Obsmon Documentation

Paulo Medeiros[*] and Klaus Zimmermann

Swedish Meteorological and Hydrological Institute (SMHI)
Folkborgsvägen 17, 601 76 Norrköping, Sweden

March 15, 2019

Contents

[*]Corresponding author: paulo.medeiros@smhi.se

1 Introduction

Obsmon is a tool for observation monitoring in the Harmonie-Arome NWP System. It is composed of a "backend" component, which is part of the scripting system and produces databases with the relevant information as part of the post-processing, and a "frontend" component which allows the analysis and visualisation of this data. *This document refers to the frontend component only.* For simplicity, however, the term "obsmon" will be employed here are a synonym of "frontend obsmon".

Obsmon is written in R using the Shiny web application framework. It can be installed either as a standalone application or inside a Shiny Server. This document contains information about how to download (Section 2), install (Section 3) and configure (Section 4) obsmon, as well as tips on how to use some of the non-trivial features of the code (Section 5). If you find that something is missing or needs to be corrected, please feel free to contact us.

2 Getting the source code

To download the code, You need a valid user account on hirlam.org. Open the terminal and enter the following command:

```
git clone https://git.hirlam.org/Obsmon obsmon
```

Enter your hirlam.org credentials as requested.

The obsmon repo contains two main branches:[1] `master` (the default) and `devel`. The `master` branch contains the official obsmon releases. The `devel` branch is generally ahead of `master` and contains code that will eventually make its way into `master` but needs some more testing before that. If you want to collaborate, you should branch off from `devel`.

3 Installing, executing and updating

Use the `install` script to install the `R` libraries needed. The main system requirement is a Linux operating system[2] with a working `R` interpreter. Instructions for the various installation modes are given in Sections 3.1.1 to 3.1.3. Section 3.2 goes through the rocommended steps to update the code. The required system packages are listed in Section 3.4, and the paths where obsmon looks for installed `R` libraries are listed in Section 3.3. Finally, the information presented in Section 3.5 about the use of pre-compiled binaries for the `R` libraries may save you a significant amount of time if you wish to install obsmon in multiple identical computers.

[1]You don't need to worry about obsmon branches if you are not familiar with git.
[2]Installation may work on MacOS, but we have not tested this.

3.1 Installing and executing

3.1.1 Standalone mode

1. Go to the `obsmon` directory and execute:

   ```
   ./install --local-install
   ```

   - The `--local-install` separates the obsmon installation from your regular R environment. This is highly recommended in order to avoid issues with incorrect versions of the R packages
   - If required system packages are missing, then the installation will stop and complain about this. Install the relevant package(s) and execute `./install --local-install` again. See Section 3.4 for more details about system requirements.

2. Create a `config.toml`. You can find a detailed discussion about the configuration file in Section 4. Also, take a look at the template config file `docs/config.toml.example`.

3. To run obsmon, just execute[3]

   ```
   ./obsmon
   ```

   The output will show something similar to

   ```
   Listening on http://127.0.0.1:5391
   ```

   Point your browser to the address you see in your output.[4]

4. Alternatively, if you want the browser to open automatically, you can run obsmon as

   ```
   ./obsmon --launch
   ```

Finally, for a list of command line options currently supported by obsmon in standalone mode, please run:

```
./obsmon -h
```

[4]This only works if the browser is running on the same machine as obsmon. Otherwise you will need to use something like X forwarding or SSH port forwarding to connect to the server.

[4]You can also put a symlink to the obsmon executable somewhere in your `PATH`. If you do so, you will be able to run obsmon as a command from any directory.

3.1.2 In a Shiny Server

If you want to offer obsmon to a larger number of users, you probably want to run it inside the Shiny Server. The installation of the Shiny Server is beyond the scope of this document. You can find more information about this at, e.g., the RStudio Shiny Server download website. In the following, we assume that the Shiny Server is already installed and running.

