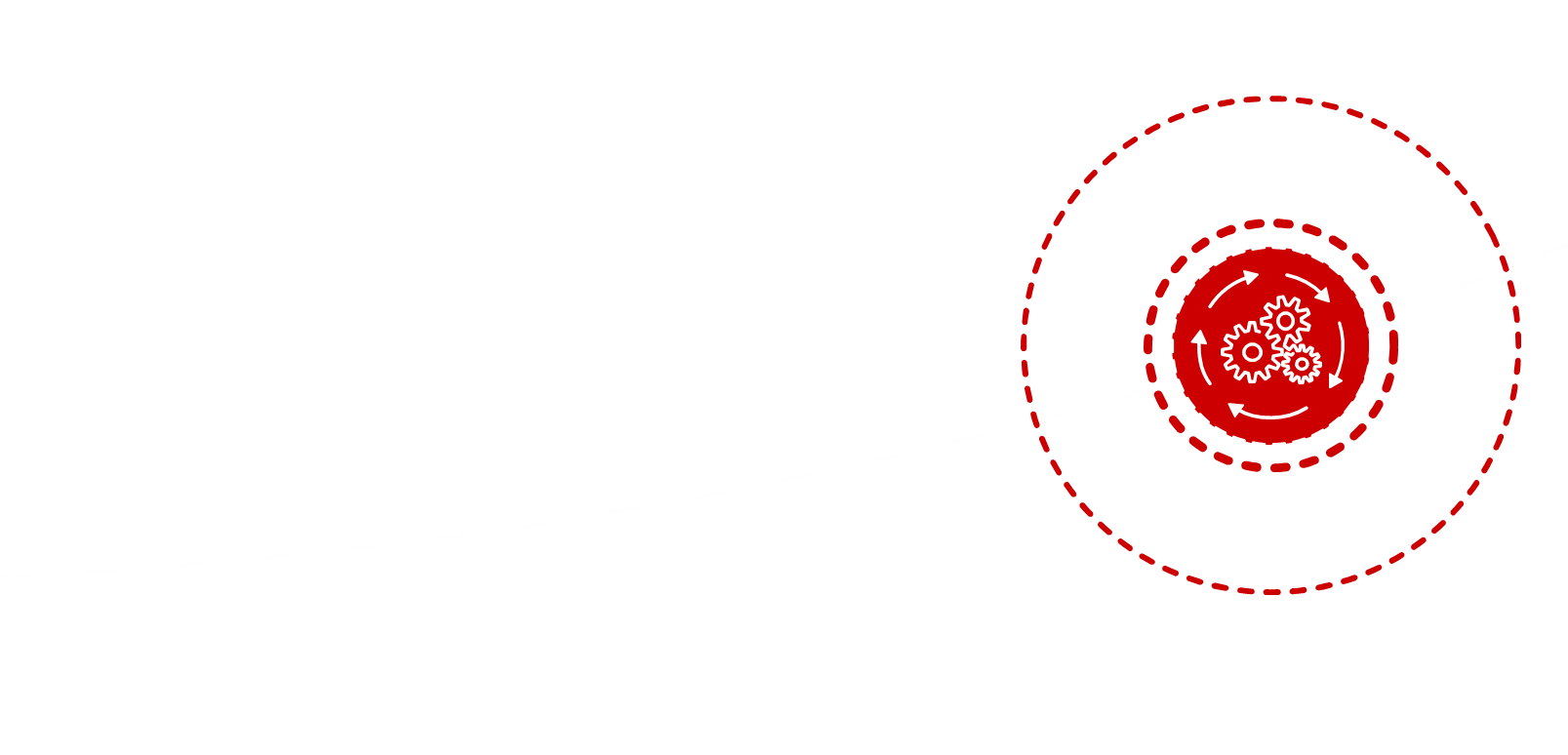
Description: C:\Users\cg2913\Desktop\In Progress\References\CapLogo.emf





**Test and Release Service**

**Automation Portal User Guide**

Origin/Author **: Harold Robson**

Approved by **: x**

Date Approved **: x**

Version : v0.2

Contents

[1 Introduction 3](#_Toc488315374)

[2 Architecture 4](#_Toc488315375)

[2.1 Support for multiple platforms 4](#_Toc488315376)

[2.2 Multiple levels of aggregation 4](#_Toc488315377)

[2.2.1 Usage Event 4](#_Toc488315378)

[2.2.2 Test Activity 4](#_Toc488315379)

[2.2.3 Test Tool 4](#_Toc488315380)

