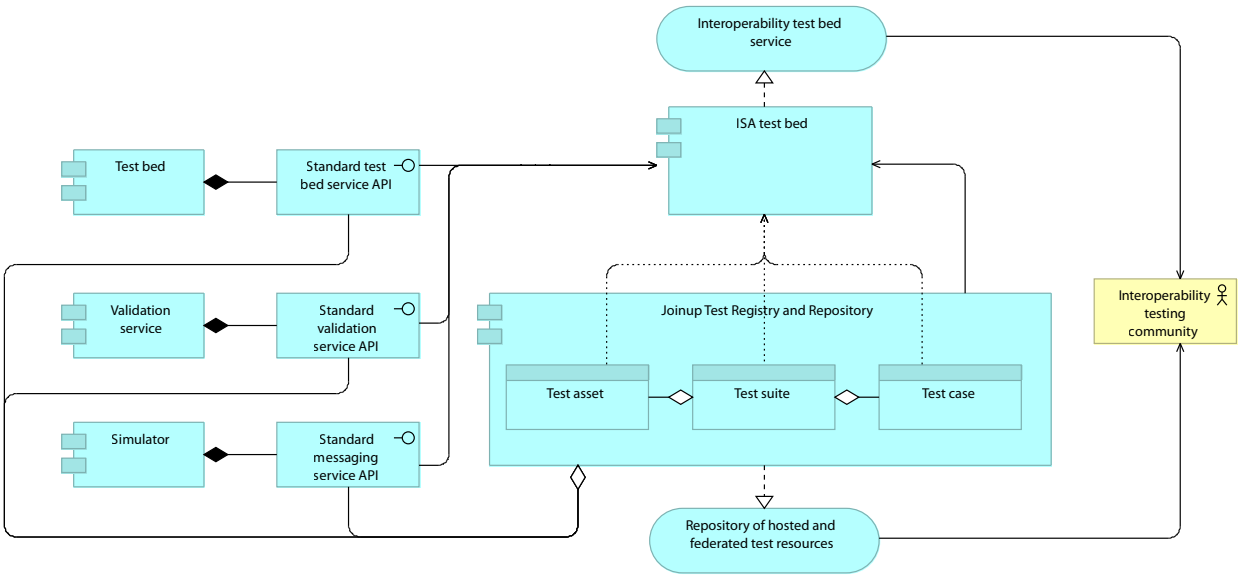
2.2 The ISA² test bed service

The ISA² interoperability test bed provides generic testing facilities to initiatives that create interoperability solutions in a cross-border context or linked to European Legislation. It offers:

- A test bed service that can run conformance and interoperability tests.
- The possibility to host reference implementations of specifications and services for clients to test against.
- A test registry and repository to store test artefacts (assertions, test cases, validation schemas etc.) and federate test services (validation services, simulator services etc.).

1 European Interoperability Framework: http://ec.europa.eu/isa/documents/eif_brochure_2011.pdf

Figure 1: The ISA² interoperability test bed service



The ISA² interoperability test bed is a central, reusable service that can be used both for interoperability and conformance testing, ranging from the verification of complex message exchanges as complete conversations, to validation of content, received through a variety of communication channels. Tests are visualised through an intuitive user interface that allows a tester to follow exchanged messages between SUTs, simulators and reference implementations, inspect and export their content, and analyse the reports of failed validations. Test progress and reports are also exposed through standardised, machine-readable formats.

The world of test beds, test services, validators and simulators is a fragmented one, with different organisations exposing distinct services per domain, often overlapping in terms of features and purpose. The ISA² interoperability test bed is designed to address this by accepting this fragmentation as a fact and seeking to enable as much as possible the reuse of existing services. This is achieved by means of standardised service APIs used to control the execution of a remote test, validate content, simulate responses, or support diverse communication means. This standardisation support is provided through the concept of service compliance established by the CEN GITB Workshop Agreement[2], which also is the source of the ISA² test bed software.

What should it not be used for?

Given that the test bed's focus is interoperability and conformance testing there are some cases that it is less suited for:

- Functional or regression testing: The test bed's focus does not lay on finding internal bugs.
- Performance testing: A test bed should not be used to stress test client systems.
- Penetration testing: To enhance connectivity and ease of use, security is relaxed and exchanged data is exposed.

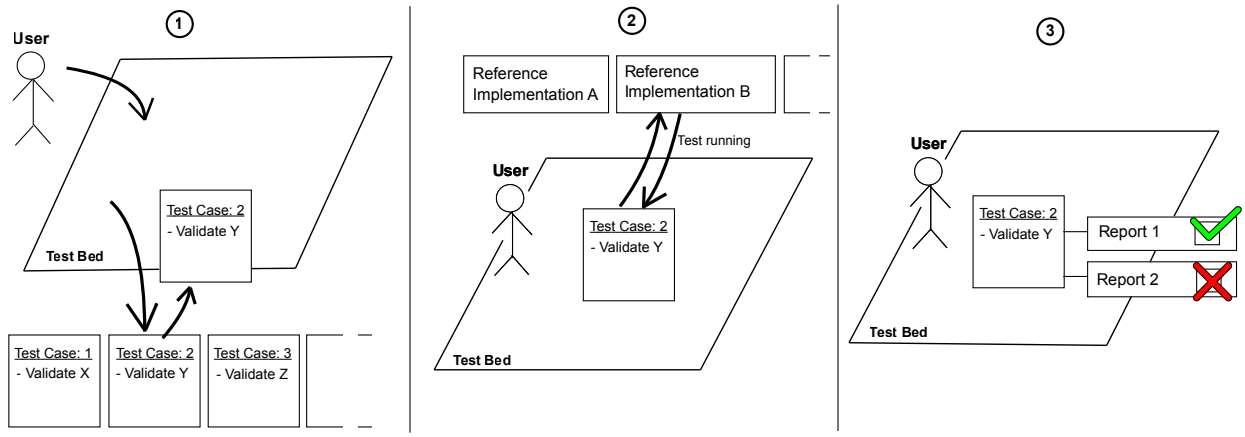
2 CEN GITB WA: http://www.cen.eu/work/areas/ict/ebusiness/pages/ws-gitb.aspx

2.3 How do I use the test bed?

Using the testbed can be summarised in three steps:

1. A user logs onto the test bed platform and selects one from a set of reusable test cases.
2. The test case executes, testing the user's system against simulators and validators. Tests can range from the verification of complex message exchanges as complete conversations, to validation of content, received through a variety of communication channels.
3. During execution and upon completion, exchanged messages and validation results are displayed and compiled in a test report that is stored for later reference.

Figure 2: Using the test bed



2.4 Who is using it?

Testing for electronic invoice exchange using the European Commission's CEF e-Invoicing DSI[3] is offered to its user community across Member States to address invoice content validation. Invoices can currently be provided for automated validation by means of web form upload, email and (GITB-compliant) web service call. At the time of writing extensions are also planned to include invoice delivery as AS2[4] and AS4[5] messages using the CEF e-Delivery DSI.

2.4.1 Who else can potentially use it?

Initiatives (public administrations, funded projects etc.) with relevance to (cross-border) interoperability can apply to have their reference implementations and test suites installed in the test bed run by ISA², for their respective communities to use. Software owners, including privately held companies who build solutions as part of a larger programme for which they would need or want to undertake interoperability testing, can then use the test bed to test their products' connectivity against supported reference implementations, or the conformance of their messages against supported specifications. The testing of software providers' solutions through the test bed results in benefiting the businesses and citizens that will eventually use the envisioned services, in that the interoperability of the building blocks used to realise them is rigorously tested.