1. Put the obsmon directory into the `site_dir` directory as configured in your `shiny-server.conf`, or, alternatively, add a `location` stance to your `shiny-server.conf` listing the obsmon directory.
    - You may want to configure in your `shiny-server.conf` the location of your log files. If you do not do so, they will most likely end up somewhere such as `/var/log/shiny-server/obsmon-*.log`, although this cannot be guaranteed
2. Install the required `R` libraries. We recommend that you use the included `install` script as `./install --local-install`. Take notice of where the libraries are being installed to.
3. We *highly recommend* that you set the `R` library search paths manually by using the `.libPaths` `R` command inside a `.Rprofile` file located inside the obsmon directory. Otherwise, you may have issues with conflicting `R` libraries between obsmon and the shiny server. See Section 3.3 for the default paths where obsmnon searches for `R` libraries.
4. Put a valid `config.toml` file inside either the obsmon directory or at `/etc/obsmon`. See the file `docs/config.toml.example` for a template. See also Section 4.
5. Run your Shiny Server and connect to it with your browser.

3.1.3 At ECMWF

Open the terminal and execute the following commands (this requires you to belong to the `hirald` group[5]):

```
module load R/3.3.1
export R_LIBS_BIN_REPO_ROOTDIR=/perm/ms/se/snz/obsmon_precompiled_libs_ecgb
```

After this, follow the instructions given in Section 3.1.1.

Note that it is important to enter `module load R/3.3.1` and not just `module load R`.[6] The `export` statement instructs the installer to use pre-compiled libraries instead of compiling everything from the scratch (see Section 3.5 for details). This saves time in general, and for ECMWF in particular this is

[5]You can check this by inspecting the output of the command `groups`.

mandatory, as they do not allow users to install some of the packages needed at compile time.

Using the browser at ECMWF is currently *very* slow. There is an alternative to make it faster by using a NoMachine remote desktop. At the moment, however, we cannot provide instructions about the configuration of NoMachine at ECMWF. Until we are able to do so, please take a look at the "NX service" section of the ECaccess Web server page, or contact ECMWF's support for more information about NoMachine.

3.1.4 At SMHI

You can follow the instructions given in Section 3.1.1, but you can substantially reduce installation time by setting the `R_LIBS_BIN_REPO_ROOTDIR` environment variable to an appropriate value (see Section 3.5). Please take a look at our internal wiki page for obsmon. If you prefer, we can also produce a package that you can install using our package manager.

3.2 Updating

In the main obsmon directory, execute the following commands[7] (in the presented order):

```
git pull --rebase
./install --local-reinstall
```

This assumes that:

- You have not modified obsmon
- You have followed the recommended approach for installation, using `./install --local-install`

You may occasionally need to install new system dependencies as well if you update obsmon. See Section 3.4.

3.3 R library search paths

Obsmon looks for R libraries installed in the following paths (listed in order of priority):

1. `./utils/build/local_R_library/R-libs`
2. The path listed in the environment variable `R_LIBS_USER`, if set
3. `$HOME/R/library`, if `R_LIBS_USER` is not set

[6]The pre-compiled binaries for the `R` libraries were produced on ECGB using that version of `R`. In particular, If `R v>=3.5` is used, then these libraries will not work and installation will most likely require help from ECMWF support.

[7]The `--rebase` switch can be dropped if your version of git does not support it. It is recommended otherwise.

4. The default R library search paths. These vary depending on your system.[8]

3.4 System dependencies

Table 1 lists the system dependencies for R packages used in obsmon under CentOS 7 or RHEL 7. If you find that any dependencies are missing, please feel free to notify us.

Required system package	First R lib that asks for it during install
libXt-devel	Cairo
cairo-devel	Cairo
libcurl-devel	DBI (via curl dependency)
openssl-devel	DBI (via openssl dependency)
libxml2-devel	DBI (via xml2 dependency)
mariadb-devel	dbplyr (via RMariaDB dependency)
postgresql-devel	dbplyr (via RPostgreSQL dependency)
geos-devel	leaflet (via rgdal dependency)
proj-devel	leaflet (via rgdal dependency)
proj-epsg	leaflet (via rgdal dependency)
gdal-devel	leaflet (via rgdal dependency)
v8-devel	shinyjs (via V8 dependency)

Table 1: System packages needed by obsmon under CentOS 7/RHEL 7. The list was produced by installing the code on a clean, minimal CentOS 7 operating system and keeping note of the system packages required during the process. Many other system packages are installed as dependencies of those listed here and hev been omitted. Last updated on 2018-03-22.

There is currently no robust way to programmatically determine the system dependencies for all Linux distributions. If you use CenOS 7 or RHEL 7, you can run the `install_sys_deps.sh` script (located under `utils/build`) to install these dependencies. For other operating systems, the best approach at the moment is to proceed as indicated in Sections 3.1.1 to 3.1.3 and install the system dependencies as they are requested during installation.