[2.2.4 Test Tool Category 5](#_Toc488315381)

[3 Usage 5](#_Toc488315382)

[3.1 Dashboard 6](#_Toc488315383)

[3.2 Resetting the statistics 7](#_Toc488315384)

[3.3 Event Log 7](#_Toc488315385)

[3.4 Tools 8](#_Toc488315386)

[4 Integrations 9](#_Toc488315387)

[4.1 Registering the tool with the portal 10](#_Toc488315388)

[4.1.1 Prerequisite details 10](#_Toc488315389)

[4.1.2 Adding the details to the repository 10](#_Toc488315390)

[4.1.3 Adding the details to the web server 11](#_Toc488315391)

[4.1.4 Manually updating the Spring Boot MySQL database via Command Line 11](#_Toc488315392)

[4.2 Example JSON Objects 12](#_Toc488315393)

[4.3 Validation rules 12](#_Toc488315394)

[4.4 Java example code 13](#_Toc488315395)

[4.1 JavaScript example code 16](#_Toc488315396)

[5 Deployments 17](#_Toc488315397)

[5.1 Development environment overview 17](#_Toc488315398)

[5.2 Live environment overview 17](#_Toc488315399)

[5.2.1 Live environment log files 18](#_Toc488315400)

[5.3 Deployment 18](#_Toc488315401)

[5.3.1 Live environment application.yml file 18](#_Toc488315402)

[5.4 Credentials 18](#_Toc488315403)

[Statement of Confidentiality 20](file:///C:\Users\cleeding\Documents\Guides%20before%20rolloff\Automation%20Portal.docx#_Toc488315404)

# Introduction

This document covers the TTAS automation portal.

The [Architecture](#_Architecture) section describes the design goals of the system ta a high level.

The [Usage](#_Usage) section describes the information displayed on the portal, how it can be accessed, interpreted, filtered, and reset.

The [Integrations](#_Integrations) section describes how other software running on the HMRCMIS network can POST updates to the portal’s REST API and the mechanism for registering new software in the portal’s database.

The [Deployments](#_Deployments) section provides a technical description of the live environment and the steps for testing and deploying a new build of the portal.

# Architecture

## Support for multiple platforms

Our team provides a range of testing and productivity tools that are delivered as standalone applications and web apps written in Java, JavaScript, and VBA, or created with WinAutomation. The goal of these tools is always to save the customer an amount of manual effort that is greater than that required for us to create the tool.

As part of our business obligations it is necessary and desirable to record the time savings generated by the use of our tools. One strategy to obtain this data is to have the users report time savings to us manually every time they use our tools.

A better strategy is to utilise automation for the reporting as well, and have the tools report their own usage statistics back to a central server which operates within the context of a secure Local Area Network.

## Multiple levels of aggregation

## Usage Event

Each usage Event is recorded in our database with metadata such as the Test Activity that ran, which Test Tool it was run under, the user who ran it, the date and time, pass and fail, number of data items generated, and data type.

## Test Activity

A single Test Activity can be mapped to one or more Test Tools. A Test Activity could be a Test Case, or it could be an activity that’s auxiliary to testing, such as data generation, or validation.

An Event is therefore not automatically a Test Run, it could be a data generation event or a data validation event.

## Test Tool

A Test Tool could be a Test Suite, but it could also be a Suite of Data Generation or Validation Activities.

## Test Tool Category

Test Tools can then be grouped into one or more categories. We have chosen to use two categories; Productivity Tool and Test Tool but the system is flexible and allows for any number of categories which can then be changed at any time.

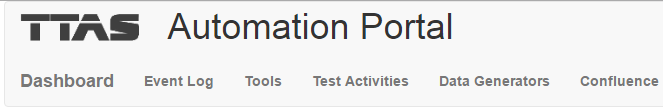
This allows us great flexibility when aggregating usage data.