Anyone, including those not eligible to directly use the ISA² test bed service, is free to download the test bed software and install a local instance in their own premises. The installation process for such a local instance is detailed in Chapter 3, "Installing the GITB test bed". Additionally, Annex I, "Risk statement", informs about risk points related to use of the GITB software.

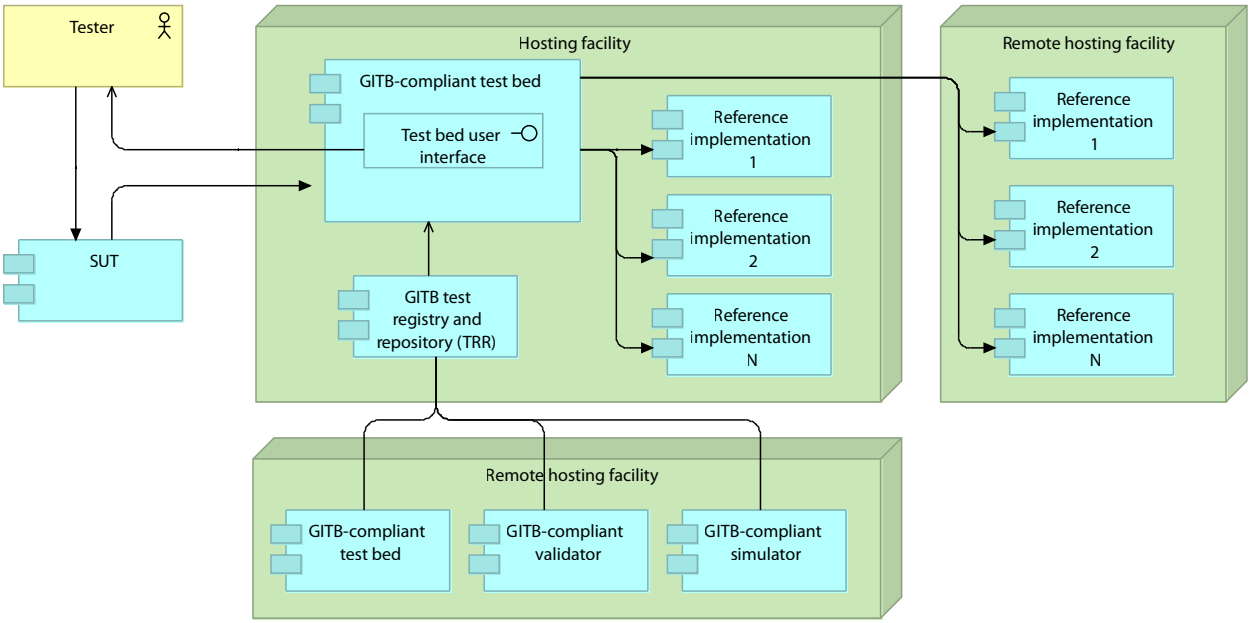
3 CEF building blocks on Joinup: https://joinup.ec.europa.eu/community/cef
4 AS2: https://www.ietf.org/rfc/rfc4130.txt
5 AS4 profile of ebMS 3.0: http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/profiles/AS4-profile/v1.0/os/AS4-profile-v1.0-os.pdf

2.5 The GITB test bed software

The GITB project represents a CEN standardisation initiative funded by the European Commission's DG GROW to provide the specifications for a generic interoperability test bed and their implementation in the form of the GITB test bed software.

The project's purpose is not only to identify what is needed to support testing but also to make test bed elements modular and reusable. This reusability extends the confines of a single test bed, in that testing resources and components can be remotely reused as long as they follow the same specifications. This results in the concept of **GITB compliance**, which includes (among other types of specification compliance) a set of standard web service APIs for test beds, validation services and simulators. These compliant web services are furthermore federated in the Test Registry and Repository (TRR)[6], implemented in the European Commission's Joinup platform, as a single point of reference.

Figure 3: The GITB test bed software used in the ISA2 interoperability test bed service



In terms of architecture, the GITB test bed serves as the intermediate between testers and SUTs on one hand, and the reference implementations or simulators that realise the offered test services on the other. Reference implementations can range from embedded test bed components to separate application instances that can either be locally hosted on the test bed's infrastructure or remotely on external hosting facilities.

2.6 Where to find the test bed and extra information?

For more information on ISA²'s test bed service and its underlying software you are invited to visit the test bed's Joinup page[7]. For more general information on the ISA² Action 2016.25 that drives the work on making this service available have a look on the Action's web page[8].

6 GITB TRR in Joinup: https://joinup.ec.europa.eu/catalogue/repository/gitb-trr
7 The action's Joinup page: https://joinup.ec.europa.eu/asset/itb/description
8 ISA² Action 2016.25: http://ec.europa.eu/isa/actions/isa2/08-supporting-instruments-for-public-administrations/25action_en.htm

3 Installing the GITB test bed

Why install a test bed locally?

The current section details the installation process for the GITB test bed software, the software that is used to power the ISA² test bed service. You would be interested in installing the GITB test bed in your premises if:

- You are considering using the ISA² test bed service but first want hands-on experience to determine if it covers your needs.
- You want to run in your local premises a GITB test bed instance, either because you are not eligible or do not want to use the ISA² test bed service.
- You want to try hands-on the GITB test bed software to get a better understanding of what a test bed can be used for. The package includes demo test cases to try out for that purpose.

To facilitate the installation of a local test bed instance, the GITB test bed's components have been made available as Docker[9] container images.

Docker is an open-source project that automates the deployment of applications inside software containers by providing an additional layer of abstraction and automation of operating-system-level virtualisation on Linux. Once Docker is installed on the machine chosen to host the test bed, the container images can be deployed as-is without any concern on lower level dependencies or necessary software packages. While Docker targets Linux platforms the test bed can also run in a lightweight Linux virtual machine for Windows or Mac designed specifically for Docker.

The installation process of the test bed is simple and fast, with most time involved relating to downloading each component. Putting aside download time, the installation can be completed in a matter of minutes.

9 Docker: https://www.docker.com/

3.1 Step 1 – Install Docker

To install Docker on the machine chosen to host the test bed, follow Docker's "Get Started" documentation at https://www.docker.com/. The test bed has been validated for Docker version 1.10.1 as a minimum.

Note for production use

If you are installing the GITB test bed for production use you need to be aware of how it handles ports. The test bed opens dynamically ports on its host machine for simulated test case actors, one set of ports per test session. Exposing a large number of ports through Docker using its default configuration can be time and memory consuming. To address this it is recommended to run Docker with hairpin NAT enabled, rather than relying on a per-port userland proxy, by setting the following value in Docker's configuration file (located at e.g. `/etc/default/docker` for an Ubuntu distribution):

```
DOCKER_OPTS="--userland-proxy=false"
```

If you are intending to run the GITB test bed for demo purposes then you can skip this configuration step.

3.2 Step 2 – Install GITB test bed

Note for demo use

The instructions in this section guide you through the installation of the GITB test bed on your local environment, resulting in a fully functional but empty test bed installation.

If your goal is to install the test bed primarily for demo purposes then it would be easier to refer to Chapter 4.2.1, "Demo installation from scratch", which results in an installation already configured to run a set of simple demos.

The GITB test bed is composed of a series of components, namely:

- A frontend server to provide the user interface.
- A backend server to provide the testing capabilities.
- A MySQL[10] relational database to record users, test cases and reports.
- A Redis[11] cache server.

