



**Management Of Networked IoT Wearables – Very Large Scale
Demonstration of Cultural Societal Applications**
(Grant Agreement No 732350)

D7.6 The MONICA Development Toolbox 2

Date: 2020-03-31

Version 1.0

Published by the MONICA Consortium

Dissemination Level: Public



Co-funded by the European Union's Horizon 2020 Framework Programme for Research and Innovation
under Grant Agreement No 732350

Document control page

Document file: D7.6 The MONICA Development Toolbox 2.docx
Document version: 1.0
Document owner: CNet

Work package: WP7 – Components & Cloud Integration
Task: T7.6 – The MONICA Development Toolbox
Deliverable type: [OTHER]

Document status: Approved by the document owner for internal review
 Approved for submission to the EC

Document history:

Version	Author(s)	Date	Summary of changes made
0.1	Peeter Kool, CNet	2019-10-19	Created ToC
0.5	Peeter Kool, CNet	2020-01-12	Added first sections
0.9	Peeter Kool CNet, all of the partners that contributed to the Toolbox	2020-03-30	Collected the parts of the GitHub site to build content in this paper document.
1.0	Peeter Kool CNet	2020-04-16	Final version submitted to the European Commission

Internal review history:

Reviewed by	Date	Summary of comments
Luca Mannella (LINKS)	2020-04-16	Accepted with minor changes
Shreekantha Devasya (FRAUNHOFER)	2020-04-15	Accepted with minor changes

Legal Notice

The information in this document is subject to change without notice.

The Members of the MONICA Consortium make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the MONICA Consortium shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Possible inaccuracies of information are under the responsibility of the project. This report reflects solely the views of its authors. The European Commission is not liable for any use that may be made of the information contained therein.

Index:

1	Executive Summary	4
2	Introduction	5
3	The Toolbox Website	6
4	Packages	8
4.1	Example Package (Sound Monitoring an event using Sound Level Meters)	8
5	Generic Enablers	14
5.1	Example GitHub Generic Enabler: MONICA COP.API	14
6	Tutorials	21
6.1	Example Tutorial COP UI Tutorial for sound monitoring	21
7	Conclusion	29
8	List of Figures	30
8.1	Figures	30
9	References	31

1 Executive Summary

This deliverable is based on work in task T7.6 The MONICA Development Toolbox. The goal of this task is to produce an open software development toolbox and generic enablers that allow developers to rapidly develop new applications to be deployed on the MONICA platform. The development platform consists of a toolbox and a set of tutorials.

The work has focused on making the MONICA Development Toolbox available in the long term after the project end and promotes reuse of the developed components and architecture. Therefore, the main part of the deliverable is the project page on GitHub <https://monica-project.github.io/> which provides an entry point for the Software Development Toolbox composed by the different tools and generic enablers together with their documentation. The Open Source code created by the project is available on <https://github.com/MONICA-Project> while readymade Docker Containers on <https://hub.docker.com/orgs/monicaproject/repositories> .

In order to promote more reuse of the complete MONICA software results, Packages are introduced. Packages, which are readymade docker environments that can easily start up a complete local private MONICA instance complete with instantiated data and simulated sensors. These can be used for demonstration purposes, but the most important part is that the developers can use them.

This deliverable describes what is available at <https://monica-project.github.io/> and provides some examples of the contents, but the full content is on the actual website.

2 Introduction

This deliverable is the result of the work in task T7.6 The MONICA Development Toolbox. The task produces an open software development toolbox and generic enablers that allow developers to rapidly create new applications to be deployed on the MONICA platform. The development platform consists of a toolbox and a set of tutorials.

The work has focused on making the MONICA Development Toolbox available and usable in the long term after the project end, promoting reuse of the components and architecture. Therefore, the main part of the deliverable is the projects page on GitHub <https://monica-project.github.io/> which provides an entry point for the Software Development Toolbox and the different tools and generic enablers.

The Open Source code created by the project is available on <https://github.com/MONICA-Project> and readymade Docker Containers on <https://hub.docker.com/orgs/monicaproject/repositories>.

The rationale for placing the MONICA development toolbox on GitHub and Docker Hub is that they will remain after the project end, as well as it is well known and used repositories for developers, where they look for Open Source components. Additionally, there is also infrastructure for issue tracking which makes it possible to maintain and update the code after the project end and to get the possibility of more developers to use and update the components.

A major effort has been put into creating Packages, which are readymade docker environments that can easily start up a local private MONICA instance complete with instantiated data and simulated sensors. These can be used for demonstration purposes, but the most important part is that the developers can have an easy first access to the platform. Configuring a MONICA instance is a very complex task, involving many individual components, but the packages can be used as templates. One of the greatest obstacles to reuse complex systems as MONICA is configuring the environment, we hope that the packages will promote reuse of more of the MONICA system.

The MONICA Toolbox is divided into four different categories:

- *Packages*. These are packaged tools intended to be used by developers. They have been developed by the MONICA project. The category is described in section 4
- *Generic Enablers*. These are re-usable software components developed by the MONICA project. MONICA generic enablers are made available in the <https://github.com/MONICA-Project> repository and can be used by entrepreneurs, start-up and established companies alike. The category is described in section 5.
- *Third Party Services and Tools*. These are some openly available third-party tools that are recommended by the MONICA project to use when building Large Scale IoT applications. MONICA Tools and Generic Enablers have available interfaces for these third-party tools and services.
- *Tutorials*. Which describe how to use a component or extend a component and have been created by the MONICA project. The category is described in section 6.

This deliverable describes what is available at <https://monica-project.github.io/> and provides some examples of the underlying contents, but the full content is on the actual website.

3 The Toolbox Website

The <https://monica-project.github.io/> is the main entry point for the MONICA Development Toolbox and provides pointers to different repositories and tutorials that make up the Toolbox.

