project-name Documentation

Release 0.1.0

author

Sep 21, 2020

For dashboard developer:

This project defines a container environment that integrates OSO-UI applications with TMC devices and a Tango control system. It defines a set of docker-compose configurations for OSO-UI applications and their dependencies so that a test integrated system can be instantiated on a developer's laptop or workstation.

The documentation is composed of three parts. The **Dashboard developer**, which explains how to use Webjive, step by step, for end user. The **developer** part, which shows technical aspects of Webjive. And **External Resources** with a list of useful documentation regarding Webjive and other tools.

CHAPTER 1

Official documentation

In this documentation, only documents regarding SKA are reported. For the complete documentation of Webjive and TangoGQL, please refer to the official documentation.

Webjive Webjive is a web-based program that allows a user to create a visual interface using widgetswhich may include charts, numerical indicators or dials that interface to Tango device back end database. WebJive General documentation is available in the following link, : https://webjive.readthedocs.io/en/latest/index.html In the documentation you will find sections as:

- Architecture: a description of the Webjive Software Architecture
- Widgets: the documentation of how the widgets works
- How to deploy a widget

TangoGQL

A GraphQL implementation for Tango, used by Webjive to access the Tango Controls Framework TangoGQL General documentation is available in the following link, : https://web-maxiv-tangogql.readthedocs.io/en/latest/

In the documentation you will find sections as:

- API Documentation
- Examples
- TangoGQL features and convention

Webjive authorization Webjive authorization (work in progress): https://webjive-auth.readthedocs.io/en/latest/

Webjive Dashboard Webjive Dashboard (work in progress): https://webjive-dashboards.readthedocs.io/en/latest/

1.1 Overview

Webjive is a web-based program that allows a user to create a visual interface using widgets, which may include charts, numerical indicators or dials that interface to Tango device back end database. Details of how this is programmati-

cally achieved is presented in a developer biased document which can be found at: https://developer.skatelescope.org/projects/ska-engineering-ui-compose-utils/en/latest/device.html

Webjive was conceived and originally created by the MAX IV synchrotron facility in Lund, Sweden. During the early User Interface identification and downselect process conducted by the SKA OSO-UI Buttons team, Webjive was highlighted as a possible candidate to be taken forward as the platform upon which the SKA Engineering User Interface could be built. In early 2019 discussions between MAX IV and OSO-UI Buttons team (overseen by SKA) were held and it was agreed that a collaborate relationship could be taken forward to develop and maintain Webjive.

Logging into Webjive presents the user with a screen showing the available Tango devices that can be interfaced with and some general statistics regarding the connected Tango database. An example of this is shown in Figure 1

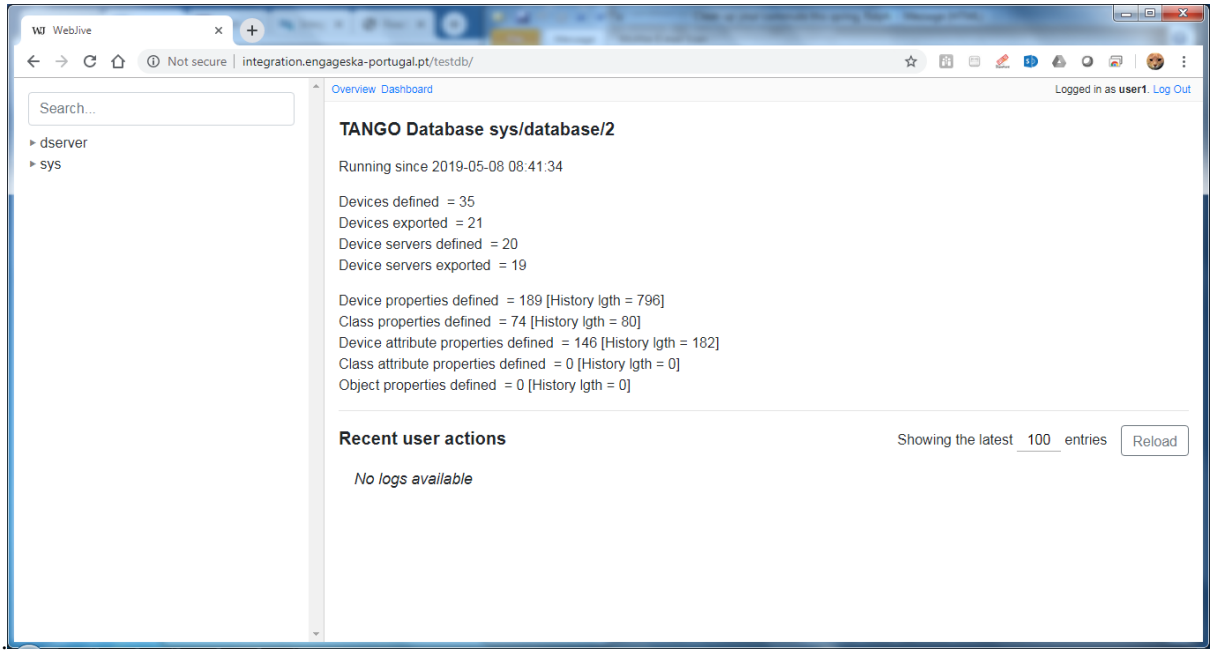


Figure 1. Screenshot to show the Webjive screen when user goes to 'localhost:22484/testdb' in web browser.

1.1.1 Webjive Widgets

The right hand side of the web interface, as highlighted in figure 2, presents the Webjive widgets which can be utilised in the creation of the Engineering User Interface by the user .

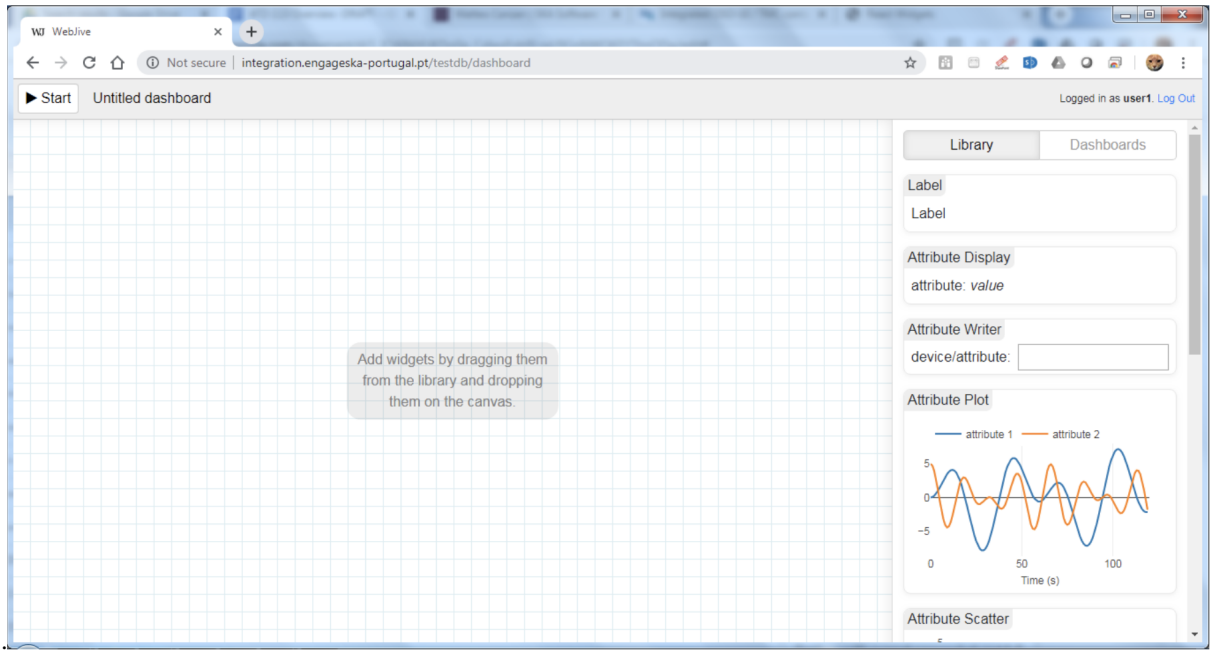


Figure 2. Screenshot to show Webjive screen when user goes to 'localhost:22484 /testdb/dashboard' in web browser. The available widgets are located on the right of the screen.

Webjive widgets, like react widgets, are components which allow the user to obtain, view and handle their data in a straightforward and repeatable manner.

Webjive widget are listed in the Webjive Official Documentation: https://webjive.readthedocs.io/en/latest/index.html

1.1.2 TangoGQL

The left hand side of the web interface houses the accessible Tango database devices. It should be possible to use the Tango Controls program Jive to access the same Tango devices database as what is presented in this column.