To install and start the test bed's components execute the following commands:

```
1  docker network create gitb-net
2  docker create -v /gitb-repository --name gitb-repo ubuntu
3  docker run --name gitb-mysql --net=gitb-net -e MYSQL_ROOT_PASSWORD=root -p
       3306:3306 -d isaitb/gitb-mysql
4  docker run --name gitb-redis --net=gitb-net -d redis:3.0.7
5  docker run --name gitb-srv --net=gitb-net -p 8080-8090:8080-8090 -e gitb.mes-
       saging.server-ip-address=$HOSTNAME -d isaitb/gitb-srv
6  docker run --name gitb-ui --net=gitb-net -p 9000:9000 --volumes-from gitb-re-
       po -d isaitb/gitb-ui
```

Note that in the commands above, the variable `$HOSTNAME` should be set (or replaced in the command itself) with the publicly accessible host name of the machine hosting the test bed.

10 MySQL: https://www.mysql.com/
11 Redis: http://redis.io/

The following table explains the purpose of each command and various configuration options to be considered.

#	Description
1.	Creates a "bridged network" to allow the individual containers to securely communicate. This is passed to the appropriate containers using the "--net" flag.
2.	Creates a storage volume to hold test suite files that are read by the GITB frontend.
3.	Creates a pre-populated MySQL database listening on port 3306 with a root password of "root". Both the port mapping and the password can be changed without impact to the test bed. For example passing "-e MYSQL_ROOT_PASSWORD=mypass -p 5506:3306" would set the password to "mypass" and expose the database on port 5506.
4.	Creates a Redis cache server instance used internally by the GITB frontend.
5.	Creates the GITB test bed backend used to drive test execution. The "gitb.messaging.server-ip" argument needs to be passed the host name of the server running the test bed (i.e. the Docker host). This needs to be a host name that can be called by test clients (i.e. a public IP address if clients are to connect over the internet) as it will be used automatically by the test bed for its simulated actors. Regarding exposed ports, this command opens the ports within the 8080-8090 range which is more than adequate for demo purposes. If the test bed is to be used for production purposes a significantly larger port range needs to be foreseen considering that each parallel test session with simulated actors requires remotely accessible ports. To allow for example 300 open ports (the currently configured maximum) provide "-p 8080-8380:8080-8380".
6.	Creates the GITB frontend that provides the test bed's web user interface. The user interface can be accessed at http://dockerhost:9000/, replacing "dockerhost" with the appropriate host name. Note that, as in the previous commands, the port 9000 can be replaced if needed. For example passing "-p 80:9000" will expose the user interface on the default port 80.

3.3 Useful commands to manage your GITB test bed installation

3.3.1 Shutdown the test bed

Shutting down the test bed basically requires its Docker containers to be stopped. This is achieved using the following command:

```
docker stop gitb-ui gitb-srv
gitb-redis gitb-mysql
```

Note here that the network (`gitb-net`) and data volume (`gitb-repo`) initially created during the installation do not need to be shut down as they do not actively consume resources.

3.3.2 Restart the test bed

If not already the case, shut down the test bed as described in Chapter 3.3.1, "Shutdown the test bed". Once stopped, issue the following commands:

```
docker start gitb-mysql gitb-
redis
```

```
docker rm gitb-ui gitb-srv
```

```
docker run --name gitb-
       srv --net=gitb-net -p
       8080-8090:8080-8090
        -e gitb.messag-
       ing.server-ip-
       address=$HOSTNAME -d
       isaitb/gitb-srv
```

```
docker run --name gitb-
       ui --net=gitb-net -p
       9000:9000  --volumes-
       from gitb-repo -d
       isaitb/gitb-ui
```

In the commands listed above, remember to replace `$HOSTNAME` with the publicly accessible host name of the machine hosting the test bed.

Note here that the database (`gitb-mysql`) and cache server (`gitb-redis`) are restarted using the "`docker start`" command to maintain their previous state. This is not the case of the GITB frontend and backend components (`gitb-ui` and `gitb-srv`) which are first removed and re-ran. The "`docker run`" commands in this case would be identical to the ones used during the test bed's initial installation. It is interesting to highlight that the test bed's frontend and backend are fully stateless and can be discarded without concern. This is especially interesting in case updated container versions are available.

3.3.3 Monitor the test bed components

Monitoring the test bed during its operation is most effectively achieved by following the desired component's log trace. For example, to monitor the logs of the test bed's service backend (`gitb-srv`) issue the following command:

```
docker logs -f gitb-srv
```

In this example providing the "`-f`" flag will result in following (tailing) the component's log output (rather than printing them once).

3.3.4 Uninstall all test bed components

The first Step in a full uninstallation would be to stop all test bed components as described in Chapter 3.3.1, "Shutdown the test bed". The second Step is the removal of all test bed components, including the test bed network and data volume. This is achieved using the following commands:

```
docker rm gitb-ui gitb-srv
gitb-redis gitb-mysql gitb-
repo
```

```
docker network rm gitb-net
```

At this point the test bed can be quickly re-installed by issuing the installation commands as described in Chapter 3.2, "Step 2 – Install GITB test bed". Given however that test bed's database (`gitb-mysql`) and data volume (`gitb-repo`) have been removed, the test bed's state (i.e. test cases, reports, users, systems) will be reset. This result could however be interesting in case you are experimenting with the test bed and would like to revert to a clean state.

4 GITB test bed demonstration

The current section introduces two demonstration test cases to help understand what can be achieved with the test bed. They are selected as scenarios that are simple to understand but also as a showcase of the different capabilities the test bed has to offer.

The subsections that follow present each demo test case and guide you through the steps needed to set them up and run them on a locally installed test bed instance.

4.1 Demonstration test cases

4.1.1 Validation of a UBL Invoice

The Universal Business Language[12] (UBL), from OASIS[13], defines a library of standard electronic XML business documents such as purchase orders and invoices. The purpose of this test case is to allow the validation of such an invoice's content that is provided to the test bed by a user through a web form upload. Once uploaded, the invoice is validated by the test bed:

- In terms of structure, using the UBL XML Schema.
- In terms of content using Schematron rules that implement the BII Core T10 Invoice Transaction Structure and BII Rules T10 Invoice Business Rules.

To realise this test case the test bed first prompts the user to provide the UBL invoice and then proceeds with its validation. In this case the test bed does not need to simulate any additional systems as test case actors.

Demo summary	
What is the demo about?	Validation of a UBL XML invoice uploaded by a user through a web form.
Why is this interesting as a demo?	✓ Shows how XML content can be validated for conformance to a specification. ✓ Shows how users can interact with a test bed to provide it input. ✓ Shows a test case including only validation, without additional simulated systems by the test bed.

12 UBL: http://ubl.xml.org/
13 OASIS: https://www.oasis-open.org/

4.1.2 Monitoring a WMS message exchange

The Web Map Service[14] (WMS) is a standard protocol from the Open Geospatial Consortium[15] for serving georeferenced map images over a simple HTTP interface. WMS 1.1.1 defines two mandatory operations that any WMS server is required to implement:

- Operation "GetCapabilities", to enable a client to discover what is supported by the WMS server, notably the layers representing the maps.
- Operation "GetMap", to enable the retrieval of specific map layers.

The purpose of this test case is to allow clients of a WMS server to test that they correctly query the server's capabilities and then, based on what is reported by the server as supported, request a specific map layer. This test case assumes that the system we want to subject to testing is such a client that would, as part of its production operations, need to query a WMS server. Exposing this test case allows this system to be tested for interoperability before production rollout and without impacting an operational WMS server.

