# Glossary

The glossary was lifted from another networking document and so may not be all that relevant just now.

**BGP** Border Gateway Protocol

**BNF** See EBNF.

**DNS** Domain Name System

**EGP** External Gateway Protocol

**EBNF** Extended Backus-Naur Format. A widely used grammar description language.

**FNC** Federal Network Council

**HINFO** DNS Host Information Record

**IETF** Internet Engineering Task Force

**IXI-CC** IXI Coordinating Commitee

**IGRP** Intermediate Gateway Routing Protocol (CISCO specific)

**MX Record** DNS Mail eXchange record

**MERIT** NSFNET operators

**NIC** Network Information Centre

**NJE** Network Job Entry (EARN/BITNET network protocol)

**NOC** Network Operations Centre

**RIPE** Reseaux IP European

**RARE** Reseaux Associates pour la Researche European

**SNMP** Simple Network Management Protocol

**SMTP** Simple Mail Transfer Protocol

**TWG** Topology Working Group (IETF Working Group)

## NAME

**libxmonet** – X11 library for monet

## SYNOPSIS

*cc* ... –**lxmonet**

## DESCRIPTION

This library provides an interface to X11 in addition to some general purpose X windows utilities. It uses the Athena widget set and the Widget Creation Library.

The features include routines for rubberbands; calling external programs from X; a simple drawing library, callbacks for popping up menus; setting and getting widget values; messages; and socket I/O.

An alternative is the monet *libio* library which provides emulators for parts of *libxmonet* without requiring X windows.

## SEE ALSO

monet(1N), libio(3N), X(1)

## AUTHORS

James Barr, NIKHEF-H, Amsterdam

## BUGS

All queries are made syncronously, this means that only one query is out standing at a time.
Priority scheme is too primitve to be useful. Currently ignores the period between queries.
Does not examine the results of other queries. Not very clever.

## AUTHORS

James Barr, NIKHEF-H, Amsterdam

## NAME

**libquery** – Monet Network Monitor/Query Library

## SYNOPSIS

**#include <monet/query/Lib.h>**

*cc* ... **–lquery**

## DESCRIPTION

The *libquery* library contains a set of modules for monitoring various types of network protocol (X.25, IP, TCP, SNMP). The results are cooked into a generic format so that the library user does not need to worry about different protocols.

It should be easy to add new protocols and thus a consistent interface needs to be defined. The interface is defined as a structure containing a set of pointers to functions. This means that **virtually no existing code** need be changed; one *#include* and one line to initialise the module, plus of course a source file to implement the module. This structure is defined in *monet/lib/query/Lib.h*.

Queries can be made syncronously or asyncronously, the latter is prefered because this give much better respose time.

## PRIORITY

New requests are given priority over existing ones so that resposes are returned quickly.

A frequency attribute can be attached to each query, this indicates the *minimum* period between querying a perticular entity. So that it can be arranged for a system to be queried every 5 minutes, say.

## SEE ALSO

monet(1N), monetd(8N)

## NAME

**monetmap** – Monet map file format

## DESCRIPTION

The maps used by *monet* are saved in an ascii format. All the files for a single database are expected to be together in the same directory with no subdirectories. They can, howerver be read in from anywhere.

Maps files do not have any connectivity information or status information, merely a list of network entities and their position and dimensions.

**ins** <class:name> <x> <y> <width> <height>

**mapset** <attribute> <value>

**map** <map name>

If the dimensions are 0 or two small they will be automatically adjusted when loaded. If the position is [0,0] it will be repositioned.

The files are actually *monetsh (1N)* scripts, this was done for flexibility.

## SEE ALSO

monet(1N), monetedb(5N), monetsh(1N)

## AUTHOR

James Barr, NIKHEF-H, Amsterdam

## NAME

**libmap** – Monet Maps Library

## SYNOPSIS

**#include <monet/map/Drawing.h>**

*cc* ... **–lmonetmap**

## DESCRIPTION

The *libmonetmap* library contains a set of routines which implement maps of network entities.

Each map is a set of drawings. A drawing represents a network entity. Each map has a class, only entities that are subclasses of a sibling of the map class can be entered on it.

The maps can be drawn on an X window but are indepent of this and so can still be used in a command line driven system or in a filter.

Functions that can be performed on maps are,

**Layout** – these untangle a map and try to lay it out in a reasonable manner.

**Visual** – the status of each entity can shown visually.

**Editting** – Interactive editing of maps.

**Output** – Maps can be drawn in an X window or sent to a postscript printer, although X11 is not required.

## SEE ALSO

monet(1N)

## BUGS

– The layout stuff is really primitive.
– does not draw lines between connected systems on different maps.

## AUTHORS

James Barr, NIKHEF-H, Amsterdam

## NAME

**libmonetio** – I/O library for monet

## SYNOPSIS

*cc* ... –**lmonetio**

## DESCRIPTION

This library is an alternative to the *libxmonet* library. Providing routines for memory allocation and registering sockets for a *select(2)* loop routine. The idea is to support applications that may or may not use X windows.

## SEE ALSO

monet(1N), libxmonet(3N)

## AUTHORS

James Barr, NIKHEF-H, Amsterdam

## NAME

**monetrc** – Monet configuration files.

## DESCRIPTION

This is a description of the format used by the *monet* configuration files.

This file is loaded in on program start up from a system directory */etc/monet* after which a local file is loaded *~/.monetrc*.

Each object should either be blank or have two items on it. The first is the name of the configuration attribute and the second is its value. The attribute is one alphanumeric word, following as many tabs and and spaces as desired is the value which takes up the rest of the line. Comments are allowed, these are started by either '#' or '–' and last until the end of the line. String values MUST not be surrounded by quotes and continuation lines are not allowed. Boolean values are given by 1(True) and 0(False).

Default values are set at compile time, some of which are obtained from the system.

For a complete list use *monetsh -help 2*.

## SEE ALSO

monet(1N), monetsh(1N)

## AUTHOR

James Barr, NIKHEF-H, Amsterdam

**MapServer** – a simple remote file server for accessing map files from a remote system. Not really very.TP 4ecific to maps. Deals with naming and version numbers. The transfer itself is done using sockets but *ftp(1C)* might be more suitable.

**config** – provides a general configuration routine which sets up a set of parameters from the command line and/or configuration files. In addition queries the system for some useful parameters.

## BUGS

– requires X11/Intrinsics header file.
– Naming of functions is not consistent.
– Communications only works over tcp/ip.
– Should generalise the pattern matching to full regexp's.
– KeyLists do not work.
– SimpleLists are inefficient.
– The term 'fid' is used in a number of functions for what should actually be an 'fd'.

## AUTHORS

James Barr, NIKHEF-H, Amsterdam
Alena Lacova, NIKHEF-H, Amsterdam

## NAME

**libmonet** – General Library for Monet

## SYNOPSIS

**#include <monet/general/General.h>**
**#include <monet/general/Channel.h>**
**#include <monet/general/SimpleList.h>**
**#include <monet/general/KeyList.h>**

**#include <monet/general/Script.h>**
*cc* ... –**lmonet**

## DESCRIPTION

The *libmonet* library contains a set of routines which implement a set of generally useful functions. It is designed to work with clients using X11 or those that do not.

**SimpleList** – for handling ordered sets.

**Channel** – for handling communications between a server and a client. Works over X11 or user supplied select routine. Pseudo asyncrous.

**wildmat** – routine for matching wildcard patterns. Should be replaced by full regex stuff.

**gather** – gather networking information.

**draw** – defines a simple generic drawing library. Postscript and a text (for debugging) version included here. An X11 version is included in *libxmonet(3N)*.

**log** – simple logging, this really needs to be extended.

**messages** – for sending warnings, information, errors, reports and debugging messages to the user. Sits over X11 or stdout.

**script** – a simple extensible script language interpreter.

## NAME

**monetedb** – Monet Network Entity Database File Format

## DESCRIPTION

These files give a logical description of a multiprotocol network. This is a textual representation of the monet entity database. The database has an inheritance scheme where classes are described in much the same way as normal entities. The difference is in the naming, for example an *ip.host* is a subclass of *host* but an instance would be given a name like *ip.host:cernvm.cern.ch.*

## FILE FORMAT

An entity description consists of a first line of the form
**Object** <name>
followed by a series of attributes
<**attribute**>= <value>
For a list of attributes use the *monetsh* command *attributes*, for the names and a description. Comments begin with the character and continue to the end of the line.

## NAME SYNTAX

Each object has a unique name. The first part is the name of its class and the seond should be a domain name. Classes do not have the second part.

**name** ::= <class name> [: <instance name>]

**class_name** ::= <class name> . <label> | <label>

**instance_name** ::= <character string>

**label** ::= <character string, without '.'s>

## SEE ALSO

monet(1N), monetsh(1N)

## AUTHOR

James Barr, NIKEHF-H, Amsterdam

## SEE ALSO

monet(1N)

## BUGS

Protocol not fully symetrical.
No status history is kept.

## AUTHOR

James Barr, NIKHEF-H, Amsterdam

## NAME

libmonetdb – Monet Network Entity Library

## SYNOPSIS

**#include <monet/db/Entity.h>**

*cc* ... **–lmonetdb**

## DESCRIPTION

The *libmonetdb* library contains a set of routines which implement a distributed database for network entities.