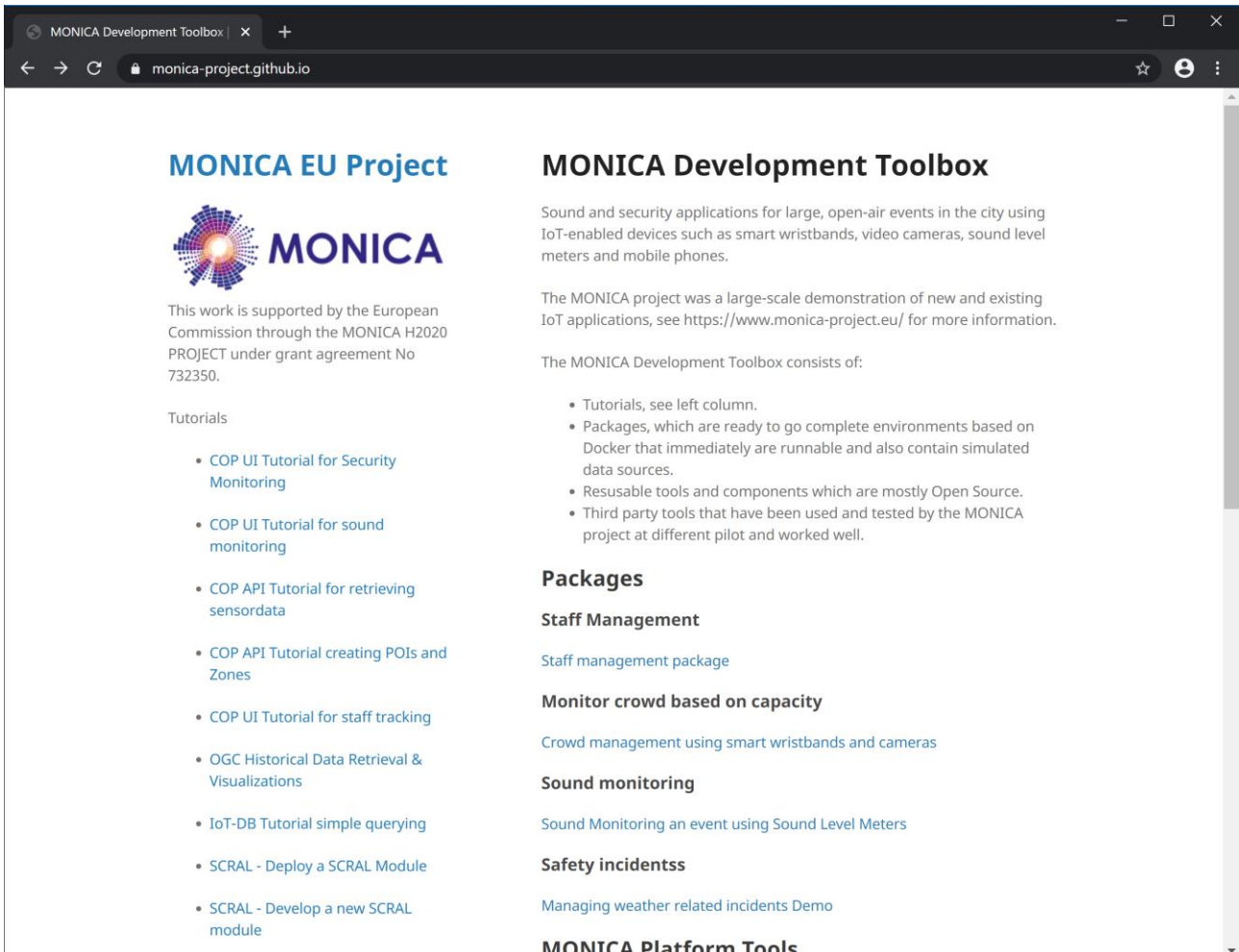


Figure 1: The <https://monica-project.github.io/> page

The page gives a short overview of the available tools, packages, components, and tutorials that are available in the toolbox. The Packages are the ready to go Docker Compose based environments that are possible to run locally providing an instance of the MONICA platform. The packages are grouped for the intended use case they provide:

- Staff Management
- Monitor Crowd based on capacity
- Sound monitoring
- Safety incidents.

The intention is that the visitor should understand and to be able to pick the right package for testing.

The MONICA Platform tools is also grouped but along:

- Generic Components: Components are generally usable in all applications. For instance, the COP.
- Sound: Components that are specific sound management, i.e. the Sound Heatmap Generator
- Cameras: Components that relate to image processing, i.e. VCA Core and the Security Fusion Node (SFN).

- Wearables: Components that relate to wearable technology, i.e. Smart glasses.
- Simulators: Components that relate to simulate different devices, i.e. Smart Wristband Simulator.
- Tools: Tools that are developed within MONICA but not part of the platform, i.e. SignalR Listener that can be used to intercept and debug SignalR communication.

The last section is the third-party tools that we have used in the project and that have worked well in the pilots, i.e. they have been proven in real life.

4 Packages

Packages are the readymade docker compose packages that can be used for starting an instance of the MONICA platform directly on your own computer. All that is necessary is to have Docker and Docker-Compose installed on the machine. The main purpose of the packages is to make it simple to start/develop/test in the MONICA platform. Since the MONICA platform is complex and built with many different components it is very hard to start from scratch to build an instance of it by oneself. As an example, the “Sound Monitoring an event using Sound Level Meters” package involve 15 components that need to be started in the right order and with the right configuration settings, this would be almost impossible for a non-MONICA developer to create, having a working configuration to start from makes the task much easier. It enables the possibility to copy and change the configuration of the docker compose yml files, thus reusing the existing pre-prepared configuration of an instance and extending or replacing some components.

There are four packages in the MONICA Development Toolbox:

- Staff management package: That contains staff tracking built from *Pütchzens Markt* pilot.
- Crowd management using *smart wristbands* and *cameras*: That contains crowd heatmaps from wristbands as well as people density using cameras. This is built using a combination of *Woodstower* and *Leeds* pilot.
- Sound Monitoring an event using *Sound Level Meters*: Contains Sound Level Meter, sound heat maps, sound contribution, sound incidents. This is built using the *Woodstower* pilot.
- Managing weather related incidents: Contains aggregates, environmental sensors, i.e. wind and temperature, and incidents with an intervention plan. This is based on the *Hamburg DOM* pilot,

Of course, that data from the sensors in the packages are either samples from some event or purely simulated depending on the package. The simulated ones are the datastreams that involve individual movement of people, i.e. the trackers and the wristbands whilst the sound datastreams are captured and replayed from real events.

4.1 Example Package (Sound Monitoring an event using Sound Level Meters)

This example comes from <https://github.com/MONICA-Project/DockerSoundDemo> on the MONICA GitHub

The package is a normal GitHub repository with the files needed for executing it and is completely self-contained, see Figure 2.

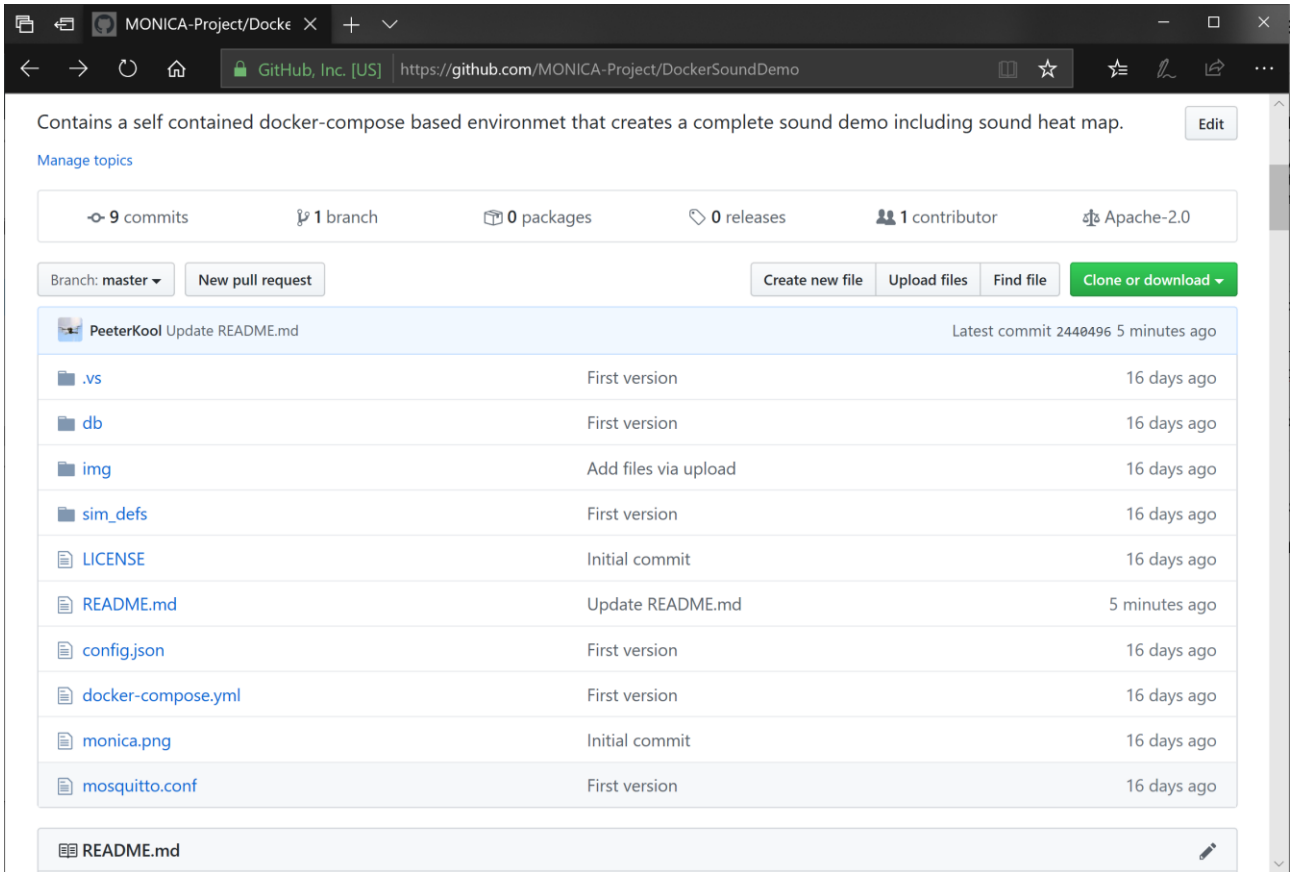


Figure 2: The GitHub repository for the package

The repository only holds the Docker Compose file and the necessary settings needed to run the package. All software is pulled automatically from Docker Hub when docker compose is invoked. The README.md is automatically shown in the repository.

The contents of the readme file first explain what the package contains, see Figure 3.

