

Obsmon Documentation

Paulo Medeiros*

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

July 20, 2022

Contents

1	Introduction	2
2	Getting obsmon	2
2.1	Downloading 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)	5
3.1.4	At SMHI	6
3.2	Updating	6
3.3	R library search paths	6
3.4	System dependencies	6
3.4.1	CentOS, RHEL or Ubuntu	6
3.4.2	Other Linux distributions	7
3.5	Using pre-compiled binaries for the R libraries	8
3.6	Advanced install options	8
4	The config file	9
4.1	Options for the [general] section	10
4.2	Options for the [[experiments]] section	12
4.3	Config file example	13
5	Using obsmon	13
5.1	Interactive plots with editable elements	13
5.2	Physical Units in the Plots	13
5.3	Explore The Data Used in Your Plots	14
5.4	Producing multiple plots in one go with multiPlots	14
5.4.1	Parameters shared by all plots within the same multiPlot	14
5.4.2	Minimal [[multiPlot]] entry configuration	15
5.4.3	Selecting what to include/exclude from a multiPlot	15
5.5	Batch mode	16
5.5.1	Configuration of batch-mode plots	16
5.5.2	Running obsmon in batch mode	18
5.6	Domain Geometry & Grid	18
5.7	Caching	19
5.7.1	Advanced cache options	19
6	Frequently asked questions	21
7	Collaborators	24

*paulo.medeiros@smhi.se

1 Introduction

Obsmon is a tool for observation monitoring in the [Harmonie-Arome NWP System](#). It allows the analysis and visualisation of data produced by `prepObsmon`.¹ *This document refers only to `obsmon`, and not to `prepObsmon`.*

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 get (Section 2.1), 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 obsmon

2.1 Downloading the source code

Obsmon v>=4.0.0 can be downloaded from [GitHub](#).² Open the terminal and enter the following command:

```
git clone https://github.com/Hirlam/obsmon.git
```

The `obsmon` repo contains two main branches:³ `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 the latter 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 requirements⁴ are:

- A Linux operating system⁵
- A working R ($\geq 4.0.5$) interpreter
- `python` ($\geq 3.6.7$)
- Internet connection⁶

¹Which is part of the scripting system (post-processing) and produces SQL databases with information filtered from ODB.

²For earlier versions, you need a valid user account on [hirlam.org](#). In this case, please use `git clone https://git.hirlam.org/Obsmon obsmon` and enter your [hirlam.org](#) credentials.

³Don't worry about `obsmon` branches if you are not familiar with git.

⁴There are also specific system requirements. See Section 3.4.

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.

3.1 Installing and executing

3.1.1 Standalone mode

1. Go to the obsmon directory and execute:⁷

```
./install
```

- If required system packages are missing, then the installation will stop.⁸ Install the relevant system package(s) and execute `./install` again. Obsmon has helper scripts to install system dependencies in some Linux distributions – see Section 3.4 for more details.
 - You may need to customise installation for some R-packages (non-standard paths for system libraries, for instance). In such cases, please use the `install` script's `-ca` and/or `-cv` arguments⁹. These options are, respectively, passed to the `configure.args` and `configure.vars` arguments of R's `install.packages` function.
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 execute¹⁰

```
./obsmon
```

The output will show something similar to

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

⁶Installation may work on MacOS, but we have not tested this.

⁶Unless you use a local R-package repository.

⁷Please run `./install -h` for more help about the install script.

⁸Please check the produced log file in case of installation failures.

⁹For instance, a recent install we performed required passing the following arguments to the install script: `-ca="--with-gdal-config=/usr/gdal23/bin/gdal-config" -ca="--with-proj-include=/usr/proj49/include" -ca="--with-proj-lib=/usr/proj49/lib"`

¹⁰You 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.