To realise this test case an actual WMS server instance is used as a reference implementation of the WMS standard which is proxied by the test bed to capture and validate exchanged messages. Upon test case initialisation the client system, the SUT, is provided with the address to be used to contact the WMS server. This address in reality is not that of the WMS server itself but of the test bed's proxy that is initialised for this test execution. Once this address is configured in the SUT, it proceeds to call successively the GetCapabilities and GetMap operations, requesting a map layer that is supported by the server. The role of the test bed is to ensure that the map layer requested through the GetMap call is included in the capabilities reported by the server through its response to the previous GetCapabilities call.

Demo summary	
What is the demo about?	Validation of the sequence and coherence of messages sent to a WMS server to retrieve a map layer's image.
Why is this interesting as a demo?	✓ Shows how the test bed can be used to validate the coherence of a message conversation, spread over multiple messages. ✓ Shows how a client system can be configured to connect as a SUT to the test bed. ✓ Shows how the test bed can proxy a real system, acting as a standard's reference implementation, to monitor and validate its communication with a SUT.

14 WMS: http://www.opengeospatial.org/standards/wms
15 OGC: http://www.opengeospatial.org/

4.2 Installing the test cases

4.2.1 Demo installation from scratch

This section details how the provided demos can be setup in an environment where the GITB test bed's components have not already been installed. The only prerequisite is the installation of Docker itself as described in Chapter 3.1, "Step 1 – Install Docker".

Alternatively, if you already have a GITB test bed installation and want to setup the demos please refer to Chapter 4.2.2, "Demo installation on an already existing test bed".

To simplify the setup process of the demos, their required resources have been pre-packaged as Docker container images. These demo-specific images include:

- The WMS server instance described in Chapter 4.1.2, "Monitoring a WMS message exchange".
- Prepopulated filesystem and database images with the already loaded test cases.

To install and start the test bed's components, including the discussed demo resources, execute the following commands:

```
docker network create gitb-net
```

```
docker create -v /gitb-repository --name gitb-repo isaitb/gitb-repo-demo
```

```
docker run --name gitb-mysql --net=gitb-net -e MYSQL_ROOT_PASSWORD=root  -p 3306:3306 -d isaitb/gitb-mysql-demo
```

```
docker run --name gitb-redis --net=gitb-net -d redis:3.0.7
```

```
docker run --name gitb-srv --net=gitb-net -p 8080-8090:8080-8090  -e gitb.messaging.server-ip-address=$HOSTNAME -d isaitb/gitb-srv
```

```
docker run --name gitb-ui --net=gitb-net -p 9000:9000  --volumes-from gitb-repo -d isaitb/gitb-ui
```

```
docker run --name geoserver --net=gitb-net -p 10080:8080 -d winsent/geoserver:2.8
```

Note that, as in the case of the test bed's installation described in Chapter 3.2, "Step 2 – Install GITB test bed", the variable `$HOSTNAME` should be set (or replaced in the command itself) with the publicly accessible host name of the machine hosting the test bed.

Details on the purpose and options of the previous commands can be found in Chapter 3.2, "Step 2 – Install GITB test bed". All commands are identical to those used to setup an empty test bed instance with the following exceptions:

- The second and third commands above, used to setup the test bed's file system repository and database are based on adapted demo images (`isaitb/gitb-repo-demo` and `isaitb/gitb-mysql-demo` respectively).
- The final command is used to install and start an open source GeoServer[16] instance as a WMS-compliant server for the purposes of the demo described in Chapter 4.1.2, "Monitoring a WMS message exchange".

16 GeoServer: http://geoserver.org/

4.2.2 Demo installation on an already existing test bed

This section details how the provided demos can be setup in an already existing GITB test bed installation.

Alternatively, if you do not already have a GITB test bed installation and are looking for the simplest way to setup the demos please refer to Chapter 4.2.1, “Demo installation from scratch”.

Step 1 – Install the WMS server

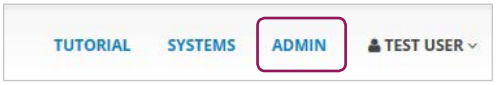
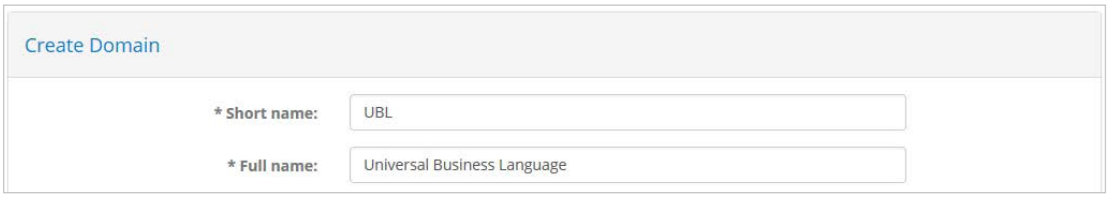
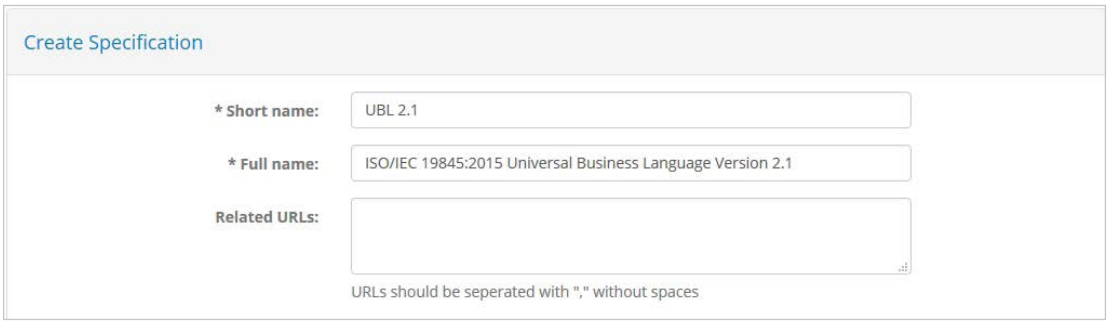
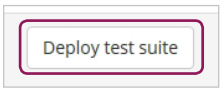
To enable the demo described in Chapter 4.1.2, “Monitoring a WMS message exchange”, a WMS server instance needs to be installed. This represents the only additional software component that needs to be setup before we can start testing.

