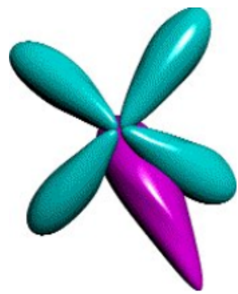


system garden habitat

Habitat User Guide

Edition 1

for

Habitat 1.0.0 and greater

Nigel Stuckey

nigel.stuckey@systemgarden.com

Copyright System Garden Ltd 2005. All rights reserved.

Table of Contents

1Tour....5
1 Getting Started....10
1.1 Collection by User....10
1.2 System Collection....10
1.3 Initial ghabitat view....10
2 Concepts....12
2.1 Architecture....12
2.1.1 Single Host: ghabitat + clockwork....12
2.1.2 Many Hosts: ghabitat + many clockworks....12
2.1.3 Many Hosts with Repository: ghabitat + clockwork + harvest....14
2.1.4 Extensible Collection Methods....15
2.1.5 System vs Private Clockwork Instances....16
2.1.6 Storage & Transport Integration....16
2.2 Data Format....16
2.3 Data Collection....17
2.4 Data Addressing....19
2.5 Data Storage....20
2.6 Ringstore & SQLRingstore....20
2.6.1 Local Data Storage....21
2.6.2 Peer Data Access....22
2.6.3 Remote Data Repository....22
2.7 Data Replication....23
2.8 User Interfaces....23
2.8.1 Command Line....23
2.8.2 Curses....24
2.8.3 Graphical....24
3 Clockwork: The Collection Agent....25
3.1 Starting....25
3.2 Stopping....25
3.3 Status....26
4 Graphical Tools....27
4.1 Data Visualisation....27
4.1.1 Data In Charts or Graphs....28
4.1.2 Data In Tables....28
4.1.3 Data In Row Popups From a Table....29
4.2 Data Navigation....30
4.2.1 Finding Data Sources....30
4.2.2 Source Exploration....31
4.2.3 Changing Timescales....32
4.2.4 Selecting Curves....32
4.2.5 Zooming and Panning Graphs....34

4.2.6 Adapting Curves....35
4.2.7 Custom Graphs....37
4.2.8 Selecting Instances....37
4.3 Import and Export....37
4.3.1 Email....37
4.3.2 External Tools....38
4.3.3 Interchange Files....39
4.4 Data Files....39
4.4.1 Saving Data in Files....39
4.4.2 Opening Data Files....40
4.4.3 Closing Data Files....40
4.5 Data Access from Peer Hosts....40
4.5.1 Peer Data over Filesystem....41
4.5.2 Peer Data over Network....41
4.5.3 Data from Repository....41
4.6 Graphical Viewer Information & Configuration....42
4.6.1 Configuration Files....42
4.6.2 Data Source History....42
4.6.3 Viewing Current Configuration....43
5 Text Terminal Tools....45
5.1 Track....45
6 Command Line Tools....46
6.1 Common Arguments....46
6.2 Data Addressing....47
6.3 habget....47
6.4 habput....47
6.5 Clockwork & killclock....48
6.6 /etc/init.d/habitat....48
6.7 Other commands....48
7 System Performance....50
7.1 Indicators....50
7.1.1 System....50
7.1.2 Storage....50
7.1.3 Network....51
7.1.4 Other Indicators....51
7.2 Adding to the standard data....51
7.2.1 Synthesising New Values....52
7.3 What is Abnormal?....52
7.4 Further Reading....52
8 Events....53
8.1 Event Queue....53
8.2 Watching Jobs....53
8.3 Watched Sources....53
8.4 Pattern-Action Data....53

8.5 Thresholds....54
9 Administration....55
9.1 Replication....55
9.2 Logs & Errors....55
9.3 Jobs....55
9.4 Raw Data....56
10 Diagnostics....57
10.1 Log Configuration....57
10.2 Collecting Less Severe Logs....57
10.3 Viewing Logs from the Choice Tree....58
10.4 Dynamic Viewing of Logs from Statusbar....58
11 Appendix....60
11.1 Manual Pages....60
11.1.1 clockwork....61
11.1.2 ghabitat....66
11.1.3 habget....71
11.1.4 habput....72
11.1.5 killclock....74
11.2 Collected Data....75
11.2.1 Linux Data....75
11.2.2 Solaris Data....78
11.3 Fat Headed Array Format....80
11.4 Job Table Format....82
11.5 Pattern Matching Table Format....83
11.6 Watched Sources Table Format....84
11.7 Event Table Format....84

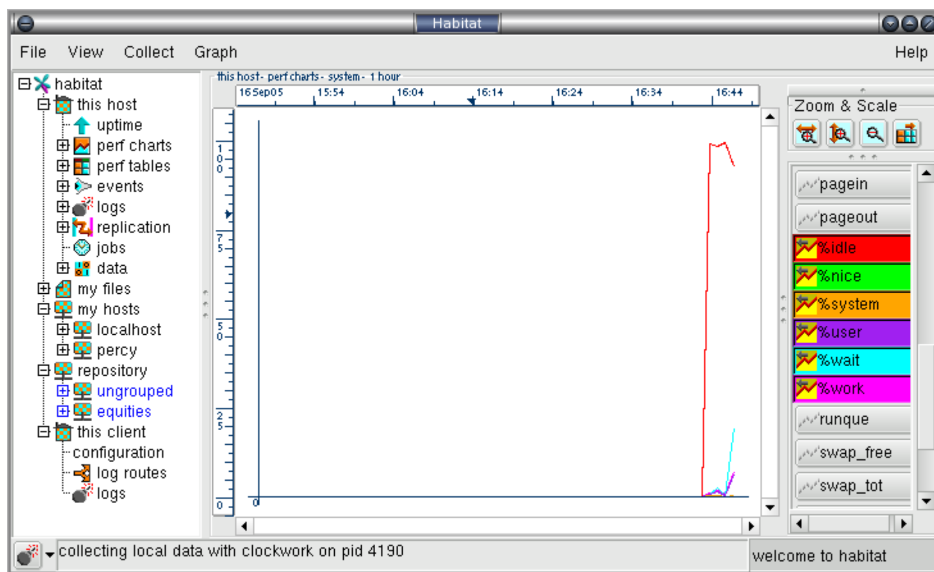
1 A Tour of Habitat

Habitat is a system capacity monitor with flexible historic data storage, easily extendible for applications or third party devices.

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 an optional archive for long term storage. 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.

A large number of measurements are taken from the system, including simple, overall usage, 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.

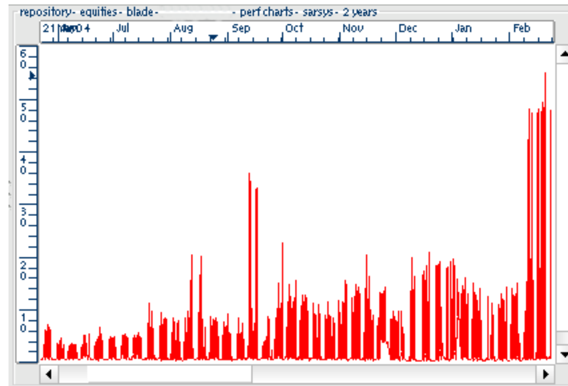
Starting **Habitat** from your desk top menu, or typing **ghabitat** on the command line will start habitat's GUI. It appears with a graph of the machine's overall utilisation over recent time.



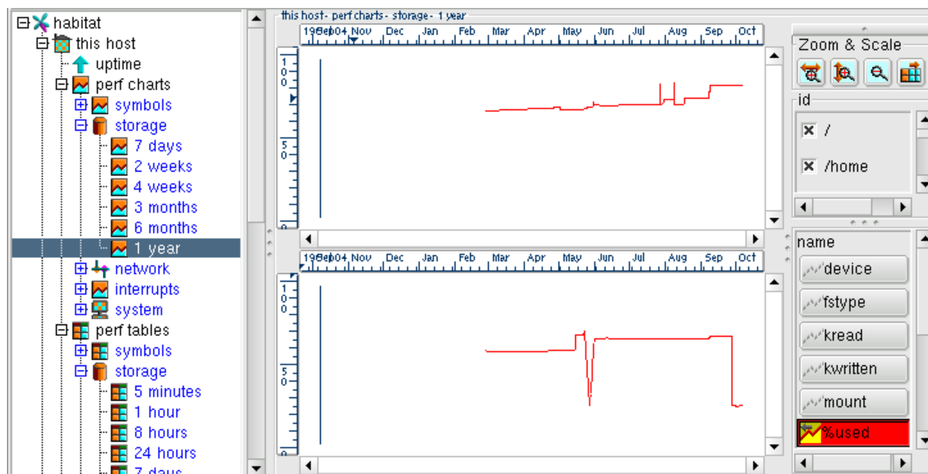
On the left hand side are the set of choices that can be displayed, with the right-hand pane showing the visualisation of the data. When the system is first started, there may be little or no data, so a blank screen may be presented for a while! By default, data is collected every minute and display is refreshed to show the new curve. In the image above, the collector may have been running for three to four samples (the same number of minutes), but if the collector had been running for longer or independently of the GUI, then there would be more data to see initially. But the a blank display will not stay blank for long!

This manual explains how to visualise data that is collected and how to navigate around the sources that are available. However, as a quick taster, this is a sample of what can be seen.

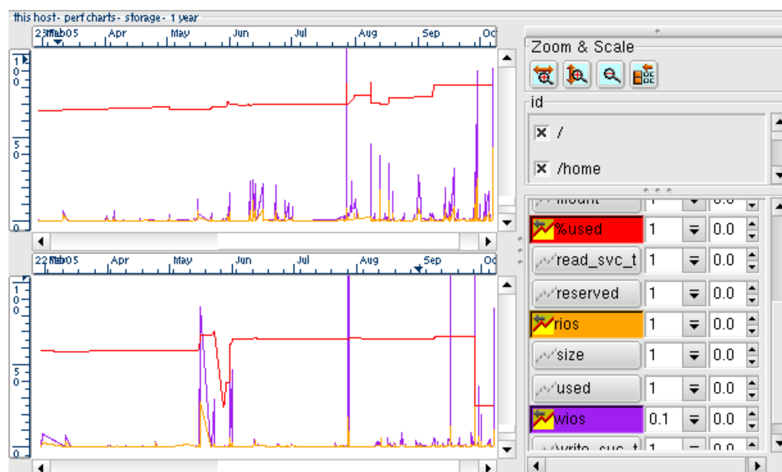
Long term growth in processor usage, shown below, shows a gradual climb over a period of nine months, with a sudden late surge of activity:



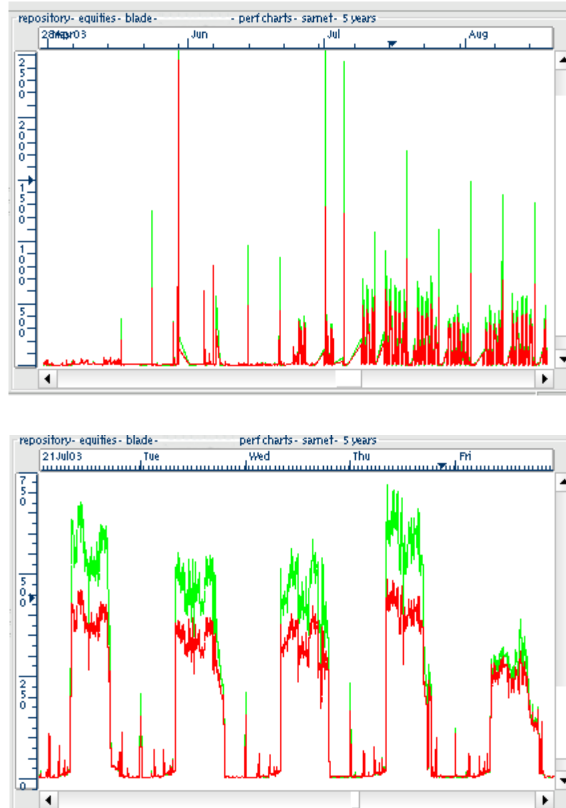
The image below shows growth in local storage on two partitions on a single machine over a period of seven months. The top chart shows the root file system (/) capacity use growing gradually in size over the period from 65% to 85%, the bottom /home chart, shows a sudden drop after a similar gradual climb.



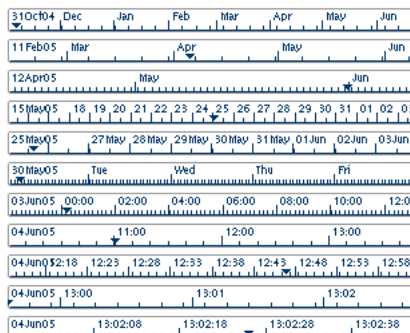
The image below takes the previous view and adds in the volume of disk requests on each chart (rios and wios for read and write requests), and scales the new curves to fit in the utilisation range. The curves are colour coded against the pick list on the right onto which the scaling controls have been added.



The next pair of images represent a busy network chart with lots of samples. Zooming into the July part of the chart (by dragging a box over the area of interest with your mouse), will expand the display to show greater detail, as shown in the lower image. Note how the time scale on the horizontal ruler changes to give the most accurate information possible, which in this case switches the display to dates of the week, with the starting date on the left.



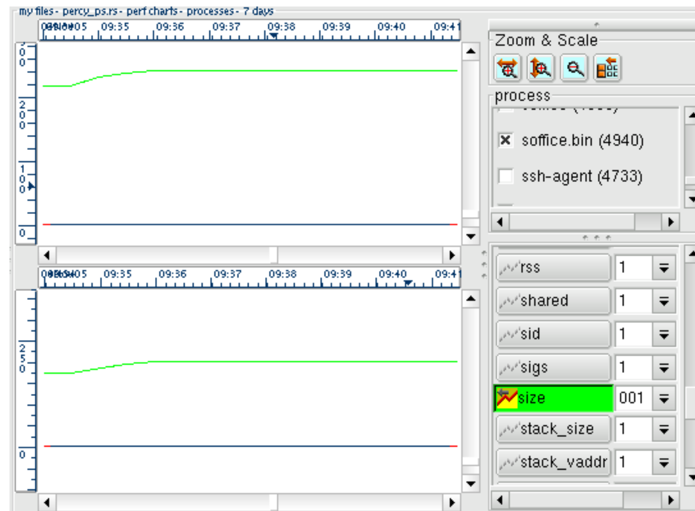
The chart zooming in habitat runs from years to seconds, as does the representations of time on the horizontal axis. A selection of time scales is shown below.



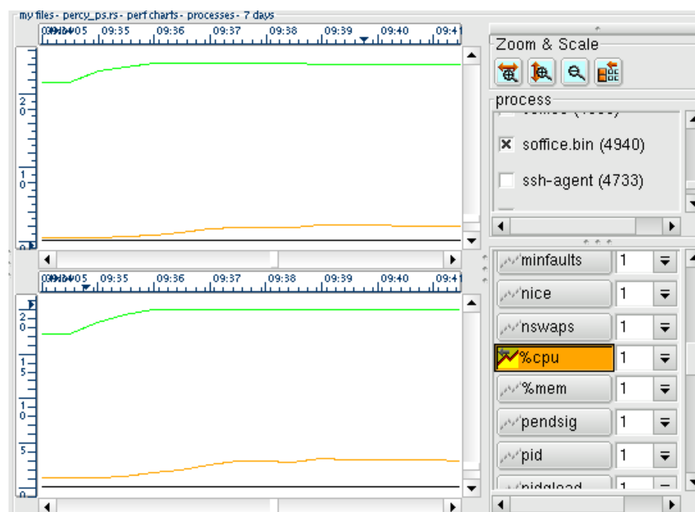
By default, habitat collects 'interesting' processes only and filters out smaller or mostly idle processes. This helps to reduce the data that needs to be stored and manipulated, critical for the successful examinations of processes.

The next image shows multiple processes being tracked for memory on a system (Open Office Writer and X-Windows in this example). Processes rarely give significant quantities of memory back to a system, so it is often useful to profile an application over time before it goes into production. The image shows the process

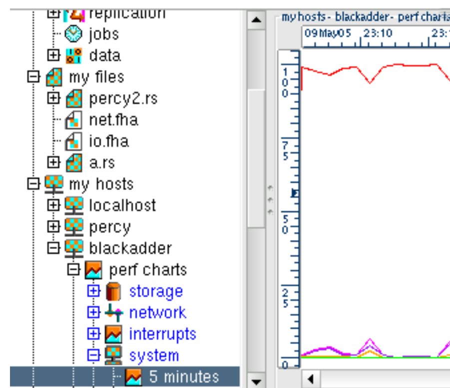
sizes in MBytes: the underlying metric is KBytes, but *ghabitat* has been used to reduce the curve values by 1/1000.



The memory usage may be augmented with the processor usage for each process by displaying %cpu, which is displayed on the same charts in a different colour. The %cpu measure is the amount of the system's cpu taken over the lifetime of the processes involved, thus peaks in demand are flattened and seen over time, it can be counter intuitive. To display both curves, the size values has be reduced in magnitude to 1/10'000, effectively making 10 MBytes units for size.

