Habitat User Manual

For Version 2.0 Alpha

System Garden **T** +44 (0) 20 7100 6627 welcome@systemgarden.com www.systemgarden.com/habitat

Table of Contents

Introduction 1

How to Start 2

Data Sources 3

The Local Host 3

Adding Hosts 3

Adding Files 3

Adding a Repository 4

Removing Files and Hosts 5

Displaying Data 6

Selecting Rings 6

Choosing Views 6

Using Charts 7

Loading Historic Data 7

Zooming and Panning Data 8

Customisation and Preferences 9

Customising the GUI 9

Preferences 9

The Collection Agent: Clockwork 11

Starting and Stopping Clockwork 11

Data Collection 12

Performance Gathering Probes 13

Data Gathering Methods 14

Job Execution 14

Data Storage 14

Command Line Utilities 16

Introduction 16

Common Arguments 16

Data Addressing 17

habget 17

habput 18

Manual and Automatic Starting 18

Other commands 19

Data Formats 20

Upload, Download and Replication with System Garden 21

Manual Pages 22

myhabitat 22

clockwork 27

statclock 31

killclock 32

habget 33

habput 34

habrs 36

habmeth 39

habprobe 41

habconf 43

habrep 45

habedit 46

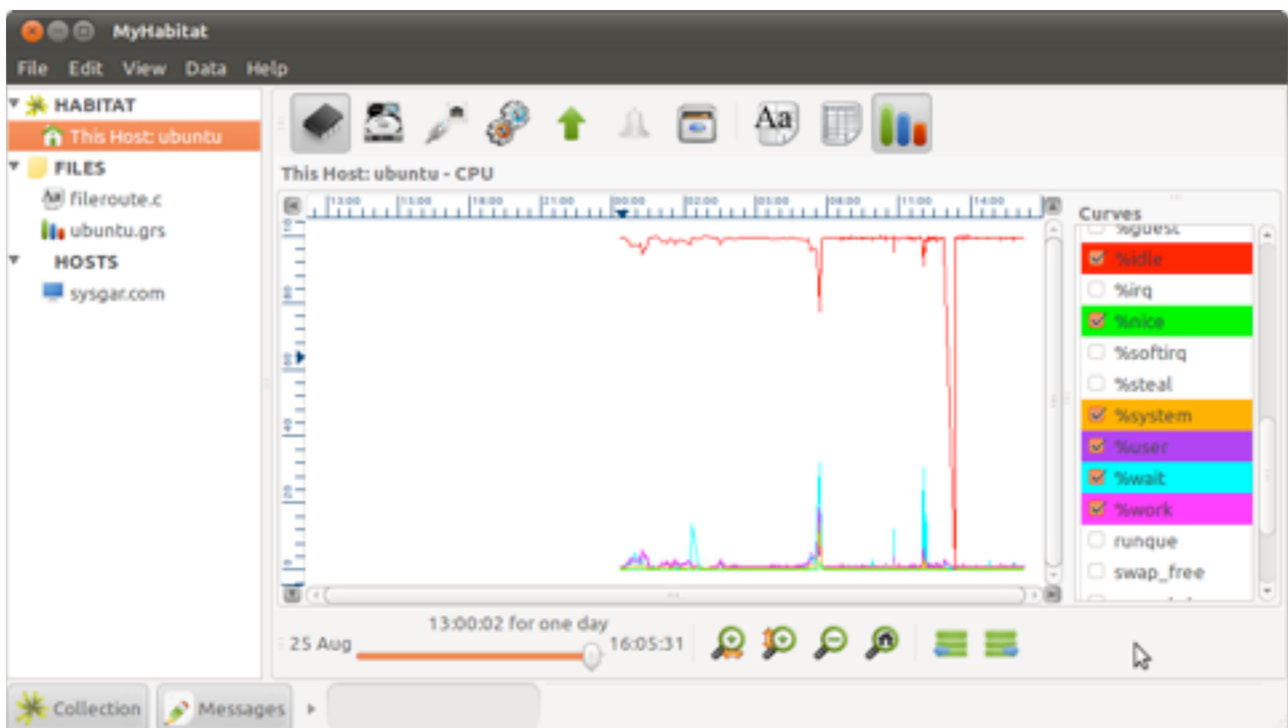
Introduction

Habitat is an extensible collector, monitor and viewer for system and application time series data. It connects to System Garden for social IT management and acts as a data gateway to upload and view streams of time series data from your organisation.

Historic data is central to the workings of habitat, with all collected information being sent to the local light weight data store and thence to the optional System Garden archive for long term storage.

A large number of measurements are taken from the system by default, including disk storage, processor utilisation and network usage. All the metrics can be examined over arbitrary time to gain a full perspective of the work the machine has done. By reducing the samples of data over time (a process called *cascading*), habitat is also able to give long term trends from only local data whilst keeping modest storage requirements.

The graphical application provided by Habitat is called *MyHabitat* and once data has been collected for a little while, you would expect to see a display like the one below.



MyHabitat showing the local machine source, two files and a remote host; the data drawn is from the local machine

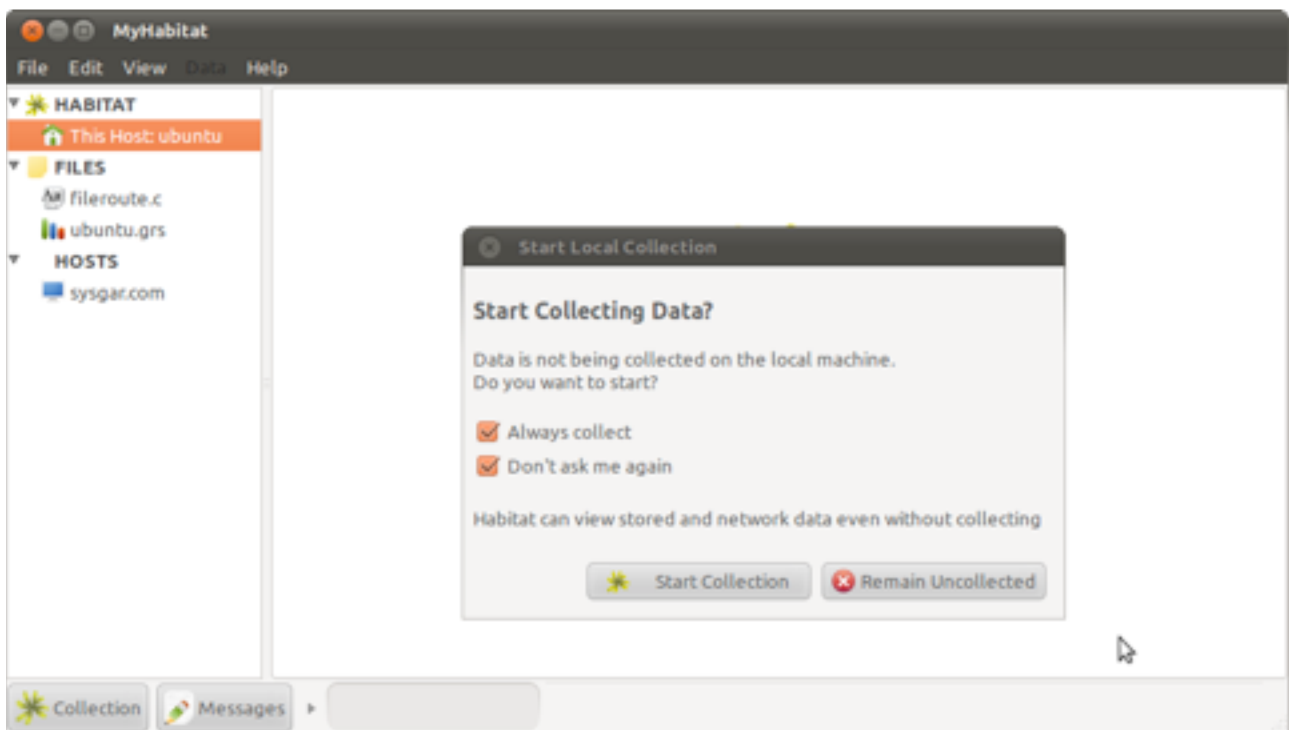
MyHabitat has a split view with data sources on the left and the selected data on the right, similar to an 'explorer' type interface.

Each source is typically a server, service or grouping of servers and services: the local host is at the top of the choice list. When selected, a default data view is shown to the right, with buttons along the top to change the data seen and how it is visualised. The buttons below help navigate and alter the appearance of the tool.

The slider at the bottom shows how much data has been loaded and can be displayed; move it to the left to load and display more data. Zooming into the chart does not affect this control.

How to Start

Just run the command *myhabitat* inside the *bin* directory or launch from the Application menu, which will start the graphical tool containing a splash screen and invite you to start the collection agent.



Click on *Start Collection* and after a while data will load in a graph format

In the current release, the default data collection rate is every 60 seconds (which can be customised) and will store this information to disk. Therefore, you will need to wait for two to three minutes for meaningful data to appear.

To start the collection daemon only with out starting the graphical application, just run *clockwork* from the *bin* directory. See the manual page *clockwork(8),* the Collection Agent section later in this manual and lastly the Habitat Administration Manual for more information about the collection agent.

Data Sources

The Local Host

The local host will be shown initially, if data is being collected, with CPU usage information over one day being the default default view.

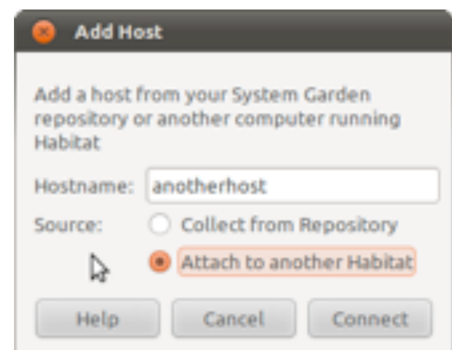
The Local Host view talks to the collection agent on its own host to obtain its data and the option will appear under **HABITAT** / This Host: *hostname* in the choice tree.

By convention and default, the collector uses a single file `var/yourhost.grs` (where *yourhost* is the name of your computer) to store its data. Inside this file are sets of data, known as ‘rings’ within Habitat, such as processor, storage, networking, processes and uptime. This is selected by choosing one of the data set buttons on the right hand pane. See Selecting Data later on.

As new data is saved, Habitat is able to show a long term history of activity on its monitored hosts, not just since the monitor was last started. As new data is collected by the agent, it will be appended to the store and the display in MyHabitat will replot.