The database model is a stack of related networks. Each layer in the stack represents one set of network protocols. This is very similar to the idea use din the OSI reference model. Links between entities in the same layer are called *interfaces* and these connect exactly two entities. Links between entities on different layers is done saying one set of entities are components of enother entity. Entities themselves are a set of attributes, each with a value. In addition each entitiy has a class which gives a general description of a set of entities. Examples of classes are *Ethernet's* or *ip networks.* Entities inherit the attributes of its class.

The library constists of four parts,

| | |
|---|---|
| **Database** management | classes, attributes and entities. |
| **Relationships** between entities | interfaces and components. |
| **Attribute,** | status information and handling changes. Trigger routines must be registered with the database, which are called when the status changes. |
| **I/O** | External access, parser, info and redirecting queries. |

## DIAGNOSTICS

Most routines return an indication that did not work, and will also send a message.

# Chapter 9

# Library Manual Pages

hooked on quite easily in a lot of cases, but this needs some serious time spent on it.

## AUTHOR

James Barr, NIKHEF-H, Amsterdam

**MonetShellInterpreterCB** similar to above but expects the shell command to return a sequence of monetsh commands which are then executed. This is used for the implementation of traceroute.

## DIAGNOSTICS

These are pretty awful, many messages are more for debugging than information. There should be better structuring so that messages are shown in better places on the map.

System errors a reported directly the stderr, not to a window. These do not always mean something catastrophic happened, rather a bug in the program has been caught.

## FILES

~/.monet - local configuration
/etc/monet - config directory
/tmp - temporary files

## SEE ALSO

monet(1N), monetd(8N), monetrc(5N), maps(5N), X(1)

## BUGS

- No undo
- Should drop into command mode if not connected to x-terminal
- Should be able to choose, motif, Xaw, and colours.
- Cannot draw lines between a systems in different maps.
- When executing an external command the output is only displayed when it has completed not as it becomes available.
- Error messages should be written some place more obvious.
- Calling shell programs while in X windows does not seem to work correctly when using SYS5.
- The action stuff will not work most of the time.
- The biggest problem is that no one has really tried to sit down and use the stuff for thoroughly. The functionality is there or could be

**Left** – A text box for entering data.

**Right** Side – A large empty map.

**Bottom** Left – A map showing all the defined classes.

## POPUPS

**Command** Menu – basic editing commands.

**Layout** Box – various commands for untangling and laying out maps.

**Load** Map – for loading saving and popping up maps.

**Remote** Maps – for accessing the remote map server.

**Options** – for setting various map options.

**Action** – for executing shell and monet shell commands. External commands and features like traceroute are hooked in here.

**Map** Menu commands for a whole map.

## CALLBACKS AND ACTIONS

Most of the functionality is provided by calling the monetsh command interpreter from a callback or action. These are also available as actions with the CB suffix replaced by ACT.

The following are for executing monetsh commands, before execution the character '&' is replaced by the name of the currently selected entity. The character '+' preceding the name of a text of dialog widget is replaced by the value of the widget.

**MonetShellCommandCB** Calls a bourne shell (sh) command and pipes the output to a window.

**MonetScriptCommandCB** executes a monet shell command. If the first character is a '+' then it is executed directly else the argument is taken as the name of a text or dialog widget which has the instruction.

## CONFIRMATION DIALOG BOXES

Whenever you press a button that may cause you to lose some work or is otherwise dangerous, a popup dialog box will appear asking you to confirm the action. This window will contain an "Abort" or "No" button and a "Confirm" or "Yes" button. Pressing the "No" button cancels the operation, and pressing the "Yes" will proceed with the operation.

## MAP EDITING COMMANDS

Virtually all the commands are available from menus, many of which have keystroke equivalents which are shown in the menus. Operations available while on a map are listed below, these are for popping up menus and using rubberbands. The maps are also available from the top row of buttons.

| | |
|---|---|
| **Ctrl**<**Button1**> | Commands Menu |
| **Ctrl**<**Button2**> | Map Menu |
| **Ctrl**<**Button3**> | Map Options Box |
| **Shift**<**Button1**> | Load maps menu |
| **Shift**<**Button2**> | User Defined Actions Menu |
| **Shift**<**Button3**> | Layout Box |
| <**Button1**> | Move |
| <**Button2**> | Select |

<**Button3**> Resize

## BASIC SCREEN LAYOUT

*xmonet* starts out with a single window, divided into a number of areas:

**Top** Row – A line of buttons with pull-down command menus.

**Top** Right – A box showing the name of the current map.

**Left** – A text box where messages are displayed.

**Left** – A group of buttons with help boxes attched.

## NAME

**xmonet** – X based network mapping and monitoring station

## SYNOPSIS

**xmonet** <monet options ...>

## DESCRIPTION

The *xmonet* program provides a window-oriented user interface to the *Monet* network mapping and monitoring system. Maps can be drawn and monitored interactively and used to build up a database of information about network entities. Maps can be loaded from local files or from a remote server, *monetservd (8)*. Monitoring information is collected by a set of distributed monitoring agents.

## MONITORING

Monitoring is done using the *monetd(8N)* agent. At start up you can connect to your local agent and all the systems on the map will be automatically monitored.

## XMONET AND THE WIDGET CREATION LIBRARY

*Xmonet* uses the Widget Creation Library to produce its windows, scrollbars, etc. This allows the widget heirarchy to be created from the X application defaults file. For more information, see the WCL documentation.

Currently there is only one resource file defined for *xmonet* which uses the Athena Widget Set. Another one could be written which uses the Motif widget set without makes changes to the code. This flexibilty and the ease of making changes were the reasons for incoperating it.

*monet* will not work unless you have entered *XMonet.ad* into the resource database. There are various ways of doing this the simplest being **xrdb XMonet.ad**.

The best documentation to the user interface is *XMonet.ad* itself and you should look at that if you want to customize the front end.

## POSTSCRIPT

Maps can be dumped in A3 or A4 postscript format. The file is dumped in the current directory. Files can also be sent straight to a postscript printer, this is done by dumping the file and then calling a script to actually print it. The scripts pathname is given by the configuration parameter *postscriptprologue*.

The files are printed with a title and trailor noting the author and giving a copyright symbol. The authors name can be set as a configuration option. There is also an option for printing without the header and trailor so that the maps can more easily be included in documentation.

There is a problem with the bounding box information. This is not written correctly when headers and footers are included

**bbfig** <name>.ps can be used to find the correct values.

This is only required if you are going to process the postscript with some other utility, such as including it in a *LaTeX* file.

## FILES

~/.monetsh - local configuration
/etc/monetsh - config directory

## SEE ALSO

monet(1N), xmonet(1N), monetd(8), monetmapd(8), monetrc(5)

## BUGS

- Currently in its infancy, implements no text processing yet or loops.
- Syntax is awful need something much more elegant.
- Select statement needs improvement.

Pipe and redirection not working properly.

## AUTHOR

James Barr, NIKHEF-H, Amsterdam

Only one map can be loaded at a time.

For a more complete set of commands use **monetsh> syntax** which list all defined functions together with a help message for each. For general help use **monetsh -exec 'help;quit'**.

## SOME COMMANDS

**ls-map** lists all the maps in the map directory.

**load-map** <filename> loads the map file and makes it the current map.

**load-edb** Loads in an entity database file, this has the status and topology of the networks.

**discard-map** get rid of current map.

**list-maps** Lists all the current maps.

**select** [attribute=]<pattern> looks in current map for systems matching the pattern and puts them in the selection list. Attributes include *contact, title, capacity, loadin ,....*

**report** [attribute=]<pattern> A short report.

**info** [attribute=]<pattern> A long report.

**list-geometry** [attribute=]<pattern> Lists the label, position and dimensions of selected systems. This is for use with the *layout(1)* program.

**list-links** [attribute=]<pattern> Lists which systems are linked together via an interface. The interfaces themselves are not listed. This is for use with the *layout(1)* program.

**help** A little introduction.

**syntax** A description of all the commands.

**quit** exit and 'q' work too.

## IMPLEMENTATION

The basic command interpreter and control structures are separate from the monet specific stuff. This was done so that new commands could be written in C and added very easily.

## NAME

**monetsh** – Command line interface to Monet

## SYNOPSIS

**monetsh** <monet options ... >

## DESCRIPTION

The *monetsh* program provides a command line oriented user interface to the *Monet* network mapping and monitoring system.

Monetsh supports a very simple script language for database queries and text processing. The idea is to provide an interface that could be used from a command line, inside a script or used to build X interfaces. In addition it is designed to use existing network tools by taking their output directly and processing it for use by monet.

A number of the features described do not work properly or well.

## FEATURES

One of the basic problems with script languages is that they should be very easy to type in on the command line yet have a decent syntax for longer programs. These actually tend to be contradictory, leading to wierd syntax as in 'sh' or consistent but awkward to type in lisp. Monetsh avoids this by having a relaxed syntax for the command line but anything nested in brackets is subject to a somewhat stronger syntax, which is basically lisp like.

Features include, function definition, recursion, loops, conditionals, arithmetic, pipe and database select.

The pipe command is similar to that used in the unix shell but can be thought of as a lisp 'apply' function. For instance it can be used to process each entity returned by the select command individually.

Entity information is loaded from a cache or from a server **monetd(8)**. If connected to the latter then status changes will be notified to the user.

There must be a map loaded before anything can be done, a map is a view of the entity database and is basically a list of entities.