3.5 Using pre-compiled binaries for the `R` libraries

When you use the `install` script to install the `R` libraries needed by obsmon, the binaries for each successfully compiled `R` library are saved into a directory located under `utils/build/local_R_library/pre_compiled`. The specific name of this directory depend on the architecture of your system.[9] Conversely, before attempting to compile any of the required `R` libraries, the `install` script looks for appropriate binaries that may have previously been compiled and stored in such a directory and in the directories listed in the environment variable `R_LIBS_BIN_REPO_ROOTDIR` (if set).

[8]You can check these out by using, for instance, the R function `.libPaths()`.

This mechanism allows installation to resume from where it stopped in case it is interrupted (e.g., if you type `CTRL+C` or if a library fails to compile). Another (perhaps not so obvious) advantage is that it can lead to a very significant reduction in the installation times whenever:

- You need to install obsmon multiple times in the same computer (e.g., for multiple users in an independent way)
- You need to install obsmon in multiple identical computers

In such circumstances, you only need to run the `install` script from scratch once. For the subsequent installs, you can copy the appropriate pre-compiled libraries directory created under `utils/build/local_R_library/pre_compiled` to some accessible location in your computer(s) – which can even be a shared drive, for instance, and then set the value of the `R_LIBS_BIN_REPO_ROOTDIR` environment value to that path. The next time you execute the `install` script, it will then find the pre-compiled binaries and, instead of spending time to recompile them, they will just be installed to the appropriate places. Again, we recommend that you run the install script as `./install --local-install`, so that the obsmon `R` libraries are kept separate from the ones that may be available in your system.

4 The config file

The next step after installation is to create a `config.toml` file under obsmon's main directory.[10] The configuration file has the main purpose of telling obsmon where to find your experiments, but it can also be used to control how the code works. It is written in TOML, which is similar to the widespread `ini` format.

A TOML config file is made up of sections (called "tables" in TOML parlance). A valid config file for obsmon:[11]

- May contain one `[general]` section
- Must contain at least one `[[experiments]]` table
- May contain multiple `[[multiPlots]]` sections

The next sections describe the config file options for each section. An example of a simple config file is presented in Section 4.3. For a more complex config file example, see the config file template `docs/config.toml.example`.

[9]It may, for instance, look like `x86_64-redhat-linux-gnu-RedHatEnterpriseClient-7`

[10] Other options exist. This is how obsmon looks for the config file (in order of priority):

1. The full file path the environment variable `OBSMON_CONFIG_FILE` points to (if set)
2. A `config.toml` file under the obsmon installation directory
3. A `config.toml` file under `/etc/obsmon/$USER`

[11] Note the mandatory double brackets in `[[experiments]]` and `[[multiPlots]]`, as opposed to the single brackets used in `[general]`. *Sections defined with single brackets cannot have repeated names, whereas multiple double-bracketed sections with the same name are allowed.*

4.1 Options for the [general] section

As the name suggests, the `[general]` section controls how the code works in general. The currently available options are:

- `cacheDir`: Directory where obsmon will store its cache.
  Default: `/var/cache/obsmon/$USER`
  For more on cache, please read Section 5.2.

- `logLevel`: Amount of detail that is logged.
  Default: `WARN`
  Valid values (in decreasing order of detail): `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR` and `FATAL`

  – In standalone mode, everything is logged to `stderr`
  – If running on a shiny server, then the log ends up in the respective shiny server log. For more details about logging while running on a shiny server, please take a look at the shiny documentation for logging and analytics

- `initCheckDataExists`: Whether or not to assert the existence of the experiments' data files at startup.
  Default: `false`
  Accepted values: `true` or `false`

  – If `false`, then Obsmon will determine the available DTGs at startup only by looking at the directories that exist under the experiments' `path`s. This is generally quick.
  – If `true`, the DTGs will only be shown as available if the corresponding data files actually exist inside the directories under the corresponding `path`s. This is slow and not generally necessary, but is safe to use if you absolutely need it.

- `maxExtraParallelProcs`: Maximum number of extra parallel tasks that obsmon is allowed to execute at any given time.
  Default: Unlimited
  Acceptable values: Any integer number greater or equal to zero. Passing invalid values will cause it to fall back to the default.

  Asynchronous/parallel tasks are used, e.g., to be able to load and cache experiments without blocking the GUI, as well as to allow having a "cancel plot" functionality.