# Usage

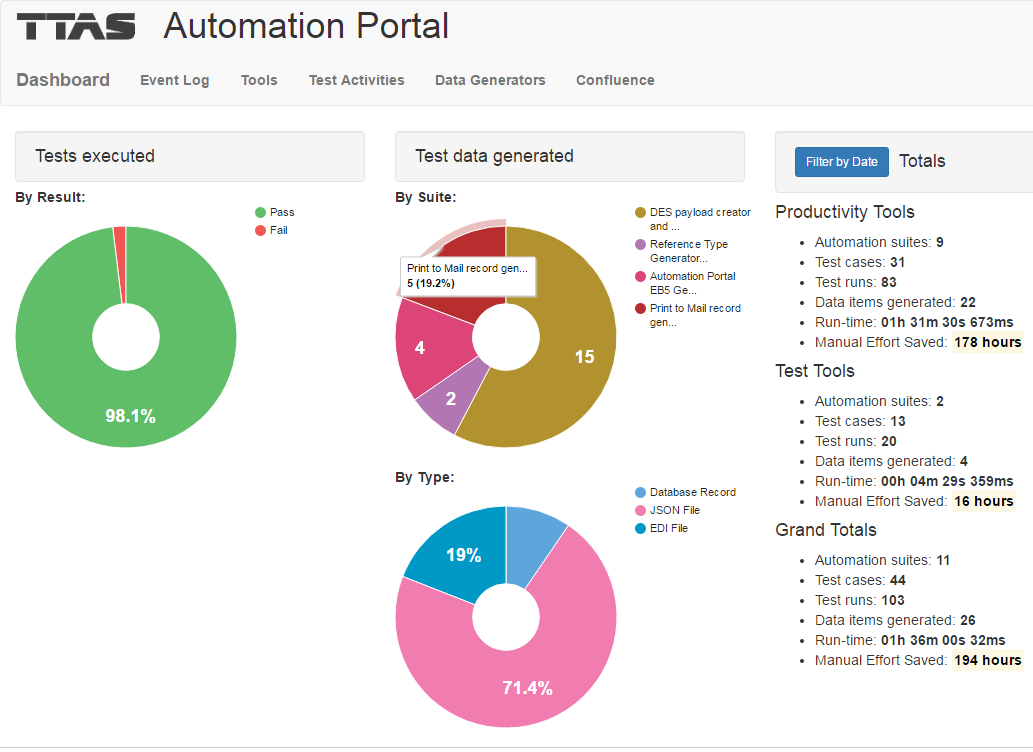
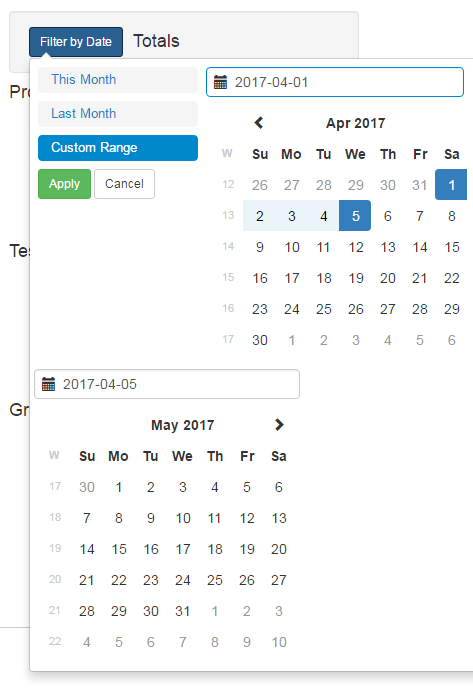
A dashboard page shows statistics and pie charts for data generated, test results, and manual effort saved.

The portal is accessed through the HMRCMIS network using the address <http://automation:8080> or [http://automation.hmrcmis.net:8080](http://automation:8080)

The following links are shown:



## Dashboard



The dashboard shows our time saving from automation, it can show the time saving for each category of tools, the screenshot above shows that most of the time saving at that time had come from the Productivity Tools, and it also shows how many suites and test cases make up each tool category. There is also a grand total time saving value, and graphs for results and data generation.

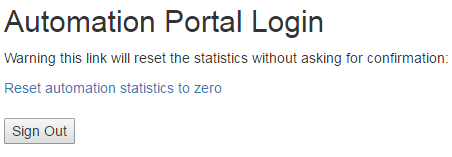
When the dashboard is first loaded, the totals shown are for every event the system has captured since it was first commissioned. To generate a report for a specific month (or any arbitrary time period), click the “Filter by Date” button in the top right. You can then specify the Start and End dates before clicking Apply:

## Resetting the statistics

To reset the statistics and graphs shown on the Dashboard, log in to the portal admin page: <http://automation:8080/portal/admin/>