## NAME

**monetdload** – Load executing Monetd agent

## SYNOPSIS

**monetdload** [-a *address*] [-p *port*] [monet options]

## DESCRIPTION

The *monetdload* program loads an entity database file from the standard input and loads it into an executing *monetd(1)* agent. The database format is described in *monetdb(5)*.

This is useful for when scripts are used to create database files, they can be loaded straight into the database where the data will be available to any connected client. It can also be used to load information obtained with other monitoring tools.

## FILES

~/.monetrc - local configuration
/etc/monet - config directory

## SEE ALSO

monet(1), monetd(8), monetrc(5)

## BUGS

Can only take input from standard input, although this could be used: **monetdload** -exec 'load <filename>'

## AUTHOR

James Barr, NIKHEF-H, Amsterdam

**FILES**

~/.monetdrc user configuration file
/etc/monet/monetdrc configuration file
/etc/monet/x25host X.25 translation table
/etc/monet/monetd.conf A list of monitoring request read in at start up.
/etc/monet/mib.txt SNMP Variable definition in ASN.1
/etc/rping Like ping but only sends one query.

**SEE ALSO**

monet(1N), monetrc(5N), variables(5N), libquery(3N), rping(1N)

**BUGS**

– There is no security in the protocol and no checking for redirect cycles. There is no notification of errors in requests or responses.

**AUTHOR**

James Barr, NIKHEF-H, Amsterdam

## CLIENT/SERVER PROTOCOL

The protocol is not really a protocol as such because it is very dumb. All communications is done in either a compressed or uncompressed form of ascii. This aids debugging and also allows the agents to be accessed via *telnet*. Each client has a set of outstanding requests for the duration of the connection. These in no way interfere with other clients, except that the same request given by more than one client will generate only one network query.

**hello <host> <port>** identifies the client, if this matches the servers address and port then the connection is closed because this is obviously a cycle. The situation occurs when chaining redirects.

**delete <#id>** – remove a request.

**clear** – remove all requests from this connection.

**close** – close this connection

**select<name>** – return full status information on the specified entity.

**redirect<host+port pattern>** – this tells the agent to chain requests on to another agent (possibly on some other machine), provided they match a wildcard pattern.

**clear_redirects** – delete existing redirects.

**request<#id name <options>>** – create a new request for this connection. Options for this include: *critical<seconds>*, specify a period of seconds after which an unreachable object is flagged as critical, zero means never; and *frequency<seconds>*, minimum time between queries.

**compact** – return results in compact ascii.

**ascii** – return results in expanded ascii.

<**name**> <**dte address**> [<**user data**>]

**nikhefh.nikhef.nl 204129004353**

A third field is also allowed, this specifies the *user data*. This is given by a string of characters and decimal values, optionally surrounded by double quotes, for example the string of 4 bytes with values 3,4,2,255 might be given by,

**"\3\4\2\255"**

Printable characters can be entered directly,

**"dummy user data"**

Or a combination of both

**"\43ABC\3Q"**

The default *user data* value will be for *pad*, however some routers are configured to have a special value for monitoring purposes.

## LOGS

All logs are kept in the standard log directory. *Monetd* stores all its logs in the same file. Each line contains a record the first field in each line is the type of query. Fields are separated by spaces or tabs. Dates are written as two fields, *yy/mm/dd* and *hh/mm/ss*. Comments are also inserted into the text.

Only changes in the state of an object are recorded. This is done using two records giving the previous and the new state. After a configurable period the state of everything is written out, once a day by default.

There is a problem with how to record changes. If we ping an IP host 20 times in an hour and it was down for all of them then we are pretty sure that it has been down for an hour. If we only pinged once at the start of the hour and once at the end we are in some doubt as to wether or not it has been up at all during the period. Therefore, not only should status changes be recorded but also some metric indicating the frequency of monitoring or a reliability statistics attached to each period.

in that matches the regular expression it is automatically redirected to the second monitor. This process is transparent to the initiating client.

There are a number of issues that still need to be dealt with, such as how to distribute the redirects in an organised matter and what to do when a monitor cannot be reached.

## SETUP

Have a look at the file *monet/etc/monetd.conf*. Enter the names of all the machines you want to monitor continously.

If you wanted to use a special monitor to do all the x25 monitoring that was running on *nikhefh.nikhef.nl, port 4444* you would enter the line,

*redirect nikhefh.nikhef.nl+4444 x25:**

If there were a monitor running on *mon.nerc.hc* that covered the whole of the *herc.hc* site, you might want to use this for monitoring rather than doing it directly. Use,

*redirect mon.herc.hc+9999 *herc.hc**

To monitor *hef-router.nikhef.nl* you first have to select a unique request number (unique in this file), 23 for example, and then enter,

*request23 ip.host:hef-router.nikhef.nl*

If you wanted a special warning if it goes down for more than 5 minutes you would convert 5 minutes into seconds, 300, and enter,

*request23 ip.host:hef-router.nikhef.nl critical300*

No spaces between *request* or *critical* and the number following.

## X.25 ADDRESS TABLE

X.25 address/name translation is done via a file *monet/etc/x25hosts*. This is because there is no standard way of mapping names and although each system has its own way of doing name lookups these either, do not provide enough information or are broken.

The format is simple, one entry per line with fields separated by spaces or tabs, for example,

sockets. The current software has not really been programmed as a daemon and so may not behave as nicely as it should.

**monetd** > /dev/null

## OPERATION

Each connection to the server may have a set of outstanding requests, each uniquely identified by the connection number and an integer. When the conenction is broken all its requests are removed. On start up a special connection is made which reads its requests from a file *monet/etc/monetd.conf*, these will be outstanding for the life time of the process.

A number of sockets are opened at start up, if they cannot be opened a warning will be given but otherwise the program will proceed as normal, without its full querying capability.

Although it is possible to send out many requests at once, this capability should be limited so as to reduce the chance of overloading part of the network. Currently only one query is made at a time, to further prevent overloading a configurable delay is introduced between each set of requests.

New requests are given priority so that users get a quick response. It is also possible to prioritise requests by giving a *frequency* option with each request, specifying the minimum time between queries. This is important because sending out too many queries can easily overload the network or a router.

## REDIRECTS

Monitoring can be distributed across a number of machines by redirecting certain queries to selected monitors of differnet hosts. These second monitors will do the actual monitoring of the specified objects and pass the status information back to the the user via the first monitor. The scheme is recursive, so requests could be passed through a chain of monitors.

The queries to be redirected a selected by associating a simple regular expression with the address of a monitor. When a request comes

## NAME

**monetd** – Distributed network monitoring and database agent

## SYNOPSIS

**monetd** [-p <port>] [-help [1|2]] [options]

## OPTIONS

| | |
|---|---|
| **-help** " level" | Gives a full list of options, if *level* is 2 then the value of each option is printed too. |
| **-p** " port" | sets the port number listened to by the *monetd(8)* daemon. |
| **-testing** | sets test mode, which enables a few untested and dangerous features. |
| **-debug** | sets debug mode, which produces a lot of messages. |

## DESCRIPTION

*Monetd* monitors networks using various protocols, processes the information into a set of status records for each object monitored, keeps logs and returns this information to clients that request it.

## USE

You can run more than one *monetd* daemon for your site, but only one for clients to connect to. Select a machine to run it on and enter the address of this machine and the ip port you selected in the system configuration file *monet/etc/monetrc*

Do **NOT** run with setuid root. It won't hurt but it won't work either. IP pinging is done by a separate program *rping*.

To start the daemon use the command below and expect a few error messages. These are just the result of it not being able to open some

placed somewhat in between these two nodes.
Nodes with only one link to another node are placed close to the node they are connected with.
Nodes without connections to other nodes are placed on the bottompart of the window (at the end of the program).
Nodes with one connection to another node with only this one connection are placed on the upperpart of the window (at the end of the program).

## LIMITATIONS

The different procedures of the program (overlap checking, force_vector determination, resizing) have not been evaluated extensively.
At this moment the number of different structures in the program make it hard not to go nuts. It must be rewritten so that we use the links counted for a node and not the different structures with nodes with 0, 1 etc. links.
It must be made more flexible to handle the nodes with for instance 5 or more links first, then the nodes with 4 links and so on.

## BUGS

The layout can be made more efficient (with a nicer spread around the window).
Time consuming with large number of nodes.

## AUTHOR

Gerard Leurs
NIKHEF-H, Amsterdam, Netherlands

## SEE ALSO

monet(1N)

## EXAMPLES

cat links.database | layout -w 1500 -h 1000 -p
In this example we use a windowwidth of 1500 and a windowheight of 1000 to layout the nodes in the file called "links.database". We also want the results to be saved in postscript.

cat nodefile | layout -r -i 150 -p.
We place the nodes in file "nodefile" at random places in a 2D window (nodes with more links closer to the center). We determine 150 times what the interactions are between all nodes. Finally we produce postscriptoutput.

Possible input consists of the following lines:
con= ip.host:hef-router.nikhef.nl ip.network:192.16.183.0
con= ip.host:hef-router.nikhef.nl ip.network:192.16.192.0
con= ip.host:cwi-gw.cwi.nl ip.network:192.16.183.0
con= ip.host:cwi-gw.cwi.nl ip.network:192.16.192.0
ip.host:hef-router.nikhef.nl hef-router.nikhef.nl 368 432 130 13
ip.network:192.16.183.0 192.16.183.0 330 394 92 12
ip.network:192.16.192.0 192.16.192.0 210 464 368 19
ip.host:cwi-gw.cwi.nl cwi-gw.cwi.nl 258 362 88 13

## DIAGNOSTICS

A large amount of time is consumed when you need a really good layout. When the number of nodes increase the time used for layout goes up and up.