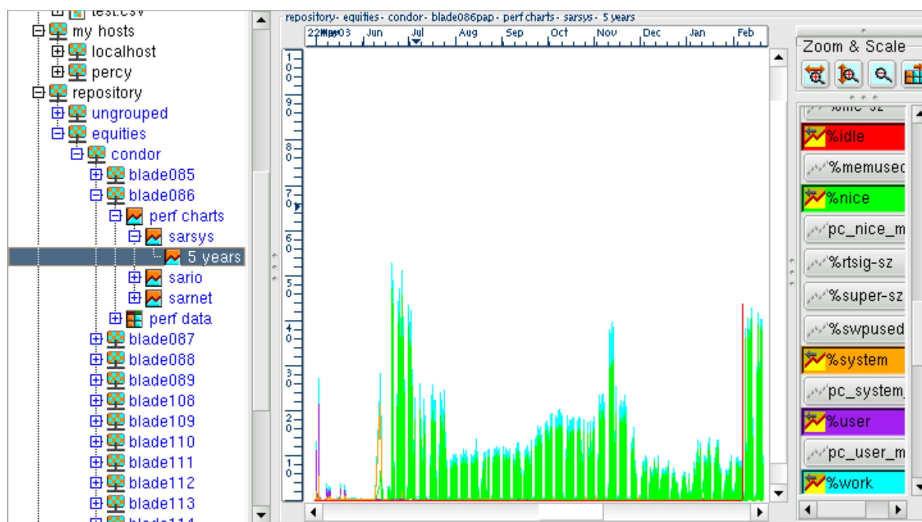


Other machines can be displaced in the same tool. Below is the choice tree with several hosts connected (under *my hosts*) and also some files from previous saved performance sets (under *my files*).



Not all data has be generated by *habitat*: both *habitat* and *harvest* can import arbitrary tabular time series data in the FHA format (explained later). In the example below, data from from the Unix data gathering tool *sar* has been imported into the *harvest* repository. The repository appears as an option in the choice tree and the machine sources have been assigned an organisational hierarchy within *harvest* to help navigation. *The node sarsys holds the system information from sar.*

Integrating the repository is simple: an administrator provides a URL, together with optional authentication information. All repository data is then grafted into the choice tree.



2 Getting Started

Habitat comes in two parts: the collection agent (called *clockwork*) and the viewing client (called *ghabitat* but others are available). It can be started in many ways, and these are described in full later in this document and in detail in the Administration manual. However, to get started, we shall describe two common ways in which habitat can be run.

2.1 Collection by User

If you install from the *tar* file, with no automatic starting of the collection daemon, the graphical tool will start it for you. This is the simplest way to start using habitat under Solaris. Firstly, extract the package from the *tar* file, then *cd* into the *bin* directory and run *ghabitat* at the command line. Below is an example of installation and running:

```
tar xvzf habitat-1.0.0-mdk1.i586.rpm
cd habitat-1.0.0/bin
ghabitat
```

Once started, *ghabitat* will look for a central instance of *clockwork* (the collection agent) that may already be running. If it does (from a previous run perhaps), then it attempts to load a default set of data from it. If it cannot find *clockwork* then it will ask the user if you want to start one. The options are described later in this document, but for now, click on *start.*

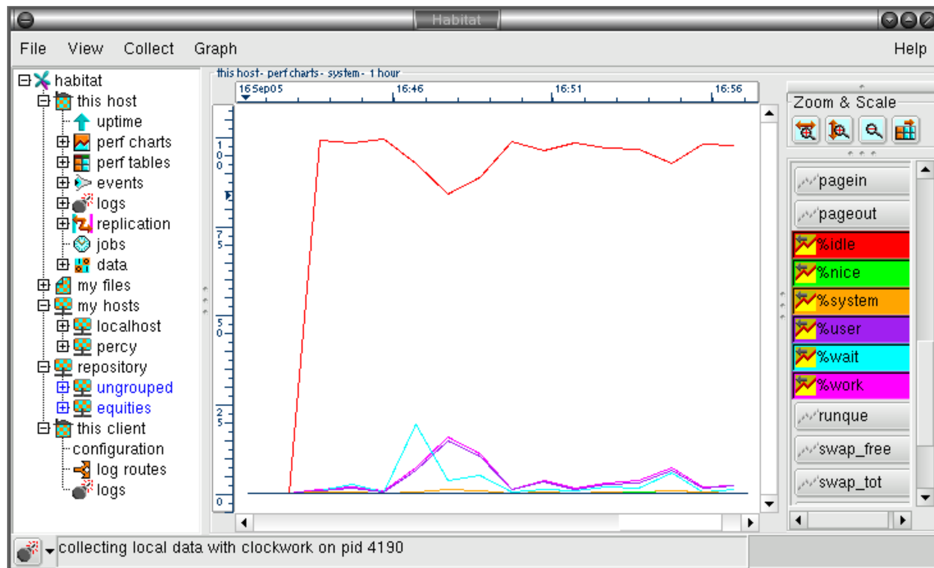
2.2 System Collection

This is the simplest of all and is the default way in which an *RPM* package is installed. When *rpm -i* is issued, habitat loads itself into the correct system location for a Linux system and automatically starts *clockwork* the collection agent. The initialisation scripts are also amended to start collection each time the system is booted.

When *ghabitat* is started from the system location (like /usr/bin/ghabitat), it will work out that collection is already running and attempt to get the initial data from it.

2.3 Initial ghabitat view

Once started, the graphical client *ghabitat*, presents the user with an *explorer*-like interface, with data choices on the left and visualisation on the right. In addition to this, the charting displays also have a curve selection list on the extreme right plus some zoom and scaling buttons.



Now we have the viewing platform running, allow *clockwork* to collect some data for a few minutes from the system and read more about the way habitat works.

3 Concepts

3.1 Architecture

Habitat in its simplest form is a two tier application: an agent process running as a daemon on each machine being examined and a client process to look at the data.

Long term or high sample rate data can be moved off to a specialist repository, called *harvest,* which also acts a reporting portal and an analytics engine. *Harvest* is covered in a separate set of documentation available at http://www.systemgarden.com/harvest and provides another optional tier to the architecture.

The following section describes the architecture of common deployments and the major concepts needed to understand *habitat*.

3.1.1 Single Host: ghabitat + clockwork

In its simplest form, *habitat* collects data and displays the timeseries on the same machine. Collection is carried out by a daemon program by the name of *clockwork*, which runs many periodic jobs, each of which collects data or carries out a calculation. The default configuration takes runs system collection jobs and inserts them into a central *ringstore* datastore.

A visualisation tool, such as *ghabitat* queries the daemon (or reads the datastore directly) and collects the data for tabular display or graphical plotting.

This scenario is found in one-off testing, when using single machines or running additional instances of habitat that are independent of a general collection infrastructure.

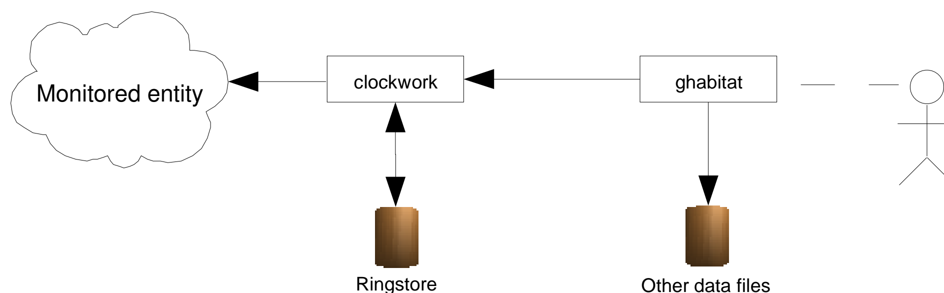


Illustration 1: Habitat running on a single machine, monitoring its own host. Ghabitat displays information obtained from a data file or from the clockwork collection agent.

3.1.2 Many Hosts: ghabitat + many clockworks

For small networks of machines with a handful of data viewers, habitat may be configured using peer level connections. By adding additional hosts to *ghabitat's* set of choices, a user may select to visualise data from remote machine.

To keep performance high, *ghabitat* only ever uses one connection at a time and polls at the same frequency of the sample interval. The connections are stateless, so the *clockwork* instances have no overhead in their role as servers of data. Additionally, the protocol is based on HTTP (but on a non standard port), that it

allows for easy routing and checking in network infrastructures.

The result is a balance between containing demand and low administration.

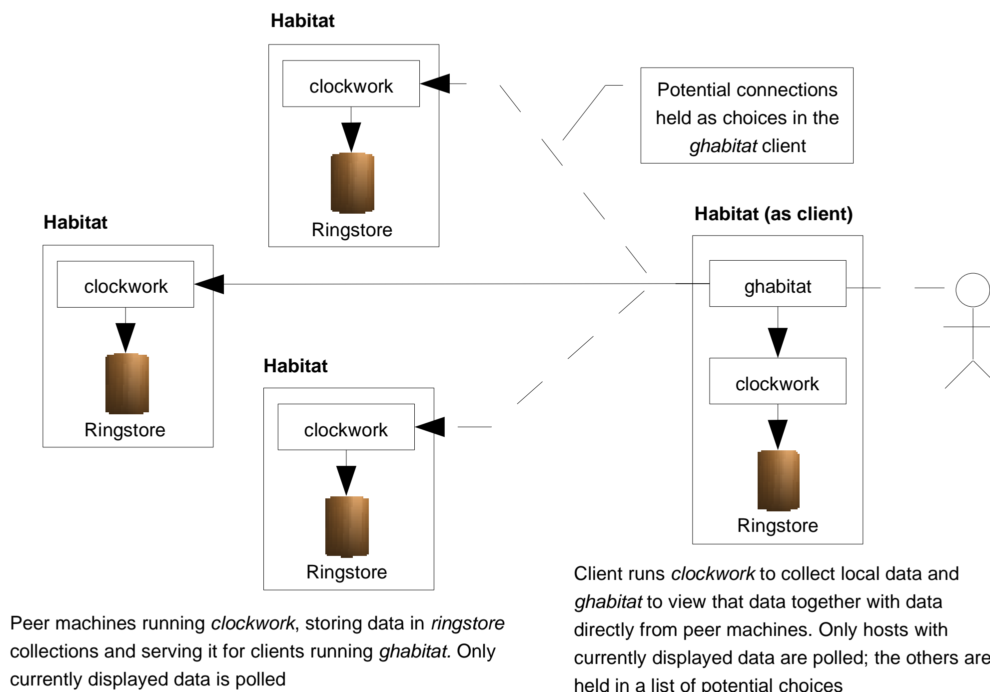


Illustration 2: Several instances of habitat, one of which is acting as a visualisation tool. Whilst many connections can be in place, ghabitat only polls for currently displayed data and at the same rate it is sampled, thus reducing the overall load to a minimum.

3.1.3 Many Hosts with Repository: ghabitat + clockwork + harvest

For larger configurations or when there are many potential viewers, an additional component is employed to remove load from monitored systems and increase scalability. This is a central repository containing an archive of machine data and is implemented using the *harvest* product, also available from system garden (http://www.systemgarden.com/harvest).

Data may be sent directly to harvest using the *route* system, although there are many advantages in indirectly posting data. *Habitat* does this by default when it replicates many pieces of data at periodic intervals, thus batching the updates and minimising another potential bottleneck.

With a repository present, *ghabitat* will use that source to obtain data for all hosts and potentially many other centrally computed statistics. In this case, there is no additional load on the monitored servers.

Only unreplicated data would need to be collected from the monitored servers, generally the case where very recent data had been collected but the replication process had not been carried out. The process of replication is configurable, both time frequency and the data rings sent and received.

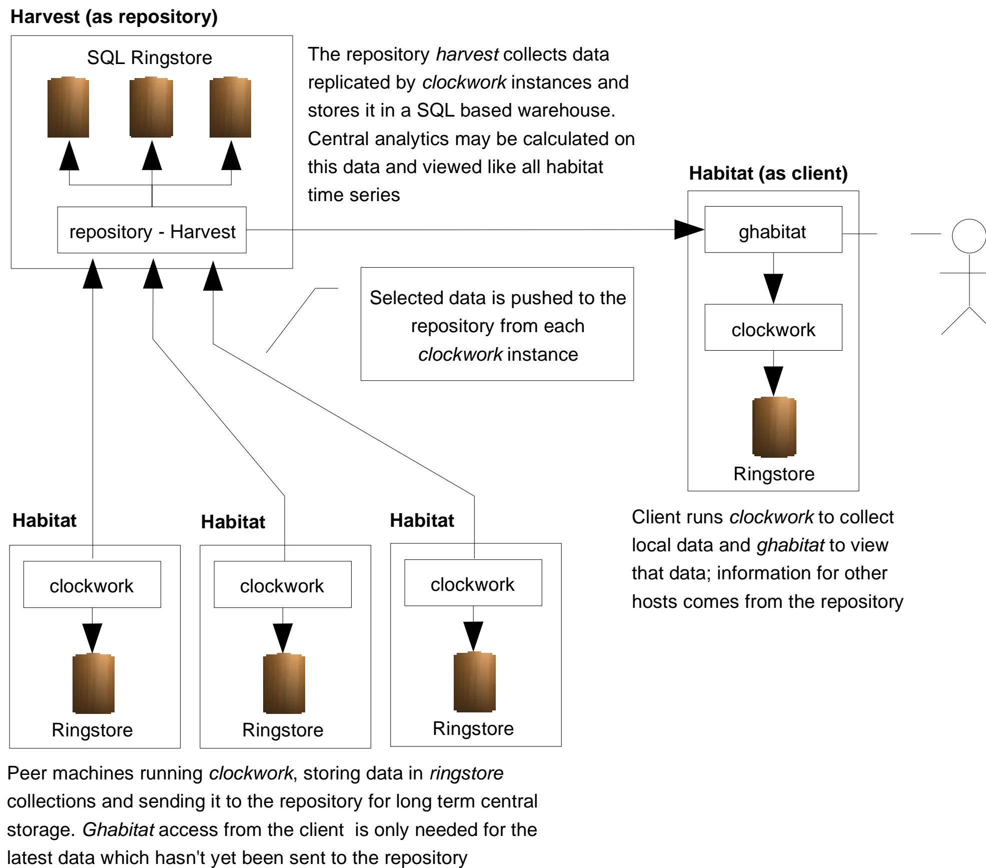


Illustration 3: Top level architecture of habitat and harvest, showing several clients running habitat, one of which is viewing data with ghabitat, and a single harvest repository. Using harvest lessens the load on the the monitored machines, stores detailed long term data and provides organisational wide analytics

The two can communicate by HTTP or by sharing the data file to which *clockwork* writes. Both these protocols are network friendly and allow multiple viewers to see the same data, or a single viewer to see many data sets.

3.1.4 Extensible Collection Methods

Within *clockwork* is the ability to extend collected data and the methods by which it can be collected. Data capture is scheduled using a single table (*clockwork's* job table) which may be different for each instance of *clockwork*. By adding or changing the jobs different data can be collected at custom time intervals. Each job runs a *method*, with several ones included in the standard distribution. New methods can be defined by adding code to *clockwork* at run time in the form of shared objects that have a standard interface. This is the most efficient way of adding new collection and computation capabilities, as the minimum number of processes are involved and overhead is kept to a minimum.

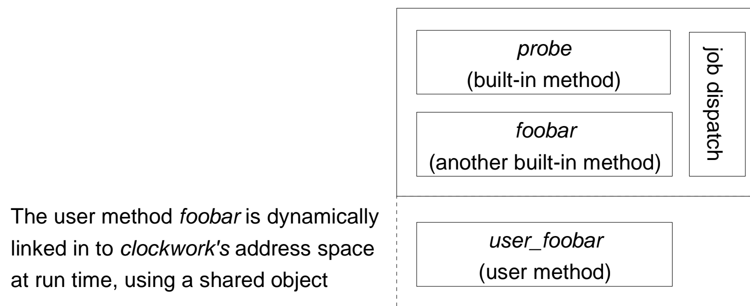


Illustration 4: The user method foobar is dynamically linked in to clockwork's address space at run time, using a shared object

3.1.5 System vs. Private Clockwork Instances

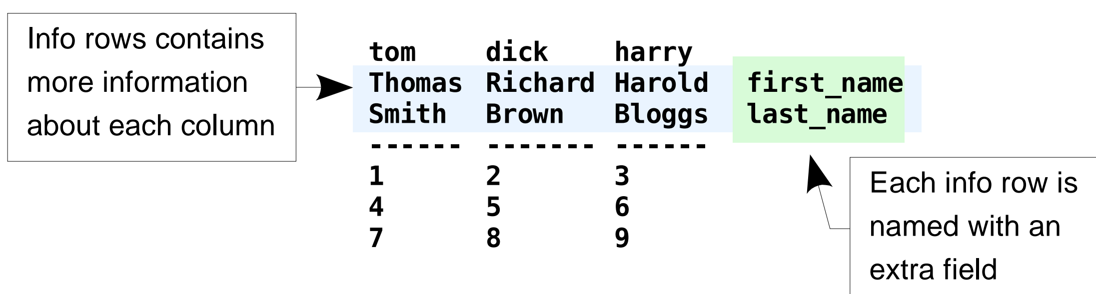
On a single machine, many instances of *clockwork* can run, but only one can provide the dependable system service, which is able to serve data over the network using the standard port. Typically, a system-wide instance would cover most eventualities, but for some issues more detail profiling is required, such as heavily monitoring an application under test. In these cases, additional, private *clockwork*s can be run, saving data to their own data files. These files can be viewed in real time or saved for later examination using the standard tools

3.1.6 Storage & Transport Integration

Habitat provides several ways of viewing collected data. Chief among them is *habget*, a command line tool to extract data for use in Unix filter pipelines, and *ghabitat*, a graphical application written in Gtk+. A more sophisticated product for larger installations and grids is under development by System Garden. It is cross platform and provides many advanced features (see separate documentation and the website http://www.systemgarden.com for more details).

3.2 Data Format

All data in habitat is tabular and can be expressed externally using a format called the *Fat Headed Array*, an example of which is shown below. It is **tab delimited** and similar in function to *CSV* (comma separated values) data, except that it can also contain sets of information attached to each column. These are known as *info rows* and are printed over several lines below the single column name row.



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*).

3.3 Data Collection

Data is gathered by habitat's agent, *clockwork,* which can **pull** the data from a source or can have data **pushed** directly into the data store or other data stream.