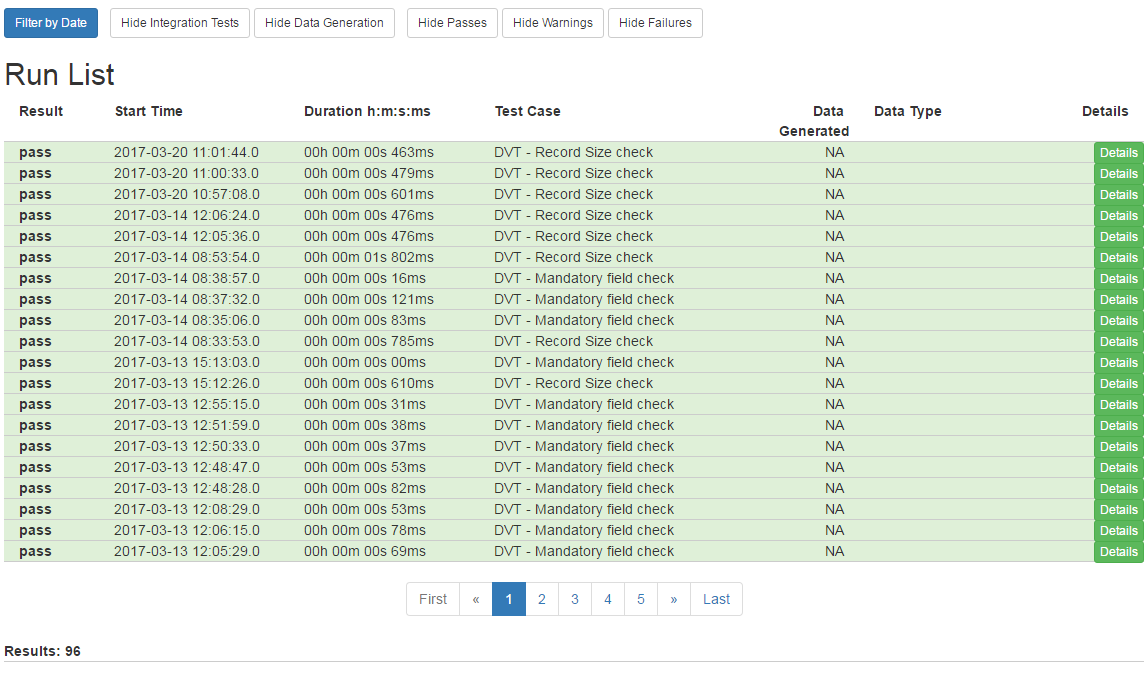
user: ttas password: portalAdmin

When you log in it will give you the link to reset the stats:



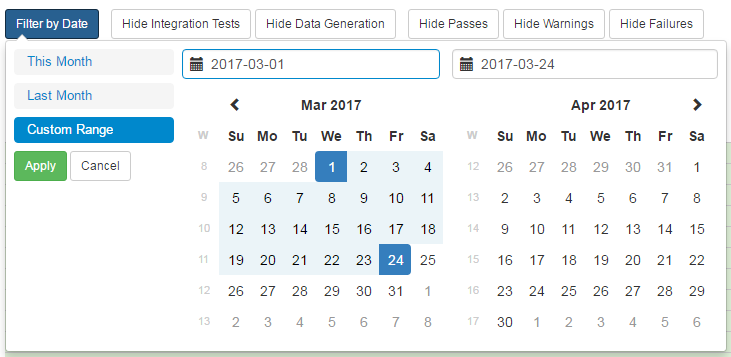
It won’t clear down the record of test runs, the test runs page will still show a complete history, and the statistics could be reconstructed from that, for any given time period, if necessary.

## Event Log



The Test Runs page is essentially an event log for events generated by TTAS tools. As discussed earlier the TTAS tools may fit into various categories, from a traditional test suite, to a productivity tool that generates data for other tests. Broadly speaking these are all testing activities, so the page is labelled as Test Runs.

The **Filter by Date** button allows you to set a time period (a start and end date) to filter the events:



The **Hide Integration Tests** button will hide all events that are not related to data generation.

The **Hide Data Generation** button will hide all events that are related to data generation.

Furthermore, the buttons **Hide Passes**, **Hide Warnings**, and **Hide Failures**, allows further filtering of the events. These filters act only on the current page of results, and not the total set.

Each Event has a **Details** button, and clicking this will reveal the user who triggered the event, and a 255 character message field that may contain error messages, stack traces, or notifications.