Adding Hosts

To connect to other hosts and get their data, select *File->Host* from the menu, type the name of the machine and click 'Attach to another Habitat' as its source. If successful, an entry for that machine will appear in the choice tree, under the **HOSTS** group.



The normal method to access local and remote data is to query the agents directly on each monitored machine. The collection agent (*clockwork*) implements a network server to satisfy these queries. When called, the primary file for that host is used to return the results.

As with local host as a source, the remote host will contain multiple data sets (rings) which are selected using the ring selection buttons from the right hand pane. It will also poll the remote host periodically and update the display when new data is available.

Adding Files

Add files to MyHabitat by selecting *File->Open...* from the menu bar and selecting the data you wish to open. Several formats are supported and MyHabitat attempts to detect the format using the file

extension and the contents. If the auto detection does not work, you can select the file type in the file choose dialogue.

Habitat generates data files that can be used later, for example when captured from a benchmark session. A GRS files is one such format and supports a secondary data set (ring) level internally. Conventional CSV files are also supported and can be displayed as a table or chart. If the file can't be read in a table structure, then it will be treated as a text file and displayed without formatting.

The supported formats are as follows, and are discussed in greater detail later in the document:

GRS	Habitat ringstore format, GDBM flavour
FHA	Fat Headed Array (discussed later)
CSV	Comma Separated Values
TSV	Tab Separated Values
PSV	Pipe (|) Separated Values
SSV	Space Separated Values

In a standard configuration, the Habitat collection agent (*clockwork)* stores data in a *ringstore* structure (see GRS above), which is held in a single file. The file is called `hostname.grs` and is held in *var* in the application directory (for the .tar distributions) or /var/lib/habitat (for the RPM distributions).

Both previously recorded and live .GRS files can be opened and viewed.

Individual users may also collect customised data for their own use, which will not be stored in the main system *ringstore* file. Typically, they will use this data in addition to the central information by mounting both files within a visualisation tool such as *myhabitat.*

The main system file is also used to provide peer data access and data replication (see below).

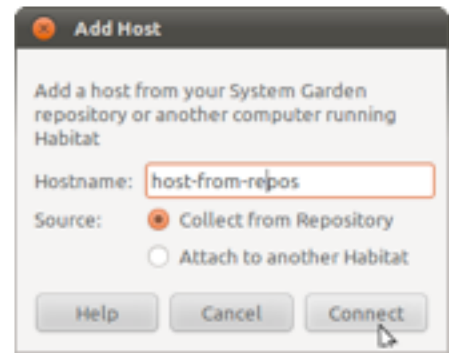
Adding a Repository

Using the standard configuration, the system *ringstore* file will grow to around 5 MBytes; more if additional data is collected or retention periods are extended. Older data is averaged down to a lower frequency than the original collected to save space.

To keep more data and to share with the rest of your organisation, a remote repository may be used to archive data and can be used as a bulk set of data sources within *myhabitat.* Such a repository is provided by System Garden, which provides social IT management (http://systemgarden.com to sign up).

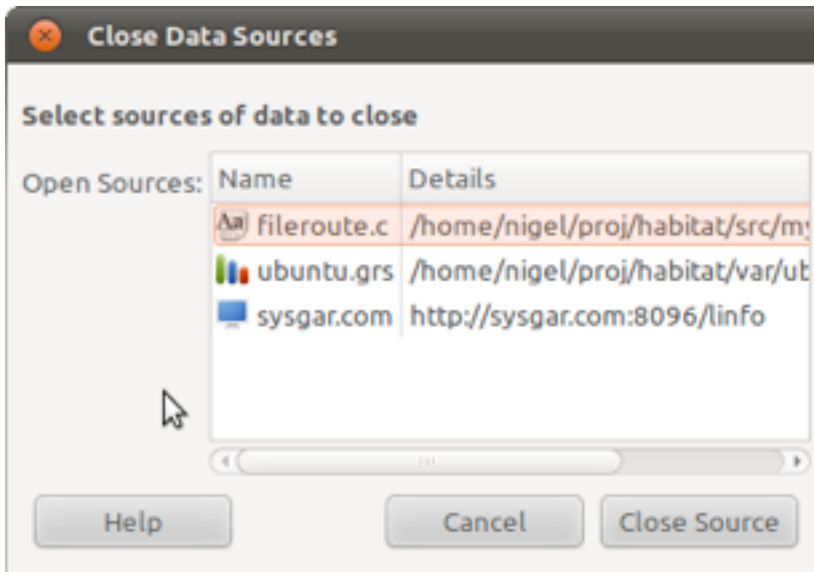
Once your account has been configured into myhabitat, the repository will appear in the choice tree under the **REPOSITORY** group. This allows the user to browse to data sources using group relationship information supplied to System Garden. For example, a British finance server may appear under the path *REPOSITORY->Finance->London->hostname*.

If you don't know the group ownership of a host, it can be sourced directly from *FIle->Host* option in the menu. Type the host name in the pop-up window and keep the repository source button highlighted. The host will appear under the **HOSTS** group in the choice tree.



Removing Files and Hosts

Remove a file or host from *MyHabitat* by clicking on *File->Close...* from the menu bar and selecting from popup window.



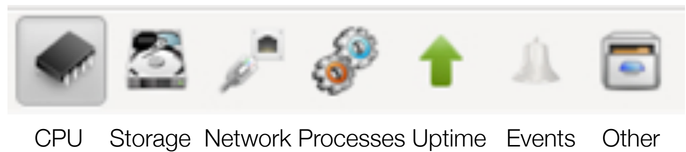
This will remove the data source from the choice tree.

Alternatively, click with the right mouse button over the source you wish to remove and choose *Close.* You can not remove the local host choice or the repository.

Displaying Data

Selecting Rings

Habitat can collect and store practically any time series tabular data. Out of the box, the following are collected and buttons for each are shown in the top viewing row in MyHabitat.



CPU	System and processor statistics
Storage	Capacity and performance of storage
Network	Networking statistics
Processes	Process table over time (potentially filtered)
Uptime	How long the machine has been running
Events	Local detection of patterns found or data thresholds crossed
Other	Menu of all the other time series data. In habitat jargon, these are also called 'rings' (as they are ring buffers)

If the ring is present in the source data, the corresponding button will be illuminated. If missing, than the button is deactivated. Sources that have no rings, for example CSV files, will not activate any buttons.

Data sets or rings of data that do not correspond to the standard set can be selected by clocking on the ‘Other’ icon (the filing cabinet draw) and choosing the item from its pull-down menu.

Each ring holds many attributes, such as processor utilisation (%cpu or %work) and these can be seen in a table or selected in a chart.

See the manual pages habget(1), habput(1) for command line extraction and import of data.

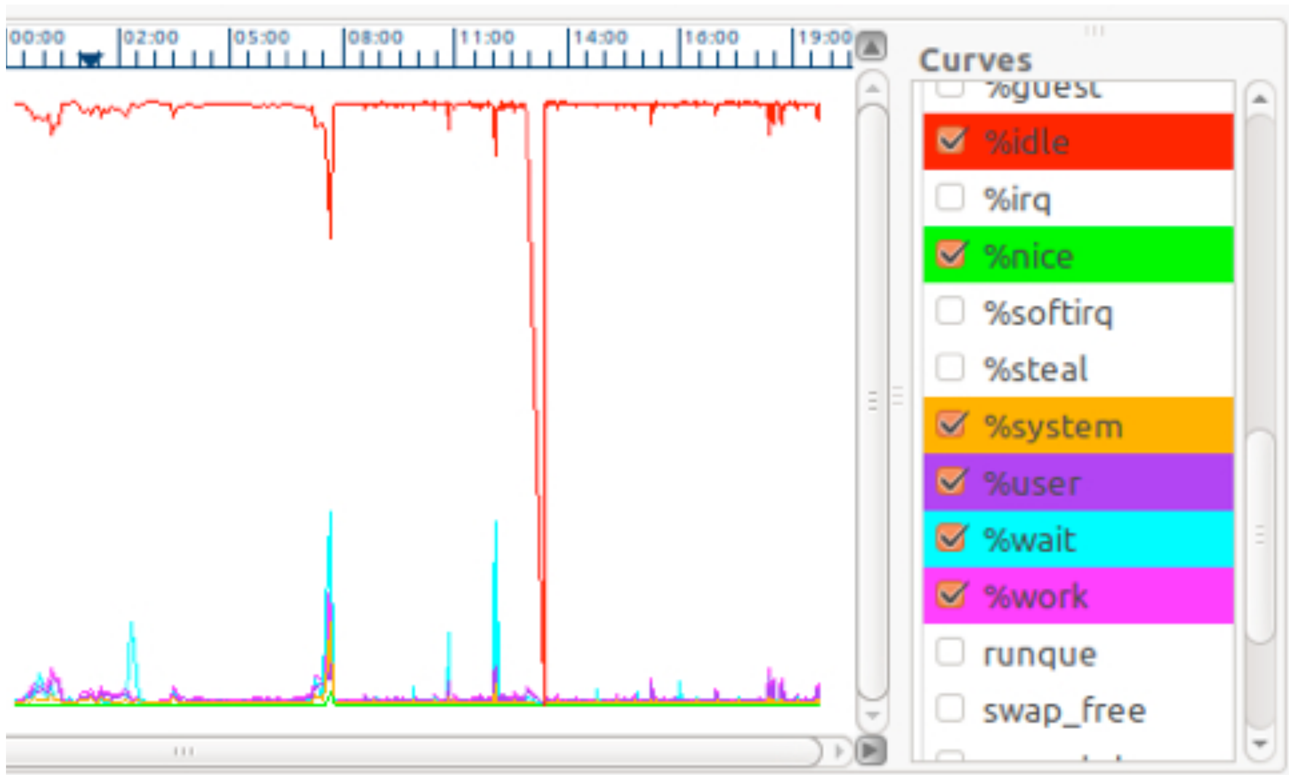
Choosing Views

Data can be viewed in three ways: chart, table and plain text, each of which is represented by buttons in the right hand pane. Providing that buttons are active and illuminated, then you can switch the view of the data between each visualisation.

During the display process and from the hints when the data source is loaded, MyHabitat attempts to check the format of the data to see if can be structured into a table. If it can, then the appropriate buttons are activated.

Using Charts

When a chart view is selected, the right hand pane is altered to a display similar to the one below. A multi-curve chart with scroll bars to pan, date stamps on the horizontal axis, values on the vertical axis and coloured list of curves on the right



The curves and the corresponding list entires share the same colour. Clicking on a curve's tick box will draw or remove the curve in the chart. MyHabitat has a list of default curves to select when a new data set is plotted and this can be customsed.