## IMPLEMENTATION

Linked nodes attract, while non-linked nodes repel. The forcevector between two nodes depents on the distance between them.
Only nodes with more than one link to other nodes with more than one link are used for the place_determination.
After every 2 iterations there is a resizing of all nodes with more links on the window (to get a nicely scaled 2D window). After every 10 iterations there is an overlap_check.
Nodes with only two connections to other nodes with more links are

## OPTIONS

**-c** s Use characterstring s for the connection-typename between the linked nodes. Default is interface=.

**-f** f Use factor f for determination of the force_vectors. This factor is used in the determination of the forces between the nodes. When you want a stronger interaction between them, you have to enlarge this factor. When you want to reduce the interaction you have to enter a factor between 0 and 1. Default is 1.

**-h** x Define the height of the 2D window to be used for the layout. This must be large enough to fit all nodes into it (and move them around). See also -w. Default is 800.

**-i** x Define the number of iterations. If you do not specify the number of iterations it will be 15. A large number of iterations (e.g. 100) is recommended.

**-p** Produce postscript-output ("/tmp/psfile"). This gives a possibility to see how the network will look like after a number of iterations. Default is no postscript.

**-r** Put the nodes at random places on the window. Nodes with more connections to other nodes get closer to the center of the 2D window. Default is off. However when a node has a zero x-coordinate it gets a random x_coordinate and when it has a zero y-coordinate it gets a random y_coordinate.

**-s** Do not check maximum boxsizes. Use this option only if there are nodes with large sizes. The size of the box depends primarily on the length of the name of the node. When a node has more connections the size is somewhat enlarged. Default is sizecheck.

**-w** x Define the width of the 2D window to be used for the layout. See also -h. Default is 1000.

## FILES

/tmp/psfile The file with postscript output of the program.

## NAME

**layout** – layout a network / graph in 2D.

## SYNOPSIS

layout [options]

## DESCRIPTION

Layout creates a 2D network graph from a list of nodenames and connections between the nodes. Nodes are drawn as rectangular boxes with the name of the node inside the box.
Connections between nodes are drawn as lines between the boxes.
Layout produces its output after a number of iterations in which the nodes get their position in the graph. It checks the size of the boxes, and uses these to prevent overlapping of nodes. layout also makes sure that the boxes stay within the window.

Layout reads from standard input, which must consist of lines with 3 or 6 fields:
The 6 field line has the following syntax:
id nodename x_coord y_coord width height.
Where id is the networkidentifier of the node, nodename is the name of the node, x_coord is the initial x_coordinate for this node, y_coord is the initial y_coordinate for this node, width is the initial width of the box and height is the initial height of the box.
The 3 field line has the following syntax:
interface= nodename1 nodename2.
This line specifies a connection between the two nodes nodename1 and nodename2.

Layout produces output which consists of lines with 3 or 6 fields. The 6 field line gives us the networkidentifier of the node, the name of the node, the final x_coordinate, the final y_coordinate, the (possibly) adjusted width of the box and the (possibly) adjusted height of the box. The 3 field line is just a copy of the 3_field_line input.

# Appendix H

# Application Manual Pages

## SEE ALSO

xmonet(1N), monetsh(1N), monetdload(1N), monetd(8N), monetmapd(8N), X(1),

## AUTHOR

James Barr, NIKHEF-H, Amsterdam

On start up the system configuration file is read *etc/monet/monetrc* followed by *~/.monetrc*. Command line options are read twice, before and after the config files. In addition two more config files will be read if they exist */etc/monet/<program_name>rc* and *~/.<program_name>rc*.

## COPYRIGHT

Copyright (C) 1991 NIKHEF-H Kruislaan 409, 1009DN Amsterdam, The Netherlands

Monet is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 1.

Monet is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Monet; see the file COPYING. If not, write to the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.

For more information please contact James Barr or Rob Blokzijl at PP NIKHEF-H, Kruislaan 409, 1009DB Amsterdam, The Netherlands

or a38@nikhef.nl, k13@nikhef.nl

## FILES

/etc/monet/* – basic directory
~/.monetrc - User configuration file.
/etc/monet/monetrc - System configuration file.
/etc/monet/x25hosts - X.25 address translation table.
/etc/monet/mib.txt - SNMP MIB File in rfc format.
/etc/monet/monetd.conf - default list of queries for monetd.
/etc/monet/maps.PS - Postscript Prolog for maps.
/etc/monet/class.edb - default definition of entity classes.
/etc/monet/class.edb - default entity database.
/etc/monet/maps - default map file directory.
/etc/monet/log - default logging directory.

**monetdload** – this utility is used to send new data to a running *monetd* agent. One use would be to get networking information from some existing monitoring tool, massage the format into that usable by monet and then load the information making it available to anyone using the system.

**monetmapd** – this daemon allows remote access to a set of map files. The idea is that once a map is completed it is deposited in one place for use by everyone else. There should be such a server running on *nikhefh.nikhef.nl*. The maps can be read, written and listed using *monetd* or *xmonet*.

**rping** – This is derived from *ping(1)* but only does one ping at a time. It is used by *monetd* for queries.

**layout** – this program takes a network and tries to lay it out nicely in 2D. It is far from perfect but takes an awful lot of the work out of doing it by hand. At the time of writing it has not been integrated into *monetsh* or *xmonet* but this will not require a lot of work. The could be a very useful program for producing maps of arbitrary bits of a network.

**makemaps** – This is a script for producing a network topology database for a set of domains. It looks in the nameserver and tries to identify those machines it thinks are the most important. Not very effective currently but could be made very powerful.

**snmp*** – these are part of the CMU SNMP distribution, they are included here because they are called by *xmonet*. In addition they have been altered slightly.

## OPTIONS

Options can be set in configuration files or on the command line and are the same for all of the programs although some may not be used. For a complete list and description use *monetsh -help 2*.

## CUSTOMIZATION

You should look at **monet/bin/xmonet/XMonet.ad** for customizing the X interface and also for specifying external commands like ping, traceroute, snmpget etc. These will probably be somewhere else on your system and you will want to include your own commands too. Comments are included in the file.

## OVERVIEW

**monetd** – you may wish to run more than one of these agents, they are used for network monitoring. Clients like *monetd* and *xmonet* can connect to them and have the network status shown as it changes. The agents can also talk to each other so that all requests for a particular type of query or queries for a particular domain can be done by an agent close to the systems being monitored. The system in fact maintains a distributed network entity database including the network topology. This can maintain any number of classes of entity although it may be able to monitor a small subset of them.

**xmonet** – this is an X windows based system for creating network and viewing network maps. It includes a reasonably powerful editor, tools for building the maps, remote access to maps, can view network status in real time and can work with scripts and other programs. The output of these programs or shell scripts can be displayed in a window or used as instructions for *xmonet*. Maps can be displayed within each other allowing a zoom facility. The system and user interface are quite configurable.

**monetsh** – this is a command line version of *xmonet*, and provides much the same functionality. It includes a very basic query/script language and so can be used in shell scripts to perform batch processing of network information.

Maps can be built fairly easily using a reasonably powerful graphics editor is available to build the maps interactively. There are also some tools built in and some scripts for auto-discovery and for automatic layout.

Changes in network status are automatically shown. In addition it is quite easy to use external programs and their output with *monet*.

The maps can be saved locally or in a remote repositary so that other people can share them. The status and topology information is stored separately from the maps, in a local file or in the monitoring agents.

## INSTALLATION

You should be able to get the compressed distribution *monet.tar.Z* from *nikhefh.nikhef.nl*. Once you have it on your machine use the following command to get the source directory.

**zcat** monet.tar.Z | tar -xf -

**cd** monet

**xmkmf**

**make** Makefiles

**make**

Monet uses *imake (1)* to create system specific makefiles. You should first edit *Monet.tmpl* to set things up for your local system. In addition you should edit *monet/lib/wcl/Wc.tmpl*. If you cannot get imake to work then try editing the existing makefiles by hand. Imake is available with the X(1) windowing system.

At the time of writing Apollo and Digital distributed only X11 Release 3 libraries. The Monet software requires X11 Release 4. This is not a problem because it is pretty easy to compile MIT release 4 software and just use the client libraries. Just ftp tape one from a convenient server, mcsun.eu.net for example. The libraries must be created with X11 Release 3

X.25 support varies greatly from machine to machine and so needs to be implemented especially for each. Currenly Apollo and Gould-NP1 is supported, but it should be simple enough for other types.

## NAME

**monet** – a generic, distributed network Mapping, Monitoring and Management Workstation

## SYNOPSIS

Monet is a generic tool for distributed gathering and viewing computer network statistics and toplogies.

The code is written in C for Unix(BSD), the mapper requires X11R4. Protocols supported are IP/TCP/X25/SNMP. Much work is needed on the monitor. Output is on the command line, X or to postscript.

Imake is used to create the makefiles, if you don't have it you can get it from the X11R4 release or simply by asking around.

The software has been developed and tested on an Apollo SR10.2 /BSD4.3 with X11R4, but a number of other systems have been used during development, including a DECStation 3100/ Ultrix, Sun, Gould/NP1, and a Silicon Graphics Iris.