  – These extra tasks are short-lived and are not usually computationally intensive (not all at the same time, at least), so we advise you not to set this option unless you have good reasons to do so.
  – The `OBSMON_MAX_N_EXTRA_PROCESSES` environment variable can also be used in order to control this setting, but the value set via the config file takes precedence.

- `showCacheOptions`: Whether or not to show advanced cache options (see Section 5.2.1).
  Default: `false`
  Accepted values: `true` or `false`

- `multiPlotsEnableInteractivity`: Whether or not to allow `multiPlots` (see Section 5.3) to be interactive.
  Default: `false`
  Accepted values: `true` or `false`

  We have chosen to make this option `false` by default to avoid memory issues, as large numbers of individual plots may in principle be generated within a single `multiPlot`. This is especially important when deploying obsmon in a Shiny Server.[12]

4.2 Options for the `[[experiments]]` section

This the section where you tell obsmon how to locate your experiment files and how you want your experiments to be named. You should include one `[[experiments]]` section for each experiment, each section defining the following two keys:

- `displayName`: Used to identify the experiment in the web interface.
  Accepted values: Any (you are free to choose the name of your experiment)

- `path`: Path to the directory containing the experiment data.
  Allowed values: Any valid experiment path. More details below.

Obsmon expects to find directories named `ccma`, `ecma`, `ecma_sfc` under the directory specified in `path` (or at least one of them). These directories are assumed to contain the databases corresponding to minimisation, screening and CANARI data, respectively. Each of these directories is expected to contain sub-directories named according to the standard dtg format `YYYYmmddHH` (one such directory for each available DTG in the experiment), which, in turn, should contain the actual database files, named, again respectively, `ccma.db`, `ecma.db` and `ecma.db` (sic).

Obsmon finds the data by combining into a single path (i) the value passed in `path`, (ii) the appropriate directory for minimisation, screening, or CANARI data, (iii) the date(s) and cycle(s) selected in the GUI, and finally (iv) the appropriate `.db` file.

4.3 Config file example

A simple config file may look like the following:

[12] Problems are known to occur occasionally in this case. We have indeed experienced a few.

```
[general]
    logLevel = "INFO"
    cacheDir = ".Rcache"

[[experiments]]
    displayName = "First Experiment"
    path = "/full/path/to/experiment1"

[[experiments]]
    displayName = "Second Experiment"
    path = "/full/path/to/experiment2"
```

For a more complete example, please take a look at the template config file `docs/config.toml.example`. A few notes:

- multiPlots are not covered by this simple example. They will be discussed in Section 5.3
- Comments can be added using a `#` character and are optional
- Indentation is also optional, but highly encouraged

5 Using obsmon

Once installed (Section 3) and minimally configured (Section 4), obsmon is typically quite simple to use. This section discusses those features of the code that may be less straightforward to use, or about which we most commonly receive questions from users. If you have any doubts that are not covered here, you recommend you to take a look at the FAQ in Section 6.

5.1 Interactive plots

Some plots in obsmon allow you to zoom, pan, hover etc. A not-so-obvious but quite useful feature of these plots is the ability to hide and show parts of the plotted data. You can do so by single- or double-clicking on the items in the legend.

Note that interactivity is disabled by default in `multiPlots` (Section 5.3). The config file option `multiPlotsEnableInteractivity` (see Section 4.1) can be used to enable interactivity in this case.

5.2 Caching

Every time you select a new {`experiment, database, DTG(s)`} combination, obsmon collects metadata such as available observation types and names, variables and station IDs from relevant data files. This information is then saved to `sqlite` cache files.[13] Such cached information is then used to populate the menus in the GUI with choices that better reflect the contents of your experiments' data files.

Caching occurs automatically. How long it takes for caching to be completed depends on factors such as how many new[14] DTGs you have selected or, for instance, whether your experiment's data files are located in the same computer as obsmon (faster) or in a network mount point (slower). Until caching is finished (of if cache information could not be retrieved for whatever reason), the menus in the GUI become populated with a set of default values. You can therefore continue to explore your data even if cached info is unavailable or incomplete.

5.2.1 Advanced cache options

Advanced cache-related options can be enabled either by setting the parameter `showCacheOptions` to `true` in the config file (see Section 4.1) or by creating a file named `.obsmon_show_cache_options` inside the main obsmon directory.[15] The GUI will then feature two buttons that allow rewriting or resetting the cache files. You would typically use this to solve problems with a corrupted cache file.