Loading Historic Data

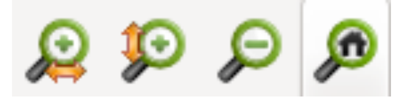
MyHabitat loads and caches a small amount of data initially to minimise the system load and keep the timings short. Typically this is up to one day, although this can be altered using *Edit->Preferences* from the menu bar.

The amount of loaded data can be increased by using the time slider at the bottom of the right hand pane. The slider shows the limits of available data from the source: in the example below, it starts on the 25th August, 13:00 today is the first data shown in the chart and ends at 20:28.

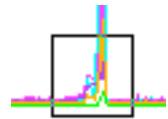
Moving the slider to the left fetches the missing data and replots the chart, caching locally to increase performance.

Zooming and Panning Data

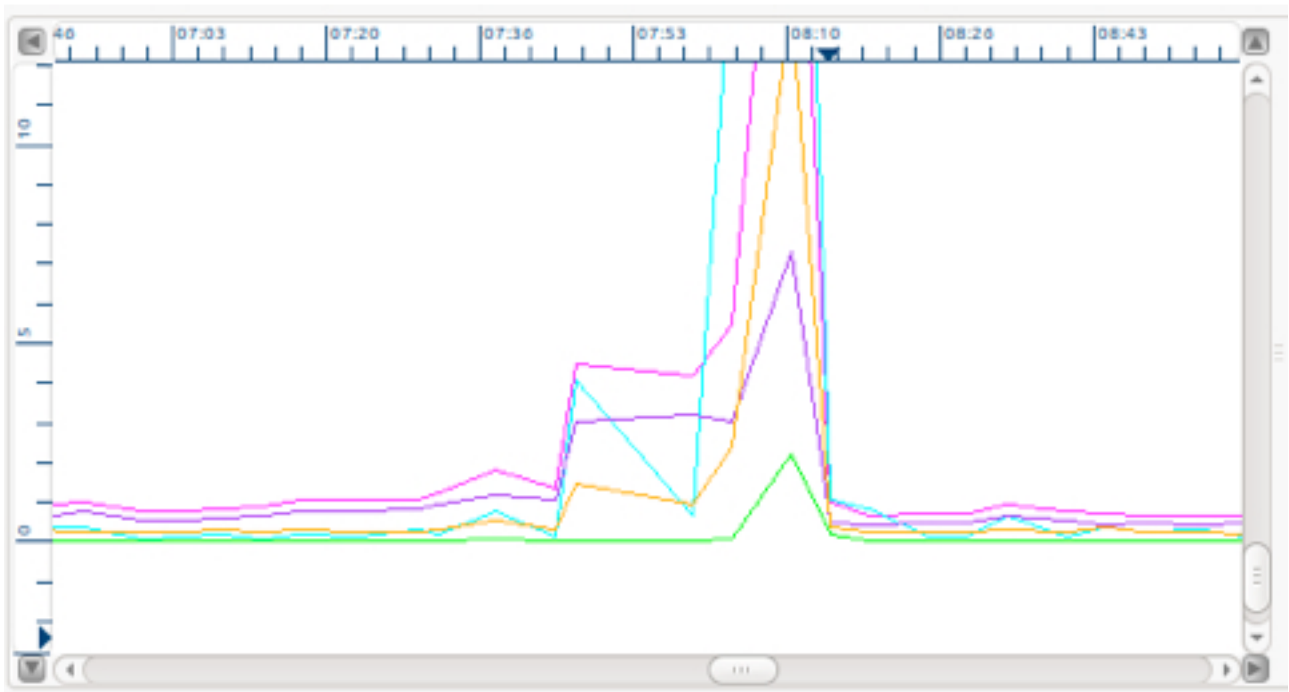
As Habitat is able to show a lot of data in a single view, it is common to zoom in to particular times to see the detail more clearly.



Using the zoom buttons allows the chart to be increased horizontally to expand the time scale, vertically to expand the value domain or both. To return to normal, there is a zoom out function and a 'home' function to rest the view to normal.



A second way to zoom, especially useful to see small areas is to drag a box over the area of interest with the left mouse button. Release the button when drawn and click inside to enlarge the horizontal and vertical areas.

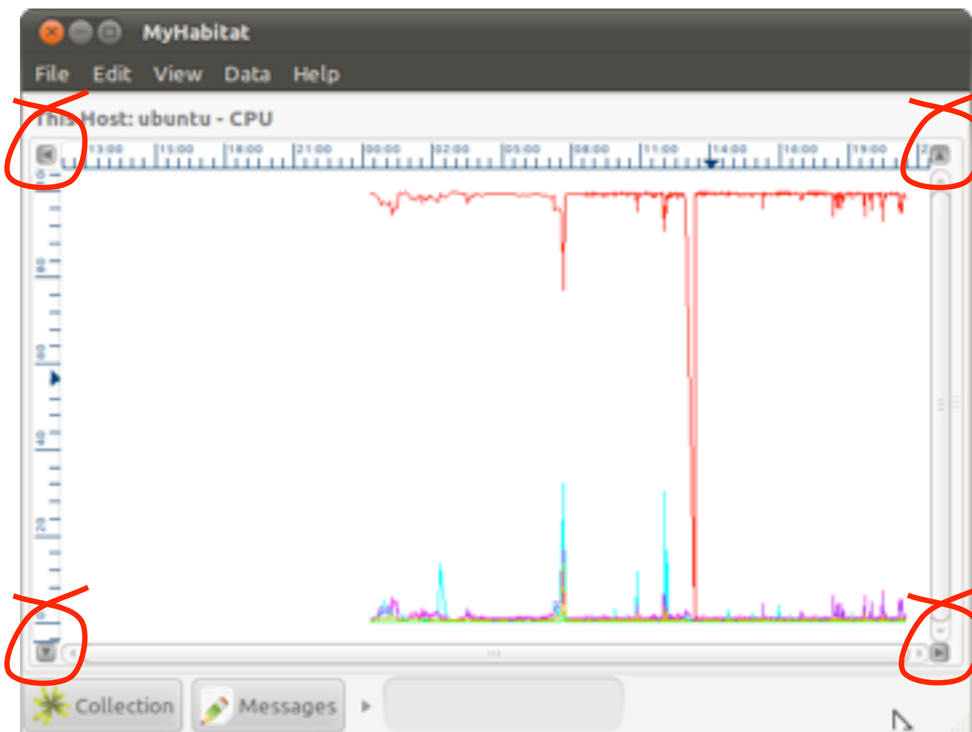


Once zoomed, move around the chart by using the scroll bars.

Customisation and Preferences

Customising the GUI

MyHabitat can be adapted to remove panels of the controls that change the visualisation, leaving just the chart. The panels are toggled using the menu items *View->Panels->Choices, View->Panels->Curves, View->Panels->Toolbars*. Another method is the push buttons at each corner of the chart which will toggle the panels. The image below shows the stripped off panels with the buttons circled in red.



Preferences

Selecting the menu option *Edit->Preferences* will allow you to change the behavior of *MyHabitat*.

Not all options are available in a self documenting GUI form, however, and need to be set from the configuration system which is common across all Habitat tools. You can edit the user level configuration file from *MyHabitat* by selecting *Edit->User Configuration...* or with your preferred text editor on the file `~/.habrc` .

(Configuration may also be set on the command line [see -C and -c switches in the manual pages of each tool] and by administrators at multiple levels, which may account for behaviour not requested by a user. For more information of the global configuration of Habitat and how to control it, see the Administration Manual.)

The first line in `~/.habrc` is the *magic number* (actually a string) that identifies the format: it must be set to **habitat 1** to show habitat its version. The remainder of the file contains settings in the form of simple assignments against property names. The values may be lists, single values and an implied positive or negative.

The following are the possible formats accepted by the configuration:

`# blah blah blah`	Comments are introduced with '#' and finish at the end of the line; they may follow any directive
`Prop`	Prop is set to true (1)
`+Prop`	Prop is set to -1
`Prop val`	Prop is set to val
`Prop=val`	Prop is set to val
`Prop val1 val2 val3`	Prop is set to the array (val1 val2 val3)

Care should be taken when manually changing the file, as the *MyHabitat* application will also write to the file when exiting or carrying out configuration tasks. To be safe, it is advisable to edit the file when *MyHabitat* is not running to use the inbuilt editor. The application will only update the lines that match the property being updated, leaving everything else alone (comments, user settings, etc).

All the parts of Habitat share the same configuration mechanism and honour the same command line switches. In addition to `~/.habrc` providing user level configuration, the file `etc/habitat.conf` is used to configure at an application level and will potentially change on each release. Site and global configuration can be achieved by setting up configuration files in central locations and instructing Habitat to configure remotely in addition to local settings. HTTP servers are ideal for this; see the manual page habconf(5) and the Administration Manual.

The Collection Agent: *Clockwork*

Starting and Stopping *Clockwork*

Habitat consists of a collection agent for data gathering called *clockwork* and a set of front ends for display and extraction of data. *MyHabitat*, is the main user interface and can be used to easily control local collection. It will check *clockwork* is running each time it starts up and will ask to start collection if it isn't already in place (although this is configurable). The menu option *Data->Collection...* or the *Collection* button on the status line gives you control over local agent.

Clockwork can also be started on the command line independently of *MyHabitat*. Run

```
$ clockwork
```

on its own and the agent will start silently as a background daemon.

To see if *clockwork* is running, run the status command

```
$ statclock
Clockwork process 4616 is running
  was started at 23-Nov-11 01:24:31 PM, user nigel, on /dev/pts/4
```

To stop the daemon, run

```
$ killclock
Stopping Clockwork
  pid 4616, user nigel and started on /dev/pts/4 at 23-Nov-11 01:24:31 PM
Stopped
```

To collect data on several machines without the GUI, just run *clockwork* on each, which as a daemon will put itself into the background and by default, log its own errors and warnings into the data store for later (see Data Sources for a data file description).

If there is a problem launching *clockwork*, starting

```
bin/clockwork -d
```

will cause diagnostic messages to be sent to stderr, including completion of collection jobs. If the failure is not obvious, send this output to support@systemgarden.com. If there are still problems, an exhaustive set of debug messages can be obtained with

```
bin/clockwork -D
```

This places clockwork into developer debug mode.

Data Collection

Data can be **pulled** from a source by *clockwork* or can have data **pushed** directly into a data store from the command line or a programming interface.