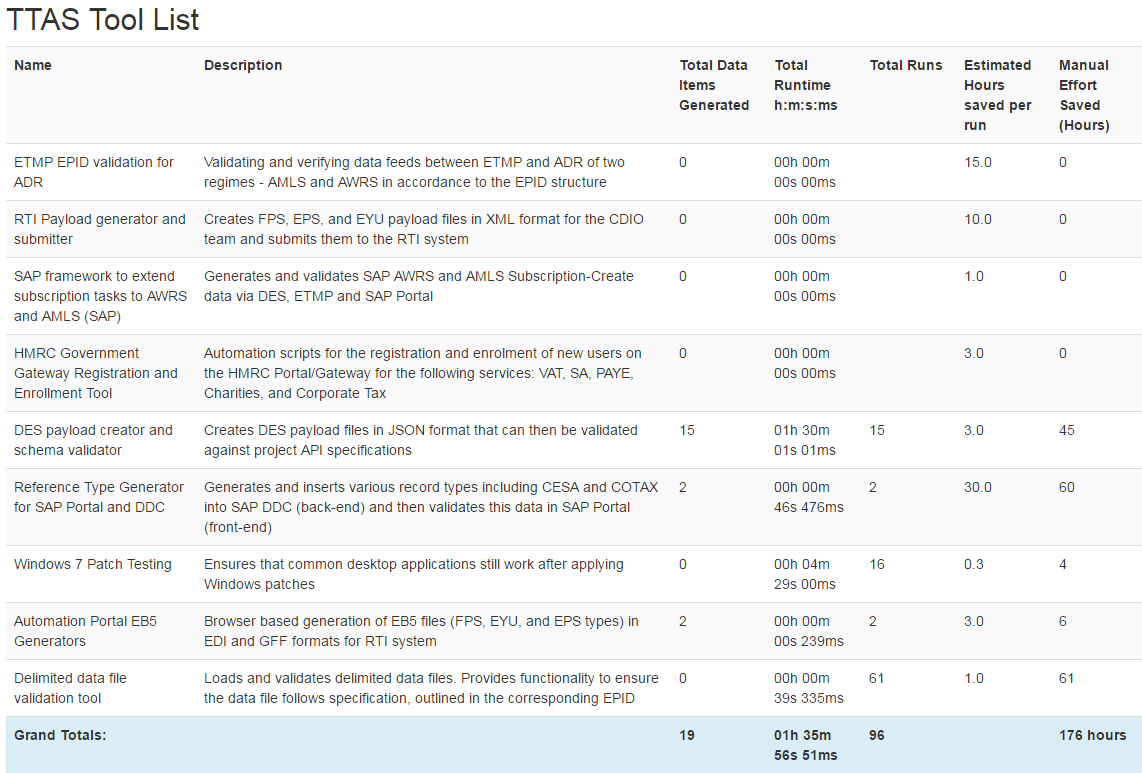
## Tools

The Tools section shows tools developed by the TTAS team. A Tool is defined as a collection of Test Activities; therefore a Test Suite is a Test Tool, as each Test Case in the Suite would be defined as a Test Activity.

But Test Activities can also be defined as data generators and data validators. For this reason we have used the generic term “Tools” rather than the specific term “Test Suites”. Each tool has a name, description, and estimated time saving.

The estimated time saving is an average value in hours that is used to define the manual time saving for one run of any of the Test Activities that make up the tool.

While all of the Activities that make up a Tool are unlikely to save identical amounts of effort, we can say that the activities within a Tool are related to each other, and therefore it is acceptable to use an average value to simplify the process of estimating time savings.



In addition to the fields I discussed earlier, you can also see four counters:

* Total Data Items Generated
* Total Runtime (machine time)
* Total Run count
* Total Manual Effort Saved (run count multiplied by time saving value)

# Integrations

When TTAS develop a new tool we aim to integrate it with the portal as the final stage of development. Tools written in the following languages have been integrated:

* Java
* JavaScript
* WinAutomation

Any language or system that includes a REST API library should work.

The integration is a two-step process that involves first registering the tool on the portal, giving it a unique ID number, and then including that unique ID along with some additional code in the tool to make the API call.

The API is called my making a POST to <http://automation:8080/testruns> once of each Event that needs to be logged.

## Registering the tool with the portal

## Prerequisite details

Before you begin, the following questions must be answered:

1. What is the tool name and description
2. How many [Test Activities](#_Test_Activity) are included in the tool
3. What are the names and descriptions of all the Test Activities?
4. Out of all the Test Activities you have just defined, what is the average manual effort saved when each one is run once? For example if a Tool has two Activities and one will save 10 minutes of effort when it is run once and the other will save 50, the average effort saved for the tool is 30 minutes per run.

## Adding the details to the repository

Once you have come up with the necessary details, you will need to add them to some CSV files for loading into the system. Three CSV files are in the following location, please clone the latest version of them.