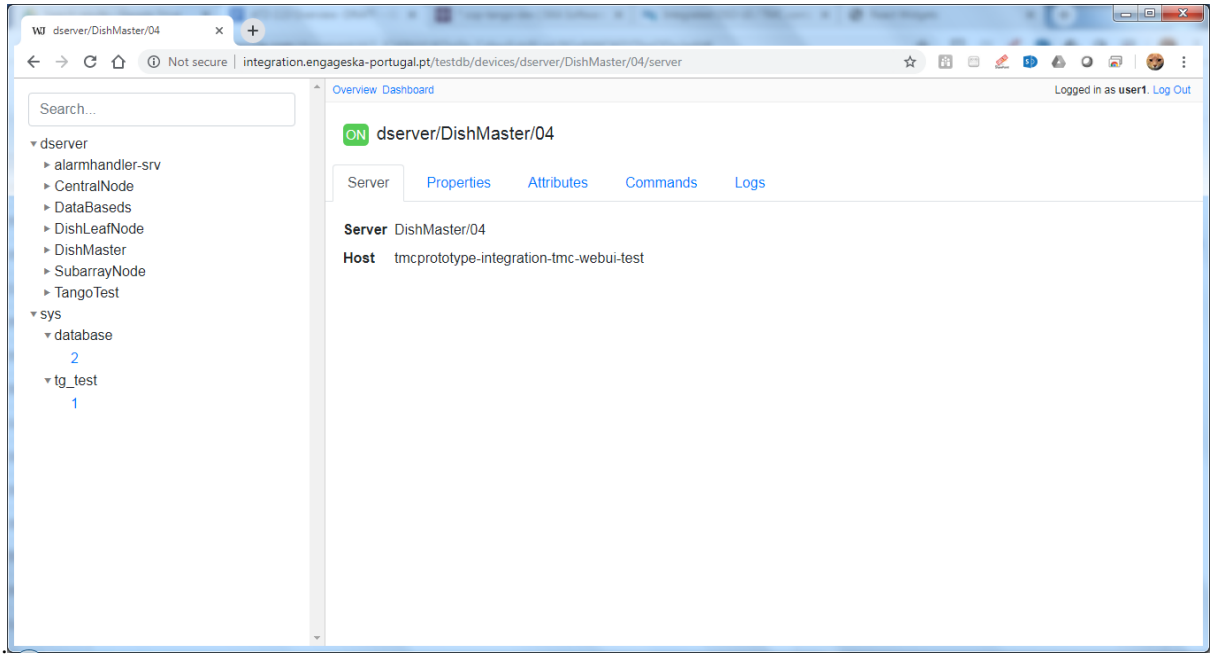


Figure 3. Screenshot to show the devices screen of Webjive. Tango devices that may be connected to are presented on the left of the screen.

Linking to TangoGQL

This activity should be viewed as a developer level activity. As such a procedure to add Tango devices to the linked database is provided in the SKA Developer portal at: https://developer.skatelescope.org/projects/ska-engineering-ui-compose-utils/en/latest/device.html

1.2 Steps to get Webjive running locally

The guide below assumes that the user has no previous versions of Webjive installed. At the time of writing there are a number of ways in which Webjive can be launched. However from a user point of view the way that should be adopted is given below because it will ensure that the 'latest' stable version is used. This guide assumes no previous versions of Webjive are present. **[#F1]_**

1.2.1 Prerequisites

It is assumed that the following are installed and working correctly before attempting to launch Webjive:

- Python 2.7.x
- Make
- Sphinx
- Git
- Docker 3.0

Steps

1. Obtain the latest ska-engineering-ui-compose-utils project from the. https://github.com/ska-telescope/ska-engineering-ui-compose-utils repository. The local ReadMe of this repository describes how to get Webjive up and running. Steps 2-4 below summarise the process.
2. Launch Webjive and TangoGQL. Using the terminal / command prompt, navigate to the local copy of ska-engineering-ui-compose-utils. Use the following make command to begin the setup process:

    make up

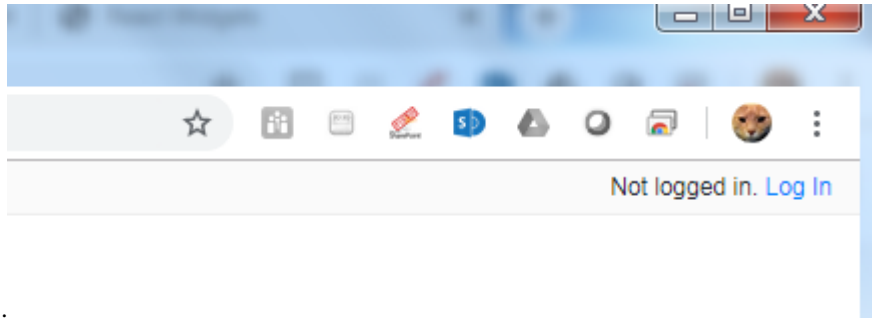
This step may take some time to complete because all of the supporting material for Webjive will be acquired from various repositories before being installed.

1. Go to your local web-browser and enter the following into the address / URL bar:

    localhost:22484/testdb/dashboard

The web browser should present a screen similar to that shown in figure 2.

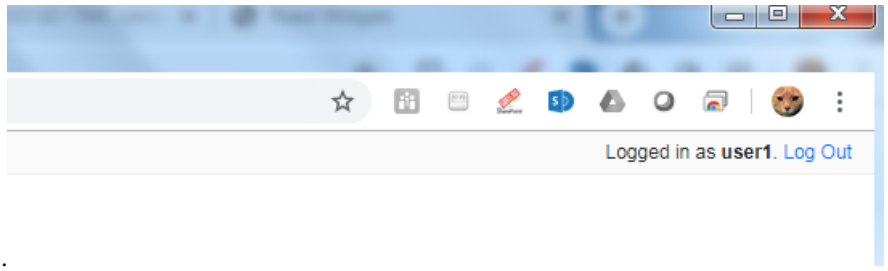
1. At the top right-hand corner of the webpage (not the browser) click on the login button and enter the following credentials**[#F2]_**.



.

Figure 4. Screenshot to show what the user should see when Webjive is running but no user logged in.

Currently SKA Webjive uses username and password reported in the *Webjive User Document*



.

Figure 5. Screenshot to show what the user should see when correctly logged in with the user1 credentials.

It should be noted that functionality is greatly reduced unless the user is logged in and it is not possible to save newly created or edited dashboards.

1.3 Quick Start

This section aims to provide a high level guide to using Webjive in terms of starting and stopping a session. It also provides an example of how a user can drag and drop a widget onto the canvas, followed by connecting it to a tango device present in the tango device database. This example is not an extensive how to guide to guide the user through all widget and device connection options, be rather an example which gives an idea about the approach that should be adopted when using Webjive.

1.3.1 Starting the Webjive session

Once the user has placed widgets on the user interface screen and connected them to the appropriate tango device, as described above, the session of Webjive can be started, i.e. data exchange between the device(s) and Webjive can commence. To do this the 'Start' button on the top left of the screen should be pressed. If started successfully, the 'Start' button name should change to 'Stop', and after a short delay pertinent data should be presented in the widget(s).

1.3.2 Stopping the Webjive session

To end a running session of Webjive, the user should press the button labelled 'Edit' in the top left of the screen, which is the exact same place where the 'Start' button was located.

1.3.3 Connecting Tango devices to Webjive widgets

Click on the Webjive dashboard button from the localhost//:22484/testdb/ page to get to the canvas and widget menu, as shown in figure 2.

Drag and drop the required widget on to the canvas. In this example the 'attribute plot' is dragged into the canvas, as shown in Figure 6.

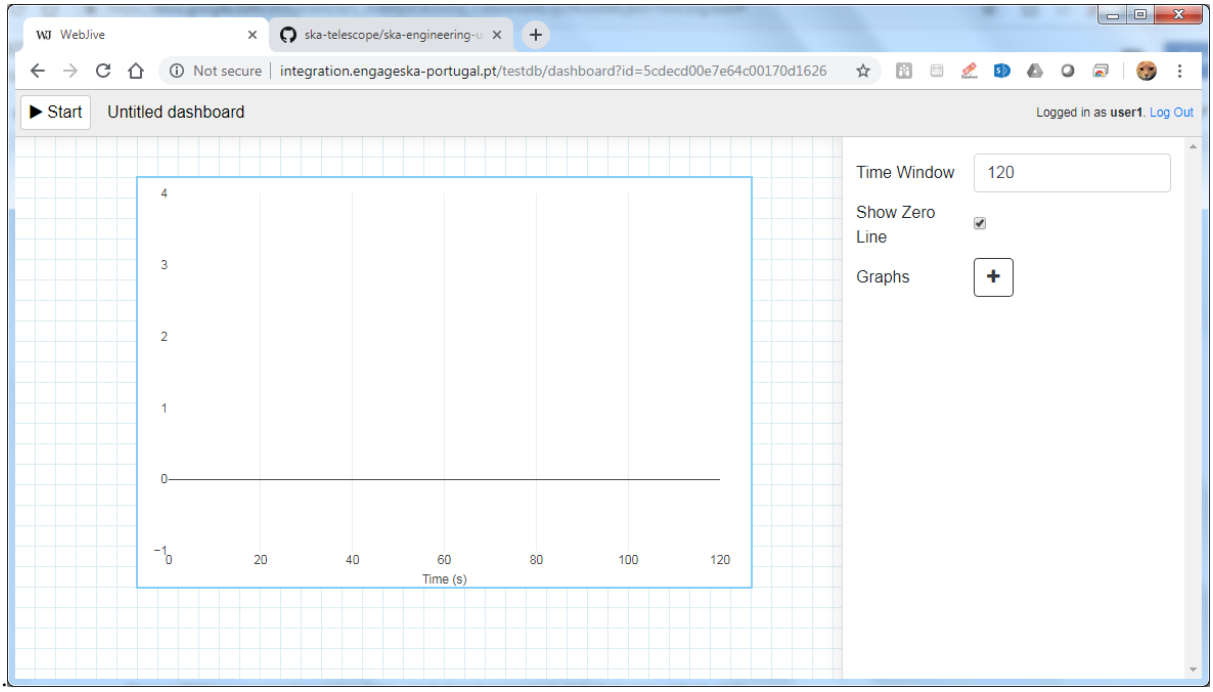


Figure 6. Screenshot showing the 'attribute plot' widget just dragged onto the canvas.

Once the widget is on the canvas the widget menu is replaced with a configuration table for the attribute plot, as can be seen in Figure 6. In this instance the user has the option to change the time frame of the data that is shown on the plot before the data begins to scroll. Click on the + labelled 'Graphs'. Begin to type in the path to the device which is to be connected to be presented with a reducing list of options of available devices (Figure 7).