In order to pull data, *clockwork* has a table of jobs that are executed at regular intervals (described later). By default these jobs run *probes* that collect all manner of system information and send it to a data stream called a *route.* These *routes* usually address local persistent data storage or remote network storage, so that information sent to a *route* will end up being saved.

Data can also be pushed on to routes independently of clockwork, by using an API or the command line tool *habput*. This will take text, in the format of a Fat Headed Array (FHA), and will send it to the route address specified.

A time series of data is built up by repeatedly storing a sequence of tables. Each table defines data at a period in time and is assigned a time stamp, sequence number and duration. This is often expressed in a tabular context by using the special columns *_time, _seq* and *_dur.*

A table of values are used for each sample, so that multiple instances may be expressed with out the use of excessive columns. For example, if habitat gathers information about storage, the FHA may look like the following.

```
_seq  _time       id    mount        kread  kwritten  rios   wios
--
200   1107372138  hda1  /            1.98   15.85     15.09  12779.53
200   1107372138  hda2  /usr         6.78   0.00      60.76  0.00
200   1107372138  hda3  /mnt/windows 0.00   0.00      0.00   0.00
201   1107372143  hda1  /            2.03   9.20      30.94  459.53
201   1107372143  hda2  /usr         57.55  0.52      4.13   0.07
201   1107372143  hda3  /mnt/windows 0.00   0.00      0.00   0.00
```

In the example above, there are two sets of three lines, with each set sharing the same sequence number: 200 and 201. These rows belong to the same sample, share the same time stamp but have different values for id, which is the instance key. In the case of storage, the instance key is a subset of the device name. Thus, to get a time series for a particular disk (say hda1), select the rows with id=hda1 and sort on _seq.

Performance Gathering Probes

A probe is a small piece of code inside *clockwork* that extracts data from a running system and is called regularly by the job system. Its job is to sample data, process it and potentially repackage for data storage or display. The output is a single table per invocation (called a Fat Headed Array when represented as text) and an error or log stream.

Probes can also act as filters on a queue of data as an input in order to perform mathematics (such as averaging) or other manipulation. In this way, a job configuration can describe a pipeline of collection and manipulation jobs state-full data stored.

You can run a probe independently of *clockwork* on the command line and see its results on standard output. For example, the following will output the system performance probe on linux; only the first six columns are included for brevity (there are 35 columns) [The info line is staggered over two lines to readability].

```
$ habprobe sys
load1      load5      load15     runque     nprocs     lastproc
1 minute load average      15 minute load average num      num of procs
           5 minute load average          num runnable procs          last proc run  info
""         ""         ""         ""         ""         ""         key
4          4          4          ""         ""         ""         max
""         ""         ""         ""         ""         ""         name
abs        abs        abs        abs        abs        abs        sense
nano       nano       nano       u32        u32        u32        type
--
0.08       0.03       0.05       1          335        5142
```

The following are the probes available for use in *clockwork* and in the command *habprobe,* all output is in table format.

intr	Interrupt statistics
io	I/O data, storage and disk statistics
names	Symbolic data from the kernel
ps	Processes
sys	System data, including CPU and memory statistics
timer	Timer data
up	Uptime data, how long the system has been up
down	Down time data, calculated from **up** probe and can give a view on outages
net	Network device statistics

To confirm the probe list, run `habprobe` on its own or look at the manual page for *habprobe(1)* to see all the other probes that are available.

Data Gathering Methods

In addition to running probes to gather data (the *probe* method), jobs can use other methods to extract data, as follows

exec	Direct submission to exec(2)
sh	Test submit command line to sh(1)
snap	Take a snapshot of a route
tstamp	Timestamp in seconds since 1/1/1970 00:00:00
sample	Sample tables from a route, carries out a mathematical process and produce a single table as a result
pattern	Match patterns on groups of routes to raise events
event	Process event queues to carry out instructions
replicate	Replicate rings to and from a repository
probe	Extract data from the built in probes (see above and habprobe(1))

Where as all the probes produce tables, the methods are not obliged to. The *probe* method is used to run probes (see above).

The utility *habmeth* runs the methods on the command line so they can be seen outside of clockwork.

Job Execution

Clockwork uses methods and probes to sample data and build streams of data which get stored locally. See the manual pages for *habprobe*(1), *habmeth*(1) and *clockwork*(8) for details of the configuration.

There are several standard job files in the Habitat distribution, prepared for different collection scenarios (see `lib/jobs.*`). Select them by running *clockwork* with `-j prefab` where prefab is one of the following

norm	Normal job file, the current default sampling once every 60 seconds
normrep	Normal job file, with 24 hour replication to the defined repository
quick	Higher frequency collection, sampling at 10 seconds

Alternately, job tables can be customised into a file or held in a route and *clockwork* started with the switch `-J route`. Route should be specified using the pseudo-URL route syntax.

In *MyHabitat* you can see the current job file used by *clockwork* by selecting *Edit->Collection...*

Data Storage

Each job line has a method, command and set of arguments. Two outputs are produced from each job, the result stream and an error stream, analogous to standard output and standard error. Each output is sent to a stream defined by a route specification.

Data in Habitat is *time series* in nature, whether it is tabular or plain text. The primary storage method is called a *ringstore,* after the technical description of the data structure, *a ring buffer*. Examples are log and uptime data which are a time series set of tables, time stamped and sequenced to make ordered and unique. The rings have descriptions and a defined maximum length (including infinite) which can be customised. Data is removed when it exceeds the retained limit. The command line utility *habrs* or *MyHabitat* directly can be used to look at this data. Following normal operating system conventions, the data file will have the permissions and ownership of the creating user.

Command Line Utilities

Introduction

Other commands exists to complement the GUI and data collector.

clockwork	The collection agent, runs in the background as a daemon
killclock	Stop the clockwork daemon
statclock	Print information about clockwork
habrs	Command line interface to ringstore storage & admin
habget	Get data from a route
habput	Send data to a route
habedit	Edit a route or ringstore (useful for configuring)
habmeth	Run a clockwork method manually
habprobe	Run one of the built-in data gathering probe manually
habrep	Synchronise data with the repository

A number of command line utilities are provided in the standard Habitat distribution. These address getting data in and out of habitat's various storage systems, maintaining *ringstore* files and being able to get data or run the suite's methods on an ad-hoc basis.

This section describes each tool and their function. Their manual pages are held separately in the appendix.

Common Arguments

Where possible, all the command line utilities share a common set of arguments. They are:-

`-c croute`	Append user configuration data from *croute* (*route* addressing format), rather than the default file `~/.habrc`. For example, `-c [/cf.dat file:cf.dat]` would load configuration from `cf.dat`.
`-C cfcmd`	Append a list of configuration directives from *cfcmd*, separated by semicolons. For example, `-C "nmalloc=1;dummy=6"` would set the configuration variables *nmalloc* to 1 and *dummy* to 6
`-d`	Place command in diagnostic mode, giving an additional level of logging and sending the text to *stderr* rather than default or configured destinations. Used for *clockwork* in daemon mode, will send output to the controlling terminal

`-D`	Place command in debug mode. As `-d` above but generating a great deal more information, designed to be used in conjunction with the source code. Also overrides normal outputs and will send the text to *stderr*. Used for *clockwork* in daemon mode, will send output to the controlling terminal
`-e fmt`	Change the logging output to one of eight preset alternative formats, some showing additional information. *Fmt* must be in the range 0-7, with format 3 being concise but useful. The formats are:- **0.** everything!! time, severity, path, process ids, file, function, line, origin, code, text **1.** upper case severity letter, text **2.** severity, text **3.** justified severity, text, file, function, line **4.** upper case severity letter, short date time, binary name, file, function, line, text **5.** time, severity, binary path, pid, file, function, line, code, text **6.** long time, epoch time, severity, binary path, pid, tid, file, function, line, origin, code, text **7.** justified severity text, justified file, line, function, text
`-h`	Print a help message to *stdout* and exit
`-v`	Print the version to *stdout* and exit

If a command does not work as expected, the user may be directed to run it with the `-d` or `-D` flags to help diagnose the problem.

Data Addressing

Most of the commands use *route* formats to address their data. The format of a *route* is similar to URLs and has been extended to cope with the formats of data storage used in habitat. It is fully explained in the concept section at the beginning of this guide.

habget

The utility *habget* opens a route specified on the command line, and redirects the output to *stdout.* As an example, the following outputs the data collected by the system probe (the data is collected at a 60 second interval):

```
$ habget rs:/var/lib/habitat/myhost.rs,sys,60
load1      load5      load15     runque     nprocs     lastproc
1 min load  5 min load  15 min load  num run procs  num procs  last proc run     info
4          4          4          ""         ""         ""             max
abs        abs        abs        abs        abs        abs            sense
nano       nano       nano       u32        u32        u32            type
--
0.00       0.00       0.04       1          142        5895
```

The data returned has been shortened & edited for brevity. In reality, the info strings are longer and there are many more columns. The six columns in the example have rows info, max, sense and type.

The most recent sample is returned, which in the example above is a single line. To get more data, use the additional *route* qualifiers `t=` or `s=`, which explicitly specify the time or sequence. For example, to return everything in the *sys,60* ring, use

```
rs:/var/lib/habitat/myhost.rs,sys,60,s=0-
```

Which will return all the records from sequence 0 on to the end. When explicit time and sequence addresses are used, the output will be augmented with a sequence, time and duration column (*_seq, _time* and *_dur*). All *routes* are valid addresses, including *http:* and *sqlrs:*.

habput

The utility *habput* inserts data onto a *route* for storage in a *ringstore* or *SQLringstore*. The following example reads the table headed `tom dick harry` and stores it in the ring called *myring* with a 0 second duration contained in the *ringstore* file `myfile.rs`.:

```
$ habput rs:myfile.rs,myring,0 <<END
tom  dick harry
--
1    2    3
4    5    6
END
```

A ring of 0 duration is the convention for data with irregular frequency. Sending the data to a *ringstore* causes it to be scanned for a table structure and if passed, will append the data to the ring.

If there is an error in the format of the table or in the address syntax, then the ring will generate an error and no data will be stored. Appended data is given an ascending sequence number and will be data stamped.

Please note: rings may be implicitly created with this utility, in which case a default size and description will be given (which can be overridden with the `-s` and `-t` switches). The default ring is circular with 1,000 slots; when sequence 1,001 is appended, the oldest will be lost. To create a queue rather than a ring buffer, use `-s 0`.

Rings may be manipulated (size, name, description, etc) using the utility *habrs*, which is described in the Administration manual.