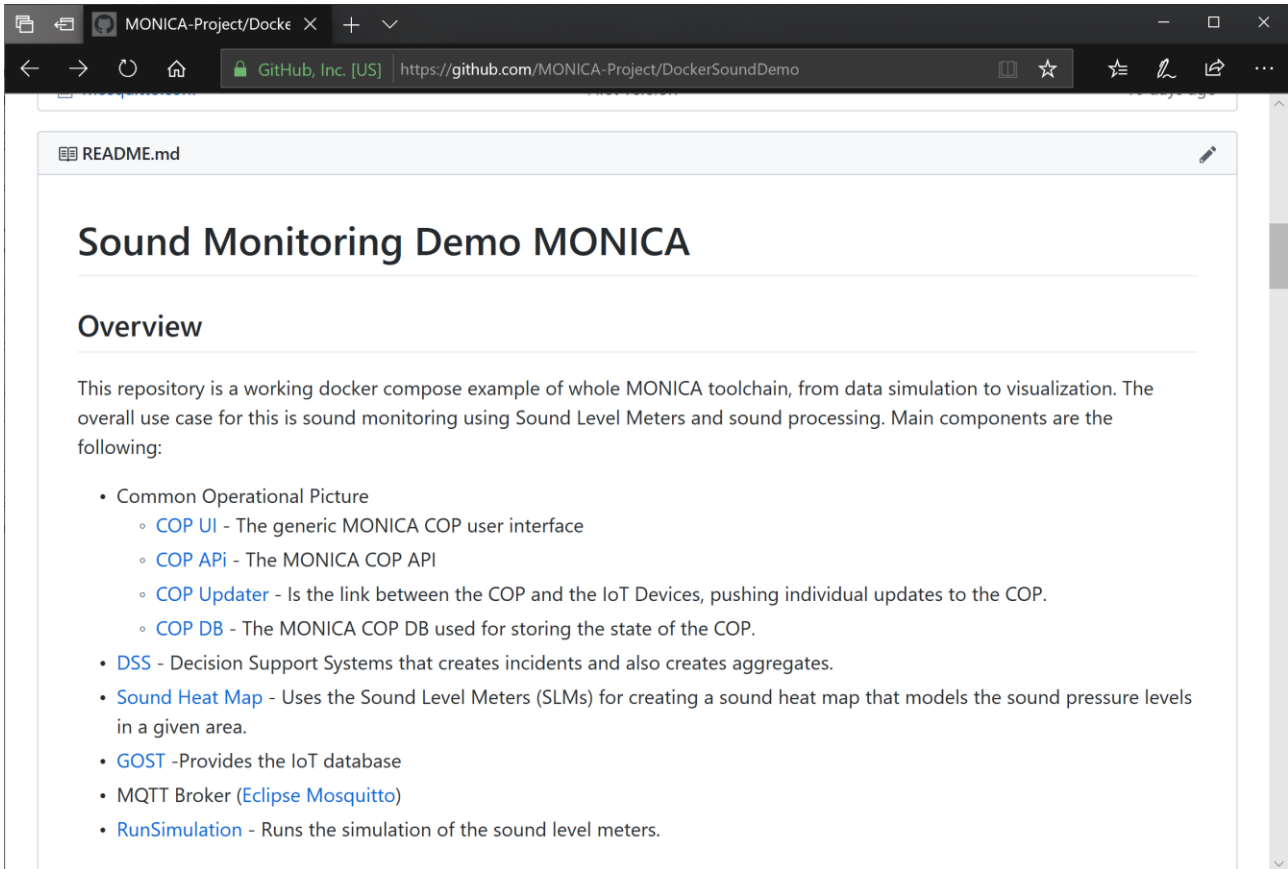


Figure 3: Readme

It details which components are involved in the package with a short description and links to their GitHub repositories, so it is possible to look at the individual components.

The next part of the readme describes how to run the package, see Figure 4

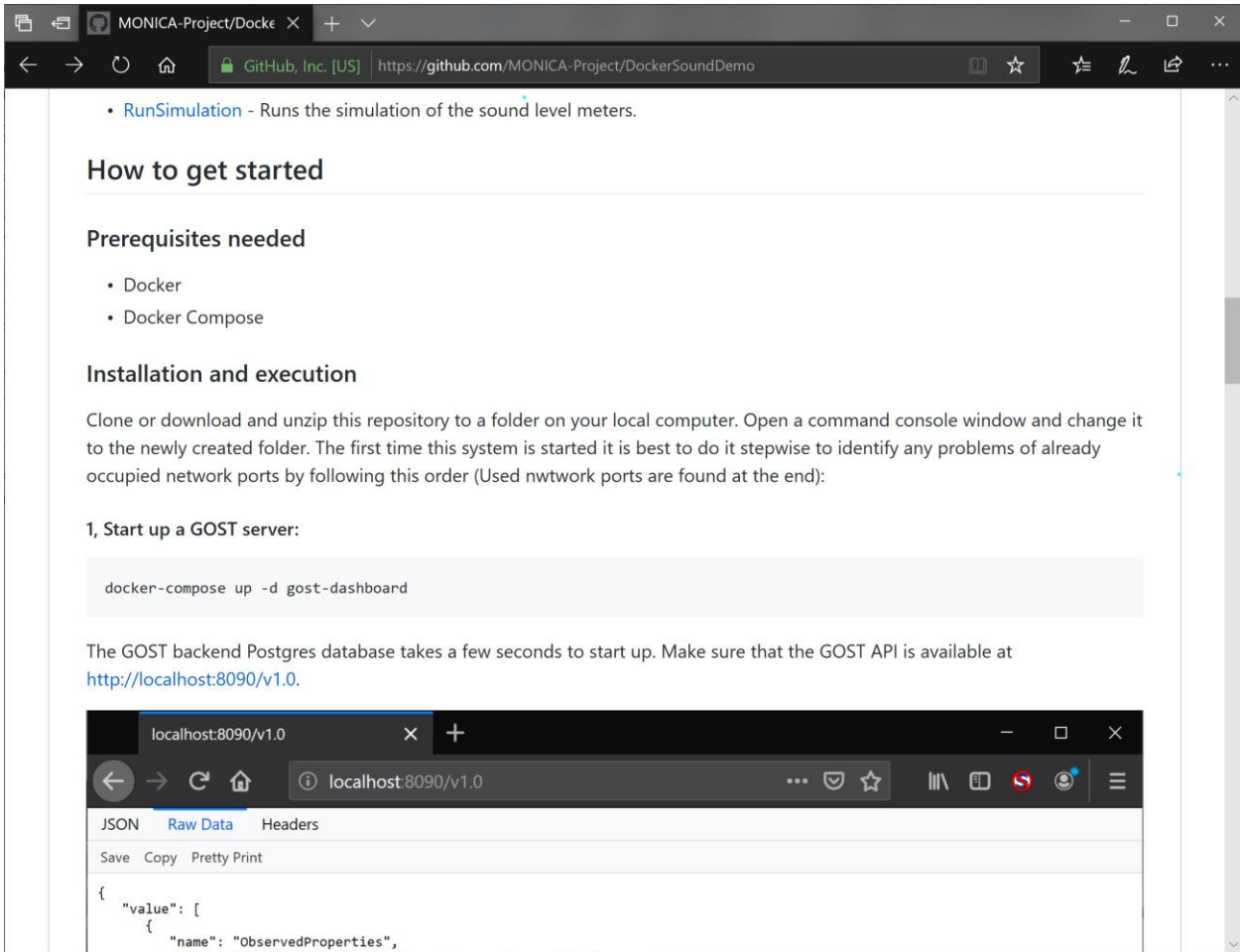


Figure 4: Package How to get started

This includes the necessary prerequisites, which like for all packages are Docker and Docker Compose. The next step is either clone or download the repository to his local disk. This is followed by step by step instructions of starting up the MONICA environment. For each of the steps there are control points to check that everything started correctly, in this case the IoT-DB (GOST) is started and verified by using the browser to see that the service is alive.

There are a number of steps that are skipped here for brevity, but they can be viewed in full at <https://github.com/MONICA-Project/DockerSoundDemo>.

Finally, the last step where the whole MONICA system is started brings up a live COP with values changing, which is basically the same COP that was used for sound monitoring at the Woodstower pilot, see Figure 5.

4, Start the COP UI

Start the service for the COP UI:

```
docker-compose up -d copui
```

Check if COP UI is up and running at <http://localhost:8900/>

NB! the userid is admin@monica-cop.com and the password SOUND2020!