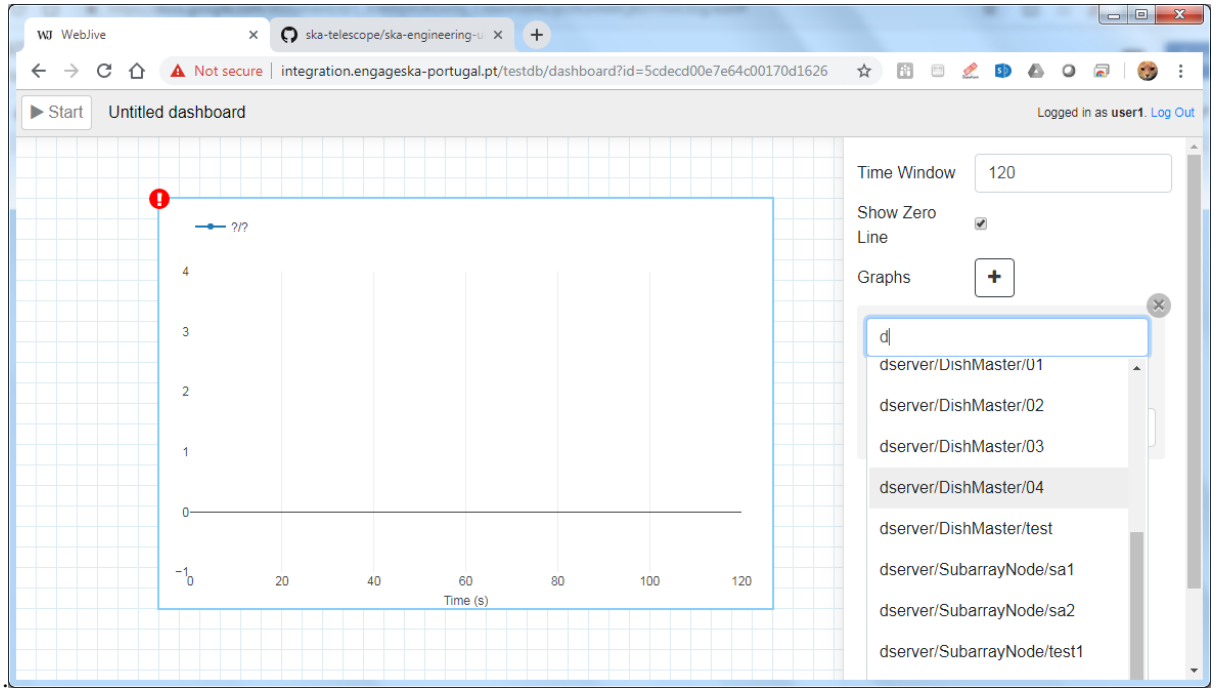


Figure 7. Screenshot showing the 'attribute plot' tango device options once the + button pressed.

Click on that which you which to be displayed in the attribute plot. In this example we will select 'sys/tg_test/1' and opt to retain the Y-axis on the left hand side of the display (Figure 8).

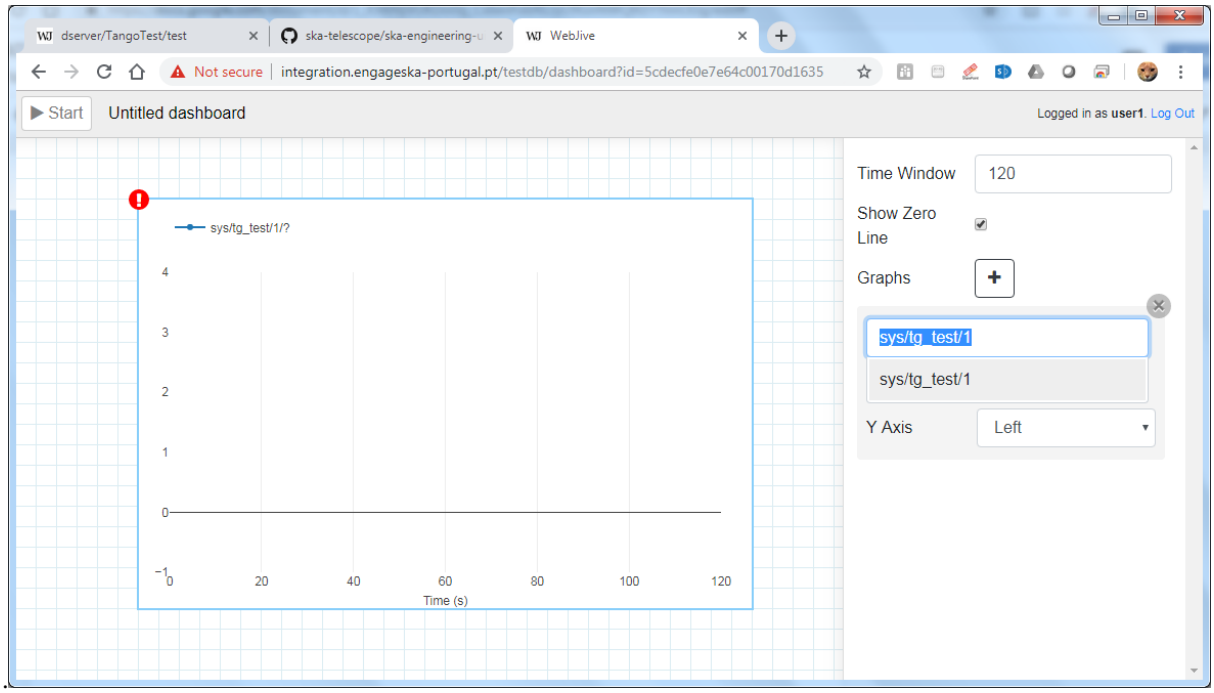


Figure 8. Screenshot showing sys/tg_test/1 being selected from Tango device options.

Now the user has to select an attribute(s) to be presented on the attribute plot. In this example we will opt to present just one attribute on the plot, which will be 'double scalar' as can be seen in Figure 9. To add further plots to this 'attribute plot', click on the + and repeat the process of selecting the device and attribute followed for 'double scalar'.

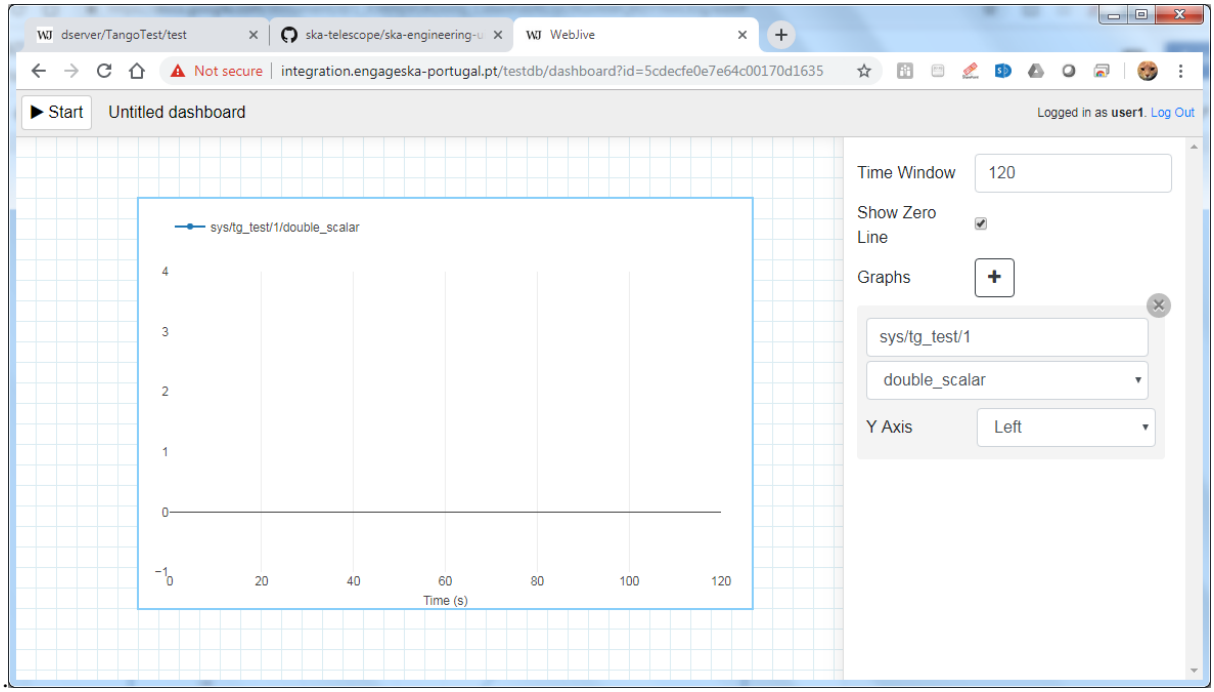


Figure 9. Screenshot showing the double_scalar attribute of 'sys/tg_test' being linked to 'attribute plot'.

Click on the 'Start' button. The buttons label changes to 'Edit', and the connection between the Tango device and the Webjive widget is established. Data will begin to flow between the device and the widget and be presented on the plot.

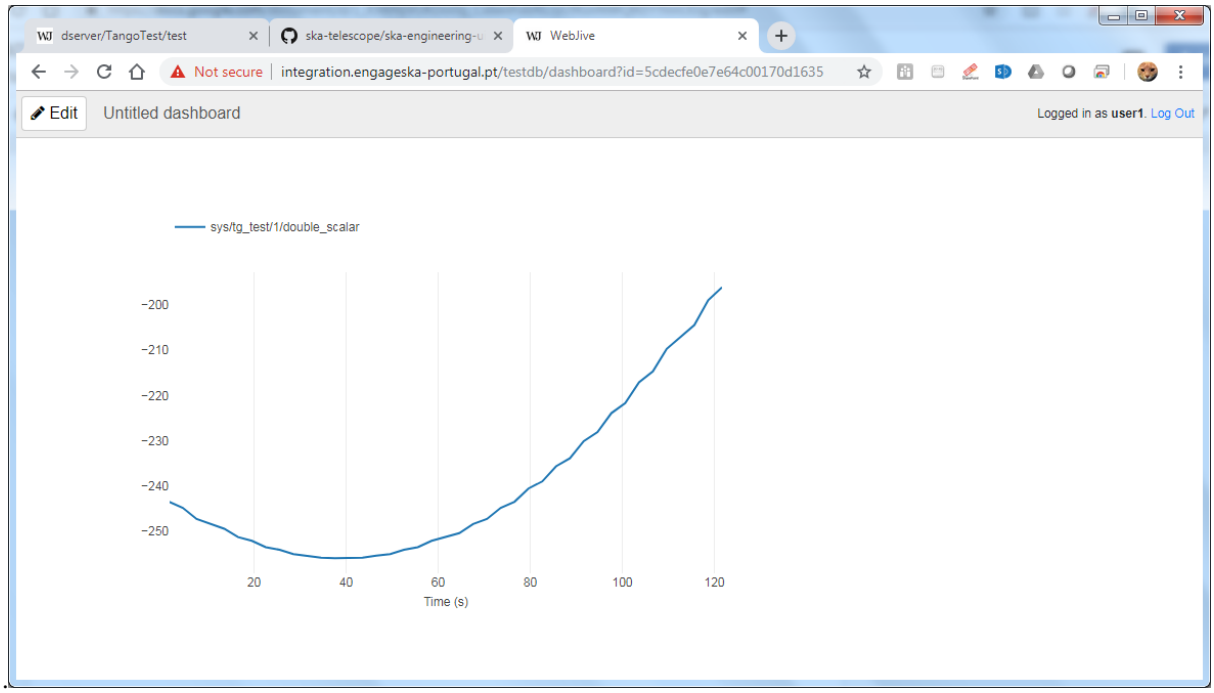


Figure 10. Screenshot showing the population of 'attribute plot' when the Webjive session is running.