In order to **pull** data, *clockwork* (habitat's agent) has a table of jobs that it executes 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, so that information sent to a *route* will end up being saved.

An Instance of *clockwork* with several probes running, each configured as a job. When a job run is triggered, the probe polls for statistics

Monitored activity or statistics

probe
probe
probe
job dispatch
ringstore route

Probes process returned data and store the results in ring buffer structures, simply called *rings* held in a file called a *ringstore*. Many rings can be stored in the file, each individually sized so that the new data overwrites old when capacity is reached

Illustration 5: The components involved in polling (pulling) monitored activities or statistics and storing them inside habitat. Ringstores are conventionally used as they are self-contained and require little maintenance

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*, and will send it to the route address specified.

A time series of data is built up by repeatedly executing a probe (or receiving FHAs from external tools) to store a sequence of tables. Each table defines data over periods of 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.*

Each probe operation or FHA store will typically store data relating to a single point in time. However, when importing or reporting sets of data, the data series is concatenated into a single table and uses the*_seq* column to distinguish between samples.

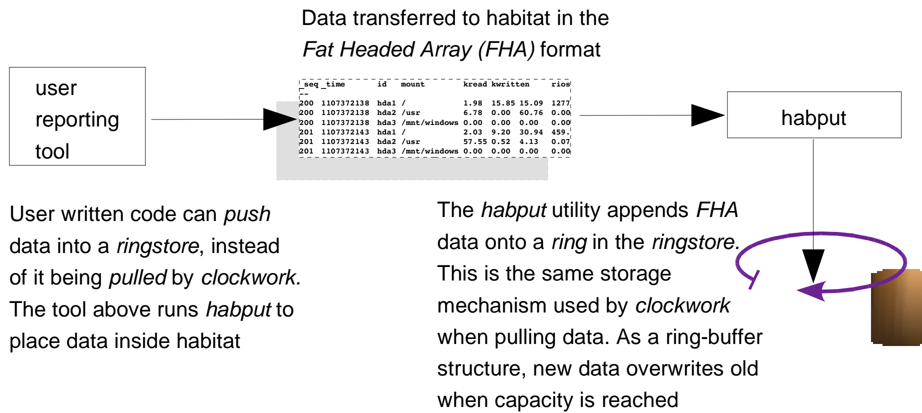


Illustration 6: The components involved in pushing data from user written utilities or applications into habitat's storage system. Usually, this will involve the creation of data in a Fat Headed Array (FHA) format and its insertion into a local ringstore using habput

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

Sample 1

Sample 2

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.

3.4 Data Addressing

Habitat and harvest share the same addressing scheme but have different implementations, suiting their particular design goals. Habitat generally uses a *ringstore* whereas Harvest uses a *SQLringstore*, which is based on more heavy weight database technology. However, as the route mechanism in habitat connects to both systems of storage, data is universally accessible across the product range.

To identify the different types of storage, a full route address contains a prefix, ending in a colon (:). For *ringstore*s, the prefix is **rs:** and for the *SQLringstore* in the harvest repository the prefix is **sqlrs:**.The type (or driver) prefix is very similar to the **URL** format used in web browsers and have some similar capabilities. The list of drivers currently supported are as follows:-

rs:	Ringstore
sqlrs:	SQLringstore
file:	Plain file. When writing will append to existing file
fileov:	Plain file. When writing will destructively overwrite existing file
http: **https:**	Hypertext transfer protocol and secure hypertext transfer protocol, the encrypted variety. Most standard URL formats are supported. Requires configuration variables to be set up to allow communication though proxies and to use accounts.
stdin: **stdout:** **stderr:**	Standard input, output and error. No further address required
none:	Output will be disposed of. No further address required

3.5 Data Storage

Data is generally collected in a local data store named a *ringstore*. This is a lightweight but structured storage system that has a lower impact on the system than a full SQL database. It is based in key-value block storage, currently implemented with GDBM.

Six dimensions are required to uniquely address a single datum, which are shown in the table below. Only the first two are needed to insert data, that of *file* and *ring* names: all the other details come from the inbound data.

File name **(or host name)**	The file name that holds the data. There is usually one file per machine with default central collection, although with personal collection, this becomes a user choice. With *sqlrs* (*SQLringstore*) from the harvest repository, this dimension is the host name.
Ring name	A collection of data tables sharing the same type or schema. Typical values are *sys* for system information, *io* for storage and *net* for network data. Note that the schema is not fixed or predefined: see below for details.
Duration	The interval between samples in seconds or 0 for an irregular sampling period. If omitted then the data is **consolidated** over all available durations (see below).
Column	Optional selection of a column name, which is able to return a single attribute of data (sets of columns may also be possible).
Sequence	Optional single or range of sequence numbers, uniquely identifying rows belonging to the same sample. Identified by pre-pending **s=** to the range.
Time	Optional single or range of time values, that can be used to extract a time series from a ring of data. Identified by pre

File name (or host name)	The file name that holds the data. There is usually one file per machine with default central collection, although with personal collection, this becomes a user choice. With *sqlrs* (*SQLringstore*) from the harvest repository, this dimension is the host name.
	pending **t=** to the range. All time values are in the Unix **time_t** format (seconds from 1/1/1970 or the *epoch*)

3.6 Ringstore & SQLRingstore

The native format for habitat is the *ringstore*, which is based on a GDBM key-value storage database and is typically held on storage local to the machine being monitored. Harvest is based on a SQL database and makes its data available using an HTTP format using a web servelet. The format is known as *SQLringstore* and has a very similar addressing format to the local *ringstore*.

The general format for *ringstore* and *SQLringstore* is as follows:-

[**sql**]**rs:**file_or_host**,**ring[**,**duration[**,**column]][**,s=**srange][**,t=**trange]

Where

[**sql**]**rs:**	Either **rs:** or **sqlrs:** for *ringstore* or *SQLringstore* respectively
file_or_host	The file name of the *ringstore*, conventionally ending with **.rs** to informally indicate its type. With *SQLringstore* this will be the host name of the machine that generated the data
ring	Ring name, such as **io**, **sys** or **net**
duration	Optional duration of samples, measured in seconds. **0** (zero) is taken to mean an irregular event. If missing on reads, the consolidated view is taken; if missing on writes, the column **_dur** will be expected in the data
column	Optional column to extract when reading. Unused when writing
s=srange	Sequence range. srange is of the form *from [- [to]]*
t=trange	Time range. trange is of the form *from [- [to]]*

The following are examples of addresses used by habitat:-

rs:myhost.rs,sys,60	Ringstore file called **myhost.rs**, returning the ring **sys** taken at **60** seconds interval
rs:/var/lib/habitat/myhost.rs,sys,300	Ringstore file called **myhost.rs** in the **/var/lib/habitat** directory, returning the ring **sys** taken at **300** seconds interval (five minutes). In habitat, this is generally created automatically in a process called **cascading**.
fileov:/home/fred/.habrc	Flat text file, being the personal configuration of habitat for the user **fred**. The file is **.habrc** in **fred**'s home directory. The type **fileov:** is chosen as the file is replaced each time it is generated by habitat.
file:mylog.txt	When used as an output, data is appended to the file **mylog.txt**. Used as inputs, **fileov:** and **file:**

	types are identical in method.
sqlrs:myhost,sys,300,*,s=428-	As an input, data is extracted from the repository (system **myhost**, ring **sys** at duration **300** seconds) and returned as a table. Only sequences **428** and greater are returned.

3.6.1 Local Data Storage

In a standard configuration, habitat stores centrally collected data in a *ringstore* structure, which is held in a single file. The file is called *hostname.rs* and is held in **var** in the application directory (for the .tar distributions) or /var/lib/habitat (for the RPM distributions).

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

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

3.6.2 Peer Data Access

The normal method to access local and remote data is to query the agents directly on each monitored machine. The agent (*clockwork*) implements a network server to satisfy queries from the front end *ghabitat* and other tools. When given a query, it accesses the central habitat *ringstore* on the local machine in order to return the results. For security reasons, it is not possible to use this method to access any other file.

In *ghabitat* your local host's data will appear under 'local host' in the choice tree. To connect to other hosts and get their data, select **File->Host**, type the name of the machine and click 'direct'. If successful, an entry for that machine will appear in the choice tree, under the branch 'my hosts'.

It is also possible to export data files using a file sharing protocol such as NFS or CIFS and mount them on remote machines. Using this technique for centrally collected *ringstores* may impact the ability the speed and reliability of data access due to the nature of file locking and network bandwidth. However, this may be the most appropriate way of sharing custom data collected by individual users as the files may not be as busy.

In *ghabitat*, connect to *ringstore* files by selecting **File->Open** and navigating to the location of the data file. The file will appear in the choice tree under the branch 'my files'.

3.6.3 Remote Data Repository

Using the standard configuration, the central *ringstore* file will grow to around 50 MBytes; more if additional data is collected or retention periods are extended. Moreover, older data is summarised at a lower frequency than originally collected to save space.

To keep data for longer, a remote repository may be used to archive data and can be used as a data source within *ghabitat* just like *ringstore* files or host attachments. Such a repository is provided by the *harvest* product, with data stored in the *SQLringstore* format.

From *ghabitat,* the repository is used by mapping the host into your choice tree. There are two methods. Firstly, by directly referencing the host: selecting File->Host from the menu, typing the host name and keeping the repository button highlighted. The host will appear under the branch 'my hosts' in the choice tree.

The second method is by browsing the repository from the choice tree. Hosts in the *harvest* repository may

be ordered by organisational structure. For example, a server in London's finance department may be reached using the following tree path:

repository->Finance->London->theserver

Regardless of the method, the tree structure below the hosts will show the data that has been transferred to the repository. See the Data Replication section for details of data transfer to the repository.

3.7 Data Replication

Habitat sends data o the *harvest* repository by a process called replication. It enables data that has been collected in the local *ringstore* to be synchronised with the repository and new data transferred.

However, replication may also work both ways by allowing centrally created data to be propagated down to satellite habitat instances. This is an ideal way of maintaining job tables or other data which needs to be independent of network connections.

Typically, replication runs once a day, but can easily be increased for sites with a policy of frequent archiving. The process is always initiated by *clockwork* using a job from its job table. If *harvest* or other similar tool wanted to pull data from habitat, *clockwork's* network service should be used using standard *route* conventions. Replication is not enabled in the standard configuration of habitat. Instructions to configure and enable it are contained in the Administration Manual.

Replication of data has a number of benefits, including:-

1. Data is backed up to an alternative location (the repository)
2. Visualisation or extraction tools can use the repository for the data source, saving capacity on the analysed machine
3. The repository can hold large quantities of data, which may be inappropriate to store locally on machines being analysed.
4. The repository can be a specialist database machine, with large storage and general purpose data handling
5. With the performance statistics of the whole enterprise in a single place, centralised analytics may be run in an efficient manner.

See System Garden's harvest repository for more details.

3.8 User Interfaces

A variety of visualisation and extraction tools exist for use with habitat, many of which are available in the package and are described below. In addition, System Garden provides other tools to enhance the interface which are available in separate packages. See http://www.systemgarden.com for more details.

3.8.1 Command Line

Habitat provides two command line utilities for data extraction and insertion. As such, they can be used in shell pipelines to build more complex commands.

habget extracts data from any supported *route* using and returns the data on *stdout.* See the manual later in this documentation for details. However, as an example:-

```
habget rs:/var/lib/habitat/myhost.rs,sys,60 | more
```

Will return the most recent data sample from the system probe and pipe the output into the utility 'more'. The data will appear as a table as shown in examples above. The file is the standard location under Linux for RPM installations.

habput is a method of inserting data onto a *route* for storage in a *ringstore* or *SQLringstore*. The manual for its use is later in the documentation, however an example is shown below:-

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

Will read the table from stdin and send it to the *ringstore* **myfile.rs**, to be stored in the ring **myring**. The table will be scanned and checked to confirm that it is tabular before it is stored. It is timestamped with the current time and a sequence allocated depending on the existing contents of the ring.

The commands **clockwork** and **killclock** are used to stop and start the collection daemon on each machine. These are explained in a section below.

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
- **irs** Interactive ringstore utility, allowing administration of the data held in *ringstore* files

3.8.2 Curses

The curses-based, dumb terminal tool is not supported in the current release.

3.8.3 Graphical

The main visualisation tool within the habitat suite is *ghabitat*. To start this, either select from the 'start' bar of your windowing environment or type **ghabitat** on the command line.

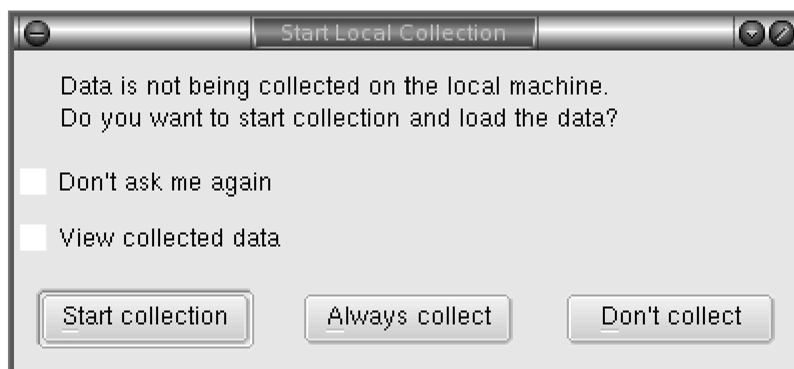
4 Clockwork: The Collection Agent

4.1 Starting

If installed from an *RPM* package, *clockwork* will already be running and will restart every time the host is booted. When *ghabitat* starts either locally or on a peer machine, it will be able to pick up data generated from the daemon. This is the most convenient way of running habitat.

If data is not available or *ghabitat* pops-up a window asking to start collection, then *clockwork* is not running. This will be the case if habitat was installed from a *.tar* package. If a full installation is required for data to be collected by the system for the benefit of all, see the Administration Manual to start up the daemon.

If the data is for you own benefit only, then the GUI *ghabitat* is all you need. The tool will ask to start collection if it is not already being done, using the pop-up below:-



Clock on **Start collection** to start *clockwork* just this once, or **Always collect** it *clockwork* should be started when ever *ghabitat* runs. This is not a substitute for system-wide collection, for which you should see the Administration Manual. Two other buttons on the pop-up ask to inhibit the automatic pop-up of collection windows and to view collected data in a separate entry in the choice tree.

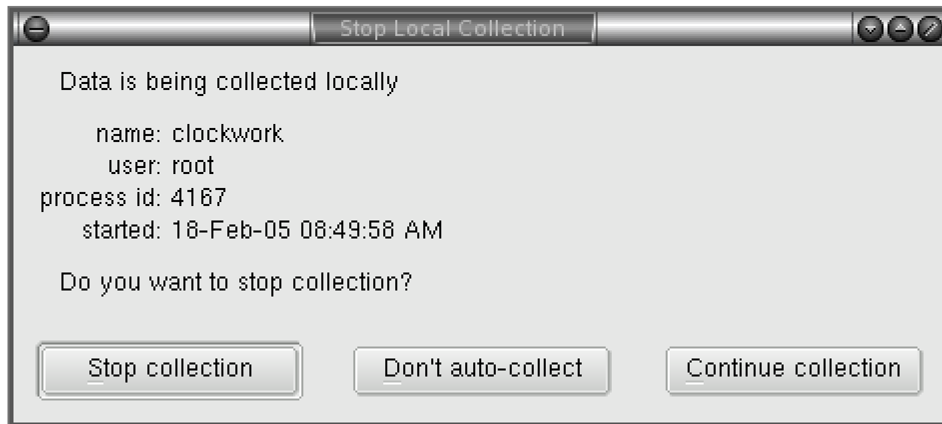
If the collection is stopped for any reason or the pop-up has been inhibited by choice, it may be restarted by selecting **Data->Local Collection** and using the pop-up.

Only one central instance of *clockwork* can run, but there is no limit to the number of user instances running specific set up collection jobs. See the section later in this document about starting custom *clockwork* instances for you own data collection projects.

4.2 Stopping

Using *ghabitat*, select **Data->Local Collection** from the menu bar.

A pop-up similar to the one below will appear



From this window, one can stop collection, continue with it or change *ghabitat's* auto-collection setting (see above). To continue with no change, click 'Continue collection' or press the *Esc* (escape) key on your keyboard. If collection is stopped then it may be restarted again with the same menu sequence.

4.3 Status

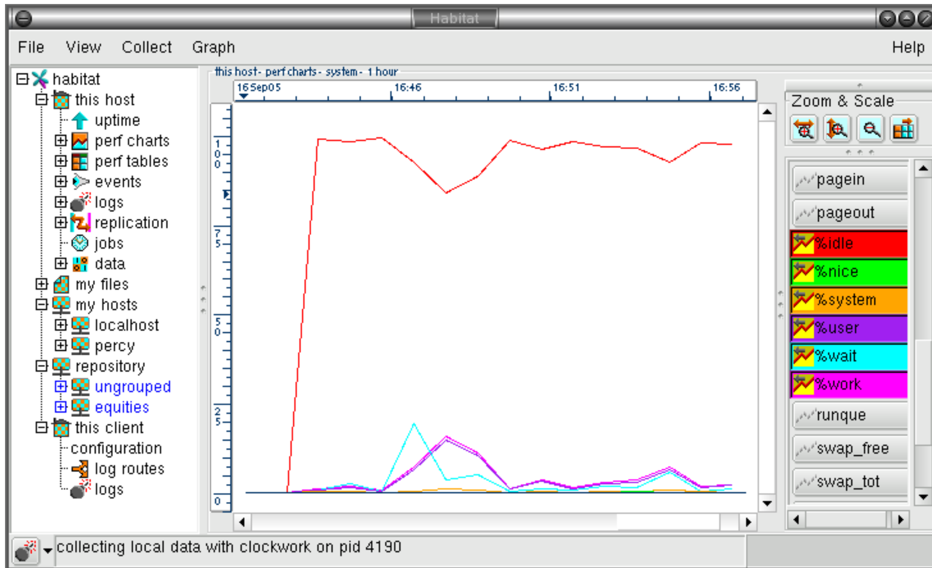
To check if collection is running locally, select **Collect->Local Data** from the menu bar. As shown in the image from the Stop Collection section above, the pop-up will show if the collection process is running and if so, who started it, when and its process ID.

