Obsmon Documentation

Paulo Medeiros* and Klaus Zimmermann

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

November 20, 2019

Contents

1 Introduction **2**

2 Getting the source code **2**

3 Installing, executing and updating **2**
3.1 Installing and executing . . . 3
3.1.1 Standalone mode . . . 3
3.1.2 Through a Shiny Server . . . 4
3.1.3 At ECMWF (ecgb) . . . 4
3.1.4 At SMHI . . . 5
3.2 Updating . . . 5
3.3 R library search paths . . . 5
3.4 System dependencies . . . 6
3.4.1 CentOS, RHEL or Ubuntu . . . 6
3.4.2 Other Linux distributions . . . 6
3.5 Using pre-compiled binaries for the `R` libraries . . . 7

4 The config file **8**
4.1 Options for the `[general]` section . . . 8
4.2 Options for the `[[experiments]]` section . . . 10
4.3 Config file example . . . 11

5 Using obsmon **12**
5.1 Interactive plots with editable elements . . . 12
5.2 Producing multiple plots in one go with `multiPlots` . . . 12
5.2.1 Parameters shared by all plots within the same `multiPlot` 12
5.2.2 Minimal `[[multiPlot]]` entry configuration . . . 13
5.2.3 Selecting what to include/exclude from a `multiPlot` . . . 14
5.3 Batch mode . . . 14
5.3.1 Configuration of batch-mode plots . . . 14
5.3.2 Running obsmon in batch mode . . . 16
5.4 Caching . . . 16
5.4.1 Advanced cache options . . . 17

6 Frequently asked questions **18**

7 Collaborators **20**

*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 deployed as a local, standalone application or even remotely through the use of a web server (for instance, using 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 recommended 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.[3] Install the relevant package(s) and execute `./install --local-install` again. Obsmon has helper scripts to install system dependencies in some Linux distributions – see Section 3.4 for more details.

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[4]

   ```
   ./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.[5]

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
```

[3] The installation will stop immediately after every failure. If you wish the install script to keep going regardless of failures found, then pass the `--ignore-build-fail` option to it. It will then keep going while ignoring what cannot be installed.

[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.

[5] This only works if the browser is running on the same machine as obsmon. Otherwise you will need to use something like SSH local port forwarding (e.g., `ssh -L 5391:localhost:5391 user@computer`) or, less preferably, X forwarding to connect to the server.

3.1.2 Through a Shiny Server

If you want to offer obsmon to a larger number of users, you may want to deploy it using a web server. A canonical choice in this case would be to use a Shiny Server. Shiny Server installation is beyond the scope of this document.[6] 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 obsmon 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 (ecgb)