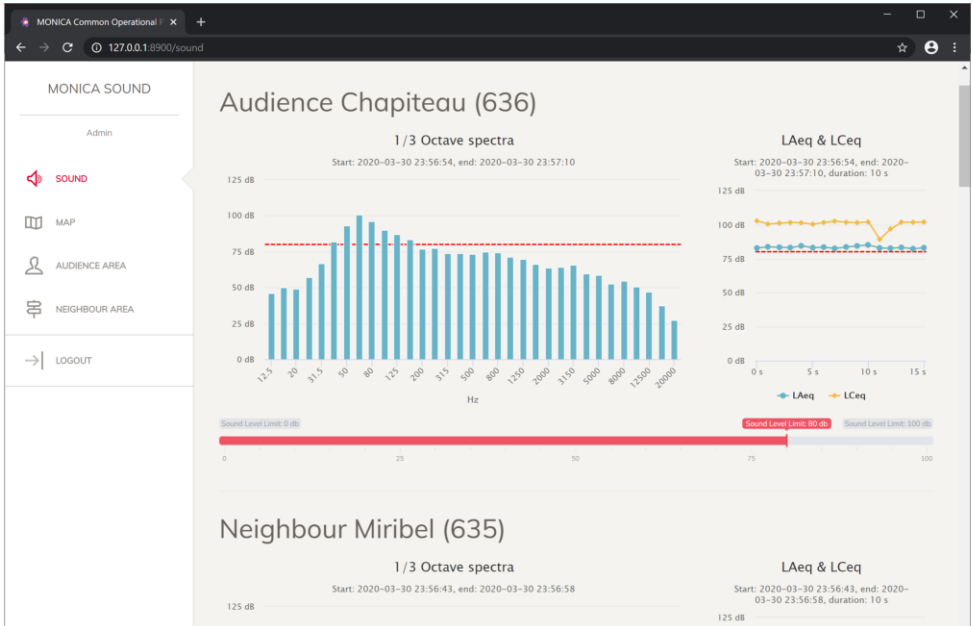
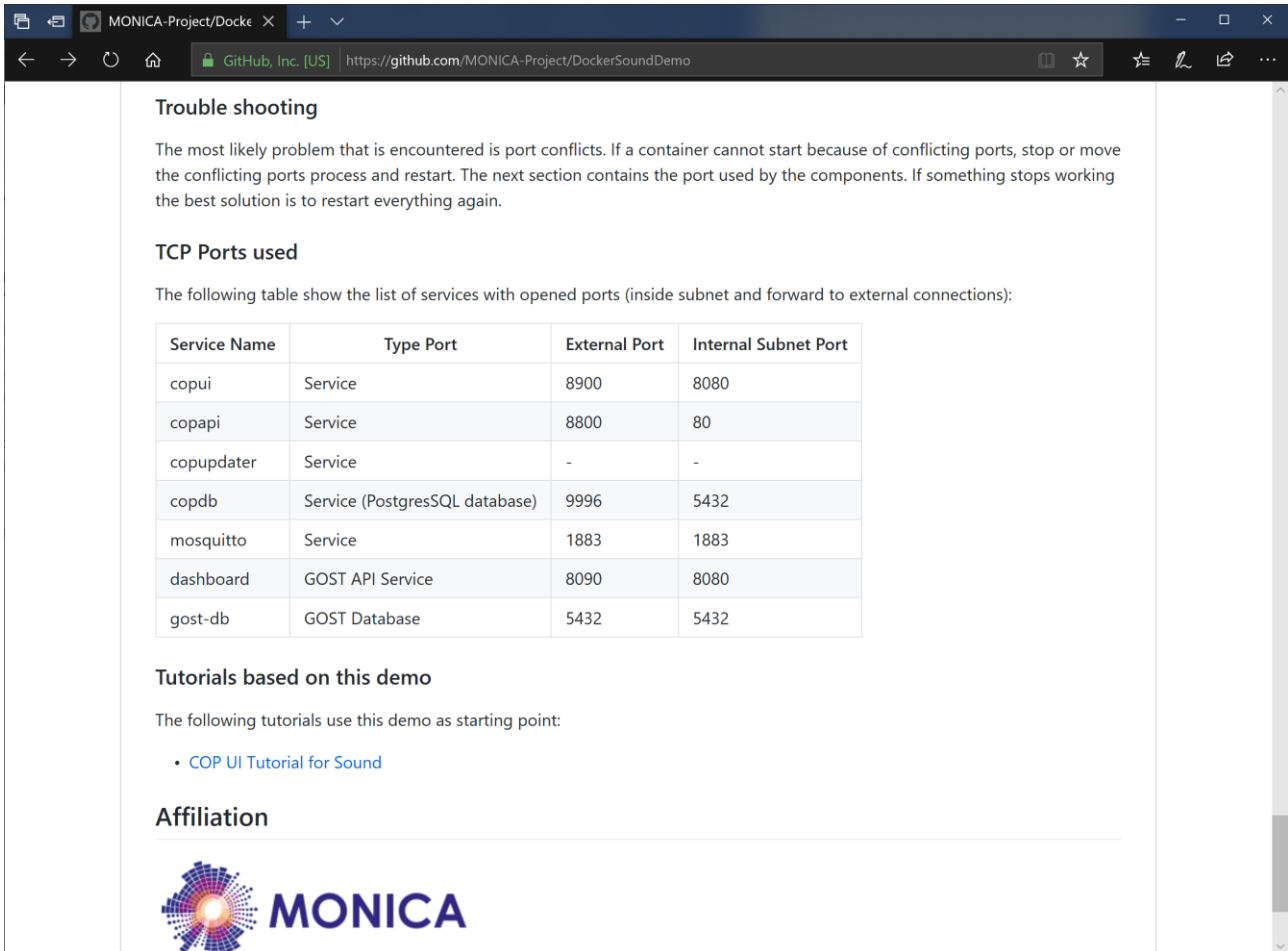


Figure 5: Package COP UI

Now the MONICA system is running on the local machine with captured replayed data. It is possible to interact with all components, querying the IoT-DB, listen and create MQTT messages, use the COP.API etc.

The last part of the package readme contains additional information, see Figure 6.



Trouble shooting

The most likely problem that is encountered is port conflicts. If a container cannot start because of conflicting ports, stop or move the conflicting ports process and restart. The next section contains the port used by the components. If something stops working the best solution is to restart everything again.

TCP Ports used

The following table show the list of services with opened ports (inside subnet and forward to external connections):

Service Name	Type Port	External Port	Internal Subnet Port
copui	Service	8900	8080
copapi	Service	8800	80
copupdater	Service	-	-
copdb	Service (PostgreSQL database)	9996	5432
mosquitto	Service	1883	1883
dashboard	GOST API Service	8090	8080
gost-db	GOST Database	5432	5432

Tutorials based on this demo

The following tutorials use this demo as starting point:

- [COP UI Tutorial for Sound](#)

Affiliation




Figure 6: Troubleshooting etc

Trouble shooting describes the most common problem when running the package, the most common problem is ports conflict. Therefore, there is a list of ports used by the package. An additional benefit is that it shows which port to use when using the components in the package. In this case we can see that COP API is available on port 8800 and the COP UI is available on port 8900.

Finally, at the end there are links to any tutorials which are based on the package.

5 Generic Enablers

The generic enablers are renamed to MONICA Platform tools on the website because we want to be closer to the normal developer language. Also, because it will be easier to find them using different web search engines.

The Generic Enablers are grouped in the following categories:

- *Generic Components*: Components are generally usable in all applications. For instance, the COP, the SCRAL, etc.
- *Sound*: Components that are specific sound management, i.e.: The Sound Heatmap Generator
- *Cameras*: Components that relate to image processing, i.e.: VCA Core and the security fusion node (SFN).
- *Wearables*: Components that relate to wearable technology, i.e.: Smart glasses.
- *Simulators*: components that relate to simulate different devices, i.e.: Smart Wristband Simulator.
- *Tools*: Tools that are developed within MONICA but not part of the platform, i.e.: SignalR Listener that can be used to intercept and debug SignalR communication.

Each of the Generic Enablers points to a repository in the MONICA GitHub repository. Most of them contain the source code and a description of the component together with instructions related to how to build and configure the component. Though some components are not Open Source, for instance the DSS, there is a description as well as a readymade Docker Container on the MONICA DockerHub repository that can be used for testing and reusing.

The aim for the Generic Enablers GitHub repositories is that a developer can get the source code, understand how it is configured, deployed, and what is necessary to further develop it. GitHub repositories also provide a possibility to create a Wiki if longer descriptions are needed. The issue tracker within GitHub gives the possibility to report bugs as well as reporting any other issue regarding repository content which puts the developer in contact with the providers of the repository.