Note that once the Webjive session starts (Figure 10), the user will be unable to edit the widget parameters or canvas layout until the session is stopped using the 'Edit' button.

To conclude the Webjive session, the user must press the 'Edit' button. This may seem counterintuitive, but in essence the 'Edit' button is the Stop button. Pressing this button does **not** imply that the user must edit the canvas.

For the purposes of this example the Tango device 'sys/tg_test/1' belonging to the Webjive Demo Tango Database was used. This is a good test device to use when setting up Webjive initially to establish correct operation. However when using your own Tango device or linked database, the user should establish the validity of the data being presented on the Webjive widget by some other means to ensure data integrity. One possible means of achieving this is to have an instance of Tango Jive running in parallel to ensure that the data being seen on webjive is the same as that seen on Jive.

Webjive Session Persistence

A key feature of Webjive is the ability to save a session layout which a User has created, so that it can be reused or edited in the future. It is important to note that exchanged data is not retained in this file, but rather the session configuration and layout.

1.3.4 Saving the Webjive session

Once a Webjive session layout has been created and appropriate links between widgets and Tango devices, it can be saved by giving the dashboard a new. A fresh dashboard automatically is named 'Untitled dashboard'. The user can simply click and delete this name and replace it with an appropriate dashboard name of choice.

1.3.5 Loading the Webjive session

To open a saved Webjive layout locate and click on the 'Dashboards' button (next to the Library button) at the top of the widget drag and drop menu. This will present the user with all available Dashboards. Locate the Dashboard the user wishes to open and click on it. After a short pause the dashboard will have loaded and its widgets displayed on the canvas.

It should be noted that editing the canvas of a dashboard will automatically modify that dashboard and will be saved as such.

Online Demo

It is possible to tryout Webjive before installing a local version. However this is limited in that the user cannot save or edit canvas or add new Tango devices to the database. The following link leads to the latest version of the demo available on the SKA repository. http://integration.engageska-portugal.pt/testdb

1.4 Webjive users

The following Users are set on Webjive:

User	Password
DEFAULT	DEFAULT_SKA
OMC	OMC_SKA
BUTTON	BUTTON_SKA
CIPA	CIPA_SKA
NCRA	NCRA_SKA
PERENTIE	PERENTIE_SKA
KAROO	KAROO_SKA
SKANET	SKANET_SKA
MCCS	MCCS_SKA
VIOLA	VIOLA_SKA

If your team is not above and you need a new user please drop a message in our #team-oso_ui channel

1.5 Docker services

The main content of this project is a set of Docker Compose files that define the containers (services) to run.

The following Docker services are defined:

Docker service	Description
tangodb	MariaDB database holding TANGO database tables
databaseds	TANGO database device server
tangogql	GraphQL interface to Tango control system
redis	Redis in-memory key/value database
webjive	WebJive container
auth	WebJive authentication service
dashboards	WebJive session persistence service
mongodb	Database for WebJive session persistence
dishmaster	TMC Dish LMC master Tango device
dishleafnode	TMC Dish leaf node Tango device
subarraynode1	TMC SubArrayNode Tango device #1
subarraynode2	TMC SubArrayNode Tango device #2
centralnode	TMC CentralNode Tango device
rsyslog-tmc	rsyslog container for TMC devices
tangotest	TANGO test device
jive	Jive - Tango GUI application
traefik	Reverse proxy used for WebJive HTTP routing
oet-ssh	Observation Tool Command Line Interface

1.6 Usage

To pull the required Docker images from the SKA Docker registry, execute:

```
# download all required Docker images
make pull
```

Optional: the images can be pulled from an alternate registry and/or account by supplying the DOCKER_REGISTRY_HOST and DOCKER_REGISTRY_USER Makefile variables respectively, e.g.,

```
# download foo/tango-cpp, foo/tango-jive, etc. from a registry at
# localhost:5000
make DOCKER_REGISTRY_HOST=localhost:5000 DOCKER_REGISTRY_USER=foo pull
```

To start WebJive and a minimal Tango system, execute:

```
# start WebJive and a Tango control system
make up
```

To start TMC devices, execute:

```
# start TMC devices
make tmc
```

Optional applications and device servers can be launched by calling the *start* make target followed by the name of the service. For example:

```
# run Jive
make start jive
# run tangotest
make start tangotest
```

To display the status of the Docker services, execute

```
# print status of Docker services
make status
```

Running services can be stopped individually or as a whole using the *stop* make target or *down* make target respectively. For instance,

```
# stop just the tangotest device server, leaving other services running
make stop tangotest
# stop all services and tear down the system
make down
```

After starting the WebJive containers and any required additional containers, navigate to *http://localhost:22484/testdb* to access WebJive. The following credentials can be used:

Username user1

Password abc123

1.7 Joint Process for Contribution between Max IV and SKA

This relates to how the teams at Max IV and SKA have agreed to collaberate on supporting and ehancing the webjive suite of tools.

1.7.1 What is this collaboration about?

This is a collaboration between software teams in MaxIV and the Square Kilometer Array (SKA) to jointly work on developing the webjive suite. This is a number of closely related products that can be used to provide a web-based interface to a Tango Control System.

1.7.2 What are the Products we are collaborating on?

Purpose	Repository	Description
A tool for creating Dashboards for interacting with the devices within a Tango Control System	https://gitlab.com/MaxIV/webjive	The tool for developing these dashboards is webjive itself.
Queryable access to a Tango Control System that can be used by the dashboard creation tool	https://gitlab.com/MaxIV/web-maxiv-tangogql	Currently, this is a TangoGQL. A GraphQL web server that integrates with the TangoDB services and cancommunicate directly with tango devices.
Storing and Sharing the configuration of developed dashboards between users	https://gitlab.com/MaxIV/dashboard-repo	A MongoDB based dashboard repository for storing webjive dashboard layouts
Authorization and Authentication for the tools	https://gitlab.com/MaxIV/webjive-auth	A simple authentication and authorization service for the webjive and TangoGQL tools that can be hooked into a corporate authentication solution
Supporting further development	https://gitlab.com/MaxIV/webjive-develop	A set of developer scripts and tools used for setting up and developing these related products.

This division is not defined in any specification, but rather has emerged from the needs of the application. In the future, services might be added, merged or made obsolete.

1.7.3 Planning Process

Priorities are agreed on a regular basis between the MaxIV Product and Feature Owners who meet regularly with their SKA compatriots. Within the SKA Planning fits within the current 3 monthly cycle for software. The outcomes of the SKA planning and joint discussions with Max IV are:

- An agreed set of common priorities between SKA and MaxIV.
- An updated backlog of webjive suite features recorded as GitLab Issues against the relevant project (general issues being raised against webjive-develop)
- A single view of all the GitLab issues across the webjive suite that summarises the current GitLab Issues from each of the contributing projects.
- A plan for the next 3 months of work covering which features are going to be worked on in which 2wk period, with these features having an approximate size, acceptance criteria, an allocated feature owner and agreement which team is best placed to tackle them.

(It is important that SKA discuss their priorities with MaxIV and vice versa before tickets are raised)

Once accepted as part of a team's work for the next period the GitLab issue should be updated to include a cross-reference to the internally tracked work item (JIRA for SKA / TAGIA for MaxIV)[1]

1.7.4 Making Changes

- **There is a single master branch that should always be deployable and usable.** This is supported by Continuous Integration by both MaxIV and SKA
- **Changes are made via short-lived feature branches that are merged back into master.** Branch names should reflect the GitLab ticket being worked on
- **All changes should have an associated GitLab ticket.**

[1] There are plugins that might help to synchronise. TBC - need to add the details of the process.

- **All features are merged via pull requests** - at least initially it would make sense to have all changes reviewed by one MaxIV and one SKA developer before being merged back into master. **HotFix/Emergency Fixes** are reviewed quickly by a second developer on the same side and communicated retrospectively **Review policy for planned changes** when a change is ready for merging with the trunk the party responsible raises a pull request, requesting a review with a priority (urgent, high medium, low priority). A priority defines generally how urgently the review should be actioned. At this early stage of the collaboration, this is on a best efforts basis without any specific deadlines. If it is not possible for the other party to review the code then the code will be merged back to the master after peer review by another developer within the collaboration.

1.7.5 Testing

The tests for these projects are not currently comprehensive in terms of coverage. The ambition is to evolve this over time. We want to ensure that new code developed is delivered with tests that demonstrate that it is fit for purpose, and is documented in a way that makes it easy to maintain.

Testing legacy code

- It is **not necessary to add tests on existing code**.
- When a **bug in legacy code has been discovered**: open a GitHub issue, fix the bug and create tests. In this way, it is possible to improve coverage.