As obsmon may be run via shiny server and/or shared by multiple users, these buttons are hidden by default to prevent individual users from changing the shared cache files. This is safe to use if you know you won't inadvertently affect other users. In the worst case scenario, completely resetting the cache files will only cause obsmon to re-cache information when necessary.

5.3 Configuration of `multiPlots`

A `multiPlot` is a collection of multiple plots performed as a result of a single plot request. It provides a way to reduce the number of operations (or "clicks") a user needs to perform in order to generate multiple plots.

The parameters for each `multiPlot` are defined in the configuration file through the use of `[[multiPlot]]` sections. In the following, we will discuss the main aspects of how to configure a `multiPlot`. There is no limit to the number of `multiPlots` that can be configured. To add a new `multiPlot`, just add a new `[[multiPlot]]` section to the configuration file. We encourage you to take a look at the template config file `docs/config.toml.example` for more concrete examples.

[13] Located under the directory defined by the `cacheDir` key in the config file (see Section 4.1).

[14] Unless explicitly requested (see Section 5.2.1), obsmon will not collect metadata for {`experiment, database, DTG(s)`} combinations that have already been cached.

[15] We recommend following the config file route to set this option, as it is more explicit. This route requires obsmon to be restarted to work. The `obsmon_show_cache_options` file strategy, however, was designed as a simple way to allow changing this configuration without having to restart obsmon (only a page refresh is required), which can be useful when running obsmon in a Shiny Server.

5.3.1 Parameters shared by all plots within the same `multiPlot`

The following parameters apply for all plots contained within a `multiPlot` and must be specified when defining a new `multiPlot`:

- `displayName`: Used to identify the `multiPlot` in the web interface.
  Accepted values: Any (you are free to choose the name of your `multiPlot`).
- `experiment`: Which experiment to use.
  Accepted values: The `displayName` of a valid `[[experiments]]` entry. See Section 4.2.
- `database`: Which experiment database to get data from.
  Accepted values: `ccma`, `ecma` or `ecma_sfc`.
- `plotType`: The type of plot that will be performed.
  Accepted values: Any `plotType` ordinarily supported by obsmon
- DTG-related parameters. How to configure these depend on whether the chosen `plotType` requires a single date value or a date range.

  Parameters to use if a single date is required:

  - `date`: Date in the `"YYYY-MM-DD"` format. E.g.: `"2018-05-12"`.
  - `cycle`: Cycle number in the `"HH"` format. E.g.: `"03"`.

  Parameters to use if a date range is required:

  - `startDate`: A date in the `"YYYY-MM-DD"` format or a negative integer number. If a negative integer `N` is passed, then it will represent a date | `N` | days before the day the plot is requested.
  - `endDate`: A date in the `"YYYY-MM-DD"` format. *Do not use this if `startDate` is a negative integer or if `nDays` (see below) is used.*
  - `nDays`: A positive integer number `N`, Sets the end date for the plot to `N` days after `startDate`. *Do not use this if `startDate` is a negative integer or if `endDate` is used.*

5.3.2 Minimal `[[multiPlot]]` entry configuration

The parameters described in Section 5.3.1 are already enough for a minimal `[[multiPlot]]` configuration. The following definition, for instance, is a perfectly valid `multiPlot` entry:[16]

```
[[multiPlots]]
    displayName = "A multiPlot with everything I can possibly have"
    experiment = "The name of my experiment"
    plotType = "Number of Observations"
    database = "ecma"
    startDate = -30
```

[16] Provided that `experiment` has a valid value. See Section 5.3.1.

Such an entry will create a `multiPlot` that will contain one plot of the "Number of Observations" type for every valid combination of observation type, name, variable, levels, station, satellite name, sensor, channels, etc.[17] The default behaviour can thus be summarised as: *Everything is included unless otherwise specified.*

5.3.3 Selecting what to include and/or exclude from a `multiPlot`

The total number of individual plots composing a `multiPlot` depends on what you choose to include and/or exclude. As explained in Section 5.3.2, obsmon adopts the convention that a `multiPlot` will include everything that can possibly make sense within its context, unless specified otherwise.

Including and/or excluding parameters such as observation types, variables, levels, stations, satellites, channels, etc. is relatively simple. There are, however, many such options, and explaining them all here would take many more lines than actually writing down the corresponding entries in the configuration file. For this reason, we have instead produced a template `config.toml.example` file containing a comprehensive selection of examples on how to setup `multiPlots`. This file can be found under the `docs` directory.