The next sub section shows one example repository of a generic enabler component.

5.1 Example GitHub Generic Enabler: MONICA COP.API

This example comes from <https://github.com/MONICA-Project/COP.API> on the MONICA GitHub. The repository contains all the code and files needed to build the component, see Figure 7.

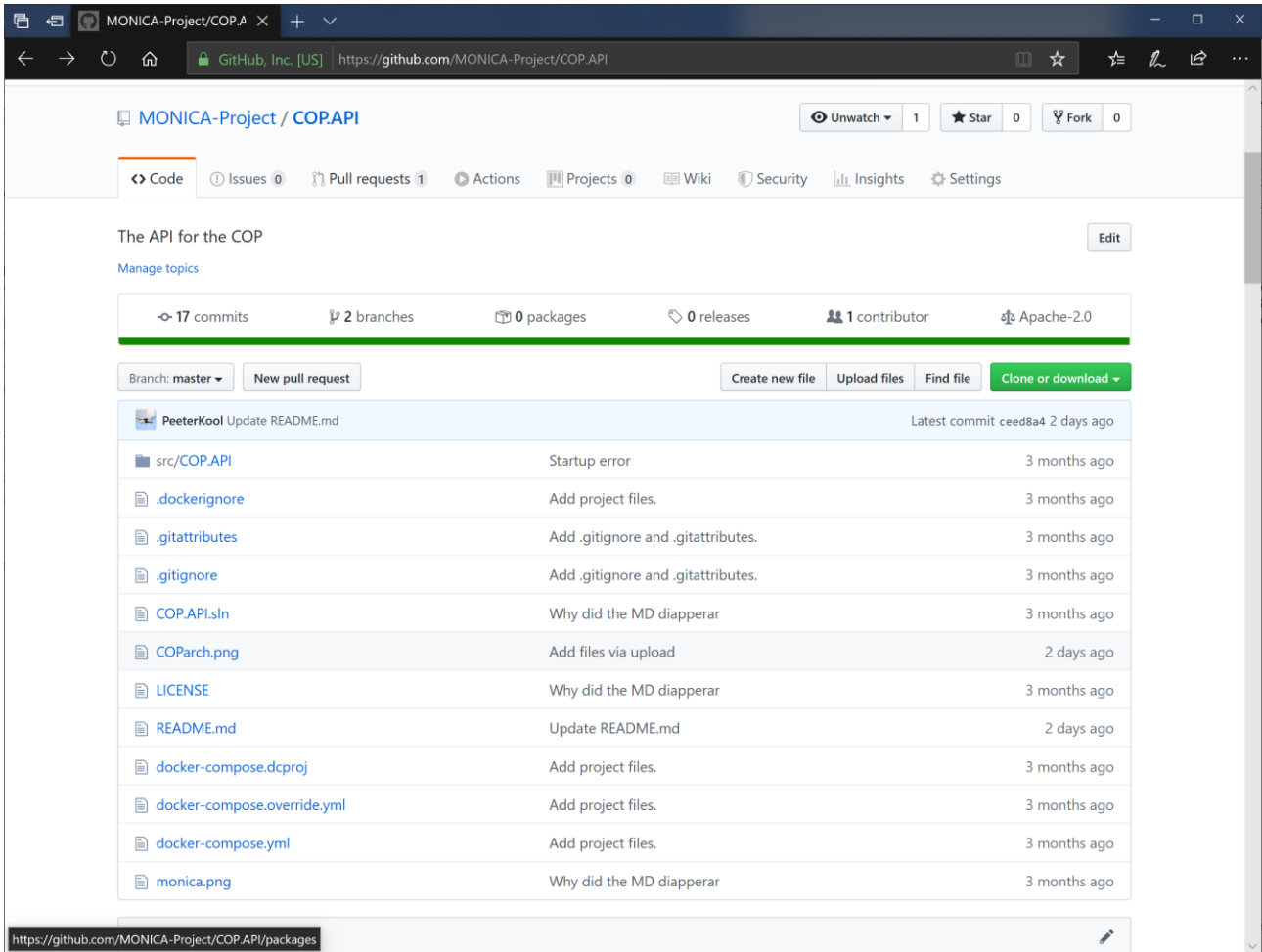
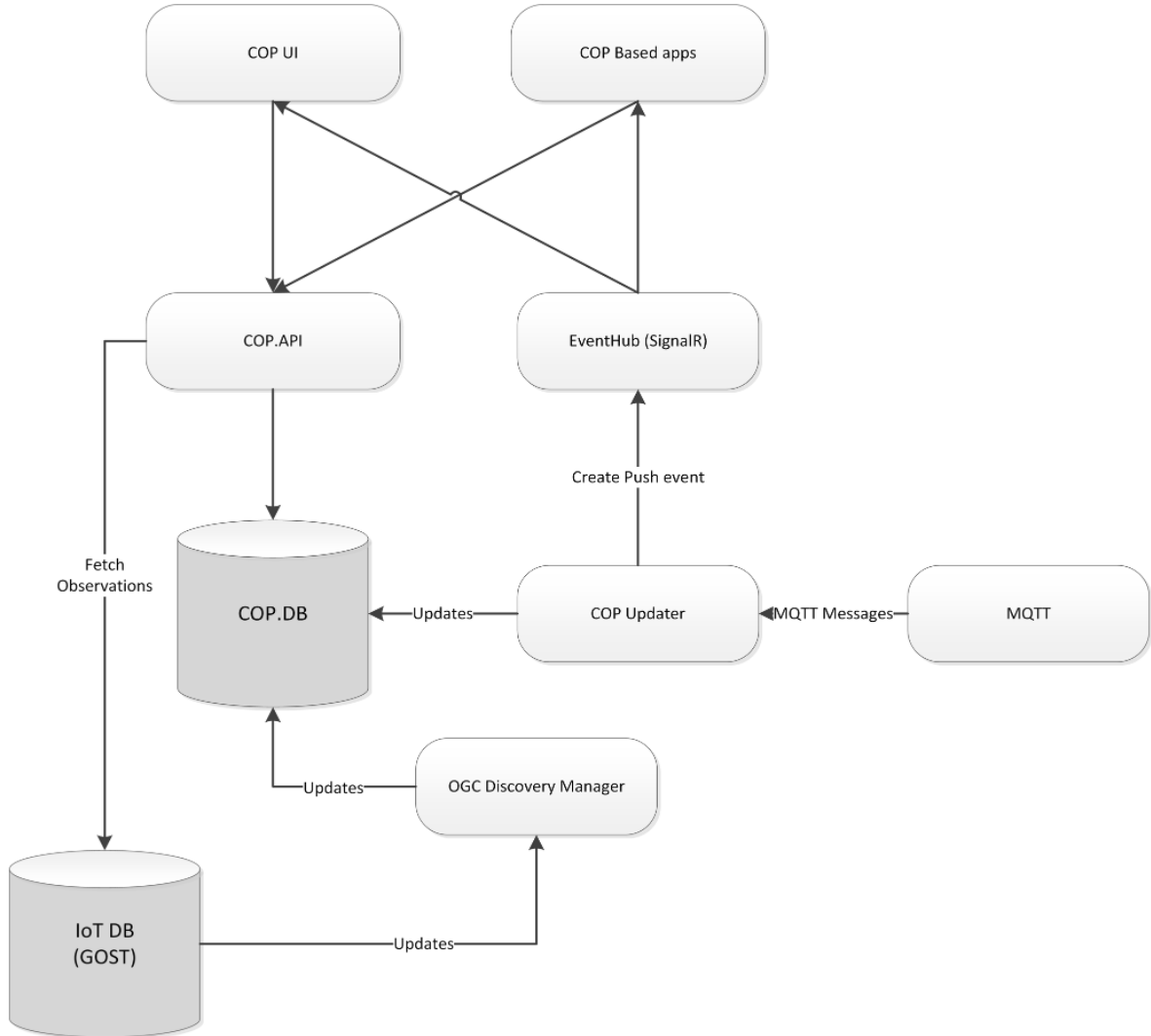


Figure 7: Generic Enabler COP.API

Included in the repository is a `README.md` file which is displayed below the file contents when entering the repository, this is used instructions on how to use and build the component. The `COP.API` is a more complex component that is part of a context of components therefore the readme is including information about how the component relates to others, see Figure 8.