To exit without changing the current state, press the *Esc* (escape) key on your keyboard or click 'Continue collection' (or 'Don't collect' if not running).

5 Graphical Tools

Habitat's main graphical tool is *ghabitat*, which can be started by typing **ghabitat** on the command line or selecting **habitat** from the start button of the graphical desktop. Data collection needs to be started from the GUI or preferably at system level using *clockwork* directly. The manual pages towards the end of this user guide show the options that can be used for launching both *ghabitat* and *clockwork* the collector. Data collection is discussed in the section above.

When running normally, a display similar to the one below will be seen, which shows around 12 minutes of data.



5.1 Data Visualisation

All manner of data can be collected by habitat. However, the default information displayed in the *ghabitat* file is shown in the table below

Uptime	When the system was last booted, plus some other information
System	Key system information, containing the processor and memory statistics.
Symbols	Entries in the form of key-value pairs that all operating systems generate when they configure
Storage	Storage capacity and performance
Network	Network performance
Interrupts	Hardware interrupts
Events	The events generated by pattern watching and crossed thresholds

These choices may be found by following the following path in the choice tree:

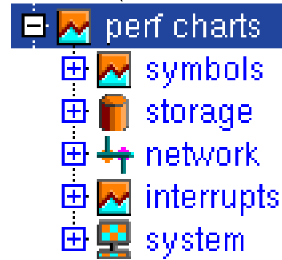
habitat->this host->perf charts

More data is collected and this can be seen by selecting the **data** node in the choice tree. This view is low level and is used when it is important to see uninterpreted data.

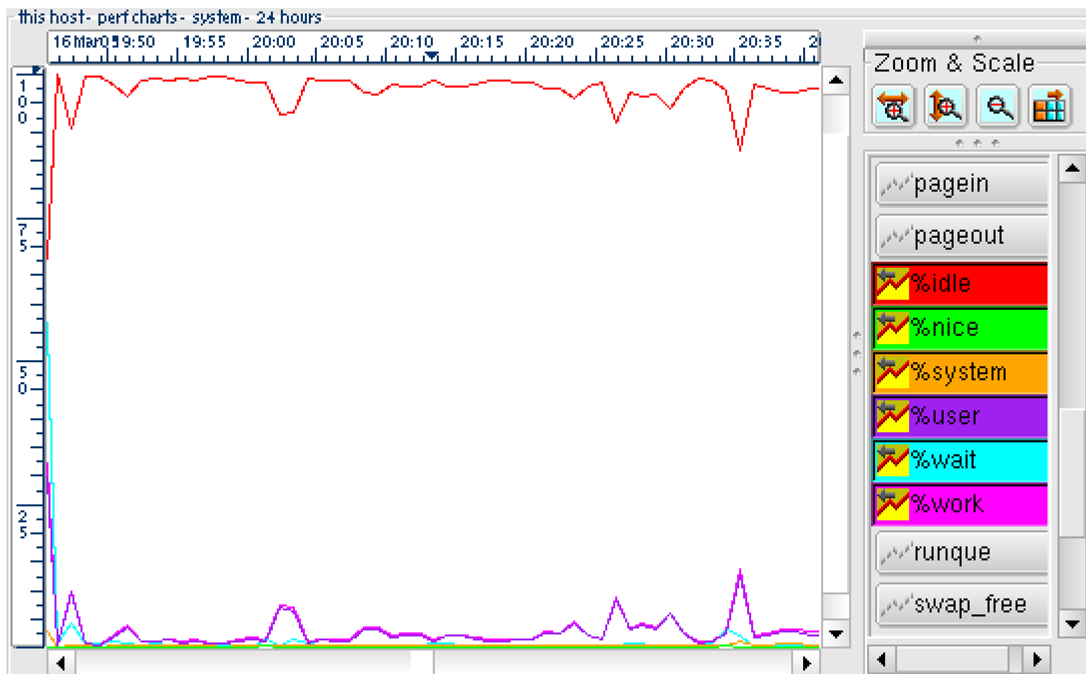
An detailed list of the performance metrics and their explanation is help in an appendix

5.1.1 Data In Charts or Graphs

In the current version of habitat, performance data is charted when **perf charts** is selected from the choice tree. Once selected, the performance data choices (shown in the section above) appear as branches.

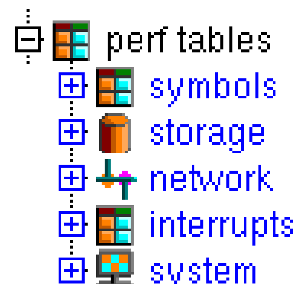


Once the leaf node in the choice tree has been chosen, the visualisation pane will change from the current display (which may be the splash screen) to a chart holding data values and a set of visualisation and navigation controls. A default set of data curves are displayed, which may be changed by using the controls to the right of the chart area. The manipulation of the data is explored below.



Please note that the chart will be updated periodically whilst it is displayed. The frequency of update depends on the time scale chosen and can be updated manually with the **^L** key or the menu bar choice **View->Update View**. Data is appended to the current view.

5.1.2 Data In Tables



In the current version of habitat, performance data is displayed in a table when **perf tables** is selected from the choice tree. Once selected, the performance data choices (shown in the section above) appear as branches. This is identical to the display of data in a chart as shown above.

Once the leaf node is chosen, the visualisation area switches to a tabular display of the selected data. Moving the mouse pointer over the column title will cause a pop up to appear with an explanation of each data attribute (where given).

_time	load1	load5	load15	runque	nprocs	lastproc	mem_tot
08-Mar-05 09:17:02	[illegible]	[illegible]	[illegible]	1.00	139.83	5527.33	515528.00
08-Mar-05 09:22:02	[illegible]	[illegible]	[illegible]	1.00	140.00	5723.20	515528.00
08-Mar-05 09:26:02	0.00	0.01	0.11	1.00	139.75	5848.25	515528.00

5 minute load average

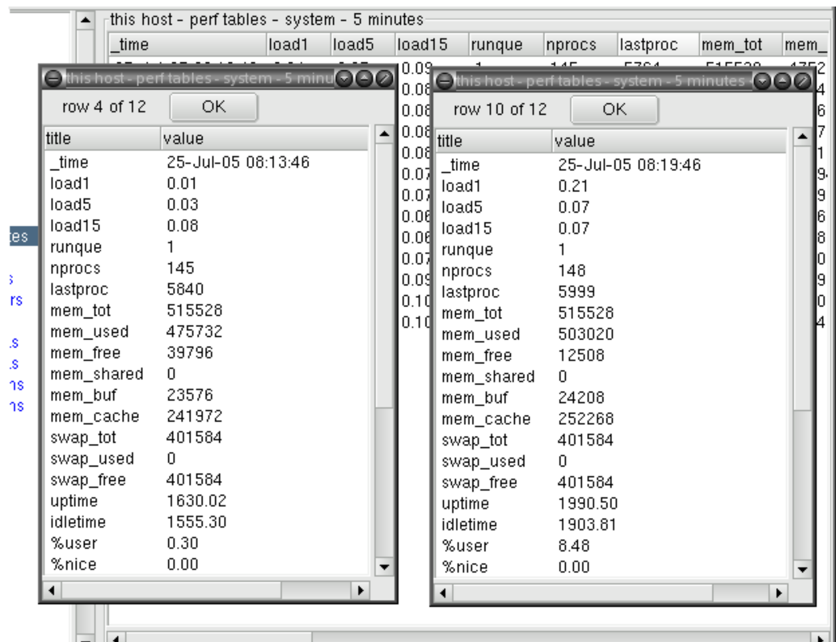
The data can be panned horizontally and vertically as expected using the scroll bars.

5.1.3 Data In Row Popups From a Table

Some times it is useful to see a row of data as a column, especially when dealing with many wide columns. By double clicking on a row in the tabular display of data, a column of data is displayed in a separate pop-up window.

Additionally, it is also useful to see one row of data whilst looking at another, which may not be easy to out of the table context.

Pictured below is an example from the **system** probe, where two popups taken from different rows are shown next to each other.

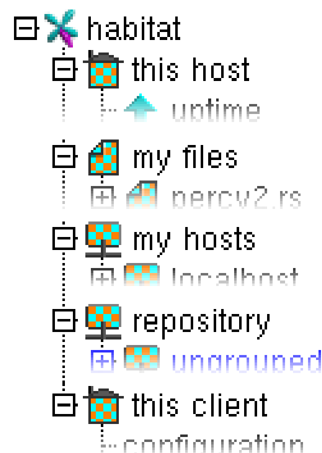


5.2 Data Navigation

5.2.1 Finding Data Sources

The choice tree, held in the left-hand area of the *ghabitat* tool, represents many of the features of habitat and each time one is selected, the visualisation area will be changed to show that data.

The first level of organisation is shown in the diagram below, showing a compressed view of the tree's top level.



The nodes are as follows:-

this host Information coming from the local host, via the network connection to *clockwork*. If *clockwork* has been configured to disable network requests *(clockwork -s*) or if a private instance is use, then this node will not be populated and the data may be accessed by file under **my files**

my files Files that can be read by *ghabitat* will be placed under this node. Open them with the menu bar option: **File->Open** and formats that can be read include *ringstore, CSV* and *plain text*. Files may be closed with **File->Close** from the menu bar and will be remembered next time *ghabitat* is run

my hosts Hosts accessed using the option **File->Host** from the menu bar will be placed under this node. Two type of access are possible: *direct*, which gets data from the monitored machine and *repository*, which obtains data indirectly from an archive

repository If a repository is configured (see appendix and Administration Manual) then this node will be populated with the organisational hierarchy taken from the harvest repository. Navigate through the organisation tree to find the host entries

this client The workings of the *ghabitat* client being used. Currently holds the configuration, log routes (log destinations) and the logs directed to the local client

Each of these choice tree paths will reach a data source, such as a file, a host, repository entry or meta source. If the choices are dynamic (itself driven by data), then the entires are coloured blue and may be updated with **View->Update Choice** option from the menu bar or the **F9** key. Automatic updates will occur periodically.

5.2.2 Source Exploration

Each data source provides arbitrary entries, which may be used to expand the functionality of habitat. However the following section describes the sources which identify themselves to *ghabitat* as conventional and have meaning to the habitat suite.

The tree above shows a typical list of the sources visible from each full data source, such as a *ringstore*. In the example above, it is the local machine that runs your *ghabitat* and named **this host.** Each release of habitat is likely to alter the organisation of choices as functionality is optimised. An explanation of the nodes are as follows:-

uptime	The amount of time that a system has been running
perf charts	Performance data visualised by charts
perf tables	Raw performance data shown in tables
events	Events from monitoring data for patterns or thresholds
logs	Logs generated from the running of data collection jobs
replication	Replication state and logs when enabled with a repository
jobs	Job table, used for *clockwork*
data	Raw view on all the data stored

Not all the nodes at this level will be populated. For example, if there was no replication taking place, the entries for it would not appear in the tree. If the *ringstore* was not generated by the standard clockwork job table, then it is likely that only the performance data would be shown; the job table causes many data sources to be run and collected in the same datastore. Flat or CSV will typically have only two choices, corresponding to charts and tabular viewing.

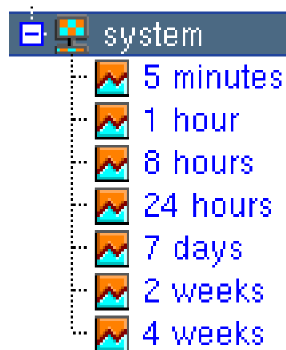
Once a source has been selected, the section below shows how the view can be tailored to show the correct data set.

5.2.3 Changing Timescales

Each row collected by habitat has an associated time stamp. When examining the data source, *ghabitat* pre select typical data views based on time scales. These are typically values such as **5 minutes**, **1 hour**, **24 hours**, **7 days**, ranging all the way to **30 years**. The more history there is in the data source, the great the time scale to display it.

When one of these time scales nodes are selected, a corresponding chart or table view will be drawn. Clicking on different time scale nodes in the choice tree will cause different amounts of data to be displayed.

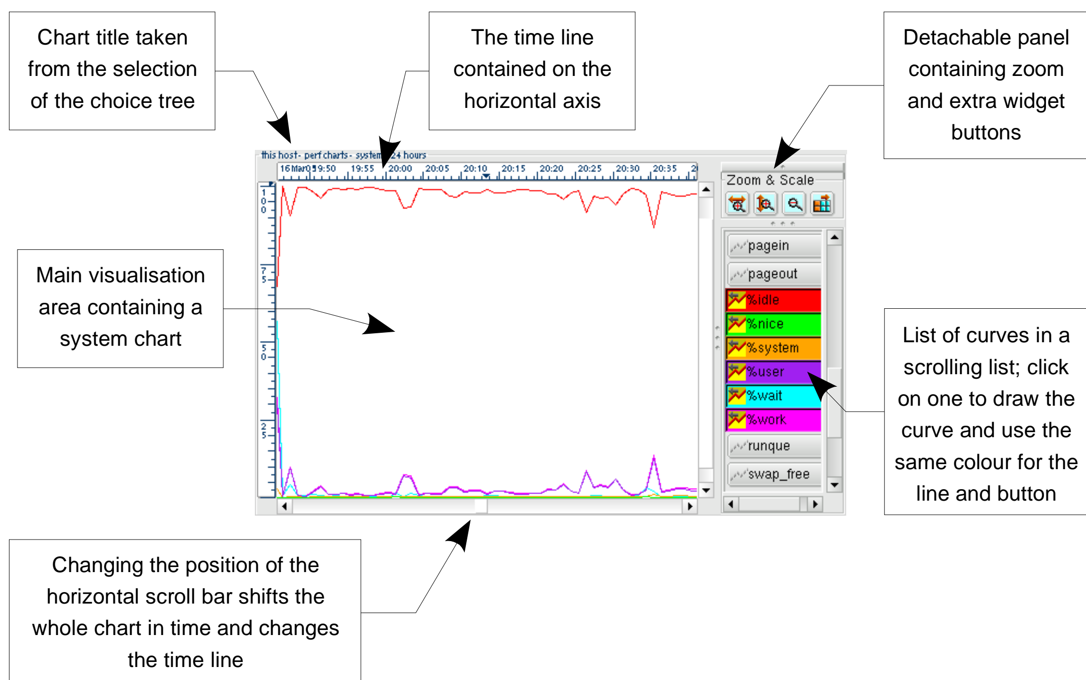
Note that the greater the time viewed, the longer it will take to retrieve from the source, the more memory used and the busier the display.



See the Zooming section to customise the views or look at interesting parts of the chart.

5.2.4 Selecting Curves

When viewing a chart, the visualisation area is split into three boxes, all of which are resizeable. The main one contains the chart itself, framed by a heading and rulers showing time and value. The smallest box contains buttons used for zooming and scaling (see the later section), leaving the remaining box which contains the curve list.



The curve list is a scrollable set of buttons that control what is drawn on the chart. Each column in the collected data may be turned into a plotted line, providing it has numeric values. Hovering the mouse pointer over a button will return the attribute's information in a tooltip. Clicking on a button changes its colour and icon and a line is drawn on the chart in the corresponding colour.

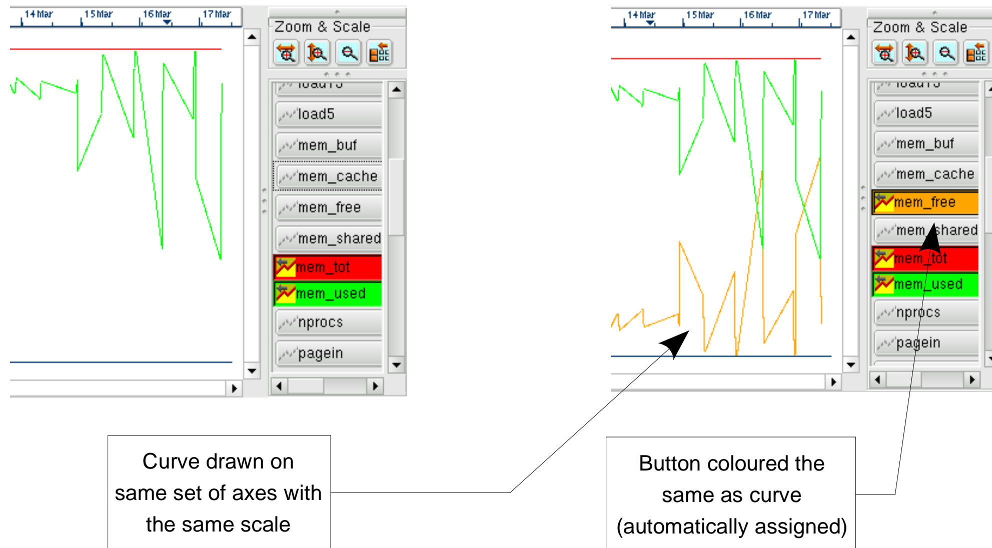


Illustration 7: Before and after: drawing a new curve on a chart with an existing set. A colour is automatically assigned and used to draw the curve line and tint the button for identification

To remove the line from the drawing, click the button to deselect.

