Table of Contents

1 Summary 3

1.1 General 3

1.2 Producing Reports 4

1.3 GUI 4

2 Detailed Specification 5

2.1 MQTT 5

2.2 Trigger Condition 5

2.3 Report Task 5

2.3.1 Configuration 5

2.3.2 Measurement Filter Items 6

2.4 Report Server Configuration 6

3 GUI Design 7

3.1 User Login 7

3.2 On-line View 7

3.3 Configuration View 7

3.4 Administration View 7

4 Report ViewerS 8

4.1 GnuPlot 8

4.2 Python 8

4.3 R language 8

4.4 Exports 8

4.5 Grafana Desktop 8

5 Comments 9

5.1 Task as Sequences 9

5.2 Validation/Export Task 9

5.3 Divide the selection into sub-selections 9

5.4 Viewing the report 9

6 References 10

# Summary

## General

The report server collects meta and measurement data from one or more measurements and produce reports for the end-user.

Webbserver

Front-End (JS)

Broker (MQTT)

Backend (C++)

User System

(Measurement Files)

MQTT

MQTT (Sparkplug B)

gRPC

MQTT (Test Bed Status)

ODS DB

Meta Data

Measurement Data

Environment

MQTT Gateway

Optional

The server has a JavaScript front-end and an ASAM ODS based back-end. The database contains meta data and references to the measurement files. This means that the actual raw measurement data values are kept in the original files not in the database.

The backend work with so-called environments. The purpose for an environment is to populate the ODS database with data The environments depend on the user system, but the most common MDF and ODS environments are supported.

Optional a MQTT gateway is required to capture event and status from the user system. Note that the gateway is used by other applications as well.

The backend application uses the MQTT interface for notification and events while the Goggle RPC is used for client/server requests.

## Producing Reports

The application doesn’t include any report viewer. Instead, are so-called reports produced, so external viewers as GnuPlot and MatPlot can be used.

The reports are either generated manually or automatic. The main purpose is to generate reports automatically. Note that this design is like the Jenkins and TeamCity builders.

## GUI

The GUI is based upon a standard web-design with a standard we-server handling most of the login and security issues. The main purpose is to configure named reports.

There is only 2 GUI needed to be developed. The configuration GUI simply configure a report. The configuration is stored in the global database.

# Detailed Specification

## MQTT

The MQTT interfaces uses the Sparkplug B standard interfaces traditionally used with SCADA systems. The purpose is to get the plug and play feature while the downside is that the payload uses the Google protobuf streaming.

The following MQTT namespace is used by the backend.

spBv1.0/ods/<message\_type>/<node\_id>.

The message types are according to [1] NBIRTH, NDEATH and NDATA. The node ID is a unique name of the application.

## Trigger Condition

A report task is started either manually or automatic. The automatic start is done by a so-called trigger. A trigger is a start condition that only have 0 or 1 value. The triggers are configured separately as it may be used to trigger many reports.

The trigger fetches its information from the MQTT broker, but that information may not be enough to generate a trigger, therefore may a trigger consist of a Lua expression.

A trigger has the following properties.

* Name of the trigger
* Description
* MQTT subscription value(s)
* Expression (Lua script)

## Report Task

The report task is a type of template. It is typically divided into 3 steps.

The first goal is to select the measurements of interest. This is done ether by searching by selecting testbed, test and measurement or by selecting on some meta data. Meta data is normally related to test object so instead of test bed name the selection is done on some test object attribute.

After selecting the measurements, the user shall select which values that should be included in the report.

The last step is to select type of report and where to put the report.

The end-user may save and or create the report.

Note that the report configuration mostly consists of filter items.

### Configuration

Each task (template) is stored in the database. It is recommended that the task can be grouped into projects or application types. A task may have the following properties.