Manual and Automatic Starting

The commands *clockwork* and *killclock* are used to start and stop the collection daemon on each machine. (Unless habitat is installed as a system service, see `/etc/init.d/habitat` below). To start

a shared collection service for the system, run *clockwork* on its own. It will become a background daemon process, requiring to be stopped with the program *killclock*, also with no argument.

If a collection process is not running when *myhabitat* is started, then a pop-up will ask if you wish to start one (see sections above). In this case, the data file will be owned by the starting user although the network service will still be available.

If a user wishes to run their own data collection in addition to system collection, they can do so by providing a custom job table to *clockwork*, like so:

```
clockwork -j jobroute
```

Jobroute must be a *route* but typically is a file created for the specific situation. Job tables are described in the Administration Manual, and describe the probes to run, their frequency and where their storage is located. The `-j` flag does not daemonise and stays attach to the controlling terminal, so that it may be controlled and stopped like a normal shell level process.

```
/etc/init.d/habitat
```

When habitat is fully installed as a system service or daemon, a single script manages the starting and stopping of clockwork. It is automatically run on on the machine's start-up.

As root, one can manually control clockwork collection using the conventional command syntax:

```
/etc/init.d/habitat [ start | stop | status ]
```

If this is the case in your installation, you do not need to manually start a personal instance of *clockwork* unless you wish to run specific jobs.

This command is covered in greater depth in the Administration manual.

Other commands

The remaining commands are covered by the Administration Manual:-

habedit	Edits configuration tables within ringstores
habmeth	Runs *clockwork* methods from the command line
habprobe	Runs built-in data collection probes from the command line
habrep	Forces a replication cycle to take place
habrs	Interactive ringstore utility, allowing administration of the data held in *ringstore* files

Data Formats

Habitat supports a number of data formats in the import and export of data. Chief of these is the *Fat Headed Array* (FHA), which a variation on *Comma Separated Values* (CSV).

FHA uses a **tab separator** between columns and uses optional quotes (“) on values when needing to embed tabs. The header line is expanded to include additional sets of information for each column, known as *info rows,* which are printed over several lines below the single row header. An example is shown below

```
tom     dick    harry
Thomas  Richard Harold  first_name
Smith   Brown   Bloggs  last_name
------  ------- ------
1       2       3
4       5       6
7       8       9
```


The single row of column names and the zero or more *info* rows, form an extended header block that is terminated by two or more dashes (--) on a single line. In the example above, the dashes have been extended to form a ruler line of the width of each column, similar to the convention of a SQL table display. Following the ruler is the tabulated array of values, which may be any sequence of characters excluding tab (\t) and the double quote (“). In summary, The fat headed array must have the same number of columns through out each row, but will be one more for the second and successive headers rows (the *info rows*).

Optionally, other formats can be used: CSV for comma (,) separated values, PSV for pipe (|) separated values, TSV for tab separated values and SSV for space ( ) separated values. None of these formats contain a *fat* header, which contain info rows and a header delimiter.

Upload, Download and Replication with System Garden

Use of System Garden's repository is not supported in the Alpha series of Habitat releases.

Manual Pages

myhabitat

Gtk+ Graphical interface to Habitat

SYNTAX

myhabitat [*-c <purl>*] [*-C <cfcmd>*] [*-e <fmt>*] [*-dDhsv*]

DESCRIPTION

The standard graphical interface for Habitat, able to view locally captured data, remote Habitat instances and repository data provided by Harvest.

When the tool starts, a check is made for the existence of the local collection agent, clockwork(8). If it is not running, the user is asked if they wish to run it and if it should autostarted as a daemon in the future.

In appearance, clockwork resembles that of a file manager, with choices on the left and visualisation on the right. If files or other data sources have been opened before, then their re-opening is attempted by **MyHabitat** and will be placed under the **FILES** and **HOSTS** nodes in the choice tree.

See the DATA SOURCES section for details of the data that can be viewed, NAVIGATION for how to interpret the data structures and VISUALISATION for how to examine the data once displayed.

On Linux, the GUI requires Xwindows; use other front ends or command line tools if you do not have that facility.

OPTIONS

-c <purl>	Append user configuration data from the route <purl>, rather than the default file ~/.habrc.
-C <cfcmd>	Append a list of configuration directives from <cfcmd>, separated by semicolons.
-d	Place **MyHabitat** in diagnostic mode, giving an additional level of logging and sending the text to stderr rather than the default or configured destinations. In daemon mode, will send output to the controlling terminal.
-D	Place **MyHabitat** in debug mode. As -d above but generating a great deal more information, designed to be used in conjunction with the source code. Also overrides normal outputs and will send the text to stderr. In daemon mode, will send output to the controlling terminal.
-e <fmt>	Change the logging output to one of eight preset alternative formats, some showing additional information. <fmt> must be 0-7. See LOGGING below.
-h	Print a help message to stdout and exit

-v	Print the version to stdout and exit
-s	Run in safe mode, which prevents myhabitat automatically loading data from files or over the network from peer machines or the repository. Use if myhabitat start up time is excessively long. Once started, all data resourcese can be loaded manually.

DATA SOURCES

Currently, data can be obtained from four types of sources:-

Local Host	Data collected from same machine running **MyHabitat** appears under **HABITAT** in the choice tree as 'This Host: *hostname*'. If data is not being collected locally, you will be asked if you wish to start on initiation. It is not essential to collect data, however, and the requests to do so can be repressed, making **MyHabitat** a viewer for the data of others.
File	Rich data is stored in format known as a *ringstore*, which is a structured format using GDBM. It allows multiple rings to be stored in single container and can contain live or historic data. Other formats include CSV files and an enhanced version known as 'Fat Headed Array' (FHA). Open them with *File->Open* or *^O* and use the file chooser. The file will appear under **FILES** in the choice tree.
Repository	Replicated data once centralised will appear under the **REPOSITORY** node in the choice tree if configured with account details. Select *Edit->Harvest...* or *Edit->Repository...* to configure. The Habitat Administration Guide discusses how to handle repository accounts in larger installations.
Network Data	Data for an individual machine can be read from the repository or a peer Habitat instance on another host (using the peer's clockwork daemon). Select *File->Host* or *^H*, type in the hostname and pick repository or host as a source. Your selection will appear under **HOSTS** in the choice tree.

Files and hosts can be removed by using the menu item *File->Close...* (*^C*) and selecting from a list or by clicking the right mouse button over the tree entry.

NAVIGATION

The choice pane on the left holds actively attached or remembered valid data sources, similar to a file browser. It is divided in to a number of sections to categorise the sources and help in navigation.

HABITAT	Data from the local machine or the operation of the collection agent
FILES	Files that have been opened by you and if they exist, files that have been remembered from previous sessions
HOSTS	Remote Habitat hosts attached now or in the past, providing that they are still running the collection agent (clockwork). Can be peer instances of Habitat or be attached directly from a repository (see below)
REPOSITORY	A central collection of performance and statistical data, able to be browsed in a hierarchical organisation tree. To get to a specific machine, one needs to know its organisational location and traverse to it in the tree. Whilst this aids browsing, one may wish to use the *File->Host* option to go directly to a machine. Habitat's collction agent will typically send local collection data to a repository for long term storage and analysis, saving excessive load on machines that collect data. Harvest (http://www.systemgarden.com/harvest) is typically used as a repository.

Opening the data source trees will display a default view or a summary and various controls to navigate the view as needed.

VISUALISATION

The right hand section of the window is used for visualisation. Its major uses are for charting and displaying tables.

Once the data source is selected from the choice pane, the source is queried to see what data it has to chart. In Habitat, these sets of data within a source are called a 'ring' (after the term ring buffer) and each is assigned a button which is displayed in the visualisation pane.:-

CPU	Processor data and other system wide information. Habitat stores these in the ring *sys*.
Storage	Disk, network (NAS) and block (SAN) storage are keyed by the device name and held in the ring *io*. Habitat shows the performance, capacity and mount point of file systems where applicable in a single collection
Network	Statistics from the network devices, keyed by the device name; Habitat stores these in the *net* ring.
Processes	Full details available from your operating system about 'interesting' processes. By default, these are ones that exceed a low threshold of utilisation, which indicates that they are more than trivially active. Habitat stores this data in the ring *ps*, indexed by the process ID and the thresholds are conventionally stored in the file *$HAB/lib/ps.conf*. A threshold is applied to reduce the amount of data from processes but at the risk of losing a complete picture. This can be customised by changing the *ps.conf* configuration file
Uptime	Accumulated up and down time of the system, stored in the ring *up*
Events	Events raised by Habitat when executing local pattern matching are stored in the ring *event*.
Other	Other is a menu button that holds all other rings in a pull down menu. Selecting one of these will change the display to that data, but the ring names are not changed: a ring name of Ifred will have an entry called fred.

The standard sets of data, such as CPU (*sys*) and Storage (*io*) have default curves that are displayed when the graph is first drawn. The list of curves down the right hand side are buttons used to draw or remove data on the graph. When drawn, the button changes colour to that of of the curve displayed.

A set of buttons change how the seleted data is seen. Options are:-

Text	Data is treated as text rather than structured tabular data. This is useful when the data is unstructured, not suitable to chart and does not parse.
Table	Data is shown as a structured table, suitable for *CSV* for *FHA* files in addition to Habitat's *ringstores*
Chart	Data is displayed as a line chart with a curve selection check list to the side. Clicking the check list will draw and remove curves from the shared charting space. All data is shown in a line chart style with a set of buttons below to zooms in and out of the displayed chart, and a set of scroll bars which can be used to panned the data

When charting, the visualisation section is divided into additional parts. The greatest is used for the graph itself, with other areas being used for visualisation type, curve selection, zooming and data held. If the data is multi-instance, such as with multiple disks, then a further area is added to control the number of instance graphs being displayed.

A time slider shows the data that is available for this ring at the source and how much of it is currently displayed. Moving the slider will load additional data and redraw more data in the display.

Data is cached in MyHabitat to minimise the number of fetches to the data source. When data is fetched, whole records are collected (row oriented rather than column oriented) which means curve selection is fast at the expense of larger data fetches.

Whilst the largest amount of data displayed is selected from the choice tree, it is possible to 'zoom-in' to particular times very easily using the graph. There are two methods: either drag the mouse of the area of interest, creating a rectangle and click the left button inside or use the x and y axis zoom buttons from the **Zoom & Scale** area. The display shows the enlarged view and changes the scale the x & y rulers. The time ruler is changes mode to show the most useful feedback of time at that scale. To move back and forth along time, move the horizontal scrollbar. To zoom out, either click the right mouse button over the graph or use the zoom-out button in the **Zoom & Scale** area.