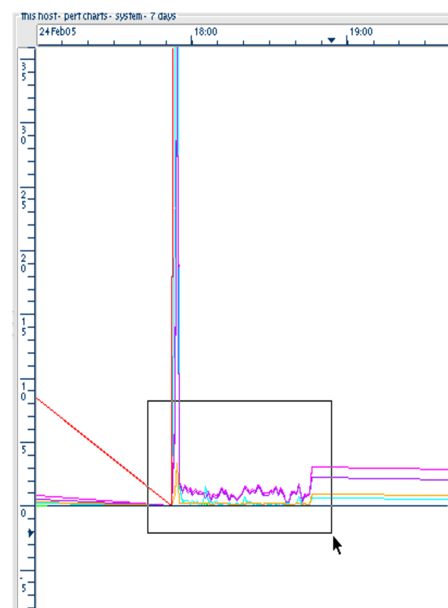
When a standard choice is selected, a default set of curves are drawn. However, when the curve selections are changed, it will be remembered when the choice is shown again or the same view is shown in another data source.

5.2.5 Zooming and Panning Graphs

One of the most useful parts of *ghabitat* is the ability to zoom into a displayed graph and for the rulers to be intelligently redrawn at the right scale.

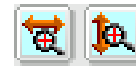
There are two ways to achieve this:-

1. Drag the mouse pointer over the interesting part of the chart whilst pressing the left mouse button. A box will be drawn over the area which should be clicked inside, again with the left mouse button. This will redisplay the graph using the dimensions of the box. The current aspect ratio will not be maintained, so that it is possible just to expand the time scale, whilst leaving the full range of values on the y-axis



2. Alternatively, click on one of the two zoom

buttons in the **Zoom & Scale** box. These will increase each dimension (the x or y axis independently) by around 50% and position the display in the middle of the previously displayed values.



Once the display is zoomed, one may pan around the chart using the horizontal or vertical scrollbars. Selecting or deselecting curves whilst zoomed will not cause the scale or position to be reset: new values will be superimposed with out resetting the current view.

To zoom out, either click on the right mouse button, or select the zoom out button. Either of these will zoom out of the current view in both dimensions by about 100% each time. Clicking the right mouse button or *zoom-out* GUI button:

several times will result in the display being returned to the full view of the chosen time scale.

5.2.6 Adapting Curves

Some attributes have such large values that the y-axis is unable to cope with the sheer length of the labels. Alternatively, if two attributes are drawn with radically differing values, all the contours of the smaller valued curve will be lost in a flat-ish line at the bottom. Examples are shown below

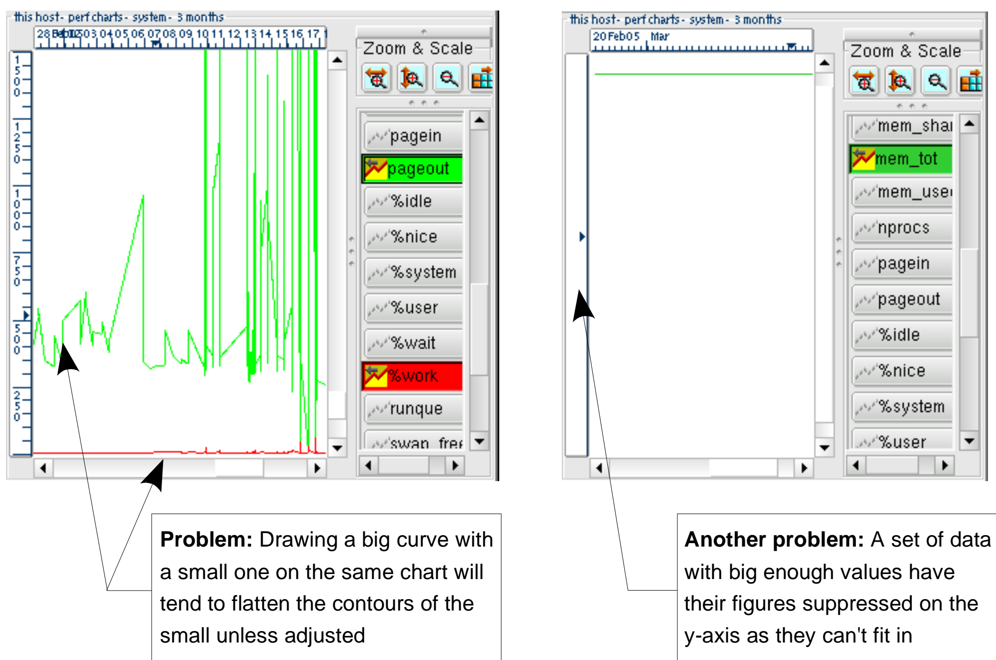
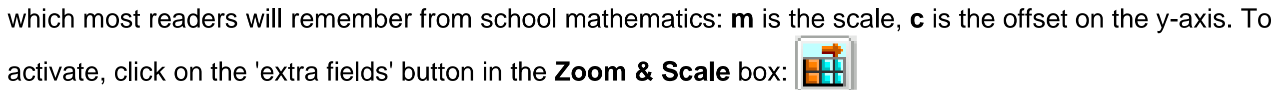


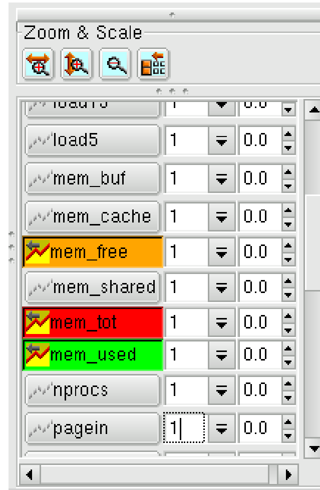
Illustration 8: Two examples of problems when using curves with large values. The image on the left showing a flattening of the smaller valued curve and on the right being too big to place values on the y-axis

For both these situations, *ghabitat* has the ability to alter the scale and offset of any curve, using the formula:

y = m x + c

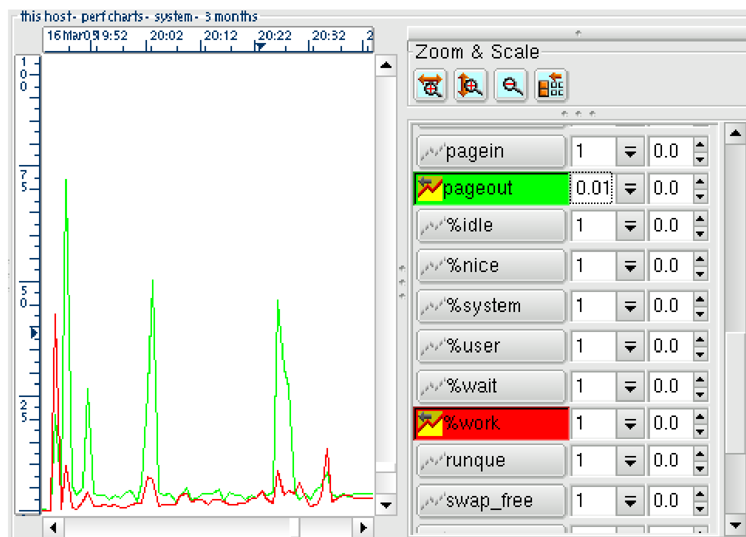
which most readers will remember from school mathematics: **m** is the scale, **c** is the offset on the y-axis. To activate, click on the 'extra fields' button in the **Zoom & Scale** box: 

This will cause the curve selection list to expand, similar to the list below:



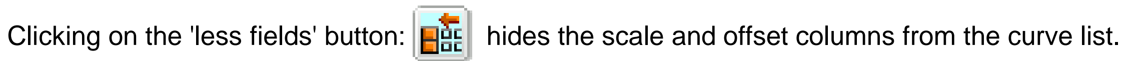
Locate the scale field of the curve that you wish to make smaller, and type in a magnitude less than 1.0 or use the pull-down to select a predetermined power of 10 (initially the best approach). The graph rescales with the values of the big curve reduced to manageable proportions. Other curves squashed down at the bottom are able to expand and show their contours compared to the giant.

The diagram below shows one of the previous examples but scaling the **pageout** data by 0.01 (dividing it by 100). The two curves fit into the same scale and can be displayed on the same chart without loss of detail.



In illustrating performance issues, it is very useful to show a small scale item such as % cpu utilisation (maxima of 100) on the same chart as memory usage (which may have a maxima of 32,000). Scaling the memory to a factor of 0.01 or 0.001 (100 or 1,000 times smaller) would achieve this and is shown in the example above.

In addition to scaling, an offset may also be applied to the values of the curve. The offset moves the curve vertically against the y-axis.

Clicking on the 'less fields' button:  hides the scale and offset columns from the curve list.

5.2.7 Custom Graphs

Custom Graphs are not currently in operation

5.2.8 Selecting Instances

Some rings involve multiple instances, such as storage or networks where the data describes multiple devices. For example, individual network interfaces or storage devices. When these rings are displayed, the selection list is joined by a scrollable instance list or check boxes. By default, the first instance is selected.

Selecting additional instances splits the chart display area evenly and draws graphs of the chosen items in each. Clicking on the curve selections draws and erases the lines on all the split graphs simultaneously.

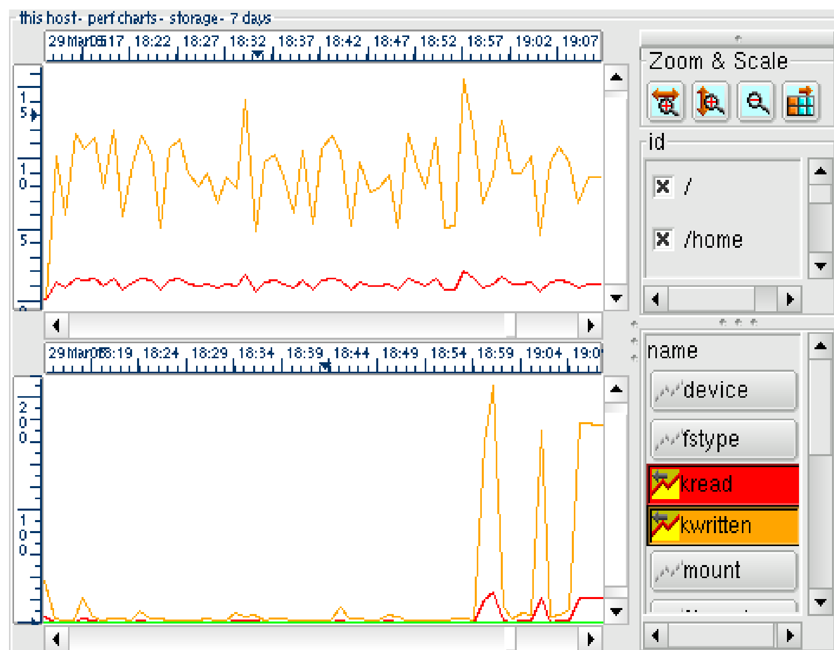


Illustration 9: Two instances are chosen from a storage ring: / (root) and /home, resulting in two sets of graphs. Each is drawn with two curves: kread and kwritten to show the volume of data transferred in both directions

Instance graphs are removed when the corresponding instance is deselected.

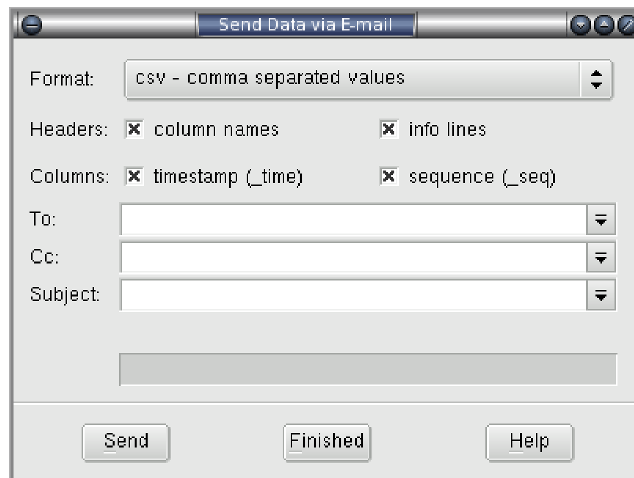
5.3 Import and Export

The conventional format for information within *ghabitat* is the *ringstore*, but data can also be directed to other formats and uses.

5.3.1 Email

When data is displayed with a graph or in tabular chart form, it can be sent to other users by e-mail. The

format will be *Comma Separated Values (CSV)* or *Fat Headed Array (FHA)* with options to alter information's appearance. Select **View->Save Viewed Data...** from the menu bar and the following pop-up will appear:

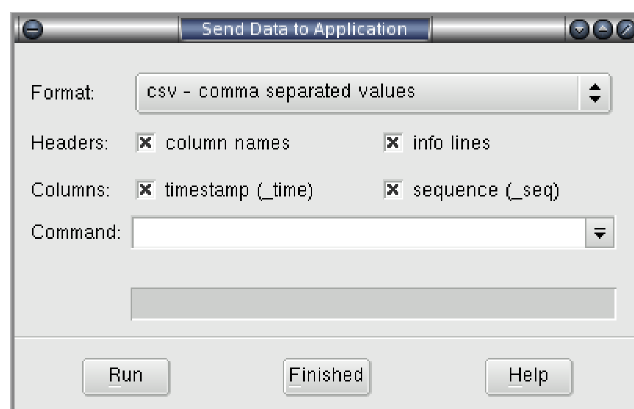


Two formats are currently available from the **Format:** widget as described above. The selection boxes in the **Headers:** line govern the printing of column names and *info* lines, the latter of which may be multiple lines terminated with the special '--'. **Columns:** select the printing of the *_time* and *_seq* columns for time stamp and duration. The **To:** and **Cc:** fields should be valid address lists, **Subject:** will identify the email.

To send the email of data, press **Send** and *ghabitat* will use the system specific email command to dispatch the data. The progress of the operation is shown in a status line above the buttons. The window stays up to deal with any problems and can be dismissed with the Finished button.

5.3.2 External Tools

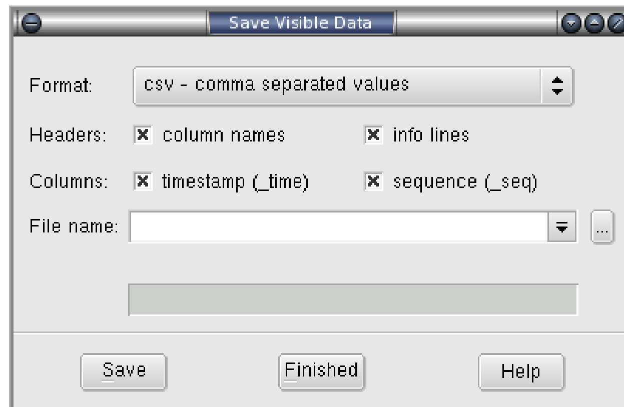
The data sent by e-mail above can also be forwarded to an external application, such as a spreadsheet application or other visualisation package. Select this features with **View->Send Data to Application...** from the menu bar. Data is sent using a pipe to *stdin* or the launched application.



The data fields are identical to the e-mail pop-up above, with the exception of **command:** which should be a standard command line valid on the host system. Progress will appear in the status line above the buttons and the pop-up is dismissed with the **Finished** button.

5.3.3 Interchange Files

Data from *ghabitat* may be simply exported to a file, using the pop-up selected by **View->Save Viewed Data...**



Data is written to a file name with the **Save** button, showing the progress in the status bar and dismissing the pop-up with the **Finished** button.

5.4 Data Files

Several type of data file that may be read and written in *ghabitat*. This section details the operations that can be carried out with them.

5.4.1 Saving Data in Files

Currently, two basic formats may be saved: *Fat Headed Arrays* (*FHA*) and *Comma Separated Values* (*CSV*). However, each can be modified to look similar to each other by customising the headers or columns. The major difference between the two is that the *FHA*'s values are separated by a single *tab* character (\t or ^I) not a comma.

Data should first be selected from the choice tree and scoped using the time scales. Regardless of the curves or instances being saved, all data covering that time will be used. Then three options are available to save data, all access from the menu bar.

1. **View->Save Viewed Data...**
2. **View->Send Data to Application...**
3. **View->Send Data via E-mail...**

Each of these options provide the opportunity to customise the appearance of the written data, then ask for specific ways in which to output. These forms are shown and described in the sections above.

5.4.2 Opening Data Files

Files may be opened for browsing using the **File->Open...** option from the menu bar and the resulting file chooser pop-up. An attempt is made to load the file using several different formats, starting with the one containing the most information, moving to the next one if the load fails.

The following formats are supported:

Ringstore	Conventionally end in *.rs* and contain many rings of time store data. The rings will appear under the file name label in the choice tree

FHA Fat Headed Array is an enriched version of a CSV files and is treated as a single ringed data source. The files conventionally have the extension *.fha.*

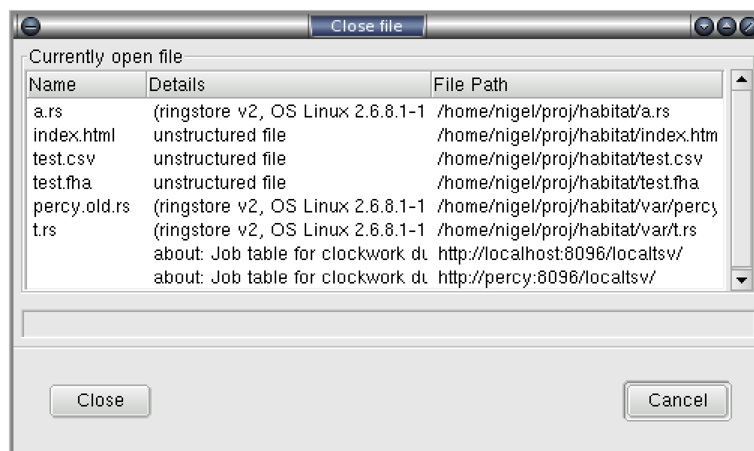
CSV Comma Separated Value files are treated as single ringed data sources. The files should conventionally end in *.csv* and the alternative *.tsv* if the values are tab (\t or ^I) separated.