Point your browser to the address you see in your output.¹¹

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
```

3.1.2 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.¹²In 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.¹³ We recommend that you use the included `install` script¹⁴ with the default settings.¹⁵
 - If you don't follow the recommended install approach, then we *highly recommend* that you manually specify the paths to the directories where you installed the R libraries. You can do this by using the `.libPaths` (R-language) command inside an `.Rprofile` file placed in the `obsmon` directory. Otherwise, you may have issues with conflicting R libraries between `obsmon` and the shiny server.
3. 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.
4. Run your Shiny Server and connect to it with your browser.

¹¹This 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.

¹²Please visit the [RStudio Shiny Server download web page](#) for information about this.

¹³See Section 3.3 for the default paths where `obsmon` searches for R libraries.

¹⁴Please run `./install -h` for more help about the `install` script options.

¹⁵In which case the libraries will be installed locally inside the `obsmon` directory, and no further `.libPaths` setup is needed.

3.1.3 At ECMWF (ecgb)

There are different options to get obsmon running at ECMWF (ecgate). In any case, the first step is to load the `obsmon` module, which is done by running the following commands (assumes that you are a member of the `hiraId` group¹⁶):

```
module use /perm/ms/se/snz/modulefiles
module load obsmon
```

After this, you can choose to:

1. Use the pre-installed version of obsmon (*recommended*):

Just set up a valid configuration file (see Section 4 for details about this) and then run the `obsmon` command.

2. Make your own install of obsmon:

Follow the instructions given in Section 3.1.1.

Take note of the `major.minor` version of R that was loaded when you installed obsmon, and *make sure to always use that same version of R later on, when executing the code*. This is because the `module load obsmon` command sets up the environment so that (i) a specific version of R is loaded and (ii) the installer can find R-libraries pre-compiled for that particular version.¹⁷

If you are not installing the latest version of obsmon, there may not be pre-compiled R-libraries available for your particular install. We unfortunately cannot provide much support if this happens, and installation of the necessary R-libraries may require help from ECMWF support. Therefore, we recommend that you use the latest official release whenever possible if you want to have your own install of obsmon.

Finally, a remark on using the browser remotely from ECMWF: *This can unfortunately be very slow sometimes*. If this is the case, then *we strongly recommend that you run obsmon in batch mode* (see Section 5.5). 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](#).

¹⁶See the output of the command `groups` if you are not sure. Support for obsmon at ECMWF is unfortunately very limited if you are not a member of the `hiraId` group.

¹⁷Using pre-compiled R-libraries (see Section 3.5 for details) greatly helps installing obsmon at ECMWF. Not only it saves an enormous amount of time, but, most importantly, it circumvents issues that arise due to the fact that users cannot generally install many of the system-wide packages needed in order to compile these R packages.

3.1.4 At SMHI

You can follow the instructions given in Section 3.1.1, but you can substantially reduce installation time by passing an appropriate value to the `-bin-repo-path` option of the `install` script (to make use of pre-compiled binaries; more on this on Section 3.5). Please take a look at our internal Obsmon wiki page for more details. If you prefer, we can also produce a package that you can install using our package manager.

3.2 Updating

Please run:

```
git pull --rebase && ./install
```

This assumes that:

- You have not modified obsmon
- You have followed the recommended installation approach when first installing the code.

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 obsmon (see Section 3.4). **N.B.:** 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):

1. `.installer_local_R-libs/R-libs`
2. The default R library search paths. These vary depending on your system.¹⁸

If you wish for obsmon to look somewhere else for R-libs, then you can, *e.g.*, edit the `.libPaths` call in the `.Rprofile` file located inside the obsmon root directory.

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.

¹⁸You can check these out by using, for instance, the R function `.libPaths()`.

Please feel free to notify us if any dependencies are missing. But please mind that these scripts are only meant to guide you in your install process. We will gladly try to help you with this should you face problems, but we can't guarantee that we will be able to do this for every system. If possible, please refer to your IT department first for problems regarding system dependencies.

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)
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.

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
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.

3.4.2 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.

3.5 Using pre-compiled binaries for the R libraries

When you use the `install` script to install the R libraries needed by obsmon, it saves the binaries for all successfully compiled R libraries.¹⁹ Conversely, before attempting to compile any R library, the `install` script looks for appropriate binaries it may have previously compiled.²⁰

This mechanism allows installation to resume from where it stopped in case it fails or is interrupted. Another (perhaps not so obvious) advantage is that it can lead to a very significant reduction in the installation times whenever:

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

For updates, you just need to follow the recommended update approach (Section 3.2). For the other cases, you only need to run the `install` script from scratch once. Then, for subsequent installs, you may (i) copy the appropriate pre-compiled libraries directory created during the first `install` run to some accessible location in your computer(s), and (ii) pass the path to this directory in the next `install` script run via the `-bin-repo-path` command line argument. By doing so, no libraries will be unnecessarily compiled.

3.6 Advanced install options

The provided `install` script uses metadata stored in the file `.installer_local_pkg_repo/src/contrib/PACKAGES` to solve R-package dependencies as well as determine which versions of them will be installed. We use this to ensure that the installs are reproducible, which greatly simplifies support and maintenance. There may be cases, however, where one may want/need to install package versions that are different from those in the aforementioned ‘PACKAGES’ file. This can be accomplished by using either one of the following options:

- (a) Creating an `.installer_pkg_versions.txt` file

¹⁹Please use the help options in the `install` script to check where the package output files generated by the code are stored (and/or change the default location it uses).

²⁰The specific name of the final compiled-binaries directory depends on the architecture of your system. It may, for instance, look like `x86_64-redhat-linux-gnu-RedHatEnterpriseClient-7`. Binaries compiled with different `major.minor` versions of R are stored separately under this directory and are not considered to be compatible with each other.

This allows you to handpick the versions of any number of used R-packages. A typical usage case would be to select specific versions of a few packages. Packages not listed in the file will have their versions determined by the default method. You can create a template file by running `./install listdeps --lock` and then edit it (keeping the same format).

- (b) Running `./install -repos=cran` instead of just `./install`

This instructs the `install` script to ignore the `.installer_local_pkg_repo/src/contrib/PACKAGES` file and, instead, use CRAN to determine the package versions and dependencies. This is the best option if you want to install the latest versions of all packages.

It is also possible to combine these options, which may be handy in some cases.

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:²¹

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

The next sections describe the config file options for the `[general]` and `[[experiments]]` tables. The configuration of `[[multiPlots]]` and `[domain]`

²¹ Note the mandatory double brackets in `[[experiments]]` and `[[multiPlots]]`, as opposed to the single brackets used in `[general]` and `[domain]`. *Sections defined with single brackets cannot have repeated names, whereas multiple double-bracketed sections with the same name are allowed.*

will be discussed in Sections 5.4 and 5.6, respectively. 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`. *Options that have default values are optional.*

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.

- **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_vMAJOR.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 5.7.

- **configName**: Name of your config file.
Default: `" "`
Acceptable values: Any string.

The text assigned to `configName` (if any) will be printed out to the logs as `INFO` (see option `logLevel` below), and will also be displayed in the GUI (on the right-hand side of the place where the code version info is displayed).

- **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 $\max\{1, \text{ceiling}(\text{nDTGs}/\text{maxAvgQueriesPerProc})\}$ groups that will then be processed in parallel.

The optimal value for this parameter is, of course system-dependent,²² and the default **"Inf"**²³ 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: $4 \times \text{\#availableCores}$
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 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 5.4) 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.²⁴

²²In our systems, for example, we have been able to get up to $2.5\times$ speedup by setting **maxAvgQueriesPerProc=24**, but one needs to test.

²³Please interpret division by **Inf** in this context as a mathematical limit.

²⁴[Problems are known to occasionally occur in this case](#). We have indeed experienced a few.

- **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 5.7.1).
Default: **false**
Accepted values: **true** or **false**

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

This is 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:

```
[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:

- 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 be switched off/on by using the config file option `plotsEnableInteractivity` (see Section 4.1). The default behaviour for `multiPlots` (Section 5.4), on the other hand, is different, and interactivity is disabled for such plots unless otherwise specified via config file option `multiPlotsEnableInteractivity`.

5.2 Physical Units in the Plots

For the plots in the main area, it is possible to customise the physical units used for some physical quantities. You can modify these by entering the name of the

new units in the appropriate GUI text input. The new units must, of course, be compatible with the default ones.

5.3 Explore The Data Used in Your Plots

For every plot, there is also available a **Query & Data** tab where you can find the SQLite query used to fetch the data along with the fetched raw data, as well as the actual data used in the plot.

Note that:

- The tables are searchable
- You can use the arrows next to the tables' column labels to sort data
 - You can *sort by multiple columns* by pressing the **shift** key when clicking on these arrows
- You can export the tables' data as text or CSV files

5.4 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.