We highly recommend that you read and understand the template config file if you want to setup `multiPlots`, and we encourage you to copy the entries from that file and adapt them to what you need. Finally, feel free to get in touch should you think something is missing.

[17]Some `plotTypes` may support multiple values of some parameters (such as levels, stations and channels, for example). In such cases, all such parameter values will be included in the same plot within the `multiPlot`, a behaviour which is consistent with how the individual plots work when performed ordinarily in obsmon.

6 Frequently asked questions

Questions about installation

Q: Do I need root permissions to install obsmon?

A: In general yes (but not at ECMWF). In most cases, you should at least be able to run your system's package installer using `sudo` (e.g., `sudo yum` or `sudo apt-get`), so you can install the system dependencies.

Q: Installation is taking too long. Is there any way to speed it up?

A: Generally not, unfortunately. Many of the required `R` libraries need to compiled and they, in turn, generally require extra system packages to be installed. *If you plan to install obsmon multiple times, however, please read Section 3.5.* Note also that if installation is restarted after failure or interruption it will resume from where it stopped, not start from scratch.

Questions about caching

Q: I see "cache info not available" (or "cache info incomplete") in one or more menus. What should I do?

A: You normally do not need to do anything. The message will disappear as soon as caching for the selected experiment/database/DTG(s) is finished. If, however, you still think it's taking too long, you can try changing one of the selected parameters back and fourth to force the menus to be updated. *You can continue to use obsmon even if this message is shown.*

Questions about plotting

Q: My plot says "Query returned no data". What does that mean?

A: That means that your experiment does not contain the data needed for the plot according to the parameters you chose. This should be a rare occurrence when cache finishes normally, but can happen more often when caching is not available or incomplete.

Q: My plot says "Could not produce plot: The required data file(s) might be inaccessible.". What does that mean?

A: That may indicate that the required experiment data files are not available. Please double-check that they exist. Contact us if they do.

Q: My plot is empty. What happened?

A: The most probably cause is that data associated with your plot request could indeed be found, but the number of observations is zero. Please take a look at the data under the "Query and Data" tab.

Q: How do I save my plot as a figure?

A: If the plot is interactive (e.g., if you can zoom in), then you will find such an option in the menu that appear on the top right of the figure when you place the mouse over it. If the plot is not interactive, you can just right-click with your mouse and choose to save it.

Q: How can I save the data used to produce my plot?

A: Go to the "Query and Data" tab and click in the appropriate button to export the data as `txt` or `csv`.

Questions about `multiPlots`

Q: I do not see a `multiPlots` tab in the GUI, even though I have configured a `multiPlot`. What happened?

A: This most likely means that there is an error in the configuration of your `multiPlots`. Obsmon will only feature a `multiPlots` tab if at least one valid `multiPlot` configuration is found.

Q: I do not see one (or more) of my `multiPlots` in the list. What happened?

A: This most likely means that there is an error in the configuration of the missing `multiPlots`. Obsmon will only show the `multiPlots` for which the configuration passed without errors.

Q: My `multiPlot` is taking too long to finish. What is going on?

A: How long it takes for a `multiPlot` to be completed depends on factors such as how much you have chosen to include or exclude from the plots (see Section 5.3), as well as where the required data files are located (faster if stored locally, slower if stored remotely). If you specify a date range, then longer time spans will naturally imply longer processing times for the `multiPlots`, as each new DTG corresponds to a new file to query data from. Additionally, some plots, such as "Station Diagnostics", require some statistics to be performed on the data before it can be plotted. This can also increase processing times.

Q: My `multiPlots` are not interactive. What happened?

A: Interactivity is switched off by default in the case of `multiPlots`. It can be switched on using the `multiPlotsEnableInteractivity` config file option. See Section 4.1.

7 Collaborators

Obsmon development greatly benefits from collaboration from users in the form of feature requests and occasional bug reports. This is appreciated and encouraged. Coding contributions are also welcome. The following people have contributed with code up to the current version:

- Klaus Zimmermann (SMHI)
- Paulo Medeiros (SMHI)
- Trygve Aspelien (Norwegian Meteorological Institute; started obsmon)
- Ulf Andrae (SMHI)

Please feel free to contact us if you wish to collaborate.