Text If *ghabitat* is unable to scan the input file as a table, it will treat the contents as plain text and allow its simple visualisation.

All files read in this way will have an entry created for them under **my files** in the choice tree. Each file types gets a different icon to help distinguish the types.

5.4.3 Closing Data Files

To close a file of any type, select **File->Close...** from the choice tree. This will display a file de-selection window, containing existing file loads.



Click on the line containing the file to remove and then the **Close** button. This will remove the file from the choice tree and dismiss the window from view.

5.5 Data Access from Peer Hosts

Data from other instances of habitat running on other hosts may be mapped into the choice tree, in addition to local data (under the node **this host**) and file sources (under **my files**). Several of these options are also explored in the section 'Finding Data Sources' earlier in this manual.

5.5.1 Peer Data over Filesystem

In addition to historic or experimental data, files may also contain dynamic data that are periodically updated from *clockwork* instances or direct insertions (from *habput* for example). If data is created by other systems, then the information may be shared using a file sharing protocol such as NFS or CIFS (SMB). The producer writes to the file on their system, the data is held on a file server and *ghabitat* on another system reads it.

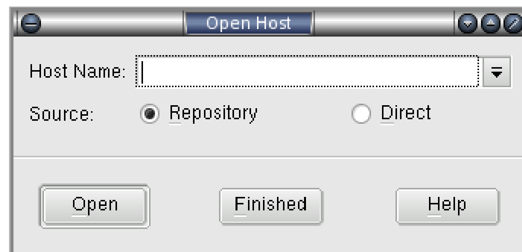
Data created in this way are loaded in *ghabitat* like static files, using **File->Open...** and made available under the label **my files**. They are removed using **File->Close...**. File based dynamic data can only be stored in a *ringstore* format.

This method of access is useful, but can suffer from bottlenecks due to file locking. When the produce stores the data, the file will be locked momentarily. Similarly, then the reader access the file, it is unavailable for

writing. Consequently, it is most useful for slower data collection or infrequent ad-hoc use.

5.5.2 Peer Data over Network

The recommended way to access data from peer systems, is by network which uses an *http* based protocol. From *ghabitat*, remote systems can be attached by using the option **File->Host...** from the menu bar. From the pop-up (shown below), type the network host name and select the **direct** option.



This will attach the storage to the choice tree under the **my hosts** node. From here one can navigate the choices like it was on the local machine.

5.5.3 Data from Repository

The disadvantage with direct access is that the machine subject to monitoring will have to spend part of its time servicing the clients that have asked for performance data. Instead a more considerate option is to query the repository instead, which obtains data indirectly from the host. The current repository is provided by System Garden Harvest, and is an archive of Habitat's replicated feeds augmented with directly imported data and computed information. The use of a repository is advised for medium or larger sites.

Repository data is attached in the same way as direct access above, except that the **repository** button should be selected. Again, the host is placed in the choice tree under the **my hosts** node.

5.6 Graphical Viewer Information & Configuration

The graphical tool itself is configurable and uses the same system of control as other tools in the habitat suite. This section shows how configuration is used by the viewer and how to display information on this and the state of the ghabitat application.

5.6.1 Configuration Files

All habitat programs read the same set of configuration files before they start, and for most users the *~/.habrc* file is the most convenient to use.

(Configuration may also be set on the command line 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 *ghabitat* 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 *ghabitat* is not running. If that is not possible, you should save the contents before *ghabitat* exits or before it starts. The application will only update the lines that match the property being updated, leaving everything else alone (comments, user settings, etc).

Some common values are shown in the Administration Manual. You may be asked to add values on the advice of an administrator or System Garden support.

5.6.2 Data Source History

When *ghabitat* exits normally, it edits the personal preference file *.habrc* in the user's home directory (*~/.habrc*). This information is used to reload files and hosts on a later invocation of *ghabitat.* It updates the lines containing the following keys:

ghchoice.myfiles.load	The list of files that should be loaded the next time *ghabitat* is started
ghchoice.myfiles.list	The list of files that should be provided for convenience in a pull-down menu when selecting files
ghchoice.myhosts.load	The list of hosts that should be loaded the next time *ghabitat* is started
ghchoice.myhosts.list	The list of hosts that should be provided for convenience in a pull-down menu when selecting files

All other lines are left alone, allowing them to be used for personal configuration (see section below).

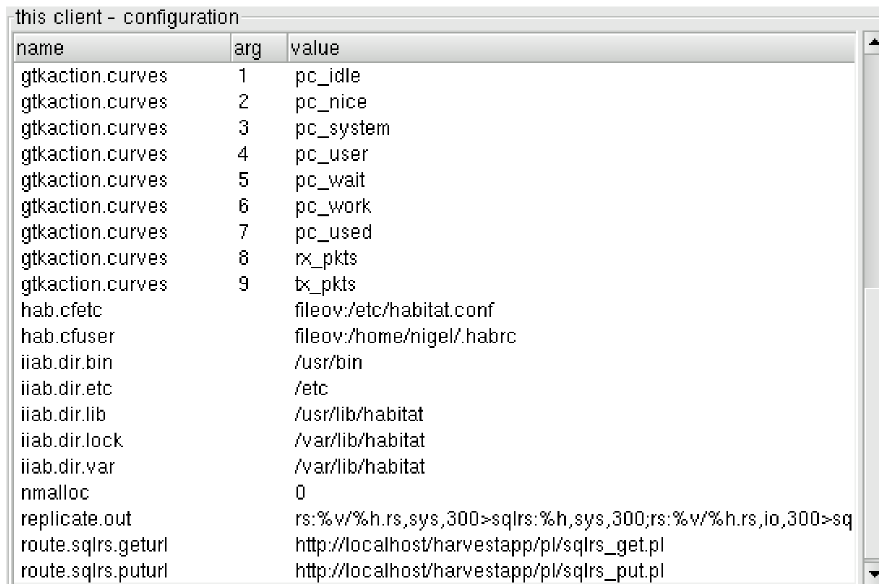
Currently, if *ghabitat* is terminated abnormally, then the configuration file is not updated.

5.6.3 Viewing Current Configuration

Under the choice label **this client** will be three choices: configuration, log routes and logs. This cluster of choices shows the current state of the ghabitat viewing tool, as shown in the image below.

Selecting **this client->configuration** from the choices will display the current configuration of the client in the visualisation area to the right. Configurations are compiled from several sources and are a mixture of personal preference and system-wide directives augmenting ghabitat's internal settings. The mechanism is discussed in an appendix in this document and also in the Administration manual.

Configurations are displayed in a three column table: name, argument and value. For scalar values (for example, a single string or integer) the argument value is blank. However, when values are arrays, multiple rows are used with the same name and the argument column is used to distinguish between them. An example configuration is shown below.



this client - configuration

name	arg	value
gtkaction.curves	1	pc_idle
gtkaction.curves	2	pc_nice
gtkaction.curves	3	pc_system
gtkaction.curves	4	pc_user
gtkaction.curves	5	pc_wait
gtkaction.curves	6	pc_work
gtkaction.curves	7	pc_used
gtkaction.curves	8	rx_pkts
gtkaction.curves	9	tx_pkts
hab.cfetc		fileov:/etc/habitat.conf
hab.cfuser		fileov:/home/nigel/.habrc
iiab.dir.bin		/usr/bin
iiab.dir.etc		/etc
iiab.dir.lib		/usr/lib/habitat
iiab.dir.lock		/var/lib/habitat
iiab.dir.var		/var/lib/habitat
nmalloc		0
replicate.out		rs:%v/%h.rs,sys,300>sqlrs:%h,sys,300;rs:%v/%h.rs,io,300>sq
route.sqlrs.geturl		http://localhost/harvestapp/pl/sqlrs_get.pl
route.sqlrs.puturl		http://localhost/harvestapp/pl/sqlrs_put.pl

In the example, the directive **gtkaction.curves** has a nine element array as a value: pc_idle, pc_nice, pc_system, ... etc.

The specific configuration of ghabitat is also available by running the application with the diagnostic (-d) or debug (-D) mode on the command line. In these modes, logging is set to be more verbose (diag or debug level) and output sent to stderr, which is overridden from the normal settings. The ghabitat manual page in the appendix has more information.

6 Text Terminal Tools

As a substitute for *ghabitat*, the habitat suite also contain a *curses*-based utility called *track*. It purpose is to provide an assisted method of data browsing using only dumb-terminals or their emulators such as *xterm*, *kconsole* or *gnome-terminal*.

6.1 Track

Track is not currently distributed. Please check its status with System Garden.

7 Command Line Tools

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 *ringstores* 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.

7.1 Common Arguments

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

-c *croute*	Append user configuration data from c*route* (*route* addressing format), rather than the default file *~/.habrc.* For example, **-c 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.

7.2 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.

7.3 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 minload  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 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:*.

7.4 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.

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.

7.5 Clockwork & killclock

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 *ghabitat* 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.

7.6 /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.

7.7 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
irs	Interactive ringstore utility, allowing administration of the data held in *ringstore* files

8 System Performance

Data gathered by *clockwork's* built-in probes are designed to aid in capacity planning and problem resolutions. Usually, that relies the trend in consumption of major resources: processor, memory, storage and networks. However, other minor indicators can also be used: highly system specific but very important.

For certain situations, log monitoring and event recording are also important, especially when showing data coincidence.

8.1 Indicators

The main indicators of capacity usage: processor, memory, storage and networking are handled in a very similar way across Unix-like systems (although memory management will vary more than other resources). In all systems, these measurements can be represented in a similar way and are the primary indicators of the capacity of a given machine. However, these should always be read with the in conjunction with the system specific indicators.

8.1.1 System

The processor and memory statistics are collected by the same probe and appended into a ring named *sys*, which appears as the leaf node **system** in the choice tree.

A full list of data that collected (at the time of writing) is held in an appendix, however a few notable indicators are described below.

Habitat's *sys* probe calculates a *%work* value (0-100 range), which indicates how much time is spent processing in all categories. In Linux this is *%user+%system+%nice* (user time, kernel time and low priority user time). In Solaris, this is just *%user+%system*.

%idle in Linux 2.4 shows what proportion of time is left over after processing has been completed; in Solaris and Linux 2.6, one needs to add *%idle+%wait* together to get the same figure. *%wait* stands for *wait I/O*, the amount of time that processes wait on blocked I/O.

The indicators *load1*, *load5* and *load15* report the 1, 5 and 15 minute load average computed by the Unix kernel. This is a traditional value of overall load that almost all Unix-like operating system report and is an exponential decline function on the number of runnable processes

Users of habitat should refer to operating system specific texts to better understand the meaning behind the indicators.

8.1.2 Storage

Storage statistics are collected by the *io* probe and appear as **storage** under the choice tree. Both capacity information (how full your disks are) as well as performance information is given in the same probe

A full list of data that collected (at the time of writing) is held in an appendix, however a few notable indicators are described below.

Capacity information is given by *size, used* and *reserved*, which fits the Unix model of reserved storage. *%used* is also calculated by the probe to shown what proportion of *(used-reserved)* taken. 100% used means all user space on the device has been taken, leaving only the *reserved* for administration working area.

Performance information is given by read and write operations and storage transferred. For systems that support it, service time for read and write is also offered, which can be very helpful in working out service levels.

The storage ring holds multi instance data, that is, each device can provide the same measuring characteristics if they are available. In *ghabitat*, this manifests as an additional scrolling list to select the devices to display (as described above).

8.1.3 Network

Network data is collected by the *net* probe and appears as network in the choice tree.

A full list of data that collected (at the time of writing) is held in an appendix, however a few notable indicators are described below.

Data collected is split between read and transmit statistics, known by their identifier prefixes *rx_* and *tx_*. Typical indicators include packets (a measure of throughput), total bytes, errors (malformed data), and collisions (high for busy shared Ethernets), etc.

A system has multiple interfaces, even if one of the is a *loopback* (typically lo0). Thus, when displaying network data in *ghabitat*, the multi interface mode operates and multiple interfaces may be selected for drawing.

8.1.4 Other Indicators

Other than the *probes* that collect the primary indicators, habitat also collects other data.

Four co-operating *probes*, named *up, down, boot* and *alive,* collect availability data which is displayed in the choice tree under the label **uptime**. Other than collecting some system specific information, the probes show when the system was last booted and create a history of down time that can be used in service levels.

Each operating system has a set of parameters which describe the operation of its kernel and the configuration. This is collected by probe called *name* which presents it data as **symbols** under the choice tree. The probe runs each time that *clockwork* started to collect the current configuration in the form of a simple key-value list.

Hardware interrupts are collected by a probe named *intr* and presented as **interrupts** under the choice tree. It shows interrupts of various sorts against real or synthetic devices. This measure can be quite system specific.

8.2 Adding to the standard data

New data is easily added to habitat and is covered in Administration and Programming manuals.

Conceptually, once the chosen data is collected in the correct format (FHA generally), it needs to be appended to its own ring in a habitat data store (using *habput* or programmatically using the *route* interface). Once there, it can be displayed using *ghabitat* in tabular or graphical form under **data** in the choice tree.

Ringstores should be used unless *harvest* is installed, when SQL ringstore (*sqlrs:*) also becomes an option. It is also possible to use a ringstore and replicate to *sqlrs:* which is another configurable process and one employed by habitat as the most convenient method.

8.2.1 Synthesising New Values

In addition to data that is recorded directly to a ring table following its measurement, a collecting probe may also *synthesise* its own values. This may be to abstract a measurement from system specific indicators or to make a more useful measurement, usually combining several native values.

This data is recorded at the same time as the native values in its own column, so that it shares the same

sample time and sequence. However, it is advised that synthetic data is indicated as such in the *info* field describing the column.

In the system probe, *%work* is an example of *synthesised* data.

8.3 What is Abnormal?

This User Manual does not attempt to provide a guide to interpreting performance characteristics, which is a significant subject in its own right, with many tests dedicated to it.

However, certain things to watch out for include: in the *system* ring, prolonged use of processors (unless by design), high paging and any significant swapping. In the *network* ring: high error or collision rates, and in the *storage* ring long service times or high usage resulting in little free space.

8.4 Further Reading

9 Events

In addition of collecting tabular performance data, habitat also has the ability to monitor data and files for patterns and thresholds, executing arbitrary jobs as a result. If the jobs generate additional logs, then it provides a complement for polled tabular data and is able to put these details into context.

It is also possible to use this mechanism to execute external processes, send email, pager or SMS messages or to carry out a specific set of data collection. In this way, habitat can change its job profile dependent on the statistical information it finds.

9.1 Event Queue

Events in habitat are used for creating jobs and are the culmination of watching data for patters, threshold crossing or direct injection.

When an event is raised by watching patterns or thresholds, it is appended to a single event queue. An event tracker job (using the clockwork method *event*) picks up new instructions and executes them. Once executed, the new state is stored so that events are not replayed if *clockwork* is restarted.

All job methods available in *clockwork's* job mechanism may also be used in event processing. See the event table format later in this document or the Administration manual for more details.

9.2 Watching Jobs

Data is watched by a set of tasks held in the *clockwork* job table. Each has a checking frequency and associates a set of named watch sources (see below) with a set of named patterns and if matched, their actions (see below).

The job method called by *clockwork* is *pattern*, which takes two arguments. This first is the *patter-action* route list, a route to a list of patterns and the second is the *watch* route, a route to a list of routes to watch.

When the pattern is matched and the embargo conditions allow, then an event is composed using information from the pattern-action table. This comprises a set of instructions

See the administration guide for more information.

9.3 Watched Sources

The sources that are watched are held in a simple list of routes, conventionally called *watched,0* (fully rs:<*hostname*>.rs,watched,0). This table is normally made available in the choice tree under **events->watched sources**. Multiple named watch lists can exist, used only when referred to by a watching job. When ever a watch list is updated, it will be reread by the watching process without the need to restart *clockwork*.

9.4 Pattern-Action Data

A table of pattern-matching data is stored in a route conventionally called *patact,0.* This is made available under the choice tree **events->pattern-action**. Multiple named tables can exist, used only when they are referred to by a watching job. When ever a pattern-action table is updated, it will be reread by the watching process without the need to restart *clockwork*.

The table has the following columns:

Pattern The regular expression to look for as a pattern, which should normally match a

	single line. Each match is considered an event.
Embargo time	The number of seconds that must elapse after the first event before another event may be raised of the same pattern from the same route.
Embargo count	The maximum number of identical pattern matches that can take place before another event is raised for that pattern and route.
Severity	How important is the event. One of: *fatal, error, warning, info, diag, debug*
Action method	The execution method of the event
Action arguments	The method specific arguments that aid in writing the event
Action message	The message template used to describe the event when it is sent on. It may contain special tokens of the form %<*char*> that describe other information about the event.

If the embargo time is reached, then the time is reset. If the embargo count is reached, then the count is reset.

The action message is used to form describing text, to be sent to the method and argument set. The methods are the same as those supported by the *clockwork's* job table.

9.5 Thresholds

Thresholds are not yet available.

10 Administration

An Administration Manual exists that explains how to configure and manage the flexibility in Habitat. However, some administrative information is useful for users to understand and some select topics are briefly covered below.

10.1 Replication

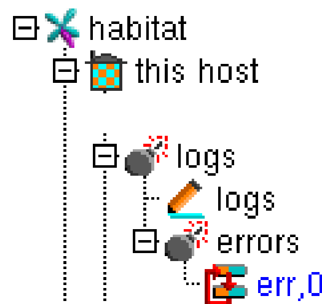
Habitat has the ability to replicate its data to a central repository (and collect data waiting for it). The replication is carried out by the a job within *clockwork*, which updates two tables once replication work has completed.