## DESCRIPTION

Monet is a set of programs for monitoring and mapping multiprotocol (Wide Area) Computer Networks. It tries to be generic by allowing users to add new types of network object without writing new code and by isolating protocol specific modules.

A set of distributed agents collect and cook data, answer queries directly or chain requests on to other agents. Non-statistical information or statistical data from other sources can be loaded into running agents. A client based on X and a command line client are avaliable, as are some scripts for producing maps.

Currently the software consists of an X11 based mapping program and a network database/monitoring agent. The monitoring program does all the logging and can support multiple clients, it requires no X11. There is also a command line version of the client and a program for loading the *database / monitoring* agent while it is running.

# Appendix G

# Installation

layout program needs to be integrated properly with xmonet
and monetsh.

. Auto-discovery

This needs a lot of work, since it is so useful. The current
Explore
stuff is primitive and not very easy to use. Shouldn't be
difficult.
Need SNMP discover.

. Heirarchical Maps

It is nice to be able to have maps drawn within maps so that
zoom
can be implemented, it also helps with layout. The basics
have been
done but needs to be fininshed. Also need to be able to use
this with auto-layout and auto-discovery.

. Whois interface

Much useful networking information is kept in a whois databse
on mcsun.eu.net.
If monetd could act as a whois server then this information
would be available
directly.

```
Stepped storage, can specify steps per attribute
64*1month
64*1week
64*1day
24*4*8*15min
Access tools, quick compressed to ascii.
Quick retrieval of history of an attribute for an entity
so graphs can be drawn interactively.
Only record changed values but keep a record of the frequency
queries were made.

Get number of values stored for
get <entity> <attribute> -> <from> <to> <#> <mean period between
queries>
Get a list of the values stored in a period.
select <entity> <attribute> <from> <to> <#> -> list of  <date,
value>

. Script Query Language

The basics of a script language for accessing monet have been
implemented
in monetsh, but much needs to be done.
This would be able to perform all of monets
functions, provide report generation support and provide string
processing
facilities so that the output of existing tools could be used
directly from
monet without using sh or awk.  Need to fix redirect an piping
so that it
can be properly ingrated with external programs.

. Layout and Untangle

Should be able to at least help with laying out networks, even
if
it does not prove possible to do the whole job automatically.
The
```

# Appendix F

# To be done

```
$Header: //gaia/group.local/computer/monet/RCS/TODO,v 1.1 1991/06/13
09:52:16 a38 Exp a38 $


. Network Query

The stuff in monet/lib/query does all the network monitoring.
This needs to be improved for efficiency, should do async
monitoring, various queries should be integrated for the same
result.  New protocols should be added.  The snmp stuff needs
a lot of improvement and fixes.

. Example Maps

The examples aren't terribly good or complete.  It would be
nice
to do the ripe map entirely using monet and then collect some
monitoring stuff on it.

. Logging

Needs a proper logging subsystem.

Real time additions, interactive access.
Compressed storage.
```

```
request31 ip.host:nikhefh.nikhef.nl frequency20
request32 ip.host:nikvx1.nikhef.nl frequency20
request10 dns.nameserver:nikvx1.nikhef.nl frequency90
request11 dns.nameserver:nikhefh.nikhef.nl frequency90
#request40 x25.host:haseoc critical3000 frequency60
#request41 x25.host:nikhef critical3000 frequency60
request42 ip.interface:192.2.3.4 frequency60
#request45 x25.host:nikhef critical3000 frequency60
```

```
# will be used, as declared in the file 'class.edb'.

# Monitor IXI

#request3211 x25.network:amsterdam
#request3212 x25.network:dublin
#request3213 x25.network:lisboa
#request3214 x25.network:brussels
#request3215 x25.network:bern
#request3216 x25.network:vienna
#request3217 x25.network:bologna
#request3210 x25.network:athens
#request1 x25.network:arhus

#request211 x25.host:haseoc
#request212 x25.host:win
#request213 x25.host:vub
#request214 x25.host:garr
#request215 x25.host:surfnet
#request216 x25.host:nordunet
#request217 x25.host:jtas
#request218 x25.host:circe
#request219 x25.host:nikhef
#request210 x25.host:rccn
#request2112 x25.host:artix
#request2113 x25.host:heanet
#request2114 x25.host:earn.ie
#request2115 x25.host:cern
#request2116 x25.host:aconet
#request2117 x25.host:switch
#request2118 x25.host:ariadne
#request2 x25.host:janet
#request3 x25.host:telepac
#request6 x25.host:ispra
#request4 x25.host:datanet1
#request5 x25.host:dcs

# Monitor some local nikhef stuff
```

# Appendix E

# Monetd default requests

```
# $Header: //gaia/group.local/computer/monet/etc/RCS/monetd.conf,v
1.1 1991/07/17 07:00:08 a38 Exp a38 $
#
# These are default requests for the monetd agent.
# you should edit this before starting of.
#

# Requests can be done locally or via a remote agent,
# these specify the TCP address <host+port> of other
# agents followed by a wildcard pattern specifying
# which requests should be redirected there.
# If the agents cannot be reached, then the queries will
# be done from the local agent.

redirect nikhefh.nikhef.nl+2567 x25.host:*nikhef*
redirect clio.nikhef.nl+2567    ip:*nikhef*
redirect idas.nikhef.nl+3333    tcp:*

# Each request should have a unique positive integer as
# an identifier.
# The critical attribute specifies a period of down time
# after which the system should be treated as 'critical'.
# The frequency is the minimum period of time between
# sending out two queries for this entity.
# If the attributes are not set here a default value
```

```
reverse_video   - Put maps in reverse video
(-map_rv) = [0]
testing         - Enable untested/dangerous features
(-testing) = [0]
debug           - Debug mode
(-debug) = [0]
tcp_dns         - Set up tcp connection to nameserver, instead
of using IP
(-tcp_dns) = [0]
```

```
(-a) = [localhost]
monetmapd_host  - Host name of map server
(-ma) = [localhost]
etc_directory   - Where monet specific files are kept
(-etc) = [/user/a38/monet/etc]
map_ps_prolog   - Postscript prologue for maps
(-mappspro) = [maps.PS]
log_directory   - Directory where logs are kept
(-logdir) = [log]
ideas_log       - Name of ideas log file
(-ideaslog) = [ideas.txt]
snmp_mib_file   - File name of Snmp MIB in RFCxxxx format
(-snmpmib) = [mib.txt]
x121_file       - File name of x25 host/address translation
table
(-x121file) = [x25hosts]
ps_lpr          - Command for sending files to a postscript
printer
(-pslpr) = [pslpr]
rping           - Name of a program for doing one-off IP pings
(-rping) = [/etc/rping]
startup_command - A monet script that will be executed at startup
(-exec) = []
x25_delay       - Period between checking x25 host/address
file has changed
(-x25delay) = [300]
reconnect_delay - Period(secs) between trying to reconnect
to monetd servers
(-reconnectdelay) = [660]
log_period      - Period(secs) between writing out batched
status to log file
(-logperiod) = [43200]
monetd_delay    - Period(secs) between sending out network
queries
(-monetddelay) = [30]
monetd_port     - Tcp port of monetd server
(-p) = [2567]
monetmapd_port  - Tcp port of map server
(-mp) = [2568]
```

# Appendix D

# Configuration Options

```
monetsh -help 2
************************************************************************
Monetsh version 0.2-Beta, Copyright (C) 1991 NIKHEF-H, Amsterdam
Monetsh comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute
it
under certain conditions.  For details see documentation.
************************************************************************

usage: monetsh [-help [1|2]]  <monet options> ...
maps_source     - Where maps are available from
(-mapssource) = [ftp nikhefh]
maps_author     - Who drew the maps
(-mapsauthor) = [a38]
maps_contact    - Who to contact about the maps
(-mapscontact) = [a38@nikhefh]
monetd_conf     - Default requests file for monetd
(-monetdconf) = [monetd.conf]
edb_file        - Cache/default Entity Database file
(-edbfile) = [monet.edb]
map_directory   - Directory where maps are kept
(-maps) = [maps]
class_edb       - File defining entity classes
(-classedb) = [class.edb]
monetd_host     - Host name of monetd server
```

# Appendix C

# Xmonet Applications Defaults

This describes the xmonet user interface, but I haven't worked out how to include it in tex yet.

```
behaviour - The behaviour of an entity, normal, chaotic, ...
future - Future of an entity, temporary, certain, planned,
...
reach - Reachability of an entity
contact - Who to contact in case of problems
ping - Instruction for monitoring entities
title - The title of an entity (domain name)
translation - Rules for translating one class of map into another
instances - Classes instances are maps, systems, itnerfaces
or none
underlying - This can be an underlying component of another
entity
unbending - Interface will be straight
anonymous - Name will be generated randomly
resizable - Can be resized
shape - Shape used when drawn on map
format - Title must match this pattern
```