Monica COP.API

COP.API The COP API is intended for components delivering data and services to the staff at the event. The main users of the API are the Common Operational Picture UI and the different apps that are part of MONICA.



The COP.API is connected to the following components in MONICA:

- IoT DB *GOST* Provides the OGC Sensorthings API database
- *COP Updater* - Is the link between the COP and the IoT Devices, pushing individual updates to the COP.
- *OGC Discovery Manager* - Is the link between the COP and the IoT DB, forwarding newly discovered devices in the IoT DB to the COP.DB.
- EventHub provides push functionality to apps and COP-UI. This component is in fact integrated in the COP.API module but as a separate component.

Figure 8: COP.API Description

It also lists those components with connected to the COP.API links to their respective repositories. This is to give the developer an opportunity to understand what the component performs. The next part of the readme file describes what technologies are used and what functionalities the component provides, see Figure 9.

The COP API is based on the following technologies:

- MQTT interface for receiving messages for updating COP status
- Odata-based API for retrieving resources from the IoT DB
- Integrates SQL database with an OGC Sensorthings database for storage and management of time series of observations.

The COP API provides the following main functionalities:

- Incident classification and management
- Division of a geographical area into zones and subzones
- Mapping of incidents, sensors, facilities and people/groups/crowds to zones
- Fast retrieval of current status of the situational objects of interest

The COP API is implemented in ASP.NET Core 2.1 using Swashbuckle so it creates its own swagger definitions, that can be viewed and used for testing.

Figure 9: COP.API Description (cont.)

The next section is the “Getting Started”, see Figure 10 below.

Getting Started

The COP.API is developed in Visual Studio using Dotnet Core 2.1.

The easiest way to build it is to clone the repository using Visual Studio 2017 or higher and then build the software or to use the DotNet Core 2.1 SDK.

There are a number of ready made Docker Compose Packages demonstration environments that include the COP.API and all dependencies and provides an easy way of testing the COP.API. In addition there are a number of tutorials for the API.

Docker Compose Packages with complete demonstration environments including the COP.API

- [Staff management package](#)
- [Crowd management using smart wristbands and cameras](#)
- [Sound Monitoring an event using Sound Level Meters\)](#)
- [Managing weather related incidents Demo](#)

Tutorials

- [COP API Tutorial creating POIs and Zones](#)
- [COP API Tutorial for retrieving sensordata](#)

Figure 10: COP.API Getting started

This section contains information about how to get started in the easiest way with COP.API, in this case it is recommended, since it is a DotNet Core application, Visual studio 2017. Besides this information it also includes which Packages include the COP.API, this allows for the developer to select one of them and install in Docker to be used for running the components the COP.API depends on. As a further bonus there will be example data as well as dynamic data running from the package, which eases that start of development and test. It also lists any tutorial which the component plays a major part. In this case there are two:

- COP API Tutorial creating Points Of Interests (POIs) and Zones: Which details how POIs and Zones can be defined using the COP.API
- COP API Tutorial for retrieving sensor data: Which details retrieving sensor data using the COP.API.

The purpose of showing the tutorials is that the tutorials give quick glance of how the component works, in this case the COP.API.

Deployment

For deployment the COP.API relies on a Postgres database for internal use as well as a connection to a GOST database. If one uses one of the demonstration docker compose environments they will automatically be created.

Docker

Environment Variables that need to be set

Variable	Description	Example
MEDIA_PATH	Shared container path used only together with smart glasses.	/ora_shared/
CONNECTION_STR	COP Database connection string	Host=copdb; Database=monica_wt2019; Username=postgres; Password=postgres;Port=5432
GOST_PREFIX	GOST	Should match the MQTT prefix use by the GOST db
TEST_TOKEN	6ffdcacb-c485-499c-bce9-23f76d06aa36	The fixed access token
USE_GOSTOBS	True if the API will retrieve Observations directly from GOST	true
GOST_SERVER	GOST Server address	http://gost:8080/v1.0/
URL_PREFIX	If the API is not deployed directly under toplevel this needs to be set	TIVOLI for deployment on http://../TIVOLI
USE_GOSTSEARCHOBS	False uses legacy model of finding datastreams	true

To run the latest version of COP.API:

```
docker run -p 8800:80 -e MEDIA_PATH=/ora_shared/ -e CONNECTION_STR=Host=copdb;Database=monica_wt2019;Username=postgres;P
```

Figure 11: COP.API Deployment

Figure 11, shows the deployment part of the readme file. This part describes what variables are necessary to deploy the COP.API component. Since the COP.API is normally only used as a Docker container the environment variables are very important. The environment variables are the means to set up the container and the component at start of the container. In effect they act as parameters when starting container.

The environmental variable lists the name of the variable, a short description, and an example typical value. For the COP.API component all environment variables are compulsory and need to be defined when creating the container in order for the component to work as intended.

To give further guidance, a pre-prepared Docker run command including the setting of the environment variables is included. If the component is part of a Package, the Docker Compose file also includes which variables to set.

Development

To start development it is enough to clone the repository and then build it either using Visual Studio or Dotnet Core SDK to build and run the API. It is recommended to use one of the Docker Compose Packages with complete demonstration environments mentioned above to have some testing data and to have the complete COP environment available.

The code for the COP.API is based on ASP.NET Core framework and for COP.DB access Microsoft.EntityFrameworkCore is used.

Prerequisite

Either one of the Docker Compose Packages with complete demonstration environments, or manually install the following components:

- GOST (IoT DB). Installation instructions are available [here](#)
- PostgreSQL. Installation instructions are available [here](#)
 - COP DB needs to be loaded in the database, instructions [here](#)
- Dotnet Core SDK 2.1 available [here](#)
 - Or use Visual Studio 2017 or higher

Build

```
dotnet build
```

Endpoints exposed

The default port that is exposed is port 80

- API with swagger is reached on <http://ip/>
- SignalR is reached on <http://ip/signalR/COPUpdate>

Figure 12: COP.API Development

The final section covers if there are any special configurations, settings etc. that are necessary to build the generic enabler component. In this case the recommendation is to use one of the Docker Compose packages to automatically create the rest of the needed environment. It is not unusual that generic enabler components need to have libraries installed before being able to build it. In this case the COP.API and DotNet Core SDK/Visual Studio will automatically install any needed libraries, so the prerequisites only contain external components that are needed to execute and develop. There is also a ready prepared statement that shows how to start the build, which is trivial in this case but in many others can be quite complicated. The section ends with a description of which endpoints are exposed by the service, if any.

Contributing

Contributions are welcome.

Please fork, make your changes, and submit a pull request. For major changes, please open an issue first and discuss it with the other authors.

Affiliation



This work is supported by the European Commission through the [MONICA H2020 PROJECT](#) under grant agreement No 732350.

Figure 13: Contributing/Affiliation section

Figure 13 shows the last part of the readme file. It contains two parts:

- **Contributing:** Stating that we are welcoming contributions to code/component and detailing how we would like it to be done. We hope that we will get some activity here.
- **Affiliation:** Showing that the component has been developed within MONICA project.

6 Tutorials

Tutorials provide guidance and examples of how to use and extend different tools, and how to use external tools in the MONICA platform. The next sub sections will walk through one of the tutorials that describe how to use the COP.UI for the “Sound Monitoring an event using Sound Level Meters” package. The intended use of the tutorial is for developers wanting to extend the COP.UI or other parts of MONICA, and for others testing the package to understand what they see and what they can do in the COP.UI.

6.1 Example Tutorial COP UI Tutorial for sound monitoring

It is implicit that the pictures reported in this tutorial could be different from the figures hosted on monica-project website. The full content of the tutorial can be found here:

<https://monica-project.github.io/sections/cop-api-tutorial%20for%20sound.html>

The tutorial starts with an introduction and a table of contents, see Figure 14.

COP UI Tutorial for sound monitoring

This is a tutorial for how to use the COP (Common Operational Picture) UI and is based on using the [Demonstration of Sound Monitoring an event using Sound Level Meters](#) simulated demonstration.

Contents

- [Prerequisites](#)
- [Introduction](#)
- [Basic Components of the UI](#)
- [The SOUND view functionality](#)
- [The MAP view functionality](#)
 - [Zoom and movement of the MAP](#)
 - [Sound heatmap](#)
- [The AUDIENCE AREA functionality](#)
- [The NEIGHBOUR AREA functionality](#)

Prerequisites

[Demonstration of Sound Monitoring an event using Sound Level Meters](#) needs to be launched in docker, follow the instructions there.