The state table (called *rstate,0* in the ringstore) is presented as **replication->state** in the choice tree. It shows the state of replication rings and the observed sequences with their times locally and remotely.

Log messages as a result of the replication process are shown under **replication->log** in the choice tree (*rep,0* is the ringstore). These are time stamped messages showing information and errors in replication process.

10.2 Logs & Errors

All jobs in the *clockwork* collection agent have the ability to generate error messages (like *stderr* in Unix). By convention, all jobs use the same ring to store their errors (the ring *err,0*), and this is presented under the label **logs->errors** in the choice tree. The log choices for the current host are shown below (**habitat->this host->logs->errors->err,0**).



Additionally, some jobs use the same ring to store non-error logs, although this is a smaller number than combine for errors. The ring in this situation is *log,0* and is presented as **logs->log** in the choice tree.

10.3 Jobs

The collection agent *clockwork* uses jobs in a table to govern its activities. This is stored in a ring called *clockwork,0* and presented under the node **jobs** in the choice tree.

The columns and meaning of them are shown below:

Start	The number of seconds after *clockwork* has started before this job should start. A value of 0 starts the job as soon as possible after running *clockwork*
Interval	The number of seconds between successive runs of this job
Phase	Not currently used
Count	The number of times that this job should repeat. A value of 0 will repeat until clockwork is terminated

Key	The name of this job, useful for identification in logs
Origin	A text description of the job's owner, useful in larger job tables and typically their e-mail address
Result	The route to which results should be sent (cf *stdout* in Unix)
Errors	The rout to which errors should be sent (cf *stderr* in Unix)
Keep	The number of slots to keep in both Results and Errors when creating the rings for the first time. Ignored when the rings already exist
Method	The job method, which at the time of writing may be one of *probe, exec, sh, snap, tstamp, sample, pattern, record, event* or *replicate*. See the Administration manual for more details
Command	The commands that are passed to the method

10.4 Raw Data

All data recorded in *ringstore* are displayed in an unadulterated form under the **data** label in the choice tree. Each ring name is represented as a node, underneath which are nodes representing different durations.

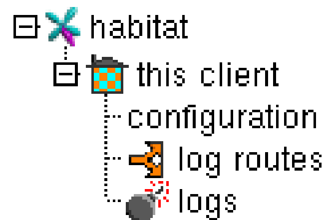
All samples are concatenated and broken into rime ranges for display as charts or tables. No consolidation of sample durations takes place, unlike the standard views (such as **this host->host data->system**).

This is the only method for viewing non-habitat data held in *ringstores* inside *ghabitat*.

11 Diagnostics

The *ghabitat* tool has the same logging & diagnostic mechanism as the rest of the habitat suite. Thus, if there are issues habitat's deployment, it can often be diagnosed using tools present in the graphical suite. The section below explains some of the features available to look at that information, which ideally should be read in conjunction with the Administration manual.

11.1 Log Configuration



Selecting **this client->log routes** shows a mapping of log severity to route. By default, this is normally set to the following:-

severity	route	format
nosev	none	%7$c\|%4$d|%5$s\|%12$s|%13$s\|%14$d|%17$s
debug	none	%7$c\|%4$d|%5$s\|%12$s|%13$s\|%14$d|%17$s
diag	none	%7$c\|%4$d|%5$s\|%12$s|%13$s\|%14$d|%17$s
info	gtkgui	%7$c\|%4$d|%5$s\|%12$s|%13$s\|%14$d|%17$s
warning	gtkgui	%7$c\|%4$d|%5$s\|%12$s|%13$s\|%14$d|%17$s
error	gtkgui	%7$c\|%4$d|%5$s\|%12$s|%13$s\|%14$d|%17$s
fatal	gtkgui	%7$c\|%4$d|%5$s\|%12$s|%13$s\|%14$d|%17$s

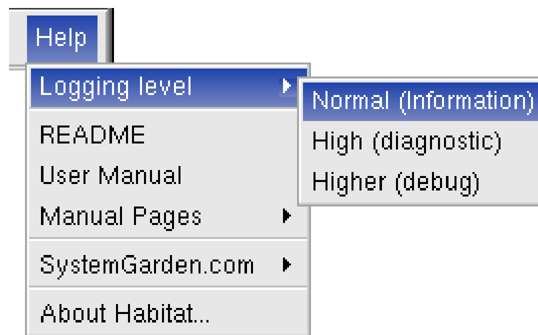
There are six severities, ranging from fatal to debug; *nosev* is a special severity conventionally discarding any inputs.

The route column indicates which route will be used to send or store log information. In the screen shot above, only two are used: *none* and *gtkgui,* both of which are route drivers without addressing. *None* discards any logs sent to it and in the example above, any messages less important than 'info' are thrown away. *Gtkgui* is a special driver provided by ghabitat that routes message logs into the graphical environment.

The log route can be customised by setting directives on the command line or in configuration files. For example, it may be useful to send logs to *stderr* or to a file (with the address *file:filename*).

11.2 Collecting Less Severe Logs

In the default configuration, only logs of severity 'info' and above are collected by the application for displaying in the main viewing area or the popup window. To collected the less severe logs 'diag' or 'debug', use the menubar item **Help->Logging level**.

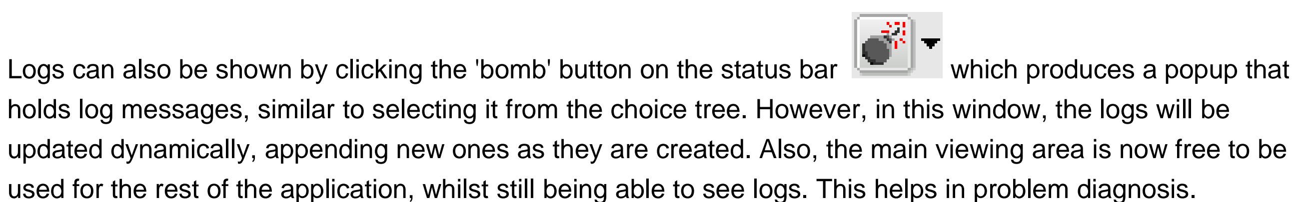


A **High** logging level will include diagnostic messages which are of general help in configuration errors. A **Higher** level will include debugging messages, useful when diagnosing or debugging application issues. **Normal** will return the collection of logs to the default state.

11.3 Viewing Logs from the Choice Tree

Selecting the **this client->logs** node from the choice tree displays a list of all the log messages sent to the *gtkgui* driver. By default, these are messages with severity of 'info' and greater but can be altered using **Help->Logging level** from the menubar, as covered above.

When selected a display similar to the one below is shown in the visualisation area


this client - logs

time	severity	message
07:48:14	info	unable to read superblock from /home/nigel/proj/habitat/data/sys.fha, assur
07:48:14	info	unable to read superblock from /home/nigel/proj/habitat/data/test.csv, assu
07:48:14	info	loading and enabling repository
07:48:32	info	collecting local data with clockwork on pid 4069

Arranged for optimum viewing is the time, severity and message of each log. However, each entry also contains the origin of the message, useful for diagnosing problems with system garden support. This takes the form of function name, file name and line number and is available by scrolling the list to expose the right hand side. As with other tabular displays in the viewing area, double clicking on any row will display the row contents in a column list inside a popup for greater inspection.

The most recent log message is always displayed in the status bar at the bottom of the main window.

11.4 Dynamic Viewing of Logs from Statusbar

Logs can also be shown by clicking the 'bomb' button on the status bar which produces a popup that holds log messages, similar to selecting it from the choice tree. However, in this window, the logs will be updated dynamically, appending new ones as they are created. Also, the main viewing area is now free to be used for the rest of the application, whilst still being able to see logs. This helps in problem diagnosis.

The dynamic logs window also has other features to aid understanding. Rows can be coloured depending on the severity of the message: black for fatal, red for error and so on. For long log histories, messages can be filtered for severity, independent of the collection of the logs. Finally, the log origin is also available by clicking for detailed message, which will widen the list being displayed.

An example is shown below of detailed coloured logs, all of 'info' severity.

Logs

All severities | Coloured text | [x] Detailed logs | OK

time	severity	message	function	file	line
07:48:14	info	unable to read superblock from /home/nigel/proj/habitat/data/s	ghchoice_loadfile	ghabitat/ghchoice.c	622
07:48:14	info	unable to read superblock from /home/nigel/proj/habitat/data/t	ghchoice_loadfile	ghabitat/ghchoice.c	622
07:48:14	info	loading and enabling repository	ghchoice_loadrepository	ghabitat/ghchoice.c	831
07:48:32	info	collecting local data with clockwork on pid 4069	gtkaction_askclockwork	ghabitat/gtkaction.c	3034

12 Appendix

12.1 Manual Pages

The following are a selection of the manual pages distributed with the habitat package, considered pertinent to the User Guide.

12.1.1 clockwork

NAME

clockwork - collection daemon for the Habitat suite

SYNTAX

clockwork [-c <purl>] [-C <cfcmd>] [-e <fmt>] [-dDhsv] [-j <jobs>]

DESCRIPTION

Clockwork is the local agent for the Habitat suite. It runs as a daemon process on each machine to be 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 and uptime 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 deligation. 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 using custom datastores 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

-j <jobs>
: Override public job table with a private one provided by the route <jobs>. 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.

-s Disable the public data service from being run, but will continue to save data as dictated by configuration.

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 $HAB/var/<hostname>.rs (the route address is rs:$HAB/var/<hostname>.rs,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 rs:$HAB/var/<hostname>.rs,<jobname>,60 and errors to rs:$HAB/var/<hostname>.rs,err_<jobname>,60

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

rs: 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 ringbuffer, 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>.rs, $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>.rs, /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>
```

AUTHORS

Nigel Stuckey <nigel.stuckey@systemgarden.com>

SEE ALSO

killclock(1), ghabitat(1), habget(1), habput(1), irs(1), habedit(1), habprobe(1), habmeth(1), habconf(5)

12.1.2 ghabitat

NAME

ghabitat - Gtk+ Graphical interface to Habitat suite

SYNTAX

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

DESCRIPTION

This is the standard graphical interface for Habitat, including the ability to view repository data provided by Harvest.

When the tool starts, a check is made for the existence of the local collection agent, clockwork. If it is not running, the user is asked if they wish to run it and what starting behaviour they wish 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 ghabitat and will be placed under the my files node in the tree.

See 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.

This GUI requires Xwindows to run, 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 ghabitat in diagnositc 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 ghabitat 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 ghabitat automatically loading data from files or over the network from peer machines or the repository. Use if ghabitat 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:-

Storage file
: The standard local data storage file known as a ringstore, which is a structured format using GDBM. Open it with File->Open or ^O and use the file chooser. The file will appear under my files in the choice tree.

Repository
: Centralised data automatically appears under the repository node in the choice tree if the configuration directive is set to a valid location. The directive is route.sqlrs.geturl which must contain the URL of a repository output interface. (route.sqlrs.puturl works in the opposite direction for replication.)

Network data
: Data for an individual machine can be read from the repository or a peer clockwork instance on another host. Select File->Host or ^H, type in the hostname and pick repository or host as a source. (Currently, host access is not fully implemented.) Your selection will appear under my hosts in the choice tree.

Route specification
: Select File->Route or ^R and type the full route specification of the data source. This is the most generic way of addressing in habitat, encompassing all of the styles used above and more.

Files can be removed by selecting their entry from the list brought up with File->Close (^C).

NAVIGATION

The repository source is special, in that the hierarchical nature of the group organisation is shown. To get to a machine, one needs to know its organisational location and traverse it in the tree. Whilst this aids browsing, one may wish to use the File->Host option to go directly to a machine.

Opening the data source trees will reveal the capabilities of the data source, which include the following:-

perf graphs
: Performance data is retrieved in a time series and will display as a chart the visualisation section

perf data
: Performance data presented in a textual form, encompassing tabular time-series data, key-data values or simple list. Visualisation is always in a table.

events
: Text scanning, pattern matching and threshold breaching functionality is clustered under this node. The configuration tables are presented here along with the events and logs that have been generated.

logs
: Logs and errors from running the jobs in clockwork

replication
: Logs and state of the data replication to and from the repository

jobs The job table that clockwork follows to collect and refine data

data Contains all the data in the storage mechanism with out interpretation.

Under the performance nodes will be the available data collections, also known as rings. The names of these collections are decided when data is inserted into the storage. For example, sending data to the route tab:fred.ts,mydata and mounting it under ghabitat, will cause the data to appear here as mydata.

There are conventions for the names of standard probes, but they will only appear in a data store if their collection is configured in the job table (usually just uncommenting it: see clockwork(1)):-

sys System data, such as cpu statistics and memory use. Labelled as system in choice tree

io Disk statistics, such as read/write rates and performance levels. Labelled as storage in the choice tree

net Network statistics, such as packets per second. Labelled as network in the choice tree

ps Process table. This can contain a significant amount of data over time, so generally only the most significant or useful processes may be included. This is dependent on the configuration of the ps probe. Labelled as processes in the choice tree

names A set of name-value pairs relating to the configuration of the operating system. Generally captured at start up only.

The final set of nodes below the ring names are a set of time scales by which to examine the data. These dictate how much data is extracted from the data source and generally the speed at which the data will be visualised. These are preset to useful values, commonly 5 minutes, 1 hour, 4 hours, 1 day, 7 days, 1 month, 3 months, 1 year, 3 years, etc.

VISUALISATION

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

When charting, the section is divided into several parts. The greatest is used for the graph itself, with other areas being used for curve selection, zooming and data scaling. 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.

The standard sets of data, such as sys and 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.

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.

It is possible to alter the scale and offset of the curves by clicking on the additional fields button in the Zoom & Scale area. This will create addition scale and offset controls next to each curve button. The values relate to the formula y = mx + c, where the offset is c and the scale is m. Moving the scale changes the magnitude of the curve, whereas the offset changes the point at which the curve originates. Using these tools, simple parity can be gained between two curves that you wish to superimpose on the same chart but do not share the same y scale.

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

Ghabitat 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 appearance: fonts, colours, styles, etc

    $HAB/lib/ghabitat.rc or /usr/lib/habitat/ghabitat.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, Netscape, Konqueror, Opera, Chimera

HOME User's home directory

AUTHORS

Nigel Stuckey <nigel.stuckey@systemgarden.com>

SEE ALSO

clockwork(1), killclock(1), habget(1), habput(1), irs(1), habedit(1), habprobe(1), habmeth(1), habconf(5)

12.1.3 habget

NAME

habget - Send habitat data to stdout

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 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 ghabitat 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 ghabitat 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 number 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 rs:var/myhost.rs,clockwork,0
```

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

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

To find errors that may have been generated by clockwork.