MENU

The File menu adds and removes file and other data sources to the choice tree. It also contains import and export routines to convert between native datastores and plain text, such as csv and tsv files.

The View menu controls the display and refresh of choice and visualisation. It also give the ability to save or send data being displayed to e-mail, applications or a file.

The Collect menu controls data collection, if you own the collection process.

The Graph menu changes the appearance of the chart and is only displayed when the graph appears.

Finally, the Help menu gives access to spot help, documentation and links to the **system garden** web site for product information. Most help menu items need a common browser on the users path to show help information.

LOGGING

MyHabitat generates information and error messages. By default, errors are captured internally and can be displayed in the visualisation area by clicking on the **logs** node under **this client**.

Also available in this area are the **log routes**, which shows the how information of different severity is dealt with and **configuration**, which shows the values of all the current configuration directives in effect.

See **habconf**(5) for more information.

FILES

Locations alter depending on how the application is installed.

For the habitat configuration

~/.habrc, $HAB/etc/habitat.conf or /etc/habitat.conf

For graphical appearence: fonts, colours, styles, etc

$HAB/lib/myhabitat.rc or /usr/lib/habitat/myhabitat.rc

For the help information

$HAB/lib/help/ or /usr/lib/habitat/help/

ENVIRONMENT VARIABLES

DISPLAY The X-Windows display to use

PATH Used to locate a browser to display help information. Typical browsers looked for are Mozilla, Chrome, Opera, Internet Explorer, Netscape, Konqueror, Chimera

HOME User's home directory

AUTHORS

Nigel Stuckey <nigel.stuckey@systemgarden.com>

SEE ALSO

clockwork(8), killclock(8), statclock(8), habedit(8), habrep(8), habconf(5), myhabitat(1), habget(1), habput(1), habrs(1), habprobe(1), habmeth(1)

clockwork

Collection daemon for the Habitat suite

SYNTAX

clockwork [-c *<purl>*] [-C *<cfcmd>*] [-e *<fmt>*] [-*dDhsfv*] [-*j <stdjob>* | -*J <jobrt>*]

DESCRIPTION

Clockwork is the local collection agent for Habitat. It runs as a daemon process on each machine being monitored and is designed to carry out data collection, log file monitoring, data-driven actions and the distribution of collected data.

The default jobs are to collect system, network, storage, uptime and some busy process statistics on the local machine and make them available in a standard place. The collection of process data and file monitoring is available by configuring the jobs that drive clockwork. Configuration can be carried out at a local, regional and global level to allow delegation. One public and many private instances of clockwork can exist on a single machine, allowing individual users to carry out custom data collection Data is normally held in ring buffers or queues on the local machine's storage using datastores held to be self contained and scalable. Periodic replication of data rings to a repository is used for archiving and may be done in reverse for central data transmission.

OPTIONS

-c <purl>	Append user configuration data from the route <purl>, rather than the default file ~/.habrc.
-C <cfcmd>	Append a list of configuration directives from <cfcmd>, separated by semicolons.
-d	Place **clockwork** in diagnostic mode, giving an additional level of logging and sending the text to stderr rather than the default or configured destinations. In daemon mode, will send output to the controlling terminal.
-D	Place **clockwork** in debug mode. As -d above but generating a great deal more information, designed to be used in conjunction with the source code. Also overrides normal outputs and will send the text to stderr. In daemon mode, will send output to the controlling terminal.
-e <fmt>	Change the logging output to one of eight preset alternative formats, some showing additional information. <fmt> must be 0-7. See LOGGING below.
-h	Print a help message to stdout and exit
-v	Print the version to stdout and exit
-s	Disable the public data service from being run, but will continue to save data as dictated by configuration.
-j <stdjob>	Select from standard job tables, allowing different modes or behaviour to be selected easily from known good configurations. See COLLECTION MODES below for values and description of <stdjob>.
-J <jobrt>	Override standard job table with a private one provided by the route <jobrt>. **Clockwork** will not daemonise, run a data service or take an exclusive system lock (there can only be one public **clockwork** instance). Implies -s and alters the logging output to stderr, unless overridden with the range of **elog** configuration directives.
-f	Run in the foreground and don't daemonise

COLLECTION MODES

default	Default mode. One minute samples recorded to disk, deriving three sets of averages: 4h@1m, 1d@5m, 7d@15m, 1mo@1h

DEFAULTS

When clockwork starts it reads $HAB/etc/habitat.conf and ~/.habrc for configuration data (see CONFIGURATION for more details). Unless overridden, clockwork will then look for its jobs inside the default public datastore for that machine, held in *$HAB/var/<hostname>.grs* (the route address is *grs:$HAB/var/<hostname>.grs,jobs,0*, see below for an explanation). If it does not find the jobs, clockwork bootstaps itself by copying a default job template from the file *$HAB/lib/clockwork.jobs* into the public datastore and then carries on using the datastore version.

The default jobs run system, network and storage data gathering probes every 60 seconds. It saves results to the public datastore using the template route
grs:$HAB/var/<hostname>.grs,<jobname>,60 and errors to
grs:$HAB/var/<hostname>.grs,err_<jobname>,60

All other errors are placed in grs:$HAB/var/<hostname>.grs,log,0

ROUTES

To move data around in clockwork, an enhanced URL is used as a form of addressing and is called a 'route' (also known as a pseudo-url or p-url in documentation). The format is <driver>:<address>, where driver must be one of the following:-

file: fileov:	reads and write to paths on the filesystem. The format is file:<file path>, which will always append text to the file when writing. The fileov: driver will overwrite text when first writing and is suitable for configuration files or states.
http: https:	reads and writes using HTTP or HTTPS to a network address. The address is the server name and object name as a normal URL convention.
grs:	read and writes to a ring store, the primary local storage mechanism. Tabular data is stored in a time series in a queue or ring buffer structure. Multiple rings of data can be stored in a single ringstore file, using different names and durations.
sqlrs:	reads and writes tabular data to a remote repository service using the SQL Ringstore method, which is implemented over the HTTP protocol. Harvest provides repository services. Stores tabular data in a time series, addressed by host name, ring name and duration. Data is stored in a queue or ring buffer storage.

CONFIGURATION

By default, **clockwork** will collect system, network and storage statistics for the system on which it runs. All the data is read and written from a local datastore, apart from configuration items which come from external sources. These external configuration sources govern the operation of all the habitat commands and applications.

Refer to the *habconf*(5) man page for more details.

JOB DEFINITIONS

Jobs are defined in a multi columned text format, headed by the magic string 'job 1'. Comments may appear anywhere, starting with '#' and running to the end of the line.

Each job is defined on a single line containing 11 arguments, which in order are:-

1. start	When to start the job, in seconds from the starting of clockwork
2. period	How often to repeat the job, in seconds
3. phase	Not yet implemented
4. count	How many times the job should be run, with 0 repeating forever
5. name	Name of the job
6. requester	Who requested the job, by convention the email address
7. results	The route where results should be sent
8. errors	The route where errors should be sent
9. nslots	The number of slots created in the 'results' and 'errors' routes, if applicable (applies to timestore and tablestore).
10.method	The job method
11.command	The arguments given to each method

See the **habmeth(1)** manpage for details of the possible methods that may be specified and the commands that can accept.

DATA ORGANISATION

Data is stored in sequences of tabular information. All data has an ordered independently of time, allowing multiple separate samples that share the same time interval. This data is stored in a ring-buffer, which allows data to grow to a certain number of samples before the oldest are removed and their space recycled. Throughout the documentation, each collection of samples is known as a **ring**, and may be configured to be a simple queue, where data management is left up to administrators.

To limit the amount of storage used, data in a ring can be sampled periodically to form new summary data and stored in a new ring with a different period. In **habitat**, this is known as **cascading** and takes place on all the default collection rings. Several levels of cascading can take place over several new rings, This allows summaries at different frequencies to be collected and tuned to local requirements.

See the **habmeth**(1) man page for more information about the **cascade** method.

DATA REPLICATION

Any ring of information can be sent to or from the repository at known intervals, allowing a deterministic way of updating both repository and collection agent.

This is implemented as a regular job which runs the **replicate** method. Data for the method is provided by configuration parameters which can be set and altered in the organisation. Thus the replication job does not normally need to be altered to change the behaviour.

See the **habmeth**(1) man page for the replicate method and the formation of the configuration data.

LOGGING

Clockwork and the probes that provide data, also generate information and error messages. By convention, these are stored in the route specification *ts:$hab/var/<host>.ts,log* The convention for probes is to store their errors in *ts:$HAB/var/***<host>**.ts,e.**<jobname>**.

To override the logging location, use the range of **elog** configuration directives, or rely on the options -d, -D, -j, which will alter the location to stderr as a side effect. See habconf(5) for details. Probe logging is configurable for each job in the job table.

The logging format can be customised using one of a set of configuration directives (see habconf(5)). For convenience, the -e flag specifies one of eight preconfigured text formats that will be sent to the configured location:-

0	all 17 possible log variables
1	severity character & text
2	severity & text
3	severity, text, file, function & line
4	long severity, short time, short program name, file, function, line & text
5	date time, severity, long program name, process id, file, function, line, origin, code & text
6	unix ctime, seconds since 1970, short program name, process id, thread id, file, function, line, origin, code & text
7	severity, file, line, origin, code, text

FILES

If run from a single directory $HAB:-

$HAB/bin/clockwork
$HAB/var/<hostname>.grs, $HAB/lib/clockwork.jobs
/tmp/clockwork.run
~/.habrc, $HAB/etc/habitat.conf

If run from installed Linux locations:-

/usr/bin/habitat
/var/lib/habitat/<hostname>.grs, /usr/lib/habitat/clockwork.jobs
/var/lib/habitat/clockwork.run
~/.habrc, /etc/habitat.conf

ENVIRONMENT VARIABLES

EXAMPLES

Type the following to run **clockwork** in the standard way. This assumes it is providing public data using the standard job file, storing in a known place and using the standard network port for the data service.

```
clockwork
```

On a more secure system, you can prevent the data service from being started

```
clockwork -s
```

Alternatively you can run it in a private mode by specifying '-J' and a replacement job file.

```
clockwork -J "file:mywork.job"
```

AUTHORS

Nigel Stuckey <nigel.stuckey@systemgarden.com>

SEE ALSO

killclock(8), statclock(8), habedit(8), habrep(8), habconf(5), myhabitat(1), habget(1), habput(1), habrs(1), habprobe(1), habmeth(1)

statclock

Reports on running clockwork, Habitat's collection agent

SYNTAX

statclock

DESCRIPTION

Reports on the running public instance of clockwork on the local machine.