[http://subversion:8080/svn/TRS\_Transformation\_Automation/Documentation/Automation Portal](http://subversion:8080/svn/TRS_Transformation_Automation/Documentation/Automation%20Portal%20)

1. Open testTools.csv
2. At the bottom of the file, add the details for your tool on a new row, use the next available id number. In the timesaving column set the [timesaving](#_Prerequisite_details) in hours.
3. In the toolClass column, set either “Productivity Tools” or “Test Tools” as appropriate.
4. Make a note of your Tool ID number
5. Save and close the file.
6. Commit and push your changes to the repository
7. Open testCases.csv
8. At the bottom of the file, add the details for your new Test Activities, creating as many new rows at the bottom of the file as you need. Use the next available id number; the id should increase by one for each row.
9. Make a note of the range of ids that correspond to your Test Activities.
10. Save and close the file.
11. Commit and push your changes to the repository
12. Open testSuites.csv
13. At the bottom of this file you will add the same number of rows as you added into testCases.csv, the id column will increase by one each time like in the other files. The testToolId column will have your Tool ID repeated on every new row, and the testCaseId column will contain the range of Ids corresponding to your Test Activities.
14. Save and close the file.
15. Commit and push your changes to the repository

## Adding the details to the web server

Once you have completed the previous step you will have three new CSV files to upload to the server.

1. Open WinSCP
2. Set the hostname to “automation”
3. Set the username and password according to the [FTP Credentials](#_Credentials).

User Name: tomcat

Password: Password1234!”£$

1. If it asks about adding the server’s certificate, click Yes.
2. Using WinSCP explorer, Navigate to the following directory on the server: /home/users/tomcat/clientdata
3. Upload the new CSV files here, overwriting the previous.
4. Log into tomcat at <http://automation:8080/manager/html> using the [Tomcat Manager Credentials](#_Credentials).
5. Stop the root app (show on screen as ‘/’)
6. Start the root app (show on screen as ‘/’)

Once the application has restarted, navigate to the [Test Tools page](http://automation:8080/portal/testtools) and [Test Activities page](http://automation:8080/portal/testcases) and check that the details you created show up. When they do you can proceed to add the relevant implementation code to your tool.

## Manually updating the Spring Boot MySQL database via Command Line

If you cannot for whatever reason make changes to the database by updating the CSV files and uploading them via WinSCP, then it is also possible to make changes manually via the Command Line prompt.

* + - 1. Open PuTTY and set hostname to “automation”
      2. Click Open
      3. Set login as “tomcat”
      4. Set password as “Password1234!”£$”
      5. Enter login session details such as Name, Team and Work Description
      6. Enter “mysql –u automation –p”
      7. Set password to “kvnSt27SQe5b”
      8. Enter “show databases;”
      9. Enter “use springboot;”

Once the springboot database is selected, you can make changes to the tables.

For example, if you would like to insert a new test case, then you can execute the following statement:

insert into test\_case(id,name,description) values (1, "name of test case", "description of the test case");

## Example JSON Objects

**Pass**

{"result":true,"startDateTime":"2016-11-13T00:00:59.059Z","endDateTime":"2016-11-13T00:01:00.059Z","username":"cg14821",”message”:”Succesfully generated one record”,"generatedDataCount":1,"generatedDataType":{"id":5},"testSuite":{"id":6}}

{"result":true,"startDateTime":"2016-11-13T00:00:59.059Z","endDateTime":"2016-11-13T00:01:00.059Z","username":"cg14821",”message”:”Test case passed”,"testSuite":{"id":7}}

**Failure**

{"result":false,"startDateTime":"2016-11-13T00:00:59.059Z","endDateTime":"2016-11-13T00:01:00.059Z","username":"cg14821",”message”:”A failure occured”,"testSuite":{"id":6}}

**Warning**

{“startDateTime":"2016-11-13T00:00:59.059Z","endDateTime":"2016-11-13T00:01:00.059Z","username":"cg14821",”message”:”Warning, task did not complete”,"testSuite":{"id":6}}

The [Java](#_Java_example_code) and [JavaScript](#_JavaScript_example_code) example code shows how to create the objects, including the correct date format.

## Validation rules

Event objects that are posted to the portal get validated according to the following rules. When they fail validation, the server will return HTTP 400 BAD REQUEST. When they pass, the server will return HTTP 201 CREATED.