Open the terminal and execute the following commands (this requires you to belong to the `hirald` group[7]):

```
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`.[8] 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

[6] Please visit the RStudio Shiny Server download web page for information about this.
[7] 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 remotely from ECMWF can unfortunately be *very* slow. *We recommend that you run obsmon in batch mode if this is the case* (see Section 5.3). There is also an alternative to have faster access to the browser by using a NoMachine remote desktop. At the moment, however, we cannot provide instructions about the configuration of NoMachine at ECMWF. For this, we refer you to the "NX service" section of the ECaccess Web server page.

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 also 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

Updating is similar to installing, except that you need to:

- First run[9] `git pull --rebase`
- Then use option `--local-reinstall` (instead of `--local-reinstall`) with the `install` script

This assumes that:

- You have not modified obsmon
- You have followed the recommended installation approach when first installing the code, using `./install --local-install`

Updating is normally faster than performing a completely new install, as obsmon tries to use pre-compiled binaries generated during the previous install/update (see Section 3.5). However, you may occasionally need to install new system dependencies when you update (see Section 3.4). Finally, you should always update the code after having switched branches.

3.3 R library search paths

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

[8] 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.

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

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
4. The default R library search paths. These vary depending on your system.[10]

3.4 System dependencies

3.4.1 CentOS, RHEL or Ubuntu

Tables 1 and 2 list the system dependencies for the R libraries used in obsmon under CentOS/RHEL and Ubuntu, respectively. You can find helper scripts under `utils/build` to install system dependencies in these Linux distributions. Please feel free to notify us if any dependencies are missing.

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: **CentOS 7/RHEL 7** system dependencies for obsmon. The list was produced by installing the code on a newly installed, minimal system and keeping note of the packages required during the process. Many other system packages are installed as dependencies of those listed here and have been omitted. Last updated on 2018-03-22.

3.4.2 Other Linux distributions

There is currently no robust way to programmatically determine the system dependencies for all Linux distributions. The recommended 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. You can nevertheless use the names of the Linux packages shown in Tables 1 and 2 as reference, as, although the names change from one distribution to another, they are generally similar.

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

Required system package	First R lib that asks for it during install
libcurl4-openssl-dev	curl
libssl-dev	openssl
libxml2-dev	xml2
libmariadb-client-lgpl-dev	RMariaDB
libpq-dev	RPostgreSQL
libcairo2-dev	gdtools
libgeos++-dev	rgeos
libgdal-dev	rgdal
libxt-dev	Cairo
libv8-3.14-dev	V8

Table 2: **Ubuntu 18.04.2 LTS** system dependencies for obsmon. The list was produced by installing the code on a newly installed system and keeping note of the packages required during the process. Many other system packages are installed as dependencies of those listed here and have been omitted. Last updated on 2019-05-28.

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.[11] 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).

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 need to run the `install` script from scratch only 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 install them to the appropriate places without further compilation. 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.

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

4 The config file

The next step after installation is to create a configuration file. Such a 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.

Obsmon looks for a configuration file named `config.toml` under the following directories (listed in order of priority):

1. `$HOME/.obsmon`
2. `/etc/obsmon/$USER`
3. The obsmon installation directory

Alternatively, you can instead provide the full path to a valid configuration file using the `OBSMON_CONFIG_FILE` environment variable, in which case you are free to choose the file name. This takes higher priority over the other options.

A TOML config file is made up of sections (called "tables" in TOML parlance). A valid config file for obsmon:[12]

- 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 the `[general]` and `[[experiments]]` tables. The configuration of `[[multiPlots]]` will be discussed in Section 5.2. 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`.

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:

- `appTimeout`: Time interval, in seconds, after which obsmon will stop and exit after all browser sessions have been closed.
  Default: `"Inf"`
  Acceptable values: Any number greater than or equal to zero. Passing invalid values will cause it to fall back to the default.

  This option is similar but not equal to the option `sessionTimeout` described further below.

[12] 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.*

- `cacheDir`: User-configurable part of the path to where obsmon will store its cache.
  Default: `$HOME/.obsmon/experiments_cache`

  The path initially passed to this parameter will be further appended by a directory named as `obsmon_v`MAJOR.MINOR. This is to avoid conflict when updating obsmon. Furthermore, cache files for each experiment will be stored separately inside directories named according to the experiments' names. For more information on cache, please read Section 5.4.

- `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

- `maxAvgQueriesPerProc`: Maximum number of database queries (in average) that a single process (computer thread) is allowed to perform when preparing plots.
  Default: `"Inf"`
  Acceptable values: Any integer number greater than or equal to one, or "Inf". Passing invalid values will cause it to fall back to the default.

  Data for obsmon plots is stored in separate files for each `DTG`. Therefore, if your plot requires `nDTGs`, then `nDTGs` independent database files will be queried. With the `maxAvgQueriesPerProc` option, these queries will be divided into `max{1, ceiling(nDTGs/maxAvgQueriesPerProc)}` groups that will then be processed in parallel.

  The optimal value for this parameter is, of course system-dependent,[13] and the default `"Inf"`[14] leads to no query parallelisation.

- `maxExtraParallelProcs`: Maximum number of extra tasks that obsmon is allowed to execute, at any given time, in parallel to the main process.
  Default: $4 \times$ `#availableCores`
  Acceptable values: Any integer number greater than 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

[14] In our systems, for example, we have been able to get up to 2.5× speedup by setting `maxAvgQueriesPerProc=24`, but one needs to test.
[14] Please interpret division by `Inf` in this context as a mathematical limit.

experiments without blocking the GUI, as well as to allow having a "cancel plot" functionality. It is worth mentioning that:

  - 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 parameter to a smaller value 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.

- `multiPlotsEnableInteractivity`: Whether or not to allow `multiPlots` (see Section 5.2) 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.[15]

- `plotsEnableInteractivity`: Whether or not to allow *regular plots* to be interactive.
  Default: `true`
  Accepted values: `true` or `false`

- `sessionTimeout`: Time interval, in seconds, after which any idle obsmon web session should be terminated.
  Default: `"Inf"`
  Acceptable values: Any number greater than or equal to zero. Passing invalid values will cause it to fall back to the default.

  Users will receive at least 60s warning that their sessions will be terminated if they remain idle.

- `showCacheOptions`: Whether or not to show advanced cache options (see Section 5.4.1).
  Default: `false`
  Accepted values: `true` or `false`

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

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

`[[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:

```
[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 5.2
- 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 with editable elements

Most plots in obsmon allow you to zoom, pan, hover etc. A less obvious but 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. You can also edit plot titles, legends, axes labels, as well as change the position of some of these elements by dragging them.

This feature is enabled by default for regular plots, and it can switched off/on by using the config file option `plotsEnableInteractivity` (see Section 4.1). The default behaviour for `multiPlots` (Section 5.2), on the other hand, is different, and interactivity is disabled for such plots unless otherwise specified via config file option `multiPlotsEnableInteractivity`.

5.2 Producing multiple plots in one go with `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.

5.2.1 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.*

5.2.2 Minimal `[[multiPlot]]` entry configuration

The parameters described in Section 5.2.1 are already enough for a minimal `[[multiPlot]]` configuration. The following definition, for instance, is a perfectly valid `multiPlot` entry:[16]

```
[[multiPlots]]
    displayName = "A minimal multiPlot config entry"
    experiment = "The name of my experiment"
    plotType = "Number of Observations"
    database = "ecma"
    startDate = -30
```

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.*

[16] Provided that `experiment` has a valid value. See Section 5.2.1.

[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.

5.2.3 Selecting what to include/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 5.2.2, 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.

5.3 Batch mode

Batch mode allows producing plots without the use of the GUI. When run in batch mode, obsmon will produce the appropriate pre-configured plot(s), save the results in individual files (under appropriately named directories, see more below), and then exit.

There are many usage scenarios where such a functionality can come in handy. For instance, remotely opening a web browser from ECMWF can be very slow, as discussed in Section 3.1.3. In such cases, one can run obsmon in batch mode and then retrieve the generated files via, e.g., `ftp`. Another usage case example would be calling obsmon from a script that runs as part of a `cron` job, thus allowing one to regularly produce plots without user intervention.

5.3.1 Configuration of batch-mode plots

Batch-mode plots in obsmon are simply `multiPlots` activated for this type of use.[18] Therefore, the first step to configure a batch-mode plot is to setup a valid `multiPlot` as described in Section 5.2. Properly configured `multiPlots` can then be activated for use in batch mode by adding, *under the main level in the corresponding `[[multiPlots]]` entry*, either:

(a) A `[multiPlots.batchMode]` table, which may be empty or contain any[19] of the the following entries:

- `enable`: Whether or not to enable the `multiPlot` for use in batch mode.

[18] In particular, `multiPlots` marked for use in batch-mode will be available as regular `multiPlots` if the GUI is used.

[19] Or all of them, unless otherwise specified.