To install and start the WMS server instance issue the following command:

```
docker run --name geo-
server --net=gitb-net -p
10080:8080 -d winsent/geo-
server:2.8
```

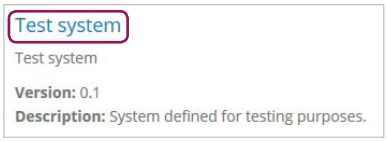
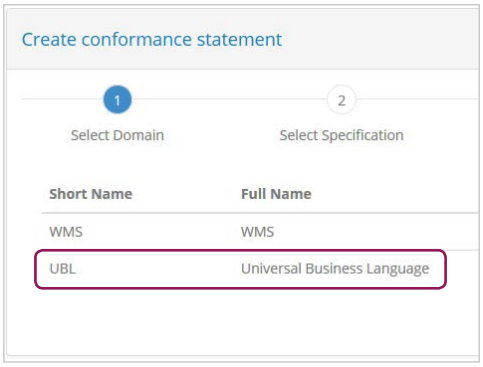
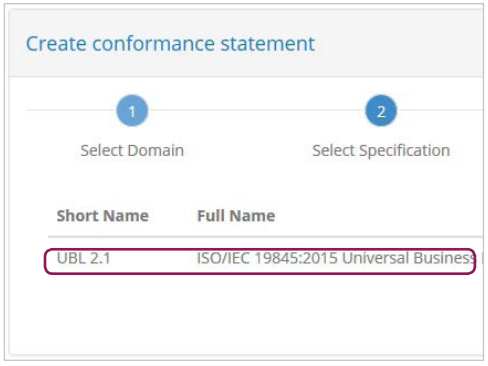
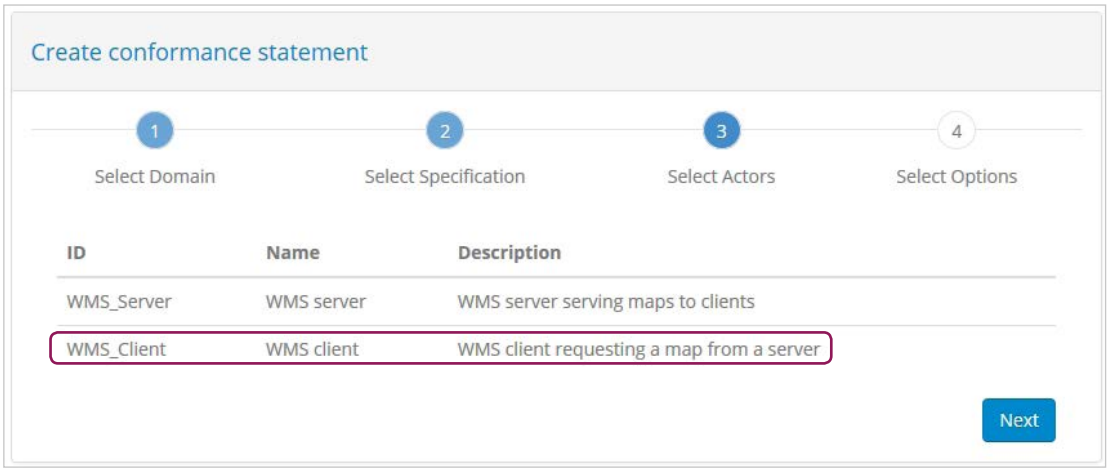
Step 2 – Add the demo test cases to the test bed

This Step requires connection to the test bed’s interface as an administrator in order to register the demo test cases and make them available to the test bed’s users. The sequence of actions below would need to be repeated for each new test case we are adding to the test bed, and in the case of the demos, once for the UBL-based demo described in Chapter 4.1.1, “Validation of a UBL Invoice” and once for the WMS-based demo described in Chapter 4.1.2, “Monitoring a WMS message exchange”.

#	Description
1	Log-in as a test bed administrator with the account “test@test.com” and password “test”.
2	Go to the “Admin” section at the top of the page, select “Domains” and create a domain.  
3	After saving the new domain select it from the table and create a specification. Provide relevant short and full names; e.g. “UBL 2.1” / “ISO/IEC 19845:2015 Universal Business Language Version 2.1” or “WMS 1.1.1” / “WMS Version 1.1.1”. 
4	After saving the new specification select it from the table, click “Deploy test suite” and upload the zip archive containing the test suite and test case.  The zip archives for each demo test case are available on Joinup: UBL: https://joinup.ec.europa.eu/svn/cen-ws-gitb/samples/UBL_invoice_validation.zip WMS: https://joinup.ec.europa.eu/svn/cen-ws-gitb/samples/WMS.zip

Step 2 – Select the test cases to test against

This Step reflects the actions that the test bed’s users would perform to select the test cases they wish to test their system against. As in the previous step, the actions listed below would need to be repeated for each test case you want to test against.

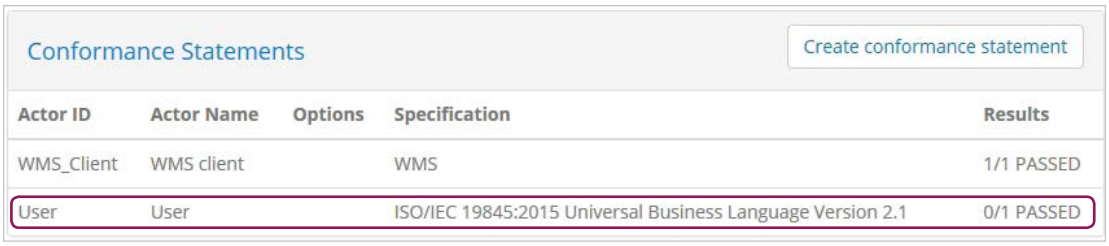
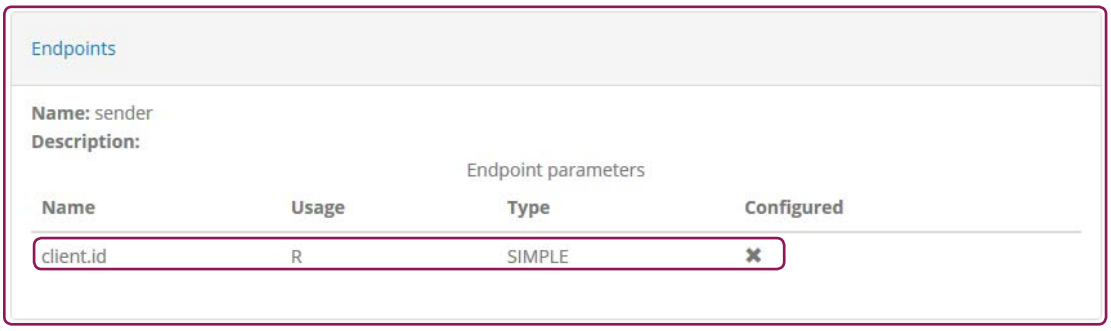
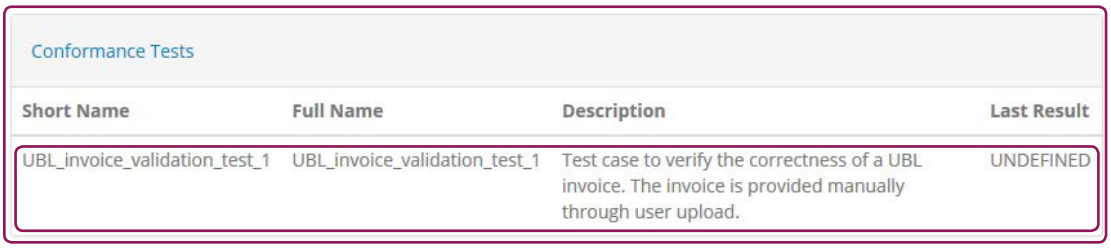
#	Description
1	Log-in as a vendor administrator with the account “admin@test.com” and password “test”. If already logged in from the previous Step you will need to first log-out.
2	Select the system that you want to test for by clicking on its name.  Now select “Conformance Statement” from the left menu and click “Create conformance statement”. In the next two steps of the wizard choose the relevant domain and specification you want to test for; i.e. “UBL” and “UBL 2.1” or “WMS” and “WMS 1.1.1”.   
3	In the third Step of the wizard you select the actor that you want to have your system test as. In the case of the UBL demo there is only one, called “User”; for the WMS demo (shown below) choose “WMS_Client” – and complete the statement creation sequence by clicking “Next” and “Finish”.