This shell script locates the lock file for clockwork, which is the collection agent for the Habitat suite. It prints the process id, owning user, controlling terminal and start time of the daemon if it is found in the process table. If not found, then the lock file is removed if it is there and a message to that effect is printed

Private instances of clockwork (started with -j option) can not be stopped by this method, as they do not register in a lock file. Instead, they should be controlled by conventional process control methods.

FILES

/tmp/clockwork.run
/var/run/clockwork.run

EXAMPLES

Typing the following:-

```
statclock
```

will result in a display similar to below and the termination of the clockwork daemon.

```
Clockwork process 17502 is running
  was started at 19-Apr-11 08:01:45 AM, user nigel, on /dev/pts/1
```

AUTHORS

Nigel Stuckey <nigel.stuckey@systemgarden.com>

SEE ALSO

clockwork(8), killclock(8), habedit(8), habrep(8), habconf(5), myhabitat(1), habget(1), habput(1), habrs(1), habprobe(1), habmeth(1)

killclock

Stops clockwork, Habitat's collection agent

SYNTAX

killclock

DESCRIPTION

Stops the public instance of clockwork running on the local machine.

This shell script locates the lock file for clockwork, which is the collection agent for the Habitat suite. It prints the process id, owning user, controlling terminal and start time of the daemon, before sending it a SIGTERM.

No check is made that the clockwork process has terminated before this script ends.

Private instances of clockwork (started with -j option) can not be stopped by this method, as they do not register in a lock file. Instead, they should be controlled by conventional process control methods.

FILES

/tmp/clockwork.run
/var/run/clockwork.run

EXAMPLES

Typing the following:-

    killclock

will result in a display similar to below and the termination of the clockwork daemon.

    Stopping pid 2781, user nigel and started on /dev/pts/2 at 25-May-04 08:08:55 AM

AUTHORS

Nigel Stuckey <nigel.stuckey@systemgarden.com>

SEE ALSO

clockwork(8), statclock(8), habedit(8), habrep(8), habconf(5), myhabitat(1), habget(1), habput(1), habrs(1), habprobe(1), habmeth(1)

habget

Send habitat data to standard output

SYNTAX

habget [-c <purl>] [-C <cfcmd>] [-e <fmt>] [-dDhv] [-E] <route>

DESCRIPTION

Open <route> using habitat's **route** addressing and send the data to stdout.

See *clockwork*(1) for an explination of the route syntax

OPTIONS

-c <purl>	Append user configuration data from the route <purl>, rather than the default file ~/.habrc.
-C <cfcmd>	Append a list of configuration directives from <cfcmd>, separated by semicolons.
-d	Place **habget** in diagnostic mode, giving an additional level of logging and sending the text to stderr rather than the default or configured destinations. In daemon mode, will send output to the controlling terminal.
-D	Place **habget** in debug mode. As -d above but generating a great deal more information, designed to be used in conjunction with the source code. Also overrides normal outputs and will send the text to stderr. In daemon mode, will send output to the controlling terminal.
-e <fmt>	Change the logging output to one of eight preset alternative formats, some showing additional information. <fmt> must be 0-7. See LOGGING below.
-h	Print a help message to stdout and exit
-v	Print the version to stdout and exit
-E	Escape characters in data that would otherwise be unprintable

EXAMPLES

To output the job table from an established datastore file used for public data collection. This uses the ringstore driver.

```
habget grs:var/myhost.grs,clockwork,0
```

To get the most recent data sample from the 60 second **sys** ring from the same datastore as above.

```
habget grs:var/myhost.grs,sys,60
```

To find errors that may have been generated by clockwork.

```
habget grs:var/myhost.grs,log,0
```

AUTHORS

Nigel Stuckey <nigel.stuckey@systemgarden.com>

SEE ALSO

clockwork(8), killclock(8), statclock(8), habedit(8), habrep(8), habconf(5), myhabitat(1), habput(1), habrs(1), habprobe(1), habmeth(1)

habput

Store data from standard input into Habitat

SYNTAX

habput [-s <nslots> -t <desc>] [*-c <purl>*] [*-C <cfcmd>*] [*-e <fmt>*] [*-dDhv*] *<route>*

DESCRIPTION

Open <route> using habitat's **route** addressing and send data from standard input (stdin) to the route.

See *clockwork*(1) for an explanation of the route syntax

OPTIONS

-c <purl>	Append user configuration data from the route <purl>, rather than the default file ~/.habrc.
-C <cfcmd>	Append a list of configuration directives from <cfcmd>, separated by semicolons.
-d	Place **habget** in diagnostic mode, giving an additional level of logging and sending the text to stderr rather than the default or configured destinations. In daemon mode, will send output to the controlling terminal.
-D	Place **habget** in debug mode. As -d above but generating a great deal more information, designed to be used in conjunction with the source code. Also overrides normal outputs and will send the text to stderr. In daemon mode, will send output to the controlling terminal.
-e <fmt>	Change the logging output to one of eight preset alternative formats, some showing additional information. <fmt> must be 0-7. See LOGGING below.
-h	Print a help message to stdout and exit
-v	Print the version to stdout and exit
-s <nslots>	Number of slots for creating ringed routes (default 1000); <nslots> of 0 gives a queue behavior where the oldest data is not lost
-t <desc>	Text description for creating ringed routes

EXAMPLES

To append a sample of tabular data to a table store, use a tablestore driver. This will create a ring which can store 1,000 slots of data.

```
habput grs:var/myfile.grs,myring
```

To save the same data, but limit the ring to just the most recent 10 slots and give the ring a description

```
habput -s 10 -t "my description" grs:var/myfile.grs,myring
```

The same data, stored to the same location, but with an unlimited history (technically a queue). To make the ring readable in ghabitat with current conventions, we store with the prefix **'.r'**

```
habput -s 0 -t "my description" grs:var/myfile.grs,r.myring
```

To save an error record, use a ringstore driver

```
habput -s 100 -t "my logs" grs:var/myfile.grs,mylogs
```

AUTHORS

Nigel Stuckey <nigel.stuckey@systemgarden.com>

SEE ALSO

clockwork(8), killclock(8), statclock(8), habedit(8), habrep(8), habconf(5), myhabitat(1), habget(1), habrs(1), habprobe(1), habmeth(1)

habrs

Command line interface to Ringstore storage

SYNTAX

habrs [-*c* <*purl*>] [-*C* <*cfcmd*>] [-*e* <*fmt*>] [-*dDhv*] [file [ring [dur]]]

DESCRIPTION

A Ringstore is a type of storage system designed to hold **Fat Headed Arrays** (FHA) in time series. It is the primary method of storing data locally on a machine in Habitat.

The command can be run with optional *file*, *ring* and *dur* arguments for names of the Ringstore file, the time series ring and its duration. Alternatively, these can be specified with the commands '**file**', '**ring**' and '**duration**' once **habrs** has started.

The prompt will contain the current file and ring name with duration.

OPTIONS

-c <purl>	Append user configuration data from the route <purl>, rather than the default file ~/.habrc.
-C <cfcmd>	Append a list of configuration directives from <cfcmd>, separated by semicolons.
-d	Place **habget** in diagnostic mode, giving an additional level of logging and sending the text to stderr rather than the default or configured destinations. In daemon mode, will send output to the controlling terminal.
-D	Place **habget** in debug mode. As -d above but generating a great deal more information, designed to be used in conjunction with the source code. Also overrides normal outputs and will send the text to stderr. In daemon mode, will send output to the controlling terminal.
-e <fmt>	Change the logging output to one of eight preset alternative formats, some showing additional information. <fmt> must be 0-7. See LOGGING below.
-h	Print a help message to stdout and exit
-v	Print the version to stdout and exit

COMMANDS

The following commands are accepted:-

! <cmd> sh <cmd>	Run <*cmd*> in a shell (see **chsh**(1) for your shell choice)
? [<cmd>] help [<cmd>]	Without <*cmd*>, print a tabular list of commands. Otherwise print an explanation and usage of <*cmd*>
bye exit e quit q	All of the above commands will exit **habrs**
close	Closes the currently open file and ring
create <file> <perm> <ring> <lname> <desc> <nslots> <dur>	Create a new time series ring, where <file> name of holstore file to contain the ring <perm> file permissions (eg 0644) <ring> name of ringstore ring <lname> long name of ring <desc> text description of ring <nslots> number of slots in ring, 0 for unlimitied <dur> secs duration of each sample, 0 for irregular

duration <dur> dur <dur>	Open the ring using the previously specified ring name in the currently open file. The duration should be a positive integer, which represents the length in seconds of each sample. Alternatively, it may be 0 to represent an irregular interval.
footprint	Print the current Ringstore file size
get	Get the next table in the time series. You should have an open file and ring
getall	Get all the tables in the series and print as a single table with the following columns:- _seq sequence number _time time _dur duration
goto <seq> jumpto <seq>	Goto a specific sequence number *<seq>* in the time series. The next call to **get** will print that table
jump <nseq>	If *<nseq>* is positive, move forward *<nseq>* places in the time series. If negative, move backwards.
ls lsrings	List the rings available in the current file
mget <nseq>	Return data from the next *<nseq>* sequences and advance to the next unread sequence. The data is printed as a single table with the following columns. _seq sequence number _time time _dur duration
open <file> [<ring> [<password>]]	Open a Ringstore file and optionally a time sries ring. If *<ring>* is not specified, then it can be specified with '**ring**' afterwards (also see **lsring**). *<password>* is not currently supported.
purge <nseq>	Remove the oldest <nseq> sequences from the time series. If more are removed that are present in the current ring, then this effectively empties the ring
put	Appends a new table to the end of the time series, which will be read from **stdin** (the keyboard or input pipe). Does not alter the current sequence to next read.
remain	Prints the amount of space free in the filesystem that houses the currently open Ringstore file.
resize	Change the number of sequence slots in the current time series ring
ring <rname>	Change the current time series ring to be *<rname>*. This will close any previously open ring but will not affect the current file. *<rname>* must exist in the current file.
rm	Remove the current ring. You will be prompted to confirm the action to prevent accidental eraseure.
rs <info>	Show the underlying ringstore structure in the form of tables. The secondary argument is needed to show the table as follows:- s | superblock superblock r | rings ring directory h | headers header hash table i | index record index
stat	Print statisticas about the current file and ring

FILES

Locations alter depending on how the application is installed.