* Name. Name used to tag the report type. No spaces allowed. Used as stem in filenames so restrict naming and ignoring case characters.
* Display Name. User friendly name used in GUI (optional).
* Description. For pop-up in GUI.
* Template Filter Configuration. The exact filters need to be defined.
* Type of report viewer. Defines the report type. If multiple outputs should be allowed needs to be defined.
* Active. Defines if the report is active (online) or not.

### Measurement Filter Items

When selecting the measurements, the following filter items should be available.

* Environment. Select all or an environment name. Assume a handful of environments.
* Test Bed. Select all or a test bed. Assume small amount of test beds.
* Meta Data. Meta data consist of a property and a value, so select a property and a value. I assume wildcard doesn’t work here. Note this actual select available tests.
* Test. All measurements belong to a test. Assume many tests. Select all, last (active) or by name. Use of wildcard is needed.
* Test Date. From and optional to a date.
* File. Each test has one or more measurement files. Wildcard filter on name.
* Measurement. Select all, last, not created or wildcard.

## Report Server Configuration

The report server also needs some common configuration, but this can be done by XML configuration files.

* Output root directory path.
* Default database source. If not defined it uses the system databases.
* MQTT node ID.

# GUI Design

## User Login

The user may login by name and password or by using active directory. This should be handled by the web server. It should exist solutions for this already.

The username or at least which group he/she belongs to, should be available for the application.

## On-line View

After the login, the user should get a list of available report tasks and their status. The end-user

When selecting a task, a list of available (created) reports should be presented. It would be nice to be able to run the report from this list, but this might not be possible, see comments.

## Configuration View

The configure view is used to adding and modifying report tasks.

## Administration View

The administration (settings) view is used for general task. The actual functionality needs to be defined.

# Report ViewerS

## GnuPlot

The GnuPlot is the basic line graph report. The GnuPlot freeware is actually a very old software that simply do line plots.

The GnuPlot needs 2 files. One script file defines how the graph should look like and the other file with tab separated data.

The report task modifies, if needed, the template and produce that data file.

## Python

This is very similar to the GnuPlot but using the Python scripting language and the MatPlotLib library. It is actually very similar but requires little bit more efforts. The design is otherwise like the GnuPlot design.

## R language

The R language is the ultimate plotting tool, but it comes with a complexity of a new language. If you need 3D graph or worse, go for it.

## Exports

If possible, we should start using customer tools to view or export reports, not just export the data. I assume this is defined by the customer.

## Grafana Desktop

There exist a lot of Web based so called dashboard software. One typical is the Grafana dashboard that we normally use for time series data. The time series data is fetched from the Influx DB which is a time series database (TSDB). The database is subscribing from an MQTT broker which in turn collects data and event from the server.

The Grafana is good on presenting trend data with absolute timestamp, but we can use it as a dashboard that for example show the progress in at test bed. Maybe a list of reports?

Most likely it requires a REST service for getting the data from ComTest.

# Comments

## Task as Sequences

As this design is like a build system as Jenkins and TeamCity, each task may be put into a sequence. For example, the user wants to Convert/Export data first and then produce its reports.

An alternative is that each task publishes when it’s done which in turn may start another task.

## Validation/Export Task

The report server can also be used as a validation server. The design is very similar just replacing the report type with some validation executable.

Exports and converters also work in a similar way. Maybe general python scripts for starting export and converters might be a solution.

## Divide the selection into sub-selections

As selection of the measurements is the most complicated configuration, a solution is to save the selection as a template that can be reused.

## Viewing the report

It is the backend that creates the actual report, so the reports can only be saved onto disk available for all clients. If the user wants to view this from his web-browser this is complicated. Finding a solution might be complicated.

The solution might be to let a stand-alone desktop application as the report explorer do the viewing of the created reports.

# References

1. [Sparkplug Specification](https://www.eclipse.org/tahu/spec/Sparkplug%20Topic%20Namespace%20and%20State%20ManagementV2.2-with%20appendix%20B%20format%20-%20Eclipse.pdf)