# Appendix B

# Entity Attributes

```
monetsh -exec 'attributes;quit'
************************************************************************
Monetsh version 0.2-Beta, Copyright (C) 1991 NIKHEF-H, Amsterdam
Monetsh comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute
it
under certain conditions.  For details see documentation.
************************************************************************

[User Error] Sorry, can't open or create config file [~/.monetrc]
Loading entity database from file [/user/a38/monet/etc/class.edb]
Reporting on [Attributes]
component - List of component entities of an entity
interface - Two entities interfaced together
action - A <monetsh> command, executed when entity is clicked
pktout - Total number of packets out
pktin - Total number of packets in
octout - Total number of bytes out
octin - Total number of bytes in
loadout - Percentage traffic out over capacity for a very recent
period
loadin - Percentage traffic in over capacity for a very recent
period
connections - Number of connections to an entity.
capacity - Capacity of entity, usually speed
```

```
For details of all defined functions, use 'syntax', or for
a specific
command, 'syntax <function>'
Functions can be grouped by enclosing them in ( and ).  Arithmetic
is
prefix and can only be done inside (...).  Inside (...) strings
must be
surrounded by quotes.  A sequence of functions can be seperated
by,
&& If left succesful then do right
|| If left unsuccesful then do right
; Do left then right
```

```
 localhost - select local host
 layout - layout/untangle map
 random - random layout/untangle map
 map - change current map
 list-maps - list loaded maps
 ins - insert <drawing> <x> <y> <width> <height>
Insert/create a drawing in a map at particular position
 mapset - set map attributes
 discard-map - discard map
Get rid of current map without saving.
 free-map - free/delete map
 promote-maps - promote common drawings in maps
Looks in the sub maps of the current map for ...
 action - Perform a class specific action
Each class of objects has an action associated with it.
-----------------
 left - left <string> n
 input - commands from a file
 * - * <a> <b>
 / - / <a> <b>
 - - - <a> <b>
 + - + <a> <b>
 loop - loop {sequence}
 if - if {condition} {do}
 sh - Unix (sh)ell command
 debug - debug on|off
 fail - always fail
 false - always fail
 true - always succeed
 def - Define a macro
 ver - which version of the software
 copyr - copyright message
 echo - echo to output
 config - set configuration parameter
 syntax - command syntax
 help - help information
 exit - quit
 q - quit
 quit - quit
```

```
 translate - translate map into another class of map
 layout - layout /untangle map, <iterations>
 rput-map - put remote maps
 rget-map - get remote map
 rls-map - list remote maps
 ls-map - list local maps
 save-maps - save all maps
 save-map - save current map
 load-map - load map file <filename>
 put-edb - send entity database to monetd
 save-edb - save entity database <filename>
 load-edb - load entity database <filename>
 disconnect - disconnect monitor
 reconnect - reconnect monitor
 list-redirect - list query redirects
Queries can be redirected to a number of agents. This command
lists them.
 clear-ping - clear ping
 request-ping - request ping
 s-list - list selected
 free - free selected
 stat - status
A quick summary of the state of the program
 select - select <pattern>
 report - short report on entities
 list-links - lists the linked systems in the current map.
any interfaces with two connections have the connections printed.
 list-geometry - lists the geometry of every drawing on the
current map
format: <name> <label> <x> <y> <width> <height>
 info - long report on entities
 shapes - list available shapes
 attributes - list defined attribute
 delete - delete entities
 contacts - display entities contacts
 join - join drawings
Link drawings together.
 detach - detach joined entities
 explore - network auto-discovery
```

```
 helpmode - Help Mode
obsolete !!!
 select-pinged - Looks in selection list for entities being
monitored
This is dubious now
 select-pingable - Looks in selection list for pingable entities
This is dubious now
 unmake-subcomponents - Removes selected entities from subcomponent
list
Must first select a super component
 make-subcomponents - Make selected entities subcomponents
Must first select a super component
 select-supercomponent - Select super component
This is used when making subcomponents
 report-supercomponents - Report on supercomponents of selected
list
 select-supercomponents - Select supercomponents of selected
list
 report-subcomponents - Report on subcomponents of selected
list
 select-subcomponents - Select subcomponents of selected list
 redraw-all - redraw all maps
 redraw - redraw map
 gather-info - gather networking information
This is a bit dubious
 report-class - status report on a class
 status-report - status report on selected list
 set-capacity - set capacity of entity, positive integer
Depends on class of entity what this actually means
 set-future - set future status on selected list
Future can be certain, temporary, planned or unknown
 getmonetmapd - return address of monetmapd
 getmonetd - return address of monetd
 currentmap - return name of current map
 gettop - get top of selection list
 straight - straighten interfaces
 dump-text - dump map as text to file
 dump-ps - dump map as postscript to file
 lpr - send map to postscript printer
```

# Appendix A

# Monetsh Commands

```
**********************************************************************
Monetsh version 0.2-Beta, Copyright (C) 1991 NIKHEF-H, Amsterdam
Monetsh comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute
it
under certain conditions.  For details see documentation.
**********************************************************************

[User Error] Sorry, can't open or create config file [~/.monetrc]
Loading entity database from file [/user/a38/monet/etc/class.edb]
Syntax of Commands
 class-map - Make current map a class map
Map must be empty
 map-set-grid - Set drawing grid for map
makes positioning lined up drawings easier
 map-set-label - Set Label state for the map
 map-set-domain - Set default domain for this map
Used to shorten the labels of drawings in the map
 mark-map - Make current map the 'marked' map
 noteidea - Note an idea
obsolete !!!
 exithelpmode - Exit help/ideas mode
obsolete !!!
 ideasmode - Ideas Mode
obsolete !!!
```

**x25.host** For X.25 end systems, should be identified by domain name.

**x25.network** This is obsolete, use router instead. It was used by accident when making the IXI map.

**x25.router** For X.25 routers.

**snmp** For modelling networks using snmp.

**snmp.host** An snmp host, should be identified by domain name.

**snmp.network** This is an ethernet or a leased line and

**snmp.interface** Describes an interface on a host. It can be linked on one side to a snmp.host and on the other a snmp.network.

**sna** If you want to do anything with SNA you best redefine all this since it probably does not reflect what is needed.

**decnet** See *sna*.

**physical** This is for objects that would be described as OSI layers 1 and 2 entities. It includes ethernets, leased line, Sun Workstations etc.

There are two basic interface classes. Their names are a bit long winded and should be shortened. They a both have the monitor instruction,

*ping= snmp_interface:public+& critical900*

If using snmp to monitor an interface it should be formated as a domain name followed by a '+' followed by the interface number of the snmp agent it is connected to.

**physical.network.line.long_distance** For long distance lines, it would be nice if they were drawn as zaggies of some other such shape.

**physical.network.line.local** For local links.

There are numerous system classes, for networks and systems. There is also one for *sites* where a line is thought of as ending at a place not a particular machine.

**Button** This ought to be a subclass of *Annotation*. *Buttons* are used for selecting sets of systems, for instance we might want a button that when *"pressed"* would select all the systems that belonged to *EUnet* or *RIPE*. This is done by making a set of objects subcomponents of the button.

**dns** There are two system subclasses,

**dns.nameserver** Instances must be named by domain names.

**dns.domain** These are given the domain name of the domain they represent.

and two interface classes,

**dns.primary** Links a domain to its primary nameserver, the identifier should probably be random.

**dns.secondary** Links a domain to is secondary nameservers.

Nameservers are linked to domains by the the *primary* and *secondary* interfaces. Currently it is only possible to monitor the nameservers but it should also be possible to check on the validity of domains and interfaces.

This was only included originally as an experiment but could actually be useful [7].

**ip** This are a mappers dream when compared to some of the others protocols, basically because the connections between systems can be rigorously checked can consistency and validity.

**ip.host** An IP host identified by a domain name.

**ip.network** An IP network identified by IP address (abc.bcd.efg.0).

**ip.interface** An IP Address. This is an interface class and is used to link hosts to networks. The title will be of the standard format (abc.bcd.efg.hij).

Hosts and interfaces can be monitored quite easily. It would be nice to use some snmp stuff to get information on networks and use *getent* to get a user-friendly name for them.

**x25** An X.25 interface is usually a leased line although it could run over a multidrop network. There is very little that can be done to monitor them.

Three system classes are defined, these can be monitored to an extent.

[7] Would anyone want a network showing MX records?

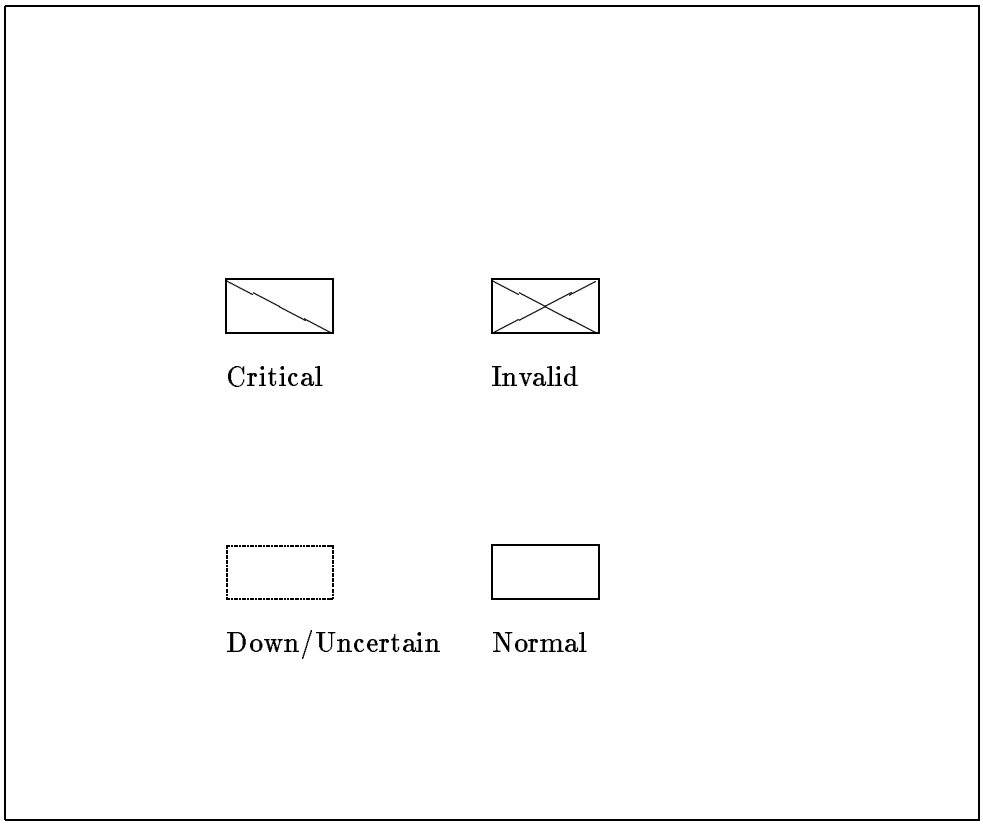


Figure 3.2: Map Symbols

**top** This is a special top level class created by the system. All others are subclasses of it. Its instances are maps which instances of any class can be inserted into. Two instances of top are always created, the universe map :Universe and the class map *:Class*.