Testing of new code or changed functionality:

- **All new or changed code should have tests**
- **All new or changed code should have documentation**
- **Trunk should always be clean and deployable** - no breaking code, all tests and linting OK. The CI/CD should run cleanly on both sides.

1.7.6 Coding Standards and Programming Language Conventions

Webjive and related JavaScript based projects

We are starting from the principle that we want to move towards a defined coding standard, but without forcing a large amount of cosmetic change on the existing codebase.

The following tools are integrated into the environments. This list may be expanded or change as the needs of the project changes.

Purpose	Tool of Choice
linting of files	eslint, @typescript-eslint/parser and @typescript-eslint/eslint-plugin
testing of code	jest, ts-jest, enzyme,
formatting of code	prettier
code coverage	jest
dependency management	npm
CI/CD	GitLab pipeline (gitlab-ci.yml file)

All existing code should as a minimum conform to the de-facto linting and formatting rules within the workspace.

These are currently relaxed in a number of areas, but the plan would be to improve the quality of the code over time enforcing stricter rules based on best practices defined by Airbnb and Microsoft after discussion.

Any exceptions to this would be documented on the publicly accessible SKA developer guidance for javascript.

For personal linting or formatting of code, it is suggested that developers use the appropriate AirBnB standards rules and plugins for their preferred editor. Guidance for set-up and configuration to be supplied as part of the readme on the projects

Use of Typescript

The use of Typescript is acknowledged and supported. It is quite acceptable for TSX files to contain JSX syntax. Typescript code should conform to the current typescript rules for static typing (currently 2.7 is enforced 3.3 is suggested for any new code)

For compatibility with the current codebase, the following rules are not enforced however any new or change code should however be written so that it would compile and run with these rules in place

- **strictFunctionTypes** Ensure that all functions can be proved to be type safe.[2]
- **strictPropertyInitialization** Ensures that all properties are initialized for every possible code path.[3]
- **noImplicitAny** Currently if the compiler cannot infer the variable type based on how it's used it silently defaults the type to any. At some point, we want to switch this to true so that if the TypeScript compiler cannot infer the type, it still generates the JavaScript files, but it also reports an error. This stricter type checking catches more unintentional errors at compile time.

Code Structure

The code should, in general, be grouped by features or routes.[4] with CSS, JS, and tests grouped together inside folders. Follow the existing webjive structure where possible:

- **dashboard** : code relating to the main dashboard display and
- **jive** : code relating to the device lists and RHS panel
- **shared** code used by both

Within this structure, there is a separation between code related to the state management and the widgets presented on the display.

The folder structure within 'components' reflecting a hierarchical view of the individual components.

TangoGQL and other Python-related projects

Any jointly developed changes should follow the SKA Python programming guidelines

1.7.7 Definition of Done

This is based on the SKA project 'Definition of Done' for software projects

[2] see description & ref article in https://www.typescriptlang.org/docs/handbook/release-notes/typescript-2-6.html
[3] see https://patrickdesjardins.com/blog/typescript-strictpropertyinitialization-should-be-turned-on
[4] For a discussion of the benefits of grouping by structure see https://reactjs.org/docs/faq-structure.html

Ticket/Story

- Code is supplied with an acceptable license.
- Code is peer-reviewed (via pull-request process).
- Code is checked into the repository with reference to GitLab ticket.
- The code has tests that have adequate (between 75% and 90%) coverage[5]
- The code compiles cleanly with no warnings.
- Code adheres to SKA and MaxIV agreed language specific style.
- Code is deployed to a continuous integration environment for both MaxIV and SKA.
- The code passes regression testing.
- The code passes 'smoke' testing.
- NFRs are met
- The story is tested against acceptance criteria.
- The story is documented.
- Story ok-ed by Product Owner.

Code documentation

- Public API exposed is clearly documented
- Code is documented inline according to language-specific standards
- Documentation is peer-reviewed by stakeholder (e.g. Product Owner for a feature or technical peer for an enabler) via the pull-request mechanism.
- Documentation is deployed to an externally visible website accessible via the SKA developer portal

Feature

- The feature has been demonstrated to relevant stakeholders
- Feature meets the acceptance criteria
- The feature is accepted by Feature owner
- The feature is integrated into both integration environments (MaxIV and SKA)
- Code documentation is integrated as part of the project documentation (and developer portal as relevant for SKA)
- SKA / MaxIV Architectural documentation is updated to reflect the actual implementation

[5] Pragmatism is assumed. We will focus testing on the core components where failure would impact multiple users. For example, users can create and deploy their own custom widgets. A failing widget only impacts the users of that widget. Here good-enough testing to cover the situations the widget has been developed for may well be more lightweight than the coverage percentages would suggest.

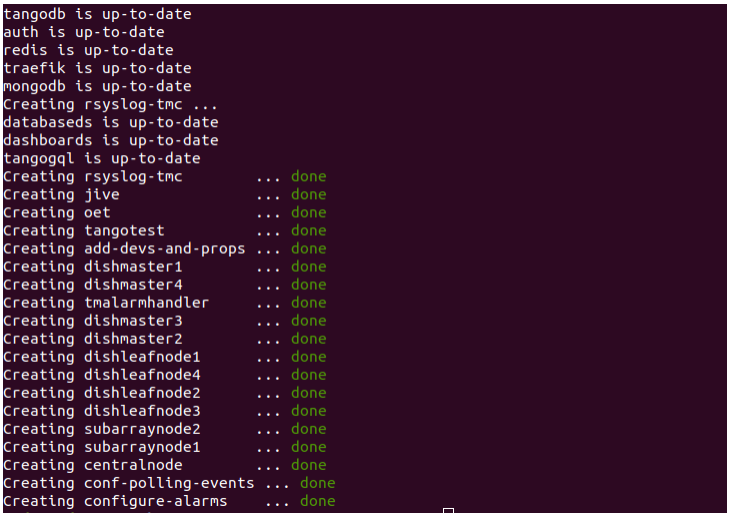
1.7.8 Notes

1.8 Basic steps to link Webjive to a real tango device

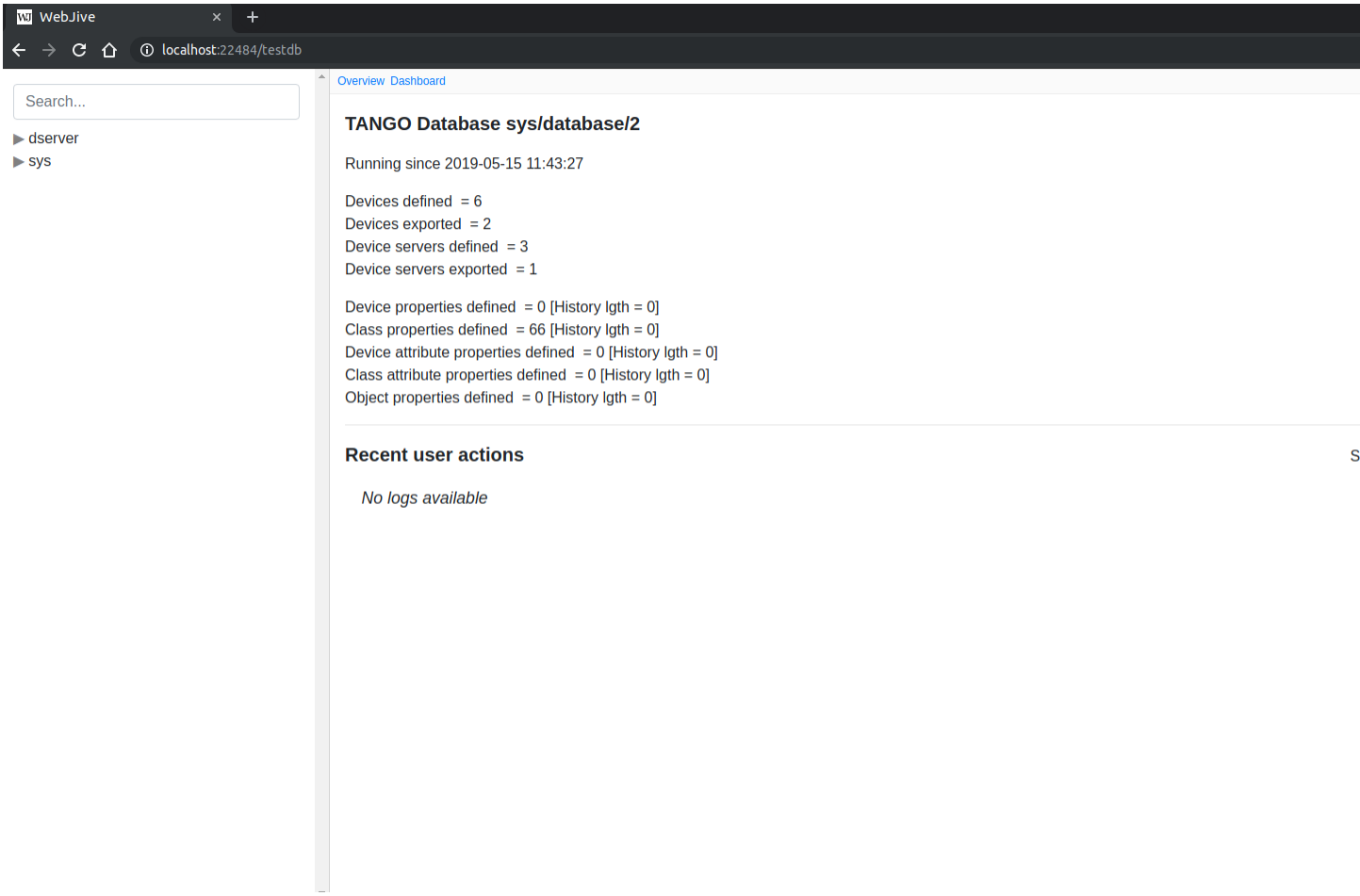
This relates to how a developer can connect a new tango device to webjive.