Introduction

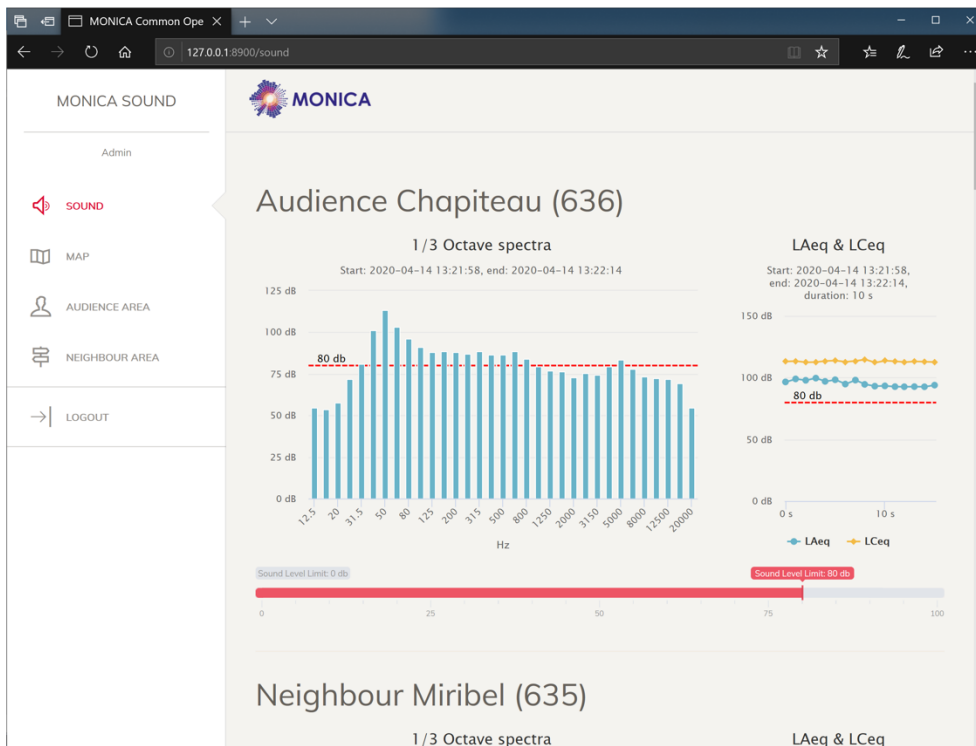
The COP UI is a simple map based interface for displaying the current state of an event. The COP UI is built using predefined Points Of Interest POI, called instance data, and mixing it with sensor and service derived dynamic information. All the functionality exposed in the COP UI is using the COP API.

Figure 14: Tutorial introduction

It is made clear that this tutorial has the package for “Sound Monitoring an event using Sound Level Meters” as a prerequisite in order to be followed.

Basic Components of the UI

The user interface is built with a main menu that selects which view the user shown.



In this case we have five modes:

- SOUND, which is the active view in this picture and shows the values from the sound level meters
- MAP, shows the MAP view including the calculated sound heatmap.
- AUDIENCE AREA, that has audience area information
- NEIGHBOUR AREA, shows the neighbour area information
- LOGOUT, that will logout the user

The number views depends on the application and can include views for sound and incident management.

Figure 15: Tutorial Basic Components of the UI.

The first section describes how to navigate in-between the different modes using the menu on the left side. It also explains shortly the content of each mode.

The SOUND view functionality

The SOUND view shows the “real time” data from the Sound Level Meters at the event. For each of the Sound Level Meters the following are shown:

- 1/3 octave spectra showing the sound pressure levels at different frequencies
- LAeq (A-weighted equivalent in decibels)
- LCeq (C-weighted equivalent in decibels)
- A limit bar that controls the redline in the graphs.

Figure 16: Tutorial Sound view

Figure 16 describes and explains what is shown in the graphs of Figure 15. Detailing some of the displayed abbreviations.

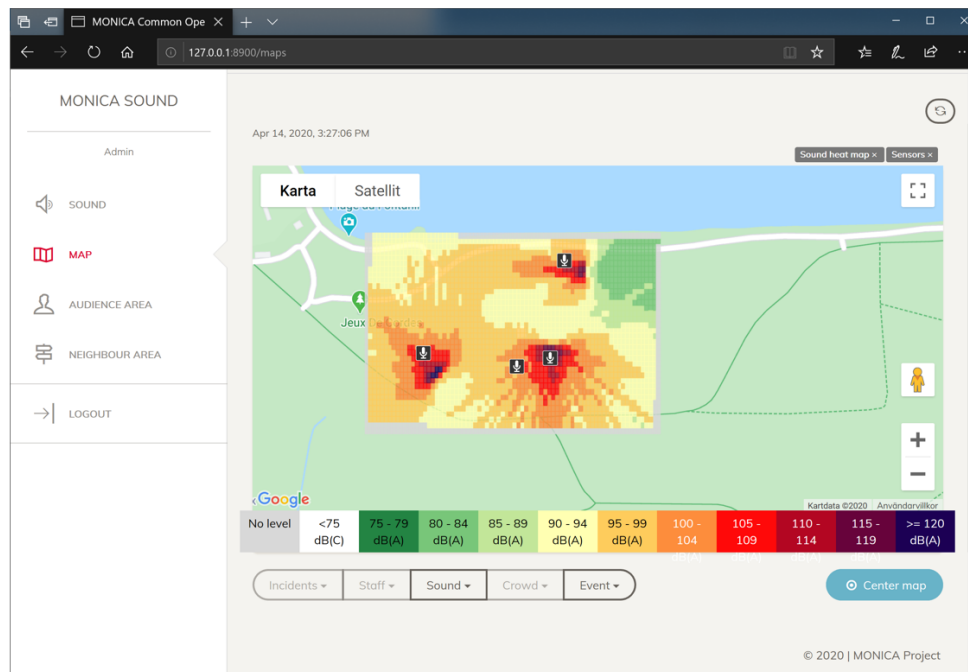
The MAP view functionality

Zoom and movement of the MAP

In the MAP view the user can zoom in and out of the map and also move the MAP. All this is achieved by using the normal Google Maps interface, i.e dragging the map with the mouse or fingers, zoom by pinching or by using the “+” and “-” controls in the map.

Sound heatmap

The map view for sound shows the calculated sound heat map. The [Sound Heat Map Generator](#) generates the sound heat map using the sound level meters.

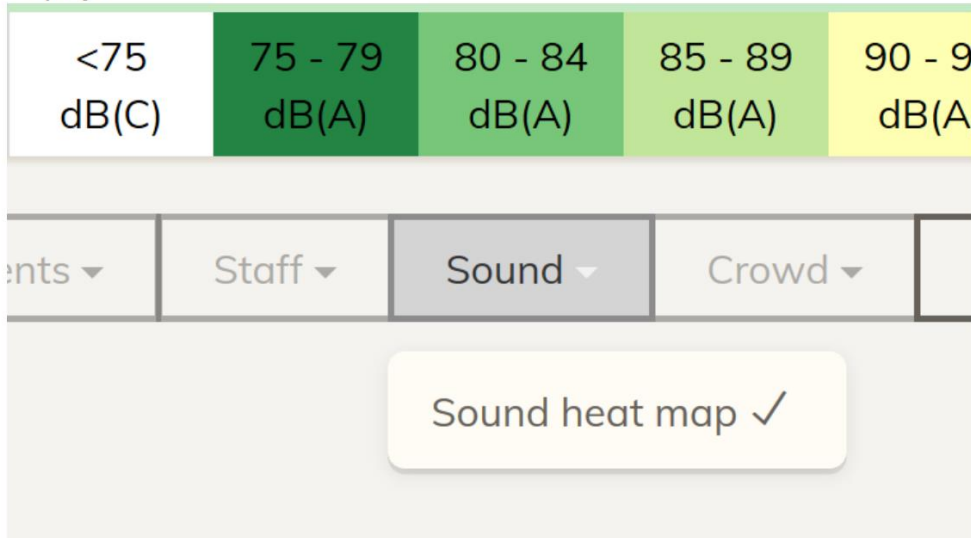


As one can see there are three active stages at this event.

Figure 17: Tutorial Map View

The Map View Functionality, Figure 17, describes how to use the map and be able to zoom and move it. Nextit describes the sound heatmap providing a link to generic enabler for the Sound Heat Map generator.