For the habitat configuration

~/.habrc
$HAB/etc/habitat.conf or /etc/habitat.conf

For the help information

$HAB/lib/help/ or /usr/lib/habitat/help/

ENVIRONMENT VARIABLES

HOME User's home directory

AUTHORS

Nigel Stuckey <nigel.stuckey@systemgarden.com>

SEE ALSO

clockwork(8), killclock(8), statclock(8), habedit(8), habrep(8), habconf(5), myhabitat(1), habget(1), habput(1), habprobe(1), habmeth(1)

habmeth

Run Habitat's clockwork methods on the command line

SYNTAX

habmeth [-c <purl>] [-C <cfcmd>] [-e <fmt>] [-dDhv] <command>

DESCRIPTION

Runs a Habitat method, that would be available to jobs within the **clockwork** collection program.

<Command> should start with the name of the method and the remaining arguments to be applied to the methods. The **probe** method is missing, but can be run on the command line using **habmeth**.

With no arguments, a list of available methods are printed. Some methods do not require additional arguments other than the method name.

OPTIONS

-c <purl>	Append user configuration data from the route <purl>, rather than the default file ~/.habrc.
-C <cfcmd>	Append a list of configuration directives from <cfcmd>, separated by semicolons.
-d	Place **habmeth** in diagnostic mode, giving an additional level of logging and sending the text to stderr rather than the default or configured destinations. In daemon mode, will send output to the controlling terminal.
-D	Place **habmeth** in debug mode. As -d above but generating a great deal more information, designed to be used in conjunction with the source code. Also overrides normal outputs and will send the text to stderr. In daemon mode, will send output to the controlling terminal.
-e <fmt>	Change the logging output to one of eight preset alternative formats, some showing additional information. <fmt> must be 0-7. See LOGGING below.
-h	Print a help message to stdout and exit
-v	Print the version to stdout and exit

EXAMPLES

To find the available methods

```
habmeth
```

run a habitat method stand alone, where methods are:-

exec	Direct submission to exec(2)
sh	Test submit command line to sh(1)
snap	Take a snapshot of a route
tstamp	Timestamp in seconds since 1/1/1970 00:00:00
sample	Sample tables from a route, carries out a mathematical process and produce a single table as a result
pattern	Match patterns on groups of routes to raise events
event	Process event queues to carry out instructions
replicate	Replicate rings to and from a repository

The tstamp method returns the time in seconds from the epoch. The example would be

habmeth tstamp

1094314985

AUTHORS

Nigel Stuckey <nigel.stuckey@systemgarden.com>

SEE ALSO

clockwork(8), killclock(8), statclock(8), habedit(8), habrep(8), habconf(5), myhabitat(1), habget(1), habput(1), habrs(1), habprobe(1)

habprobe

Run Habitat's data probes on the command line

SYNTAX

habprobe [-*c* <*purl*>] [-*C* <*cfcmd*>] [-*e* <*fmt*>] [-*dDhv*] <*probe*> [<arguments> ...]

DESCRIPTION

Runs a built in Habitat probe to extract data from the system and print the resulting table on standard out. The data is represented in Fat Headed Array (FHA) format. <Probe> should be one of the probe names shown below and some take additional arguments.

This command is the equivalent of the following line in the job file of the **clockwork** collection tool

> ***probe*** *probename "arguments..."*

The difference being that output is sent to a route for storage or onward data stream.

With no arguments, a list of available probes for you system are printed along with usage.

OPTIONS

-c <purl>	Append user configuration data from the route <purl>, rather than the default file ~/.habrc.
-C <cfcmd>	Append a list of configuration directives from <cfcmd>, separated by semicolons.
-d	Place **habprobe** in diagnostic mode, giving an additional level of logging and sending the text to stderr rather than the default or configured destinations. In daemon mode, will send output to the controlling terminal.
-D	Place **habprobe** in debug mode. As -d above but generating a great deal more information, designed to be used in conjunction with the source code. Also overrides normal outputs and will send the text to stderr. In daemon mode, will send output to the controlling terminal.
-e <fmt>	Change the logging output to one of eight preset alternative formats, some showing additional information. <fmt> must be 0-7. See LOGGING below.
-h	Print a help message to stdout and exit
-v	Print the version to stdout and exit

PROBE NAMES

The following probe names are accepted:-

intr	Interrupt statistics
io	I/O data, storage and disk statistics
names	Symbolic data from the kernel
ps	Processes
sys	System data, including CPU and memory statistics
timer	Timer data
up	Uptime data, how long the system has been up
down	Down time data, calculated from **up** probe and can give a view on outages
net	Network device statistics

AUTHORS

Nigel Stuckey <nigel.stuckey@systemgarden.com>

SEE ALSO

clockwork(8), killclock(8), statclock(8), habedit(8), habrep(8), habconf(5), myhabitat(1), habget(1), habput(1), habrs(1), habmeth(1)

habconf

Habitat configuration

DESCRIPTION

Every major **habitat** utility and program gathers its configuration from external sources in the same way. The formats are shown below, but this section details how the data is gathered.

Firstly, each program can have individual directives specified on the command line with multiple '-C' flags.

A whole file of replacement directives can be used by using the '-c' flag. As this can be a route address, the data can originate from a web server. This feature allows one to keep the application and user defaults for mainstream use, and also allows special instances to be used simultaneously.

The user file *~/.habrc* is read next (although the location can be overridden), generally used for specific customisations. For example, the data files held in user interface history.

Then the application config is read from **$HAB/etc/habitat.conf** or **/etc/habitat.conf** (either of which again can be overridden). This contains the default supplied by the developers and may be amended by the local administrators.

Further configuration sources are possible, to allow for administrators to configure on a regional or global basis. Thus, the application configuration file does not have to be altered or making regional varients built. These sources may be web servers or other directory locations.

PARAMETERS

To customise at a user level, the file **~/.habrc** should contain directives in the following format.

[-]directive [[=] value]

Directive may appear on its own, or have a value associated with it. Values may be arrays, by separating their elements with whitespace. Each directive must appear on a new line. Comments may appear on any line, including ones with directives. They start with the '#' character and end with the newline character. The file must start with the magic string 'habitat 1', which confirms the contents and intent of the file to the configuration system.

Common directives include:-

elog.above <sev> <route> elog.below <sev> <route> elog.set <sev> <route> elog.all <route>	send errors and other messages above (or below) level <sev> to the route <route>. The .set form configures just that severity level and the .all form sends messages from all severities to <route>. <sev> can be 'fatal', 'error', 'warning', 'info', 'diag' and 'debug'.
elog.format <sev> <format> elog.allformat <format>	The specified severity level <sev> should print using <format>. The .allformat form sets <format> to all message sevelities. See the full manuals for more details of the format.
hab.cfetc <route> hab.cfuser <route>	Locations of the user configuration file (*~/.habrc*) in route format and the application wide file (*$HAB/etc/habitat.conf*). Because they are read early in the process of starting an application, these have to be specifed on the command line using -c or -C flags.

nmalloc	Turn on the memory checking, which will identify memory leaks. Normally this is off on stable releases, but developer releases may have it activated within the code.
replicate.out <links> replicate.in <links>	Specifies a list of replication links, used by the replication job. The purpose is to associate a local storage ring with a remote one on a repository. See the replication method's manual for information on its format.

FILES

~/.habrc
$HAB/etc/habitat.conf or /etc/habitat.conf

SEE ALSO

clockwork(8), killclock(8), statclock(8), habedit(8), habrep(8), myhabitat(1), habget(1), habput(1), habrs(1), habprobe(1), habmeth(1)

habrep

Replicate data between Habitat and System Garden's repository

SYNTAX

habrep

DESCRIPTION

Replicates data from the running public instance of clockwork on the local machine with System Garden's repository.

This shell script is equivalent to a replication command in Clockwork's job table, but rather than waiting for its scheduled time, replication will start at once. The job line (starting with method) is

    replicate replicate.in replicate.out "grs:%v/%h.grs,rstate,0"

The replicate method is run with the inbound replication contained in the 'replicate.in' configuration value, outbound replication driven from the 'replicate.out' configuration value. State is recoded in the GRS driven ringstore $HAB/var/<hostname>.grs,rstate,0.

As data is protected by sequences, habrep can be run any number of times without duplicates and corruption. Thus, the scheduled replication in the job table will be unaffected and its time does not need to change.

Replication is the process of sending new data from local rings to a repository and the same for remote to local. The state is recorded in a table in the central instance's ringstore.

FILES

    $HAB/var/<hostname>.grs

EXAMPLES

Typing the following:-

    habrep

will result in the following if successful

    Replicating to repository now...

or the folowing on a failure

    habrep: replication failed

AUTHORS

Nigel Stuckey <nigel.stuckey@systemgarden.com>

SEE ALSO

clockwork(8), killclock(8), statclock(8), habedit(8), habconf(5), myhabitat(1), habget(1), habput(1), habrs(1), habprobe(1), habmeth(1)

habedit

Edit data stored in a Habitat route or ringstore

SYNTAX

habedit <*route*>
habedit <*file*> <*ring*> <*duration*>

DESCRIPTION

Read data from a ringstore (using the three argument addressing) or other location (using habitat's **route** addressing) and amend it using your favorite editor.

If the object does not exist, you will be asked whether you wish to create it before the editing starts. Answering no will exit the command.

An attempt is made to recognise free text format data within the specified object. If found, then the time and sequence columns are suppressed along with the header so that the text alone is edited and saved.

If the data is tabular, then a tab separated format will be sent to the editor, with a header of column names. The sequence will be overridden with the next ordinal number when writing back to the object.

If the editor returns a failure (non 0 return code), then the resulting edit is not stored back to the route and the utility is abandoned.

See *clockwork*(1) for an explination of the route syntax

ENVIRONMENT

VISUAL If set with a path to a valid binary, this will be used as an editor of the data.

EDITOR If VISUAL is not set, this environment variable will be used instead.

EXAMPLES

To edit the job table from an established datastore file used for public data collection. This uses the ringstore driver.

```
habedit rs:var/myhost.grs,clockwork,0
```

Alternatively, if the three argument form is used, the ringstore driver will be assumed and the **rs:** part is not needed.

```
habedit var/myhost.grs clockwork 0
```

AUTHORS

Nigel Stuckey <nigel.stuckey@systemgarden.com>

SEE ALSO

clockwork(8), killclock(8), statclock(8), habrep(8), habconf(5), myhabitat(1), habget(1), habput(1), habrs(1), habprobe(1), habmeth(1)