4.3 Running the test cases

4.3.1 UBL Invoice validation

The series of steps below describe how to execute the test case for UBL invoice validation described in Chapter 4.1.1, "Validation of a UBL Invoice".

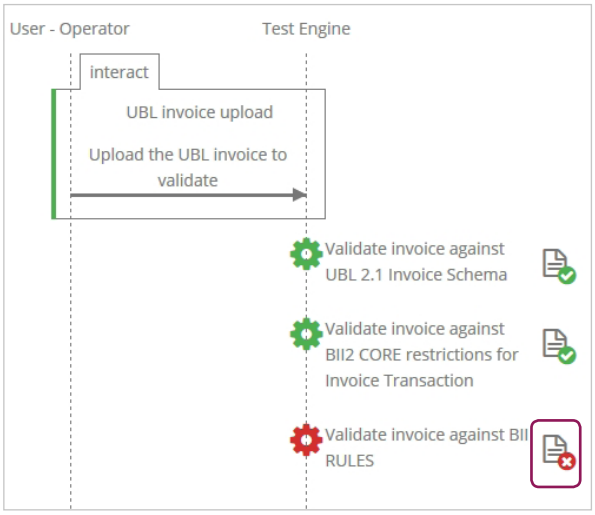
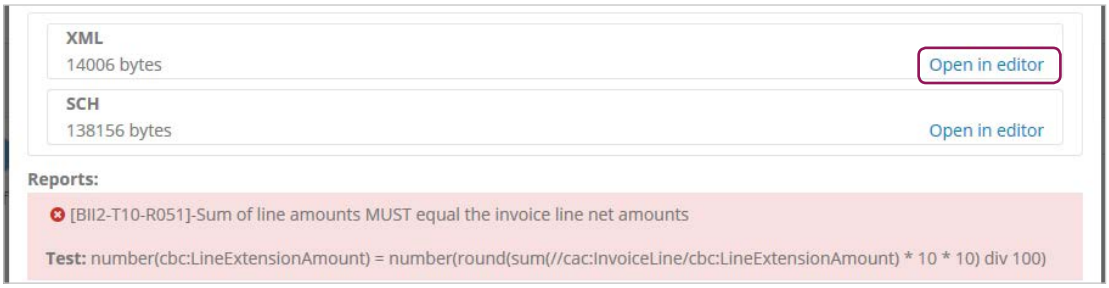
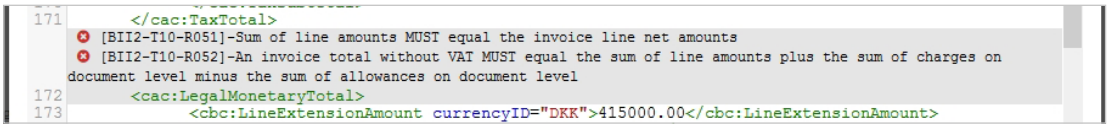
Step 1 – Test case selection

#	Description
1	Log-in with the SUT vendor administrator account "admin@test.com" and password "test".
2	Select the test system "Test system", and then "Conformance Statement" from the left menu. From the listed conformance statements select the one corresponding to the "User" actor within the UBL specification.  Conformance Statements Create conformance statement Actor ID | Actor Name | Options | Specification | Results WMS_Client | WMS client | WMS | 1/1 PASSED User | User | ISO/IEC 19845:2015 Universal Business Language Version 2.1 | 0/1 PASSED
3	In the "Endpoints" section you provide the configuration that the test bed expects from the SUT. For this simple test case click on "client.id" and provide a value of "0" (any value will do). Note that once provided, the test bed will remember this and not require it again.  Endpoints **Name:** sender **Description:** Endpoint parameters Name | Usage | Type | Configured client.id | R | SIMPLE | ✖
4	The "Conformance Tests" section at the bottom of this screen lists the test cases in the selected test suite (only one in this case). Select the conformance test named "UBL_invoice_validation_test_1".  Conformance Tests Short Name | Full Name | Description | Last Result UBL_invoice_validation_test_1 | UBL_invoice_validation_test_1 | Test case to verify the correctness of a UBL invoice. The invoice is provided manually through user upload. | UNDEFINED

Step 2 – Test case initiation

#	Description
1	Click “Next” on the system configuration screen; this acts as a confirmation that expected configuration from the user is complete.
2	Click “Next” on the preliminary initialisation screen as there is nothing specific to initialise.

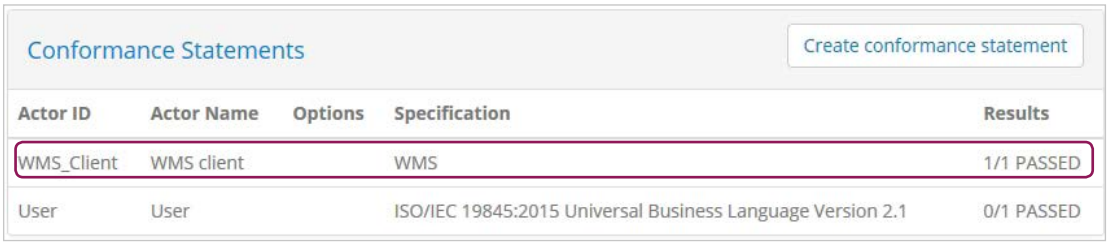
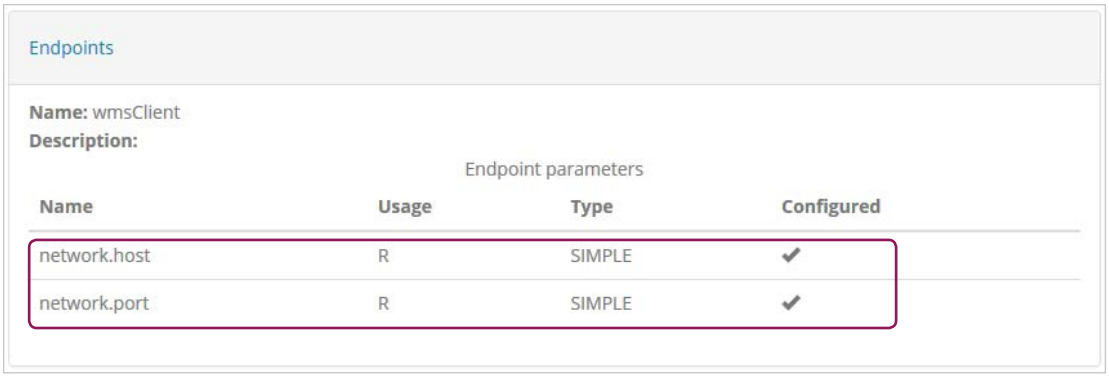
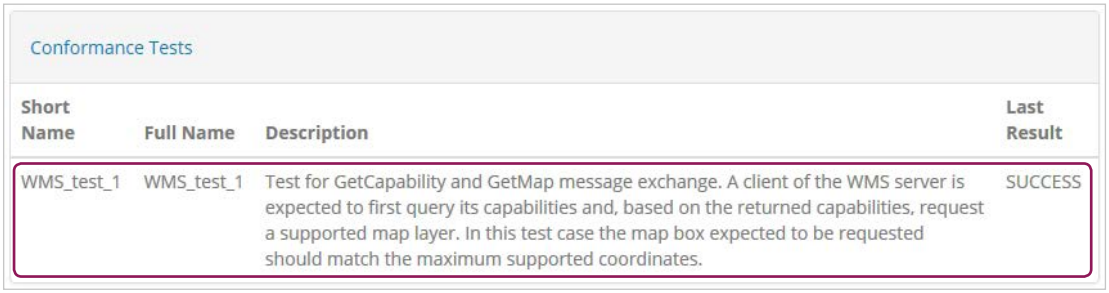
Step 3 – Test case execution