1.8.1 First step setup the local developer environment

In order to have WebJive and a minimal Tango system working on your local environment you should follow the guidelines on *Usage*. After the *make up* command you should see something like this on the command line:



To verify that everything is running well after those steps just go to and you should see something like this:



1.8.2 Create a new device to add on webjive

To create a new device on your local environment, just follow the documentation on . Alternately to test you can just use the already built docker image on *nexus.engageska-portugal.pt/tango-example/powersupply:latest*

1.8.3 Setup tango device connection to webjive

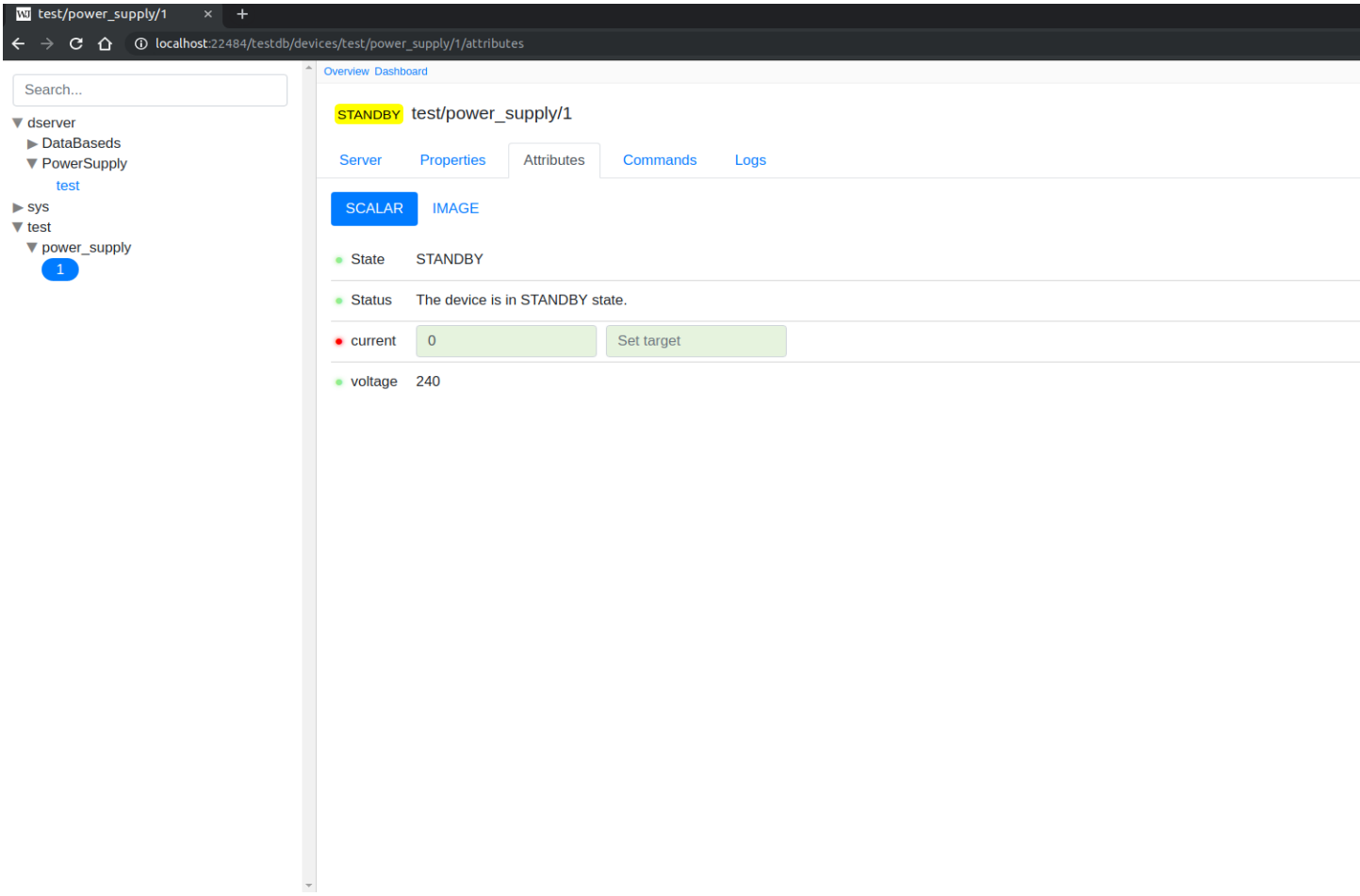
The connection between webjive and a tango device or server is done by the var *TANGO_HOST* for more info about this go to

To see how to setup the connection between tango-example and webjive just follow

This *tango-example.yml* file is already in this project, to start up tango-example just do

```
make start tango-example
```

By the end of the command if you go to http://localhost:22484/testdb you should see the following:



1.8.4 Debug the created tango device

In order to be able to debug the device you just created you can for example run the following code[1]:

```
TANGO_HOST=databaseds:10000 NETWORK_MODE=tangonet MYSQL_HOST=tangodb:3306 CONTAINER_
↪NAME_PREFIX= COMPOSE_IGNORE_ORPHANS=true docker-compose -f tango-example.yml up
```

Then you should see this on the command line:

```
code@code-VM:~/ska-engineering-ui-compose-utils$ TANGO_HOST=databaseds:10000 NET
WORK_MODE=tangonet MYSQL_HOST=tangodb:3306 CONTAINER_NAME_PREFIX= COMPOSE_IGNORE
_ORPHANS=true docker-compose -f tango-example.yml up
Starting powersupply ... done
Attaching to powersupply
powersupply      | wait-for-it.sh: waiting 30 seconds for databaseds:10000
powersupply      | wait-for-it.sh: databaseds:10000 is available after 0 seconds
```

This is the output of tango-example device (powersupply), this will vary from different devices

In order to debug all the devices and webjive-suite itself you can just run[2]:

```
make debug
```

You should see something like this on the command line:

[1] The vars my change on different machines, running *make up* you get your vars on the command line
[2] This will also start jive

```
(KHTML, like Gecko) Chrome/74.0.3729.157 Safari/537.36"
webjive               | 172.19.0.4 - - [15/May/2019:14:01:47 +0000] "GET /favicon.ico HTTP/1.1" 200 8213 "http://localhost:22484/testdb/devices/test/power_supp
Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.157 Safari/537.36"
webjive               | 172.19.0.5 - - [15/May/2019:14:24:05 +0000] "GET /testdb HTTP/1.1" 200 1097 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KH
7 Safari/537.36"
webjive               | 172.19.0.5 - - [15/May/2019:14:24:06 +0000] "GET /favicon.ico HTTP/1.1" 200 8213 "http://localhost:22484/testdb" "Mozilla/5.0 (X11; Lin
, like Gecko) Chrome/74.0.3729.157 Safari/537.36"
webjive               | 172.19.0.5 - - [15/May/2019:14:24:25 +0000] "GET /testdb/devices/test/power_supply/1/properties HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11;
TML, like Gecko) Chrome/74.0.3729.157 Safari/537.36"
webjive               | 172.19.0.5 - - [15/May/2019:14:24:25 +0000] "GET /favicon.ico HTTP/1.1" 200 8213 "http://localhost:22484/testdb/devices/test/power_supp
Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.157 Safari/537.36"
webjive               | 172.19.0.5 - - [15/May/2019:14:25:05 +0000] "GET /testdb/devices/test/power_supply/1/attributes HTTP/1.1" 200 1097 "-" "Mozilla/5.0 (X1
(KHTML, like Gecko) Chrome/74.0.3729.157 Safari/537.36"
webjive               | 172.19.0.5 - - [15/May/2019:14:25:06 +0000] "GET /favicon.ico HTTP/1.1" 200 8213 "http://localhost:22484/testdb/devices/test/power_supp
Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.157 Safari/537.36"
webjive               | 172.19.0.7 - - [15/May/2019:14:35:41 +0000] "GET /testdb HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML
afari/537.36"
webjive               | 172.19.0.7 - - [15/May/2019:14:35:50 +0000] "GET /testdb/devices/test/power_supply/1/server HTTP/1.1" 200 1097 "-" "Mozilla/5.0 (X11; L
ML, like Gecko) Chrome/74.0.3729.157 Safari/537.36"
webjive               | 172.19.0.7 - - [15/May/2019:14:35:50 +0000] "GET /favicon.ico HTTP/1.1" 200 8213 "http://localhost:22484/testdb/devices/test/power_supp
x x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.157 Safari/537.36"
dishleafnode1         | wait-for-it.sh: waiting 30 seconds for databaseds:10000
dishleafnode1         | wait-for-it.sh: databaseds:10000 is available after 0 seconds
subarraynode1         | wait-for-it.sh: waiting 30 seconds for databaseds:10000
dishmaster3           | wait-for-it.sh: waiting 30 seconds for databaseds:10000
dishmaster3           | wait-for-it.sh: databaseds:10000 is available after 0 seconds
rsyslog-tmc           | 2019-05-15T14:52:46.962397+00:00 59cbd0e6be94 sudo: pam_unix(sudo:session): session closed for user root
rsyslog-tmc           | 2019-05-15T14:52:47.950494+00:00 59cbd0e6be94 sudo: pam_unix(sudo:session): session closed for user root
rsyslog-tmc           | 2019-05-15T14:52:49.071409+00:00 59cbd0e6be94 sudo: pam_unix(sudo:session): session closed for user root
rsyslog-tmc           | 2019-05-15T14:52:49.911920+00:00 59cbd0e6be94 sudo: pam_unix(sudo:session): session closed for user root
add-devs-and-props exited with code 0
subarraynode1         | wait-for-it.sh: databaseds:10000 is available after 0 seconds
dishleafnode3         | wait-for-it.sh: waiting 30 seconds for databaseds:10000
tangogql              | [13:36:02] Starting aux server at http://localhost:5005 ◆
tangogql              | [13:36:02] Starting dev server at http://localhost:5004 ●
tangogql              | 2019-05-15 13:36:02,975 - DEBUG - Logging setup done. Logfile: /var/log/tangogql/256323b60e2d.log
tangogql              | [13:36:06] ● POST /db 200 810B 7ms
tangogql              | [13:36:06] ● POST /db 200 527B 221ms
tangogql              | [13:36:11] ● POST /db 200 531B 222ms
tangogql              | [13:36:37] ● POST /db 200 1.3KB 13ms
tangogql              | [13:36:37] ● POST /db 200 7.3KB 31ms
tangogql              | [13:36:41] ● POST /db 200 1.3KB 5ms
tangogql              | [13:36:41] ● POST /db 200 772B 8ms
tangogql              | [13:36:41] ● POST /db 200 312B 5ms
tangogql              | [13:37:29] ● POST /db 200 1.3KB 7ms
tangogql              | [13:37:29] ● POST /db 200 772B 7ms
tangogql              | [13:37:29] ● POST /db 200 312B 7ms
```