5.4.1 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.

- **startDate**: A date in the "YYYY-MM-DD" format or an integer number N less than or equal to zero. If the integer number format is used, 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..* The default, if **startDate** is not a negative integer and **nDays** is not passed, is 'today' – whatever day today is.
- **nDays**: A non-negative integer number. Sets the end date for the plot to **startDate** + **nDays** - 1. *Do not use this if **startDate** is a negative integer or if **endDate** is explicitly set.*
- **cycles**: Cycle number(s) in the "HH" format. E.g.: "03" (to select a single cycle) or ["00", "03", "12"] (to select multiple cycles).

5.4.2 Minimal [[multiPlot]] entry configuration

The parameters described in Section 5.4.1 are already enough for a minimal [[multiPlot]] configuration. The following definition, for instance, is a perfectly valid multiPlot entry:²⁵

```
[[ 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.²⁶ The default behaviour can thus be summarised as: *Everything is included unless otherwise specified.*

5.4.3 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 5.4.2, obsmon adopts the convention that a multiPlot will include everything that can possibly make sense within its context, unless specified otherwise.

²⁵Provided that **experiment** has a valid value. See Section 5.4.1.

²⁶Some **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.

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.

5.5 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.

5.5.1 Configuration of batch-mode plots

Batch-mode plots in `obsmon` are simply `multiPlots` activated for this type of use.²⁷ Therefore, the first step to configure a batch-mode plot is to setup a valid `multiPlot` as described in Section 5.4. 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 any²⁸ of the the following entries:
 - **enable**: Whether or not to enable the `multiPlot` for use in batch mode.
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.

²⁷In particular, `multiPlots` marked for use in batch-mode will be available as regular `multiPlots` if the GUI is used.

²⁸Or all of them, unless otherwise specified.

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.²⁹

- **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 5.4.2):

```
[[ 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"
```

or, if you want to keep it simple,

(b) `batchMode = true`

²⁹For more details, see the documentation of the `ggsave` function from the `ggplot2` R package.

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 5.4.2):

```
[[ 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.

5.5.2 Running obsmon in batch mode

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

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

5.6 Domain Geometry & Grid

Interactive maps support the use of a domain's geometry and grid. If set, these are employed so that:

- Display and domain projections match
- Initial zoom in config is based on the domain's boundaries
- Plots of the "Average Maps" type can perform grid-averages

To enable domain use, just enter valid configurations in the fields under the **Domain Geometry & Grid** tab in the GUI. The geometry and grid resolution parameters used in `obsmon` are a subset of those specified in *Harmonie*.³⁰

To set default parameters, which will be loaded in the GUI at startup, you need to add a `[domain]` section to your config file. For example, to configure a coarser version of the the `METCOOP25C` domain,³¹ you can add the following section to your config file:

```
[domain]
  nlon = 90
  nlat = 96
```

³⁰See the [Harmonie System Documentation on Model Domain](#) configuration. Mind, however, that you can set these to any valid values, not only those used in *Harmonie*.

³¹See the contents of the [Harmonie_domains.pm](#) file.

```

lonc = 16.763011639
latc = 63.489212956
lon0 = 15.0
lat0 = 63.0
gsize = 25000
lmrt = false

```

N.B.: Please pay special attention to the `nlon`, `nlat` and `gsize` parameters when copying configs from Harmonie domains. The Harmonie grids are normally much thinner than you would typically want to use in obsmon, and using them without tweaking these parameters will most likely lead to long processing times for plots involving grid averages.

5.7 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.³² 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*³³ 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.

5.7.1 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.³⁴ 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.

³²Located under the directory defined by the `cacheDir` key in the config file (see Section 4.1).

³³Obsmon 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 5.7.1).

³⁴We 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.

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.

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.

Q: I want to install a different version of R-lib X instead of the one that the `install` script is using. How do I do this?

A: See Section 3.6.

Questions about caching

Q: The popup info in the some UI menus says that cache is ongoing. What does that mean and what should I do?

A: That 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' choices. The presented values may change in such cases. Whenever applicable, you will be offered choices that combine some defaults and whatever has already been cached. 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 5.4), 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.