#	Description
1	Click “Start” to run the test. You will immediately be prompted to upload a UBL invoice for validation. Sample UBL invoices, both valid and invalid that can be used to demonstrate the test bed’s reporting capabilities are available in Joinup: https://joinup.ec.europa.eu/svn/cen-ws-gitb/samples/UBL_invoices.zip
2	 The test bed presents the validation results of the test case in a graphical way. Arrows indicate an exchanged message or, in this case, the upload action. The cogs on the other hand show the validation steps performed by the test bed. Steps coloured in green indicate success, whereas red indicates failures. In all cases the document icons next to each Step can be clicked to inspect the full details of the validation. Clicking for example a failed validation Step would list the overview report of the validation including reported warnings and errors. The input sources, in this case the XML invoice and Schematron rule file, can also be inspected clicking the “Open in editor” link.  Finally, clicking on a specific error message will result in displaying the offending section of the received UBL invoice. 

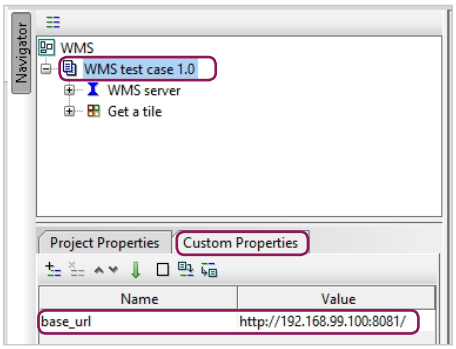
4.3.2 WMS message exchange

The series of steps below describe how to execute the test case for a client system of a WMS server described in Chapter 4.1.2, "Monitoring a WMS message exchange".

Step 1 – Test case selection

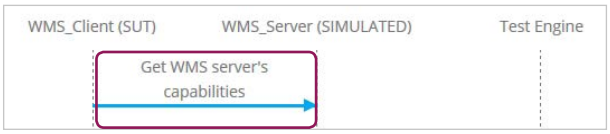
#	Description
1	Log-in with the SUT vendor administrator account "admin@test.com" and password "test".
2	Select the test system "Test system", and then "Conformance Statement" from the left menu. From the listed conformance statements select the one corresponding to the "User" actor within the UBL specification.  Conformance Statements Create conformance statement Actor ID | Actor Name | Options | Specification | Results WMS_Client | WMS client | WMS | 1/1 PASSED User | User | ISO/IEC 19845:2015 Universal Business Language Version 2.1 | 0/1 PASSED
3	In the "Endpoints" section you provide the configuration that the test bed expects from the SUT which in this case is the address it will connect from. This is used by the test bed to isolate this test session by ignoring messages received from other addresses once the test session starts. In other words, once a test session is in progress, the test bed will allow it to progress only after receiving messages from the address that has been configured in this step. For the "network.port" you may provide any port value (e.g. "80") as this will not be used in this specific case. For the "network.host" you need to provide the IP address your SUT is connecting from. If testing against a remote test bed over the internet this would be your IP address as viewed by the remote test bed. In this case however, assuming you will be connecting locally using the provided SoapUI project (see subsequent steps), you need to provide your local IP address as expected by the host running the test bed[1].  Endpoints **Name:** wmsClient **Description:** Endpoint parameters Name | Usage | Type | Configured network.host | R | SIMPLE | ✔ network.port | R | SIMPLE | ✔
4	The "Conformance Tests" section at the bottom of this screen lists the test cases in the selected test suite (only one in this case). Select the conformance test named "WMS_test_1".  Conformance Tests Short Name | Full Name | Description | Last Result WMS_test_1 | WMS_test_1 | Test for GetCapability and GetMap message exchange. A client of the WMS server is expected to first query its capabilities and, based on the returned capabilities, request a supported map layer. In this test case the map box expected to be requested should match the maximum supported coordinates. | SUCCESS

Step 2 – Test case initiation

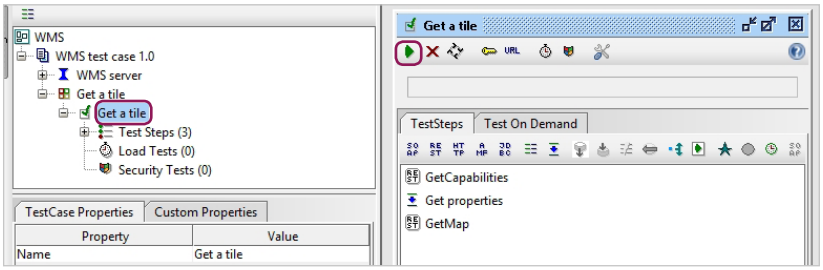
#	Description
1	For the purposes of the demo we need a simple approach to provide the SUT that would call the WMS server. A SoapUI[2] project is available for this that is configured to make in sequence all expected calls to the WMS server. This SoapUI project is available to download from Joinup: https://joinup.ec.europa.eu/svn/cen-ws-gitb/samples/WMS-soapui-project-1.0.xml
2	Click "Next" on the system configuration screen in the test bed; this acts as a confirmation that expected configuration from the user is complete.
3	 The following screen, regarding preliminary initialisation, gives the address and port for the WMS server, where the client needs to send its messages. In other words, this is the configuration that is reported to the user by the test bed that the user is expected to configure in his SUT. It is worth noting that these configuration values are distinct for each test session; if a parallel testing session is to be launched the test bed will assign a distinct port to ensure test executions are isolated. To configure these values in our SoapUI project, select within SoapUI the "WMS test case 1.0" project, select the tab "Custom properties", and provide as the value of the "base_url" property the address in the form of http://network.host:network.port/. Once your SoapUI project is configured you can, in the test bed, proceed to start the test case by clicking "Next".

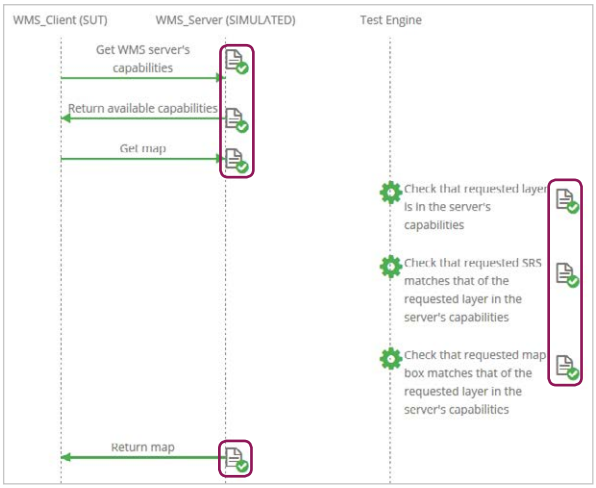
Step 3 – Test case execution

#	Description
1	Click “Start” to have the test bed begin the test case’s execution. Once started the test bed will expect the initial “GetCapabilities” request from the WMS client. This is reflected by the arrow coloured in blue that indicates an expected but not yet received message.



To send this request switch to SoapUI and trigger the “Get a tile” test suite. This will send the initial “GetCapabilities” request, extract a supported layer from the server’s response and use it in the subsequent “GetMap” request.





The result we expect to see in the test bed is an indication of all message exchange and validation steps having completed successfully. As in the case of the UBL invoice validation demo, all document icons can be clicked to inspect the details of the validation steps and the complete information received by the client and responded by the server.