This will update with the runtime debug, you can just refresh the webjive page to see the output

1.8.5 Notes

1.9 Webjive Suite Publish-Subscribe Mechanism

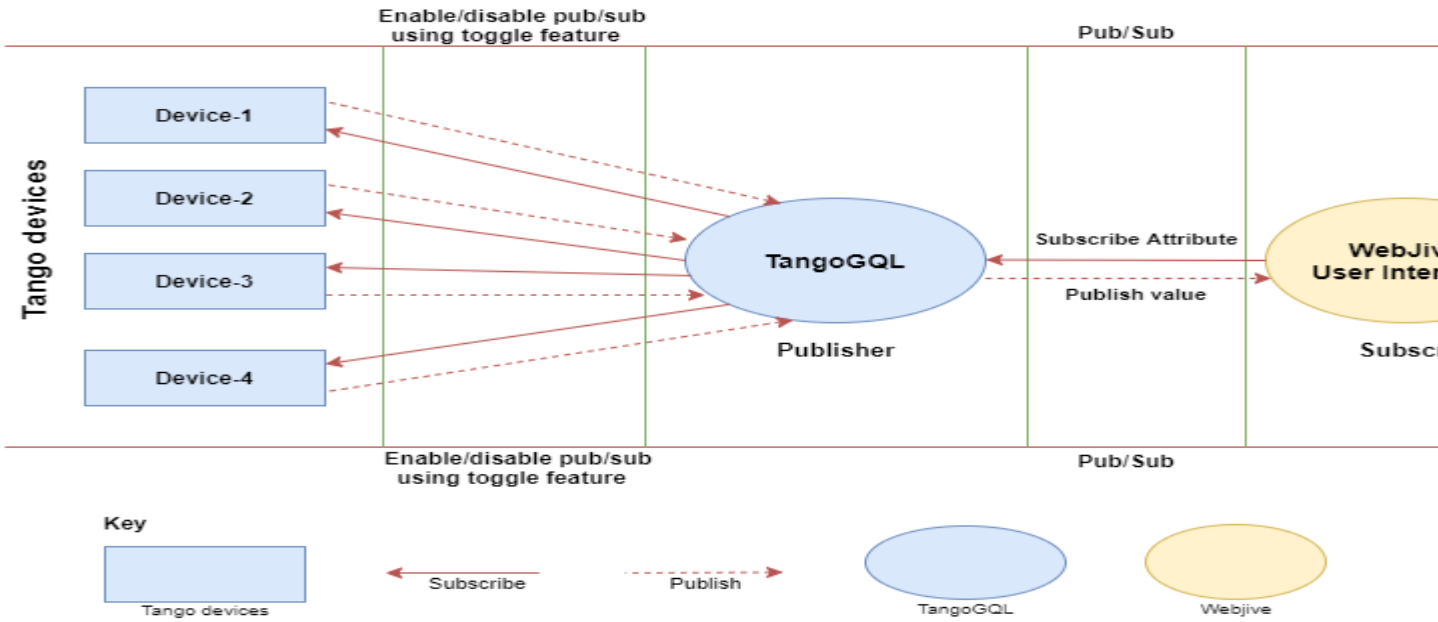
1.9.1 Introduction

Publish/Subscribe is a software design pattern that describes the flow of messages between applications, devices, or services in terms of a publisher-to-subscriber relationship.

publish/subscribe mechanism works like this: a publisher (i.e. any source of data) pushes messages out to interested subscribers (i.e. receivers of data) via channels . All subscribers to a specific publisher channel are immediately notified when new messages have been published on that channel, and the message data (or payload) is received together with the notification.

In the Webjive suite, TangoGQL works as publisher and Webjive User Interface works as subscriber.

When subscription to an attribute is made, the Webjive suite establishes the type of polling mechanism that attribute uses. With this knowledge the Webjive suite determines and selects the most appropriate way to interact with Tango. If Tango events are set up for that attribute, they will be used. However if they aren't the Webjive suite reverts back to using the original mechanism of TangoGQL polling the attribute then publishing updates to the Webjive user interface when it changes.



.

1.9.2 Enable/disable publish-subscribe feature from TangoGQL

TangoGQL has a function called features toggle capable of controlling some features such as publish-subscribe. There is a file inside tangogql/ called tangogql.ini, the file looks like this:

```
# this configuration file is used to hold details of which features
# currently enabled in TangoGQL ( True = enabled False = disabled)

[feature_flags]
# Publish Subscribe is currently disabled for SKA
publish_subscribe = False
```

Changing the *publish_subscribe = True* will enabled pub/sub on TangoGQL, in this case, TangoGQL will try to Subscribe to changeEvents on the device, if it fails it tries PeriodicEvents, and if that fails it falls back to polling

1.9.3 Setting up publish-subscribe on Webjive suite

Getting Started

This page shows how to set up the Webjive suite publish-subscribe (pub/sub) mechanism using the 'tangotest' and 'webjivetestdevice' tango device servers. When setting up pub/sub for your own tango devices, you should substitute the aforementioned tango devices with your own.

To follow this guide you will need to have an instance of Webjive suite and Tango Jive running locally. The easiest way of doing this is to use the images available through the Gitlab repository ska-engineering-ui-compose-utils.

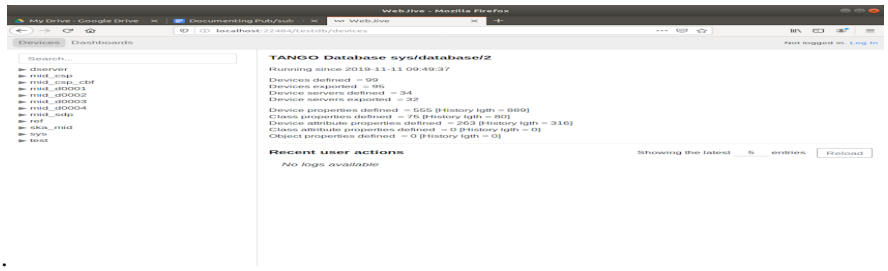
Using a terminal, go to the ska-engneering-ui-compose-utils. Using the following make commands create tango DB and register all devices, start the mvp, tango test and webjive test device images. Webjive suite will be launched as part of this.

- make ds-config
- make mvp
- make start tangotest

- make start webjivetestdevice

Go to a web-browser of choice and open the Webjive suite:

http://localhost:22484/testdb/devices



.

Figure 1. Screenshot to show Webjive suite 'Devices' screen when user goes to 'localhost:22484/testdb' in web browser.

Tango Jive Set up

To run up an instance of Jive in order to set up device polling and events.

- make start jive

After a short pause Jive should launch and present all available Tango devices. Locate the TangoTest tango device. Navigate through TangoTest–>test–>TangoTest–>sys/tg_test/1. At this level you should be able to see 'Polling' and 'Event', which when clicked on will allow the user to modify device attribute settings.

Figure 2. Screenshot to show location of sys/tg_test/1 using Tango Jive .

Modifying Polling characteristics

In Polling, change polling attribute of double_scalar from default (3000) to 100 (ms). Also ensure that Polled is selected and ticked as shown below.

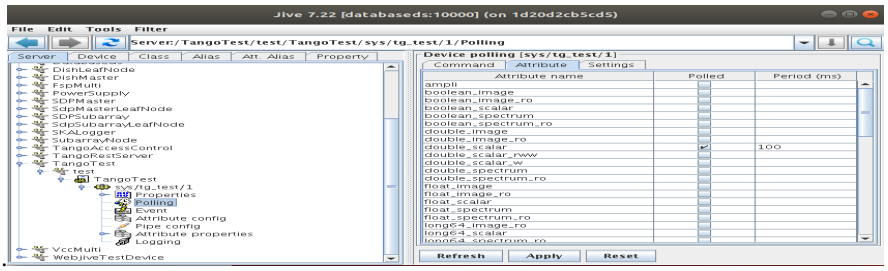


Figure 3. Screenshot to show the Attribute tab of sys/tg_test/1 Polling characteristics.

Modifying Event characteristics

In Event, select “Periodic event”, then for attribute “double_scalar” from default (3000) to 100 (ms)

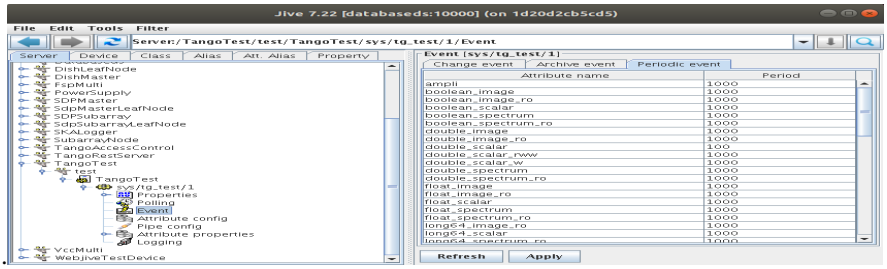


Figure 4. Screenshot to show the Attribute tab of sys/tg_test/1 Event characteristics.

Verifying in Webjive Suite

Note. If Webjive suite is already running, in order to apply these new polling and event settings, it is advised to stop and then restart Webjive suite.