* When an event is posted, it should not contain the ID field. The service assigns IDs to the events after they have been posted, so a post with the ID field already populated will not be accepted.
* The start and end DateTime fields must both be populated, allowing the service to work out the total runtime.
* A Test Suite ID must be provided.
* The Test Suite record for that ID must contain a valid Test Tool ID.
* If data count field is supplied, data type field must be supplied.
* If data type field is supplied, data count field must be supplied

## Java example code

The Java implementation uses a **reporter** class with a **createReport()** method**.** The **createReport()** method is overloaded so that the arguments relating to test data generation are optional.

In our Selenium framework, the reporter is a component of the **evidenceCollector** which is already instantiated in all of our test classes; you should follow the implementation used in any of the reference tests in the **nondigital** package.

For Java code developed outside of the Selenium framework, you can copy the **reporter** class (shown below) and the three domain objects (**Test Run, Test Suite, RecordType**) into your own code base.

**public class Reporter** {

private String message;

// returns true if success

public void createReport(Date startTime, Date endTime, Boolean result, int testSuiteId, int generatedDataType, int generatedDataCount) throws Exception{

// create the test case

TestSuite testSuite = new TestSuite();

testSuite.setId((long) testSuiteId);

// test data type

RecordType recordType = new RecordType();

recordType.setId((long) generatedDataType);

// create test run

TestRun testRun = new TestRun();

testRun.setResult(result);

testRun.setUsername(System.getProperty("user.name"));

testRun.setGeneratedDataType(recordType);

testRun.setGeneratedDataCount((long) generatedDataCount);

testRun.setTestCase(testSuite);

testRun.setStartDateTime(startTime);

testRun.setEndDateTime(endTime);

if (message != null && message != ""){

testRun.setMessage(message);

message = null;

}

Gson gson = new GsonBuilder().setDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'").create();

sendReport(gson.toJson(testRun));

}

public void createReport(Date startTime, Date endTime, Boolean result, int testSuiteId) throws Exception{

// create the test case

TestSuite testSuite = new TestSuite();

testSuite.setId((long) testSuiteId);

// create test run

TestRun testRun = new TestRun();

testRun.setResult(result);

testRun.setUsername(System.getProperty("user.name"));

testRun.setTestCase(testSuite);

testRun.setStartDateTime(startTime);

testRun.setEndDateTime(endTime);

if (message != null && message != ""){

testRun.setMessage(message);

message = null;

}

Gson gson = new GsonBuilder().setDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'").create();

sendReport(gson.toJson(testRun));

}

private static Boolean sendReport(String json) throws Exception {

try{

URL obj = new URL(Configurations.REPORTING\_SERVER);

HttpURLConnection con = (HttpURLConnection) obj.openConnection();

con.setRequestMethod("POST");

con.setRequestProperty("Content-Type", "application/json");

con.setDoOutput(true);

OutputStream os = con.getOutputStream();

os.write(json.getBytes());

os.flush();

if (con.getResponseCode() == 201){

return true;

}

else{

String error = String.format("Failure to submit %s, HTTP code %d", json, con.getResponseCode());

System.out.println(error);

return false;

}

}

catch (Exception e){

System.out.println(e.getMessage());

String error = String.format("Failure to submit %s", json);

System.out.println(error);

return false;

}

}

public String getMessage() {

return message;

}

public void setMessage(String message) {

this.message = message;

}

}

**public class TestRun {**

private Boolean result;

private Date startDateTime;

private Date endDateTime;

private String username;

private String message;

private Long generatedDataCount;

private RecordType generatedDataType;

private TestSuite testSuite;

public Boolean getResult() {

return result;

}

public void setResult(Boolean result) {

this.result = result;

}

public Date getStartDateTime() {

return startDateTime;

}

public void setStartDateTime(Date startDateTime) {

this.startDateTime = startDateTime;

}

public Date getEndDateTime() {

return endDateTime;

}

public void setEndDateTime(Date endDateTime) {

this.endDateTime = endDateTime;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public Long getGeneratedDataCount() {

return generatedDataCount;

}

public void setGeneratedDataCount(Long generatedDataCount) {

this.generatedDataCount = generatedDataCount;

}

public RecordType getGeneratedDataType() {

return generatedDataType;

}

public void setGeneratedDataType(RecordType generatedDataType) {

this.generatedDataType = generatedDataType;

}

public TestSuite getTestCase() {

return testSuite;

}

public void setTestCase(TestSuite testSuite) {

this.testSuite = testSuite;

}

public String getMessage() {

return message;

}

public void setMessage(String message) {

this.message = message;

}

}

**public class TestSuite {**

private Long id;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

}

**public class RecordType {**

private Long id;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

}

Once you have these four classes in your codebase, you can instantiate the reporter and make a one line method call to **createReport()** from the appropriate location in your code. After you instantiate the reporter, and before you call **reporter.createReport(),** you can call **reporter.setMessage()** to set the an optional message to be displayed with the report on the portal. The **reporter.createReport()** method then resets the message so that it will not be resent.