**Interface** This was a kind of general purpose link between any two system classes. It is obsolete and must not be used!

**Annotation** It is nice to be able to add a few extra features to the maps for clarification and so on. Subclasses of annotation are used for this, eg. *Annotation.string* and *Annotation.line*. Currently these are not well implemented and should not really be used.

### 3.5.1 Map Symbols

The status of objects drawn on a map is indicated visually. The list below is a quick note of how this is done. It must be remembered that each object is drawn as a rectangle.

The traffic going into and out of a system is indicated by marking links from it. If 20a thick line it indicates a 20is traffic/capacity.

**Flag** If it is up then the system is reachable, if down then it is not. If half way then we don't know.

**Gray** This mean that the future of the object is uncertain or unknown.

**Single Diagonal Line** The system has gone critical, it has been unreachable for more than a short while.

**Two Diagonal lines** The object is crossed out. This means the system is in some sense invalid, eg. a host not in the nameserver.

### 3.5.2 Shapes

A number of shapes are provided for drawing objects. A few basic ones are listed here, for a full list try

$ *monetsh -exec 'shapes; quit'*

**simple** Simply a rectangle surrounding the system name.

**network** A filled in rectangle surrounding the system name.

**interface** This is a tiny square, it cannot be resized.

**doublering** For token rings.

## 3.6 Standard Classes

A number of classes come with the distribution, some of them are required and will generate error messages if not present [6]. For a full list look in the file monet/etc/maps/:Class or simply expand the class map by selecting from the map menu.

[6] This will probably change in a following release.

operation performed on an IP host would draw all the networks it is connected to.

This is a nice feature for creating maps. Unfortunately, it will only work for a limited number classes.

**Traceroute** Select a host on the map and all the intermediate machines between it and your machine will be selected. Currently only available for IP [5].

All systems found are listed in the report box, only systems actually on a map will be selected.

**Report** Produce a detailed status report on a selected object.

**Translation** It is useful to create a physical map of the area of interest and then derive ip/x25/etc maps from it. Translations are monets way of helping this process. It works by defining a mapping from classes in the source map to classes in the new one.

For instance we might say that all drawings of class *physical.system* will be translated into *ip.host*. This would translate all drawings of any subclass of *physical.system* into *ip.host*, including for example *physical.system.apollo* or *physical.system.sun*.

Translations are defined in the entity database entry for the target map. For example the table for *ip.topological* might be,

translation= physical.system ip.host *physical.linkip.interface* physical.network ip.network *x25.hostip.host* x25.interface ip.interface

**Layout and Straighten** Monet provides a simple procedure for laying out maps on the screen with the minimum of overlaps. This is very useful for when the maps are generated outside xmonet, by scripts for example and have no geometry information.

The algorithm used simply treats the nodes in the network as mutually repulsive particles in space if they are not linked and attracting particles if they are. This requires many iterations to get right and is O(n*n-n) so will be very slow for large networks.

There is also a procedure for positioning interfaces so that they lie between or next to the systems they connect and the connections will be drawn straight if possible.

[5] IP version not yet finished.

**Setting Attributes** The user can set a number of attributes on a system such as its *future* status, name and its capacity. Some of these may be changed if monetd returns a different value.

**Straighten Links** Links that are either horizontal or vertical are pleasing to the eye. The command tries to rearrange links into nicer positions. It is pretty good, but does not look to see if any objects are in its way.

It should not take much improvement before this could simply be done automatically every time an object is moved.

**Update/Gather** These instructions are currently a bit of a mess because they try to query the network directly instead of using monetd. They will be used to make monetd work a bit harder and return as much status data as it can and use this information to check the consistency of the database.

**Make Component** Select one object as the supercomponent and then select all the objects you want as sub- or super-components of it.

**List Component** You can list the sub or super components in the report box or put them in the selection list. This is very nice if you want to know which parts of the network belong to EUnet say and which to RIPE. The effect of selections is cumulative so you much free all selections before starting.

**Free Selection List** Deselect all the items on the selection list, this is a menu option and can also be done quickly with the mouse.

**Help** This has not been well implemented as yet. The idea of *help mode* is to press a button and get some help text instead of the actual operation. The help text would look very much like that in the section.

**Join** This is used to connect two systems together or an interface and a system. In the former case the user must select the class of the interface to be used.

If the classes do not agree or restrictions apply then the join will be refused.

**Detach** Disconnect an interface and a system.

**Explore** This option finds all the neighbours of a selected object and creates new objects on the map linked together. So an explore

**Creating Classes** If you want to monitor an SNA network, you will have to write software for doing it. Drawing a map is not such a problem, however, as these are be defined in monet/etc/maps/:Class.

You need to define at least four classes. One for the protocol *sna*, one for maps *sna.map*, one for systems and one for interfaces. As the author knows nothing about SNA, nothing will be said about what to count as an SNA system or and SNA interface. That is for the user.

See the definition of *ip ...* for use as a template.

Classes can be created interactively but not all the possible options can be set.

**Actions** It is nice to be able to associate an action with an entity or a class of entities. This is done using the *action=* attribute in the *class.edb*, class definition file. The value is a *monetsh* instruction. Some standard ones are listed here.

**Button** select entities which have a name similar to the buttons. For example an entity called *Button:cern* would select every entity which matched the pattern "*cern*".

**System/Interface** status report.

**Class** class report.

**Deleting Objects** Beware, no Undo operator is provided. Operations for deleting one or deleting many objects are provided, the latter prompts for confirmation. Maps currently cannot be deleted, edit the map files for this.

**Technical Contact** Finds the address of someone to contact if there is a problem with a network entity. [4]

**Map Format** Maps may be drawn in more than one format so that different aspects of the database can be seen.

**Map** Each object is placed at a particular (x,y) position, with links between connected objects drawn as lines. State is indicated in a hopefully striking manner, so this would be the normal view.

**List** One line per objects with a list of current state information expressed textually.

[4] Doesn't work as yet.

**Components** an object can have links to a set of objects we call its *sub-components*. In turn a object can have one or more *super-components*.

The subcomponents of some Object Types have special meaning,

**Class** a classes subcomponents are its instances. (Sure?)

**Map** a maps subcomponents are are the objects placed on it.

**Button** pressing a button should list its sub-components. Components can be of any class. Circular components should not be allowed but is currently not completely checked for (Sure?).

Components are useful for defining the relationship between objects. for instance we could have a Button called ABC-NET and all the systems and interfaces that made up ABC-NET would be subcomponents of the button. As an object can have many super-components, a system that was also part of XYZ-NET would also be a subcomponent of another Button.

More awkward would be to define the various components that actually made up an IP network. These might be simply one or more ethernet segments drawn on a 'protocol.physical' map. Or they could be the a set of leased line and routers running various other level three protocols, which in turn would have subcomponents.

**Contact** A technical or administrative contact for a network object. An e-mail address of someone to contact in case of emergency. This is printed out in case an object goes critical, it could also be used to automatically generate e-mail.

**Current Map** This is the map curent operations are performed on.

## 3.5 Basic Operations

This section describes some of the basic functions provided by *monetsh* and *xmonet*.

**Creating Instances** Before creating a new object, you must selected a Class. If you create an instance object then it will be an instance of the chosen class.

**Button** these objects can a set of subcomponents. When the button is 'pressed' all its subcomponents are selected. This is defined as a system currently but ought to be annotation and should not be interfaced to.

**Object names** these have the syntax given in figure 3.1

| | | |
|---|---|---|
| object_name | ::= | <class name> [ : <instance name>] |
| class_name | ::= | <class name> . <label> |
| | \| | <label> |
| instance_name | ::= | <character string> |
| label | ::= | <character string, without '.'s> |

Figure 3.1: Object Names

A class object will obviously have no <instance name> part.

**Status** Each system or interface is a *model* of a real network object. The status of the real object is reflected by the Objects status. This defines a small set of parameters that hopefully apply to all sorts of network device regardless of protocol of operation. The status parameters currently include *reachability*, *capacity* and *load*.

It would be nice to record a status history too.

**Class Map** all Classes are placed on this map. They are automatically positioned and the user cannot move them.

**User maps** the user can create and delete these at will. An object can be placed at any position on the maps the user chooses.