5 Glossary

The following two tables list acronyms mentioned in the document text and definitions of terms used.

Table 5-1 Acronyms

Acronym	Definition
API	Application Program Interface
CEF	Connecting Europe Facility
CEN	European Committee for Standardisation
DSI	Digital Service Infrastructure
EIF	European Interoperability Framework
GITB	Global e-Business Interoperability Test Bed
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
ISA	Interoperability Solutions for public Administrations
NAT	Network Address Translation
OGC	Open Geospatial Consortium
SUT	System Under Test
TRR	Test Registry and Repository
UBL	Universal Business Language
WMS	Web Map Service
XML	Extensible Markup Language

Table 5-2 Definitions

Term	Definition
Reference implementation	Application component that serves the purpose of realising a service that is made available to SUTs to perform interoperability tests against. It is termed a "reference implementation" in that it typically provides an implementation of a specification that is assumed to be correct so that other implementations of the same specification (or just other instances of the reference software) can test against. Note that in case no standard or specification is involved, the reference implementation is effectively a simulator of the service that is exposed for testing. Reference implementations can range from being fully independent IT systems to plug-in components.
System under test	System whose owners want to perform interoperability testing upon by connecting and running test scenarios against services available to test against.
Test assertion	Artefact that defines a statement to capture the goals of a testing activity and its prescription level, making the link between the normative source of the assertion (e.g. a legislation article) and the test details elaborated in test cases. An example of a test assertion would be "The system needs to respond with a receipt to a valid request within one minute".
Test case	Artefact that defines in detail the message exchange, processing and validation steps to be performed in realising a test assertion. A test case realises an assertion possibly as a part of a set of related test cases and can potentially be reusable between multiple assertions. A possible test case definition for the assertion "The system needs to respond with a receipt to a valid request within one minute" would include the steps to generate a valid request, send it to the system in question, receive its receipt, validate it and check the time it was received.
Test suite	Collection of test assertions forming a cohesive set. In the same way test assertions group a number of related test cases, test suites group assertions into a cohesive set. As an example consider a test suite that groups all the assertions needed to test a system's conformance against a specific standard (e.g. AS4).

6 Annexes

Annex I. Risk statement

The current annex lists risk points related to use of the GITB software. It is important to point out that these points are relevant when considering a local installation of the GITB software, not when using the ISA² test bed service. The goal is to provide maximum clarity on such matters to parties that would potentially host the GITB test bed.

Use of Docker

The GITB test bed software is comprised of multiple components that need to be built from source[17], with additional components, namely its database and cache server, which require separate installation. These components need to be subsequently configured appropriately so that they operate as a cohesive whole. Although by no means an overly complex task, this requires that interested parties have sufficient development skills and invest time in understanding the details of each component.

To streamline the installation process of the GITB test bed, both for demo purposes and also for the management of ISA²'s own test bed service, the Docker tool was selected as an approach to greatly simplify installation and operation by packaging the test bed's components in lightweight and modular containers. Although software delivery using Docker is quickly becoming an Industry best practice, it is not a standard and the choice to make use of it for the GITB test bed adds a dependency to third party software. This dependency manifests in the definition of the Docker containers themselves and also in the fact that their installation involves them being fetched from the "isaitb" account on the central Docker Hub[18].

If the Docker software is discontinued or use of its services is restricted due to future licensing constraints or costs, an alternate approach to ship and operate the GITB test bed software will be made available.

GITB test bed software licensing

As mentioned in Chapter 2.5, "The GITB test bed software", the GITB test bed software has been produced as part of the CEN GITB Workshop Agreement, using funding from the European Commission (DG GROW). The initial requirements assumed that all software produced would be open source and free of licencing constraints. Moreover the produced software's source is currently available for download from GitHub. However, considering that this is the result of a CEN standardisation process, it remains possible that CEN requires a fee for its use. Discussions on the software's licencing are still, at the time of writing, pending and explain why the source code on GitHub is not accompanied by a clear licence statement.

This lack of licencing clarity is not considered a problem for the ISA² test bed service but needs to be made aware to potential third parties envisioning direct use of the software itself. As this is a pending process the reader is requested to contact ISA² at isa@ec.europa.eu in case information on the latest developments is required.

17 The GITB test bed software is available on GitHub: https://github.com/srdc/gitb
18 ISA² ITB Docker Hub repository: https://hub.docker.com/r/isaitb/

Annex II. Troubleshooting common GITB test bed issues

The current annex lists solutions to common problems that could be encountered by parties hosting an instance of the GITB test bed software. Highlighting these is important especially considering that parties running the test bed for demo purposes will lack the experience and time to investigate encountered issues in depth.

Accessing the test bed's user interface results in a blank page

When accessing the user interface of the GITB test bed it could be the case that the user encounters a blank page. This is typically due to experimentation with the test bed's components and specifically with a reinstallation of its cache server (Docker container `gitb-redis`). One of the uses for the cache server is to store HTTP session identifiers that are maintained in a client's web browser as a cookie. Experiencing a blank page is due to an existing cookie providing a session identifier that cannot be matched in the cache server because the latter has been removed and reinstalled. To address this the user should clear his locally stored cookies and reconnect to the test bed. Note however that under normal operations this issue should never be encountered.

The test bed does not receive messages sent by a SUT as part of an open test session

Test cases that require message exchange with a SUT expect the SUT's address to be provided by the user as configuration before a test session is started. This is reflected as an "Endpoint" configuration property named "network.host". The value provided for this needs to be the address of the SUT as received by the test bed and is used by the test bed to ensure that only messages received from the configured address are considered for a specific test session. If the wrong value is configured the experienced result will be that communication initiated by the SUT does not get responded to and eventually times out, with the relevant test session in the test bed remaining stuck expecting a message.

If accessing the test bed over the internet, finding the correct IP address to configure is simple and can be provided by a multitude of free services (even Google searching "what is my IP"). It is more challenging to determine the correct address to use if you are accessing the test bed locally, especially if the Docker host used to run the test bed is running on a virtual machine (as is currently the case on Windows or Mac machines).

The simplest way to determine the correct address to use is to inspect the test bed's log. As a first Step attempt a communication from your SUT and, assuming this has no effect, provide the following command:

```
docker logs gitb-srv
```

This command will print the tail of the test bed's log file and will include in this case an entry matching the following:

```
... TCPMessagingServerWork-
er - Received [192.168.99.1]
but expected [192.168.1.1]
```

This message indicates that you have configured value "192.168.1.1" as the "network.host", whereas the communication received matched the address "192.168.99.1". To correct this, stop the test case execution and configure the expected value for the "network.host" property (in this case using value "192.168.99.1").

HOW TO OBTAIN EU PUBLICATIONS

Free publications:

- one copy:
  via EU Bookshop (http://bookshop.europa.eu);
- more than one copy or posters/maps:
  from the European Union's representations (http://ec.europa.eu/represent_en.htm);
  from the delegations in non-EU countries (http://eeas.europa.eu/delegations/index_en.htm);
  by contacting the Europe Direct service (http://europa.eu/europedirect/index_en.htm) or
  calling 00 800 6 7 8 9 10 11 (freephone number from anywhere in the EU) (*).

  (*) The information given is free, as are most calls (though some operators, phone boxes or hotels may charge you).

Priced publications:

- via EU Bookshop (http://bookshop.europa.eu).

NO-05-16-009-EN-C

Publications Office

ISBN 978-92-79-63502-1