Default: `true`
Accepted values: `true` or `false`

- `parentDir`: Where to put the directory containing the plots produced by the `multiPlot` when run in batch mode.
  Default: The directory obsmon is being executed from.
  Accepted values: Any path where the user running obsmon has write access to. Both relative and absolute paths are accepted. Relative paths are assumed to be relative to where obsmon is being executed from. A `parentDir` may be shared by multiple `multiPlots`.
- `dirName`: The name of the directory where to put the produced plots. *This is just the directory name.* It will be prepended by `parentDir` to generate a full path.
  Default: `obsmon_batch_MPNAME_TIMESTAMP`, where `MPNAME` is a version of the `multiPlot`'s `displayName` in lowercase and with any non-word characters (or sequence of such characters) replaced by a single underscore, and `TIMESTAMP` is the time (in %H%M%S format) when the directory was created.
  Accepted values: Any valid directory name such that the full path `parentDir/dirName` does not already exist.
- `fileType`: The type of the graphics files produced.
  Default: `png`
  Accepted values: Most of the commonly used file types that support saving graphics (e.g., `pdf`, `jpeg`, `tiff`, `png`, `bmp`). Note, however, that this is system-dependent.[20]
- `dpi`: Resolution of the generated figures (in dots per inch).
  Default: `300`
  Accepted values: Any integer greater than zero.
- `figHeight`: Height of the generated figures (in inches).
  Default: `6`
  Accepted values: Any number greater than zero.
- `figWidth`: Width of the generated figures (in inches).
  Default: `10`
  Accepted values: Any number greater than zero.

Example (compare with Section 5.2.2):

```
[[multiPlots]]
    displayName = "An arbitrary multiPlot"
    experiment = "My experiment"
    plotType = "Number of Observations"
    database = "ecma"
    startDate = -30
    [multiPlots.batchMode]
        parentDir = "/home/user/obsmon_batch_mode_plots"
```

[20] For more details, see the documentation of the `ggsave` function from the ggplot2 `R` package.

or, if you want to keep it simple,

(b) `batchMode = true`

And that is it. This is of course much simpler than going via option (a), but it does not allow any customisation. Example (compare with Section 5.2.2):

```
[[multiPlots]]
    displayName = "An arbitrary multiPlot"
    experiment = "My experiment"
    plotType = "Number of Observations"
    database = "ecma"
    startDate = -30
    batchMode = true
```

Using option (a) with an empty `[multiPlots.batchMode]` table has the same effect as using option (b). Finally, setting `batchMode = false` in option (b) is also allowed, in which case the corresponding `multiPlot` will, rather unsurprisingly, not become activated for use in batch mode.

5.3.2 Running obsmon in batch mode

Just use the `--batch` command line option:

```
./obsmon --batch
```

5.4 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 the relevant data files. This information is then saved to `sqlite` cache files.[21] Cached information is used to populate the menus in the GUI with choices that better reflect your experiments' data.

Caching occurs automatically and asynchronously. How long it takes for it to be completed depends on factors such as how many *new*[22] 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). While caching is not finished (or if cache information cannot 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.

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

[22] Obsmon will not re-collect metadata for {`experiment, database, DTG(s)`} combinations that have already been cached, unless (i) the cached information is older than the last-modified time of the corresponding experiments' data files, or (ii) upon a direct user request to do so (see Section 5.4.1).

5.4.1 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.[23] 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.

These buttons are hidden by default to prevent individual users from changing the contents of cache files when using a single shared obsmon installation (be it available locally or via web). Nevertheless, is safe to use these advanced cache-resetting options if you know you won't inadvertently affect other users. In fact, it is actually safe to manually remove the cache files if the same circumstances apply. In the worst case scenario, completely resetting/removing the cache files will only cause obsmon to re-cache information when/if necessary. Your experiments' data files will not be affected if you click on any of these buttons.

[23] 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.

6 Frequently asked questions

Questions about installation

Q: Do I need root (admin) 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 be 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 "incomplete cache & defaults" in the title of one or more menus. What does this means and what should I do?

A: This just means that obsmon has not yet cached all information it needs about the selected experiment/database/DTG(s) in order to accurately populate the menus, and that you are seeing a combination of whatever the code has already cached and some defaults. You normally do not need to do anything, the message will disappear as soon as caching is finished. *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 appears 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 indicates that there may be 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 5.2), 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 user input in the form of feature requests and occasional bug reports. This is appreciated and encouraged. Coding contributions are also welcome. The following people have made code contributions to obsmon up to the current version:

- Klaus Zimmermann (SMHI)
- Paulo Medeiros (SMHI, main developer at the moment)
- Trygve Aspelien (Met.no; original author of obsmon)
- Ulf Andrae (SMHI)

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