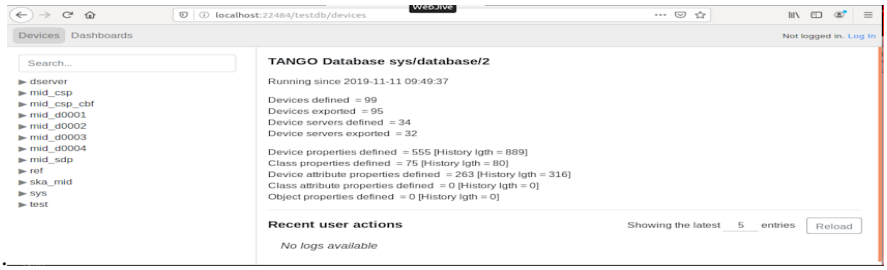


Figure 5. Screenshot to show the available Tango devcies in Webjive suite.

Now you should go in to the devices list and ensure that “sys/tg_test/1” is in a running state. This can be confirmed by looking at the top of the right hand side pane of the browser, a green box with “RUNNING” written in it should be visible. If it is not present, the tango test image was not successfully launched, and so this step should be run using the ‘make start tangotest’ command from the terminal.

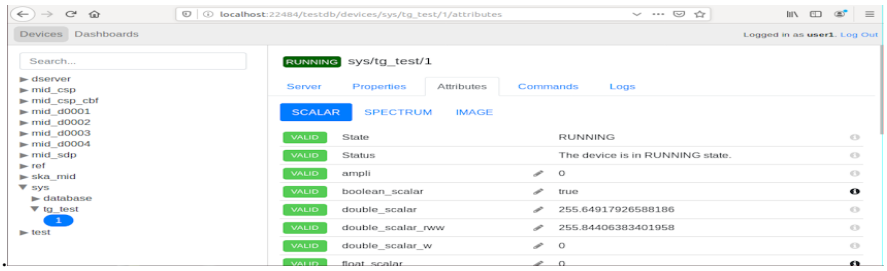


Figure 6. Screenshot to show the Scalar Attributes of the sys/tg_test/1 Tango device.

Once the tango test device is confirmed as RUNNING, go to the ‘Dashboard’ of the Webjive suite. From the right hand side widget menu, select the “Attribute Display” widget and drag and drop an instance over onto the left hand side canvas. Configure the widget as:

- Device: sys/tg_test/1
- Attribute: double_scalar

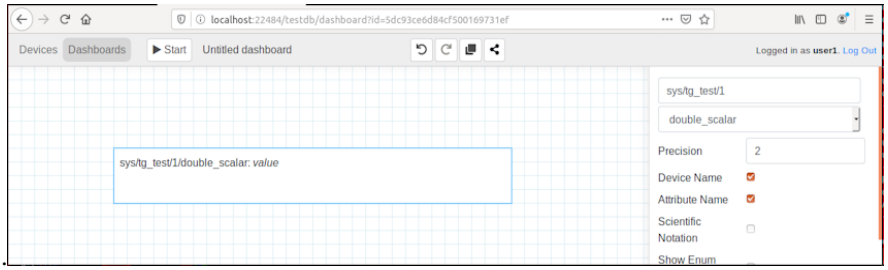


Figure 7. Screenshot to show the Attribute display widget being set up on the Webjive suite dashboard.

Once set up, click on the "Start" button to run the dashboard. After a short pause you should see the displayed attribute value update.

.

Figure 8. Screenshot to show Webjive session running and showing the double_scalar value on the attribute display widget.

1.9.4 Comparison

In order to demonstrate how the pub/sub can be used to allow different device attributes to be presented at different periodicity, the same process should be repeated for the device webjivetestdevice. The Tango device webjivetestdevice was created to allow the pub/sub mechanism to be demonstrated. It facilitates this by allowing a greater ability to configure polling and event periodicity that what can be achieved with the tg_test device. The tg_test device is limited to only changing its value every second - so even if polling is set to more frequently you won't see any difference, hence webjivetestdevice was written which does not have this restriction.

- Tango Device: test/webjivetestdevice/1
- Attribute: RandomAttr

Figure 9. Screenshot to show Tango Jive and the location of the WebjiveTestDevice in the sever listing.

Using Jive go to the Polling icon of "WebjiveTestDevice–>test–>WebjiveTestDevice–>test/webjivetestdevice/1". For attribute RandomAttr, set the polling period to 500(ms) on the Attribute tab. Ensure that the polled option is ticked.

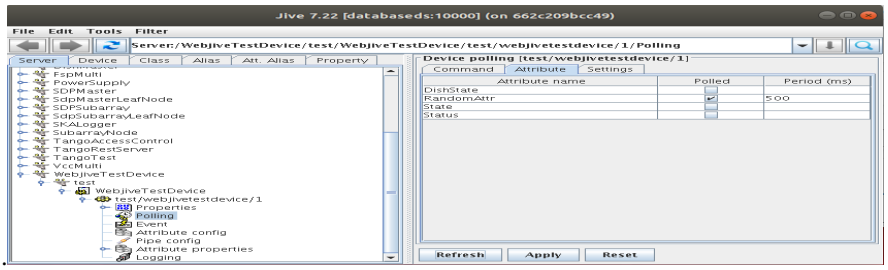


Figure 10. Screenshot to show Tango Jive and Attribute tab in which the Polling characteristics of the selected attribute needs to be activated and an interval be stated.

For the same Tango Device, select the Event icon. For the RandomAttr attribute set the period to 1000 (ms) on the Periodic event tab. Furthermore, RandomAttr has the Change Event set in order to send events if the current value differs by 1% from the previous value

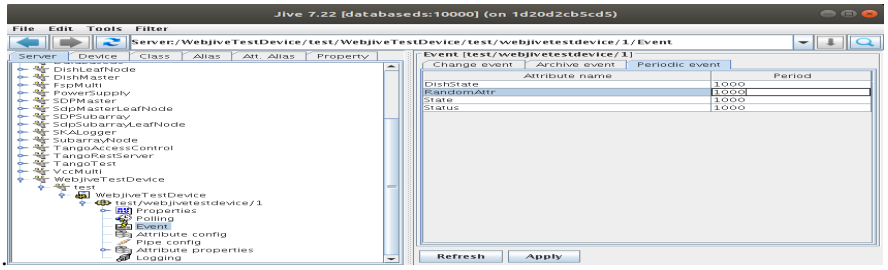


Figure 11. Screenshot to show Tango Jive and Attribute tab in which the Event characteristics of the selected attribute needs to be activated and an interval be stated.

Once the tango devices have been set up in Jive, go back to the Webjive suite and drag a new Attribute Display widget onto the canvas. Set up the Attribute display widget to present the RandomAttr device attribute values in Webjive.

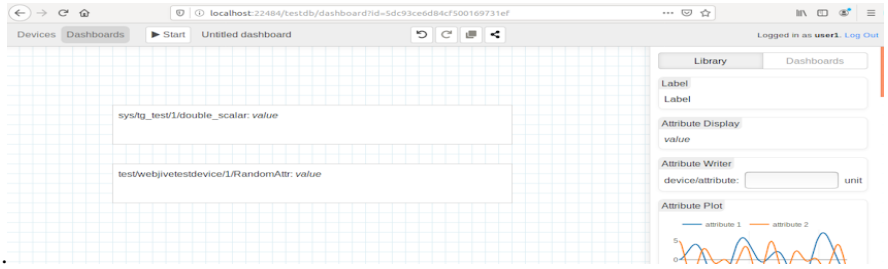


Figure 12. Screenshot to show Webjive dashboard showing the double_scalar value of tg_test and webjiveTest-Device RandomAttr on seperate attribute display widgets.

Now run the Webjive suite dashboard by clicking on the Start button. If set up correctly you should see the two individual device attributes update at different intervals (as defined by the polling and event intervals set via Jive).

1.10 TangoGQL Logging in SKA

TangoGQL logging system uses a file called *logging.yaml* by default to configure the logging capabilites.

Please refer to the TangoGQL Official Documentation for the logging details: https://web-maxiv-tangogql.readthedocs.io/en/latest/logging.html

To change the format of the logging, for example to the SKA standard one can simply change this line:

```
format: "1|%(asctime)s.%(msecs)03dZ|%(levelname)s|%(threadName)s|%(funcName)s|
↪%(filename)s#%(lineno)d|%(message)s"
```

Optional: There is a way to pass a new file to tangoGQL using the *LOG_CFG* var, and example can be found on the *tangogql.yml* file

```
version: "2.2"

 volumes:
 tangogql-logs: {}

 services:
 tangogql:
     image: web-maxiv-tangogql_tangoql:latest
     container_name: ${CONTAINER_NAME_PREFIX}tangogql
     network_mode: ${NETWORK_MODE}
     command: /bin/bash -c "source activate graphql && adev runserver tangogql/
↪aioserver.py --app-factory=dev_run --port=5004"
     depends_on:
     - databaseds
     - redis
     volumes:
     - tangogql-logs:/var/log/tangogql
     - ./ska-logging.yaml:/tangogql/ska-logging.yaml
     environment:
     - LOG_CFG=ska-logging.yaml
     - TANGO_HOST=${TANGO_HOST}
     - LOG_PATH=/var/log/tangogql
     # If this is not set, the output of python is delayed and only shows when the␣
↪docker container restarts
     - PYTHONUNBUFFERED=1
     labels:
     - "traefik.frontend.rule=Host:localhost; PathPrefix: /testdb/db, /testdb/socket,␣
↪/testdb/graphiql; ReplacePathRegex: ^/testdb/((?:db|socket|graphiql.*?)/?)/?$$ /$$1"
     - "traefik.port=5004"

 redis:
     image: redis
     container_name: redis
     network_mode: ${NETWORK_MODE}
```

By default two files are created, one called *info.log* and *error.log* if using ska-logging file this files will go to */var/log/tangogql*

CHAPTER 2

Prerequsities

To use this project, Docker >= v18 and GNU Make must be installed.

CHAPTER 3

Development of the OSO-UI applications

The development of the OSO-UI webjive application suite is a collaboration between software developers at the Max IV Laboratory in Lund and the SKA Buttons team.

Any development work on the webjive suite follows an agreed *Joint Process for Contribution between Max IV and SKA*