If the heat map is not shown, check the filter setting so that is selected for display:



In the map the positions of the SLMs are also indicated. By clicking on the symbol a detail view will be shown with the name and id. There are additional SLMs placed in the neighbouring areas, by zooming out the map these can be seen.

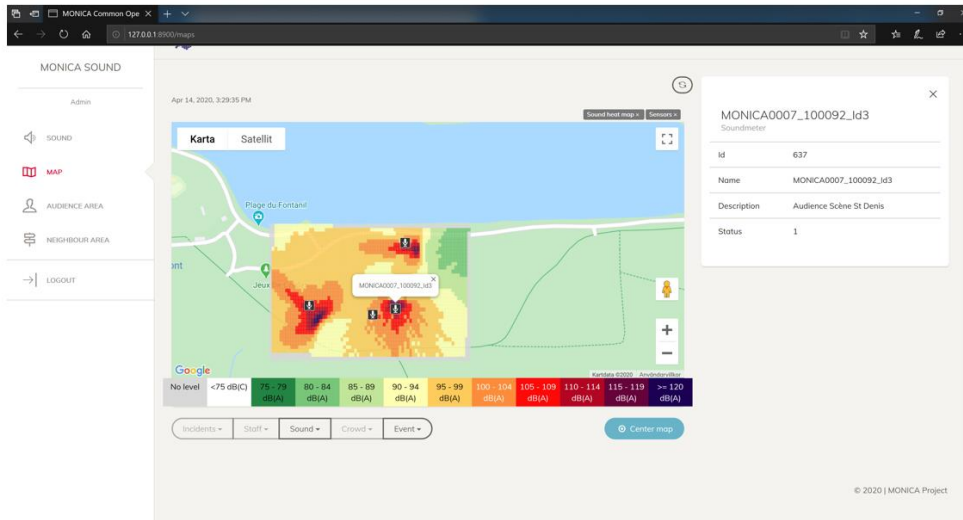
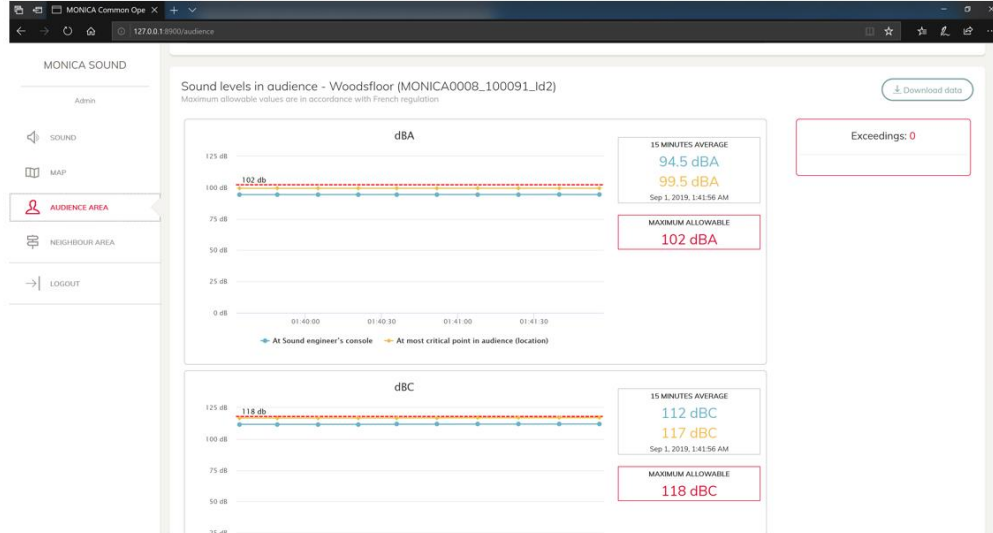


Figure 18: Tutorial Map View (cont.)

The next section in the map viewed in Figure 18 shows how to use the map filtering function, in normal cases the sound heat map should always be shown. The last part of map view shows how to see details and indicates also that there are more Sound Level Meters in the neighbouring area.

The AUDIENCE AREA functionality

The AUDIENCE AREA displays information about sound pressure levels in the audience area:



The sound pressure values for dBA and dBC are processed by the Decision Support System (DSS) producing a sliding average over time. Any exceedings of limits will generate incidents from the DSS which are shown in the Exceedings “box”.

Figure 19: Tutorial Audience Area

The audience area view, Figure 19, explains the information shown in the view but also provides a link to the Decision Support System (DSS) generic enabler. As can be seen in the picture no exceeding’s have occurred yet.

The NEIGHBOUR AREA functionality

The NEIGHBOUR AREA displays information about sound pressure levels in the neighbouring areas.

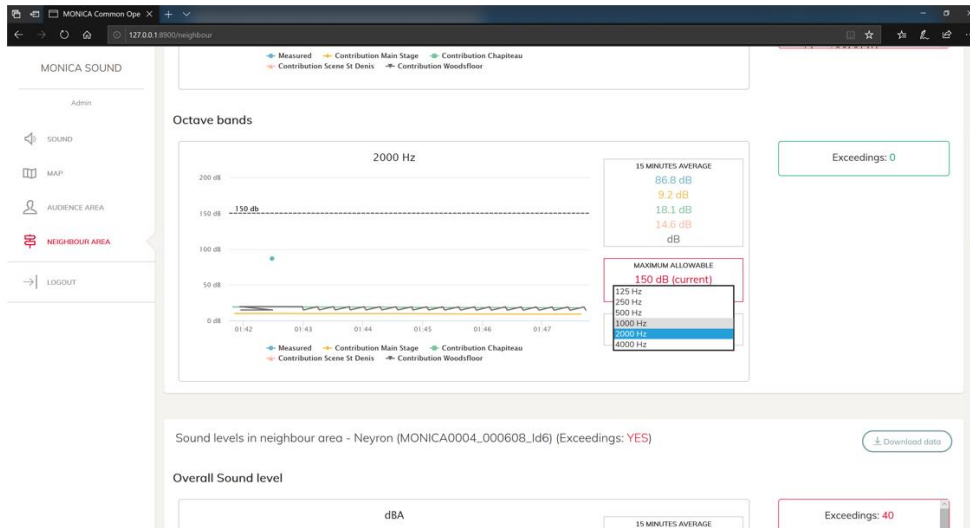


It shows the sound pressure level and a 15 minute average. In addition to this it also displays the calculated contribution of sound from the different stages. The calculations are done by the (DSS). In this case the limits have been exceeded a number of times and they are shown in the exceedings box.

Figure 20: Tutorial Neighbour Area View

Figure 20 shows the first part of the neighbour area view. It describes the graph with calculated contribution from each stage and points out the exceedings of allowed levels.

In addition to the dBA there is also a view of the calculated octave bands.



This also shows the calculated sound contribution from each of stages. You change the selected band by using the drop down list.

Figure 21: Tutorial Neighbour Area View (cont.)

Finally, Figure 21, shows how to interact with the calculated octave bands and how to change the selected band.

7 Conclusion

This is the final paper version deliverable “The MONICA Development Toolbox” but we hope that the MONICA toolbox will live on and evolve in the GitHub full version at <https://monica-project.github.io/>. We also hope that the hard work with the premade packages will be fruitful, from one side in marketing the MONICA results and on the other side for showing functionality live (that always gives a better impression related to showing just documentation). We hope also that developers will find the Toolbox useful when extending or changing the MONICA platform.

8 List of Figures

8.1 Figures

Figure 1: The https://monica-project.github.io/ page	6
Figure 2: The GitHub repository for the package	9
Figure 3: Readme	10
Figure 4: Package How to get started	11
Figure 5: Package COP UI	12
Figure 6: Troubleshooting etc	13
Figure 7: Generic Enabler COP.API	15
Figure 8: COP.API Description	16
Figure 9: COP.API Description (cont.)	17
Figure 10: COP.API Getting started	17
Figure 11: COP.API Deployment	18
Figure 12: COP.API Development	19
Figure 13: Contributing/Affiliation section	20
Figure 14: Tutorial introduction	21
Figure 15: Tutorial Basic Components of the UI.	22
Figure 16: Tutorial Sound view	23
Figure 17: Tutorial Map View	24
Figure 18: Tutorial Map View (cont.)	25
Figure 19: Tutorial Audience Area	26
Figure 20: Tutorial Neighbour Area View	27
Figure 21: Tutorial Neighbour Area View (cont.)	28

9 References