## JavaScript example code

The following code has been developed for including into JavaScript applications:

document.getElementById('saveoutput').addEventListener('click',function() {

payloadObject = new Object();

var d1 = new Date (),

d2 = new Date ( d1 );

d2.setMinutes ( d1.getMinutes() + 10 );

payloadObject.startDateTime = d1;

payloadObject.endDateTime = d2;

payloadObject.result = true;

payloadObject.testSuite = new Object();

payloadObject.testSuite.id = 15

payloadObject.generatedDataType = new Object();

payloadObject.generatedDataType.id = 3;

payloadObject.generatedDataCount = 1;

new Date("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'");

try {

xhr = new XMLHttpRequest();

var url = "http://automation:8080/testruns";

xhr.open("POST", url, false);

xhr.setRequestHeader("Content-type", "application/json");

xhr.send(JSON.stringify(payloadObject));

}

catch(err) {

alert(err);

}

finally {

saveAsFile(this, 'output', 'output');

}

});

# Deployments

## Development environment overview

Requirements:

* Eclipse
* Gradle
* MySql 5
* AutomationStatisticsPortal.zip

## Live environment overview

The automation portal is running on port 8080 of the virtual server 10.102.83.149, which we have mapped to the DNS ‘A’ record ‘automation’. The portal is accessed through <http://automation:8080> or [http://automation.hmrcmis.net:8080](http://automation:8080)

For queries relating to the server, contact [dev-ops1@hmrcaspire.com](mailto:dev-ops1@hmrcaspire.com)

## Live environment log files

Using WinSCP connect to the live server (see credentials section) and find the log files in the following location:

/opt/app/tomcat/apache-tomcat-8.5.8/logs

The first log file to check is usually:

Catalina.out

## Deployment

1. Run “gradlew build” and ensure the tests pass
2. Copy over the production version of the application.yml file
3. Run “gradlew build -x test” - this will build the application again but some of the tests will fail as it can’t reach the production database, so we add the ‘-x test’ switch to turn testing off. However we have just run the tests using the development environment configurations, which will be sufficient.
4. The application will be packaged as a \*.war file for deployment to our tomcat server: \*\AutomationStatisticsPortal\build\libs\AutomationStatisticsPortal.war
5. Copy the \*.war file to a folder where you will store all live releases of the portal, rename it to ROOT.war
6. Log into tomcat at <http://automation:8080/manager/html>
7. Undeploy the root app (show on screen as ‘/’)
8. Upload the new file and Click Deploy
9. Check that the portal is running at <http://automation:8080>

## Live environment application.yml file

load-files: "/home/users/tomcat/clientdata"

spring:

  datasource:

    type: com.zaxxer.hikari.HikariDataSource

    url: jdbc:mysql://localhost/springboot

    username: automation

    password: kvnSt27SQe5b

  jpa:

    database-platform: org.hibernate.dialect.MySQL5InnoDBDialect

    database: MYSQL

    show-sql: true

  properties:

    hibernate.hbm2ddl.auto: update

## Credentials

Automation Portal Admin:

u: ttas

p: portalAdmin

Tomcat Manager:

u: tomcatmgr

p: 1234qwer

SSH and FTP:

u: tomcat

p: Password1234!”£$

MySql:

u: automation

p: kvnSt27SQe5b

**Document Control**

**Status:**

|  |
| --- |
| **Author** |
| **Last Updated By** |
| **Status** |

Change Log:

| Version | Date | Comments *(please include names of approvers in this section)* |
| --- | --- | --- |
| v0.1 | 17/05/2017 | First draft prepared by Harold Robson |
| v0.2 | 20/07/2017 | Updates concerning the manual insertion of data into MySQL db by Connor Leeding (sub-section 4.1.4) |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Storage:

|  |  |
| --- | --- |
| **Master Copy** |  |
| **Review Record** | None |
| **Approval Record** | None |

Statement of Confidentiality

© HM Revenue & Customs 2016.

All rights reserved. Capgemini and the Capgemini logo are trademarks of Capgemini SA.