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

12.1.4 habput

NAME

habput - Store habitat data from stdin

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 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 ghabitat 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 ghabitat 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 behaviour 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 tab:var/myfile.ts,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" tab:var/myfile.ts,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" tab:var/myfile.ts,r.myring
```

To save an error record, use a timestore driver

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

AUTHORS

Nigel Stuckey <nigel.stuckey@systemgarden.com>

SEE ALSO

clockwork(1), killclock(1), ghabitat(1), habget(1), irs(1), habedit(1), habprobe(1), habmeth(1), habconf(5)

12.1.5 killclock

NAME

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(1), ghabitat(1), habget(1), habput(1), irs(1), habedit(1), habprobe(1), habmeth(1), habconf(5)

12.2 Collected Data

The tables below shows the data that is collected by the standard probes in **habitat**, one table per operating system. It may not be up to date, so always check with the application itself.

12.2.1 Linux Data

Probe	Measure	Description
system (sys)	load1	1 minute load average
	load5	5 minute load average
	load15	15 minute load average
	runque	number of runnable processes
	nprocs	number of processes
	lastproc	id of last process run
	mem_tot	total memory (kB)
	mem_used	memory used (kB)
	mem_free	memory free (kB)
	mem_shared	used memory shared (kB)
	mem_buf	buffer memory (kB)
	mem_cache	cache memory (kB)
	swap_tot	total swap space (kB)
	swap_used	swap space used (kB)
	swap_free	swap space free (kB)
	uptime	seconds that the system has been up
	idletime	seconds that system has been idle
	%user	% time cpu was in user space
	%nice	% time cpu was at nice priority in user space
	%system	% time cpu spent in kernel
	%idle	% time cpu was idle
	pagein	pages paged in per second
	pageout	pages paged out per second
	swapin	pages swapped in per second
	swapout	pages swapped out per second
	interrupts	hardware interrupts per second
	contextsw	context switches per second
	forks	process forks per second
storage (io)	id	mount or device identifier
	device	device name
	mount	mount point
	fstype	filesystem type
	size	size of filesystem or device (MBytes)
	used	space used on device (MBytes)
	reserved	reserved space in filesystem (KBytes)
	%used	% used on device
	kread	volume of data read (KB/s)
	kwritten	volume of data written (KB/s)
	rios	number of read operations per second
	wios	number of write operations per second
	read_svc_t	average read service time (ms)
	write_svc_t	average write service time (ms)

Probe	Measure	Description
network (net)	device	device name
	rx_bytes	bytes received
	rx_pkts	packets received
	rx_errs	receive errors
	rx_drop	receive dropped packets
	rx_fifo	received fifo
	rx_frame	receive frames
	rx_comp	receive compressed
	rx_mcast	received multicast
	tx_bytes	bytes transmitted
	tx_pkts	packets transmitted
	tx_errs	transmit errors
	tx_drop	transmit dropped packets
	tx_fifo	transmit fifo
	tx_colls	transmit collisions
	tx_carrier	transmit carriers
	tx_comp	transmit compressed
uptime (up)	uptime	uptime in secs
	boot	time of boot in secs from epoch
	suspend	secs suspended
	vendor	vendor name
	model	model name
	nproc	number of processors
	mhz	processor clock speed
	cache	size of cache in kb
	fpu	floating point unit available
downtime (down)	lastup	time last alive in seconds from epoch
	boot	time of boot in secs from epoch
	downtime	secs unavailable
processes (ps)	pid	process id
	ppid	process id of parent
	pidglead	process id of process group leader
	sid	session id
	uid	real user id
	pwname	name of real user
	euid	effective user id
	epwname	name of effective user
	gid	real group id
	egid	effective group id
	size	size of process image in Kb
	rss	resident set size in Kb
	flag	process flags (system dependent)
	nlwp	number of lightweight processes within this process
	tty	controlling tty device
	%cpu	% of recent cpu time
	%mem	% of system memory
	start	process start time from epoc
	time	total cpu time for this process
	childtime	total cpu time for reaped child processes
	nice	nice level for cpu scheduling
	syscall	system call number (if in kernel)
	pri	priority (high value=high priority)

Probe	Measure	Description
	wchan	wait address for sleeping process
	wstat	if zombie the wait() status
	cmd	command/name of exec'd file
	args	full command string
	user_t	user level cpu time
	sys_t	sys call cpu time
	otrap_t	other system trap cpu time
	textfault_t	text page fault sleep time
	datafault_t	data page fault sleep time
	kernelfault_t	kernel page fault sleep time
	lockwait_t	user lock wait sleep time
	osleep_t	all other sleep time
	waitcpu_t	wait-cpu (latency) time
	stop_t	stopped time
	minfaults	minor page faults
	majfaults	major page faults
	nswaps	number of swaps
	inblock	input blocks
	outblock	output blocks
	msgsnd	messages sent
	msgrcv	messages received
	sigs	signals received
	volctx	voluntary context switches
	involctx	involuntary context switches
	syscalls	system calls
	chario	characters read and written
	pendsig	set of process pending signals
	heap_vaddr	virtual address of process heap
	heap_size	size of process heap in bytes
	stack_vaddr	virtual address of process stack
	stack_size	size of process stack in bytes
hardware interrupts (intr)	name	device name
	hard	interrupts from hardware device
	soft	interrupts self induced by system
	watchdog	interrupts from a periodic timer
	spurious	interrupts for unknown reason
	multisvc	multiple servicing during single interrupt
system values (names)	name	name
	vname	value name
	value	value of symbol

12.2.2 Solaris Data

Probe	Measure	Description
system (sys)	updates	
	runque	+= num runnable procs
	runocc	++ if num runnable procs > 0
	swpque	+= num swapped procs
	swpocc	++ if num swapped procs > 0
	waiting	+= jobs waiting for I/O
	freemem	+= freemem in pages
	swap_resv	+= reserved swap in pages
	swap_alloc	+= allocated swap in pages
	swap_avail	+= unreserved swap in pages
	swap_free	+= unallocated swap in pages
	%idle	time cpu was idle
	%wait	time cpu was idle waiting for IO
	%user	time cpu was in user space
	%system	time cpu was in kernel space
	wait_io	time cpu was idle waiting for IO
	wait_swap	time cpu was idle waiting for swap
	wait_pio	time cpu was idle waiting for programmed I/O
	bread	physical block reads
	bwrite	physical block writes (sync+async)
	lread	logical block reads
	lwrite	logical block writes
	phread	raw I/O reads
	phwrite	raw I/O writes
	pswitch	context switches
	trap	traps
	intr	device interrupts
	syscall	system calls
	sysread	read() + readv() system calls
	syswrite	write() + writev() system calls
	sysfork	forks
	sysvfork	vforks
	sysexec	execs
	readch	bytes read by rdwr()
	writech	bytes written by rdwr()
	rawch	terminal input characters
	canch	chars handled in canonical mode
	outch	terminal output characters
	msg	msg count (msgrcv()+msgsnd() calls)
	sema	semaphore ops count (semop() calls)
	namei	pathname lookups
	ufsiget	ufs_iget() calls
	ufsdirblk	directory blocks read
	ufsipage	inodes taken with attached pages
	ufsinopage	inodes taked with no attached pages
	inodeovf	inode table overflows
	fileovf	file table overflows
	procovf	proc table overflows
	intrthread	interrupts as threads (below clock)

Probe	Measure	Description
	intrblk	intrs blkd/prempted/released (switch)
	idlethread	times idle thread scheduled
	inv_swtch	involuntary context switches
	nthreads	thread_create()s
	cpumigrate	cpu migrations by threads
	xcalls	xcalls to other cpus
	mutex_adenters	failed mutex enters (adaptive)
	rw_rdfails	rw reader failures
	rw_wrfails	rw writer failures
	modload	times loadable module loaded
	modunload	times loadable module unloaded
	bawrite	physical block writes (async)
	iowait	procs waiting for block I/O
	pgrec	page reclaims (includes pageout)
	pgfrec	page reclaims from free list
	pgin	pageins
	pgpgin	pages paged in
	pgout	pageouts
	pgpgout	pages paged out
	swapin	swapins
	pgswapin	pages swapped in
	swapout	swapouts
	pgswapout	pages swapped out
	zfod	pages zero filled on demand
	dfree	pages freed by daemon or auto
	scan	pages examined by pageout daemon
	rev	revolutions of the page daemon hand
	hat_fault	minor page faults via hat_fault()
	as_fault	minor page faults via as_fault()
	maj_fault	major page faults
	cow_fault	copy-on-write faults
	prot_fault	protection faults
	softlock	faults due to software locking req
	kernel_asflt	as_fault()s in kernel addr space
	pgrrun	times pager scheduled
	nc_hits	hits that we can really use
	nc_misses	cache misses
	nc_enters	number of enters done
	nc_dblenters	num of enters when already cached
	nc_longenter	long names tried to enter
	nc_longlook	long names tried to look up
	nc_mvtofront	entry moved to front of hash chain
	nc_purges	number of purges of cache
	flush_ctx	num of context flushes
	flush_segment	num of segment flushes
	flush_page	num of complete page flushes
	flush_partial	num of partial page flushes
	flush_usr	num of non-supervisor flushes
	flush_region	num of region flushes
	var_buf	num of I/O buffers
	var_call	num of callout (timeout) entries
	var_proc	max processes system wide

Probe	Measure	Description
	var_maxupttl	max user processes system wide
	var_nglobpris	num of global scheduled priorities configured
	var_maxsyspri	max global priorities used by system class
	var_clist	num of clists allocated
	var_maxup	max number of processes per user
	var_hbuf	num of hash buffers to allocate
	var_hmask	hash mask for buffers
	var_pbuf	num of physical I/O buffers
	var_sptmap	size of sys virt space alloc map
	var_maxpmem	max physical memory to use in pages (if 0 use all available)
	var_autoup	min secs before a delayed-write buffer can be flushed
	var_bufhwm	high water mrk of buf cache in KB
	var_xsdsegs	num of XENIX shared data segs
	var_xsdslots	num of slots in xsdtab[] per segmt
	flock_reccnt	num of records currently in use
	flock_rectot	num of records used since boot
processes (ps)	pid	process id
	ppid	process id of parent
	pidglead	process id of process group leader
	sid	session id
	uid	real user id
	pwname	name of real user
	euid	effective user id
	epwname	name of effective user
	gid	real group id
	egid	effective group id
	size	size of process image in Kb
	rss	resident set size in Kb
	flag	process flags (system dependent)
	nlwp	number of lightweight processes within this process
	tty	controlling tty device
	%cpu	% of recent cpu time
	%mem	% of system memory
	start	process start time from epoc
	time	total cpu time for this process
	childtime	total cpu time for reaped child processes
	nice	nice level for scheduling
	syscall	system call number (if in kernel)
	pri	priority (high value=high priority)
	wchan	wait address for sleeping process
	wstat	if zombie the wait() status
	cmd	command/name of exec'd file
	args	full command string
	user_t	user level cpu time
	sys_t	sys call cpu time
	otrap_t	other system trap cpu time
	textfault_t	text page fault sleep time
	datafault_t	data page fault sleep time
	kernelfault_t	kernel page fault sleep time
	lockwait_t	user lock wait sleep time
	osleep_t	all other sleep time
	waitcpu_t	wait-cpu (latency) time

Probe	Measure	Description
	stop_t	stopped time
	minfaults	minor page faults
	majfaults	major page faults
	nswaps	number of swaps
	inblock	input blocks
	outblock	output blocks
	msgsnd	messages sent
	msgrcv	messages received
	sigs	signals received
	volctx	voluntary context switches
	involctx	involuntary context switches
	syscalls	system calls
	chario	characters read and written
	pendsig	set of process pending signals
	heap_vaddr	virtual address of process heap
	heap_size	size of process heap bytes
	stack_vaddr	virtual address of process stack
	stack_size	size of process stack bytes
storage (io)	device	device name
	nread	number of bytes read
	nwritten	number of bytes written
	reads	number of read operations
	writes	number of write operations
	wait_t	cumulative wait (pre-service) time
	wait_len_t	cumulative wait length*time product
	run_t	cumulative run (service) time
	run_len_t	cumulative run length*time product
	wait_cnt	wait count
	run_cnt	run count
System values (names)	name	name
	vname	value name
	value	value
system timers (timer)	kname	timer name
	name	event name
	nevents	number of events
	elapsed_t	cumulative elapsed time
	min_t	shortest event duration
	max_t	longest event duration
	start_t	previous event start time
	stop_t	previous event stop time
uptime (up)	uptime	uptime in secs
	boot	time of boot in secs from epoch
	suspend	secs suspended
	vendor	vendor name
	model	model name
	nproc	number of processors
	mhz	processor clock speed
	cache	size of cache in kb
	fpu	floating point unit available
down-time (down)	lastup	time last alive in secs from epoch
	boot	time of boot in secs from epoch
	downtime	secs unavailable

Probe	Measure	Description
hardware interrupts (intr)	name	device name
	hard	interrupt from hardware device
	soft	interrupt self induced by system
	watchdog	interrupt from periodic timer
	spurious	interrupt for unknown reason
	multisvc	multiple servicing during single interrupt

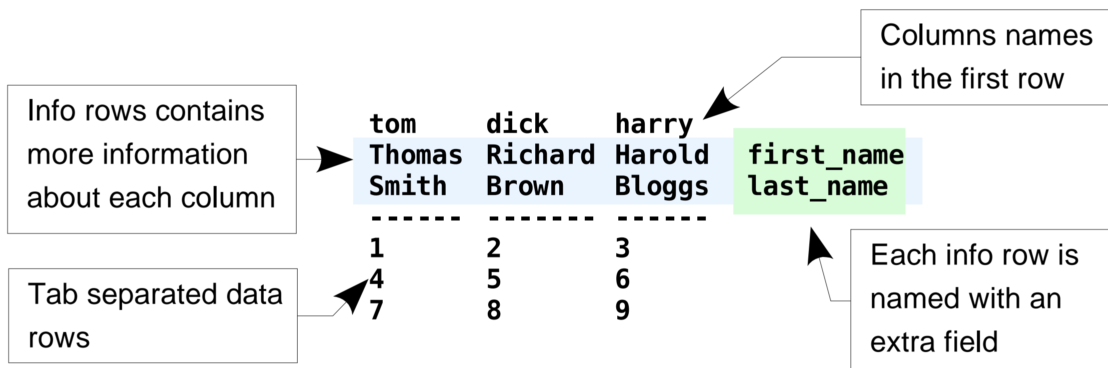
12.3 Fat Headed Array Format

The Fat Headed Array is a table of information designed for transportation between systems and external representation. Habitat uses FHAs when reading *clockwork*'s central store and for I/O work with *harvest*. It is also used when loading data into harvest's repository when linking with other systems.

There are three parts to the data in the following order:

1. Column names: tab separated attribute names that must be in line 1
2. Info Rows: zero or more lines of meta information for each attribute. Each meta record takes one line, separating its fields with tabs which must be in the same order as the attribute names. Each info row is named with a trailing supplementary field, such that these rows have one more column than the data or column name rows. Info rows are terminated with a row which must start with two dashes '--'.
3. One or more data rows to follow the column names and info rows.

The following is an example of a FHA.



To represent an empty value, two quotes should be used (“”). Occasionally, this may also be represented as a single dash.

To represent a value that contains tab characters (\t), the value should be contains in quotes (eg “embedded \t tab“).

Column names may only contain characters accepted by SQL database servers to ensure compatibility. This is generally accepted as the range [A-Za-z_]. See the info row **name** below for greater expression.

In addition to pure character formatting, *ghabitat* the graphical client, also understands certain named info rows.

1. **info** – the text in the row is used as informational help in the client. This can be seen when hovering the mouse over a column name in a table or a curve's button when displaying charts. The information is

contained in graphical 'pop-ups' or 'tool tips'.

2. **max** – Optional value which, if present, sets the maximum expected value for an attribute. This helps in making charts more understandable.
3. **type** – the data type of the column. In version 1, the data types are relatively simple:-
   1. **i32** – 32-bit signed integer
   2. **u32** – 32-bit unsigned integer
   3. **i64** – 64 bit signed integer
   4. **u64** – 64bit unsigned integer
   5. **nano** – nano second precision when used for timers. Currently also used for floating point values with more restrictive accuracy
   6. **str** – string value
4. **key** – The column that is the primary key of the table contains a *1,* all the other values contain no value (""). May be expanded to show secondary or tertiary keys in later versions.
5. **name** – The unrestricted full name of the column, if it is not possible to express it in the column name. If blank, the column name is used as the attribute's label. This is used to include punctuation characters such as '-' or '%' in the label as they are disallowed by the SQL naming standard but can be very useful for compact expression.

12.4 Job Table Format

Clockwork reads a job table and uses the information to establish repeating and timed jobs. It is similar to the Unix scheduler *cron*, but with greater flexibility.

When first run, clockwork boot straps an initial version of its jobs from the file *lib/clockwork.jobs*. The resulting table is stored in the ringstore location *var/<hostname>.rs,jobs,0*. Subsequent runs of clockwork will use this table, so any amendments should be made using *habedit* on the ringstore route.

Clockwork may also be started with an alternate job table by using the the *-j* switch to *clockwork.* In this mode, clockwork runs a private data collector with out starting a network service for the whole machine.

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

ringstore and SQL ringstore).

10. **method** – the job method

11. **command** – the arguments given to each method

Part of a job table taken from the default file *lib/clockwork.jobs* is printed below. The top line runs the *sys* probe every 60 seconds, gathering system data (which becomes the **system** node in the choice tree). The remaining lines use the collected high frequency data and transform it to lower frequencies using an averaging process, running every five minutes, fifteen minutes and one month (300, 900 and 3600 seconds).

start name nrings phase results command

```
0 60   0 0 sys,%d sysgar rs:%h.rs,%j  rs:%h.rs,err,0  240 probe   sys
0 300  0 0 sys,%d sysgar rs:%h.rs,%j  rs:%h.rs,err,0  288 sample “avg rs:%h.rs,sys,60”
0 900  0 0 sys,%d sysgar rs:%h.rs,%j  rs:%h.rs,err,0  672 sample “avg rs:%h.rs,sys,300”
0 3600 0 0 sys,%d sysgar rs:%h.rs,%j  rs:%h.rs,err,0  672 sample “avg rs:%h.rs,sys,900”
```

count errors period requester method

The job running every 300 seconds creates a storage ring with 288 entries, allowing a full day's data at five minute intervals to be collected. The other jobs collect seven days at 15 minutes and one month at hourly intervals.

The methods are **probe** which is given the command **sys**, and **sample** which has the command **avg** and the route to sample as its argument. These methods are available on the command line using *habmeth*. The probe data is similarly available using the command *habprobe*.

Note that special tokens are used which expand when *clockwork* is running. These are %d for the ring's duration and %h for the hostname. Other tokens are available which are explored in the Administration manual.

12.5 Pattern Matching Table Format

The pattern matching table is user configurable, using the processing mechanism is described earlier in this document.

The pattern-matching table, which defines the bahaviour has the following columns:

1. **pattern** – the regular expression to look for as a pattern, which should normally match a single line. Each match is considered an event.
2. **embargo time** – number of seconds that must elapse after the first event before another event may be raised of the same pattern from the same route.
3. **embargo count** – maximum number of identical pattern matches that can take place before another event is raised for that pattern and route.
4. **severity** – importance of the event. One of: *fatal, error, warning, info, diag, debug*

5. **action method** – event's execution method
6. **action arguments** – method specific arguments
7. **action message** – text message to describe the event. It may contain special tokens of the form %<*char*> that describe other information about the event.

When the event is detected and is not subject to embargo, then an event is raised. A text message is prepared which is turned into an instruction using the action method and arguments. Then, it is appended to the event ring for execution (see below).

12.6 Watched Sources Table Format

The watched sources table defines a set of *routes* associated with a identifier. A watching job then ties together a set of sources with a set of patterns and executes them periodically. When the watching job starts, it checks all the *routes* defined in this table for changes in size. Those that have changed will be checked for pattern matches (see the details above).

The format of the table is simple: one entry per line, with each being a valid *route* format.

12.7 Event Table Format

The event table is filled from the activities of pattern matching and threshold detection. When there is a match not covered by an embargo, an event will be raised, which is a instruction to execute a method supported by the *habitat* job environment. The instructions are queued as separate sequences in the event ring, which is stored in the system ringstore.

The table format is simple:-

1. **method** – execution method as supported by the *habitat* job environment.
2. **command** – command to give to method
3. **arguments** – command arguments, which may contain spaces. The '%' character must be escaped if it is to be used in an argument (see below)
4. **stdin** – input text to the method, which must be introduced with '%' to separate it from the argument. Successive % characters represent new lines. To actually print %, escape it with backslash (\%).

When an event has been completed, the next sequence to be processed is stored in a state table. The event ring is a finite length, so old events will be removed automatically over time.