**Selection List** this is a list of Instances that have been selected by some operation. Those objects on the list are drawn with an extra border so are easily seen. The user can select the objects in various ways using the mouse, in addition some operations will respond by putting a set of objects on the list. Listing the instances of an Class for example. Classes cannot be added to the selection list.

Some operations will subtract from the correct selection list, eg. selecting monitored objects will look through the current list of selected objects and remove those that are not being monitored.

Some operations will empty the list before adding new objects to it, others will not. An abortive operation usually, will not affect the list.

**Instance Types** in addition to the class mechanism we say that a Class is one of three basic types, this is inherited by the instances and cannot be overridden. The types are,

**System** These objects can only be connected together via Interfaces. They are the primary building blocks of the map. A Host or a network may be represented by a System.

**Interface** This is a link between two systems and thus can have no more than two connections. In other ways it is identical to a System. On an IP Map, for example , a particular IP address might be represented by an Interface and would connect an IP Host to and IP Network.

**Map** Maps are defined as hierarchy. Each map is of a particular class and only instances (Systems, Classes, sub-Maps) of sibling or subclasses of the map can be inserted in the map.

For example, if the map had either of these classes,

**ip.topological**

**ip.political**

then instances of the following classes could be placed on it,

**ip** [3]

**ip.political**

**ip.interface**

**ip.host.router**

while instances of these classes could not be inserted,

**x25.topological**

**sna**

An Object can appear on only one map at a time, this is really an unnecessary restriction.

**Annotation** this is not currently well defined or implemented. These are things added to the map to for clarification only and have nothing to do with the network objects.

[3] Actually this is a dummy class and can have no instances

A map may have child maps. These are strictly heirarchical.

## 3.3 Politics

The model should include something about politics since it is does intrude a great deal on practical networking.

## 3.4 Definitions

Everything that appears on a map is a *Visual Object*. This is a generic data type which can have a set of attributes and a set of links associated with them. The object orient model the data-model is based on divides objects into two types.

**Classes** these Objects describe a type of object, for example, ethernets or X.25 hosts.

**Instances** these Objects describe a particular object, for example, the Ethernet thats going to be installed in my corridor if we have enough money, or the machine in the other room running it's version of the X.25 standard.

In the model used, each Instance Object must be linked to exactly one Class Object.

**Attributes** a list of attributes is associated with each object. A set of attribute types are defined internally and the user may define their own (although this is not currently useful). An attribute may have a string, integer or boolean value.

**Inheritance** rather than setting all the attributes for each Instance, an attribute value can be set for a Class Objects and all its Instances will use this value, unless overridden. This provides a useful defaulting scheme.

For example, we might define the shape attribute of a token ring class to be a circle. This attribute would only be set once in the definition of the Class Object and be 'inherited' by all the Instances.

The inheritance mechanism provided is not fool proof or particularly regular.

### 3.1.3 Namespace

The naming of entities is modelled on Domain Names but with the the name of the entities class prepended to it. In fact the class heirarchy can be looked at as a new domain.

So the entities are also arranged in a heirarchical namespace which may have little little to do with topology. A future version allow inheritance of attributes from domain parents.

## 3.2 Maps

It is very wonderful to have a nearly complete, nearly up to model of the world, but you just get confused if you try to take it all in at once. Thus, we need to be able to take slices of the model, selecting only a small set of related entities. One such slice is a *map*.

A map is one part of a layer of the database stack. For example, all the ip networking in the *nl* domain, or all the ethernets in this building or the *IXI X.25 Network*.

A map needs very little specification, just a list of entities and a class. Only entities of that are a subclass of the maps superclass [2] are allowed on the map.

Maps can be parts of other maps. This is conventient for visualisation because it provides a way of clustering related entities in the same area and allows important systems to be drawn larger than more detailed ones.

> A map is a picture of a network, each node of which represents a network entity. Each node is drawn on the screen as a rectangle and of a certain width and height. There are two kinds of node, *system nodes* and *interface nodes*. The latter are used for linking the *system nodes* togther, they are normally drawn as small boxes with a pair of lines connecting to other nodes.
>
> Each map has a class. Only entities that are siblings or subclasses of its parent class can be placed on a map. For example an *ip.host* node can be put on an *ip.topological* map because they both share the same parent class *ip*. However, a *x25.host* node could not be place don the map.

[2] we say superclass so that we can define different varieties of maps, for instance ip.topological, ip.political, ip.geographical.

Some entities are known as *interfaces* and these can connect two non-interface entities, provided they are all on the same layer of the stack.

### 3.1.2 View from the inside

An Example

A networking program *host* returns the name of two a machines; their types; a *Acne Corp. SuperStation* and a *08xz*; and a list of their IP addresses.

We have at least five kinds of objects here, ip host, ip address, two particular kinds of machine and ip network. These are all *Network Entities* or *Entities* for short. This leads to a series of questions such as how do we specify connections between hosts and networks and in which layer do we place an entity.

**In which layers do entites go?** We allow an IP layer, in which all IP entities are placed, and a physical layer in which the machines (which are physical entities) are placed.

**How to specify connections between hosts nad networks.** We say that the IP address entities are *interfaces* and that they can connect an ip host to an ip network.

**How do we specify the type of an entity?** This is done by defining a set of entity classes; for example the class of all IP entities, the class of all X25 entities or the class of all IP hosts. Then we say that an entity belongs to a particular class when we create it.

In addition, the set of IP hosts is a subset of all IP entities. So we say that the *IP hosts class* is a *subclass* of the *IP entities* class.

**What syntax do we use?** Lets say that an IP host abc.com has class *ip.host* so use the syntax *ip.host:abc.com* and an ip network 192.16.199.0 might be *ip.network:192.16.199.0*.

Entities are described by their name, their class and and their attributes. Attributes are a collection of values. A class is a special kind of entity that describes a type of entity rather than any particular system. Instead of setting all the possible attributes for each entity we just set them up for the class and entities of that class will *inherit* these values. [1]

The <name,class> pair uniquely identifies an entity.

[1] Single inheritance only.

# Chapter 3

# Concepts

This chapter aims to describe some of the concepts and structures used is monet. This includes with a basic description of the model used, features provided and some of the standard classes and shapes.

## 3.1 Entity Database

The database is a description of the networking universe. The idea is to build a model of any interesting networks and try to keep the model up to date automatically.

### 3.1.1 View from the outside

An Example

An ip network might be built upon a network of ethernets. Here we have two seperate networks with topologies quite distinct from each other.

Thus in our model we treat them as separate parts of our model. We need to make connections between them of course and this can be done by listing all the parts of an ethernet that make up an ip network. We say these are the *components* of the ip network.

*The OSI reference model is similar in its layering.*

> The entity database is structured as a stack of networks. The elements of these networks are known as *entities*, and these can be linked to each other in two ways.
>
> Entities in these networks can components of other entities in other layers of the stack.

### 2.8.1 Pragmatics

The nature of network objects varies greatly as do the protocols that can be used to monitor them. So for some object classes it is not possible to get some or any of the specified data. Some of the reasons for this are listed below.

**No reliable source** You can get a lot of information from the nameserver for IP objects but where do you find out the official name of an X.25 host?

**Shakey Definition** The capacity of an ethernet is pretty clearly 10Mbps. But what is the ordinal capacity of a sun workstation or an IP address.

**Difficult to determine** Determining a list of nearest neighbours for an IP host is trivial. Finding the domain name of the machine at the other end of the leased line connected to your local cisco is possible at least in some circumstances but is not simple.

More through accident than intention, the information returned by *gethostbyname* corresponds quite closely with the status information defined above. Wouldn't it be great if all networking objects where listed there too.

### 2.7.7 Domain Name Server

This is a very good source of information. It provides an accurate source of naming and topology for networks and hosts but is limited because only necessary information is ept up to date.

## 2.8 Interpreting Queries

monet is concerned with monitoring the status of network objects, thus a definition of status is required. The status information must be general enough to cover all types of network objects by a set of common attributes. In addition to the status at one point we need to keep track of the status history so that back references can be made to particular events and so that statistics over the whole or part of the history can be made.

The raw statistics will need to be processed by monetd to produce results in generic format. In addition these may need to be further processed by clients, for example to produce statistics over a period or to produce *current status* values.

A number of meta-statistics may be generated by monetd such as *behaviour*. Although clients should be able to produce this for themselves from the more basic values, it is very convenient to have this done by monetd.

In addition to instantaneous status values, it would be useful to have *mean* values for a period. This would be basically to save space, as much data for a period would be compressed into single values. It is likely that instantaneous values would be required for the most recent period, with means for periods further back, the size of the mean periods increasing as we go further back. For example keep the instantaneous values for the past day, the mean values for every day in the past month and the means for every day in the past year. Of course an instantaneous values is just a mean value for a very small period.

Keeping history lists could use up memory. In addition to its retention by monetd it may also be cached by the clients. Periodic saving to disc would also be required in a format readable by humans and by monet on startup.

In future we might like monetd to handle all consistency checking for network objects, this would be used by clients such as the xmonet mapper to keep the maps in touch with reality.

a time because many networks limit the number open at any one time, in addition this caused at least one log file to over flow.

The user data field should be set up for a particular host usually this is very simple but some hosts will refuse a pad connection and have special user data field for monitoring purposes.

Address translation is done using a very simple lookup table that comes with the distribution but each site will need to update this themselves. The user data field for a host can be set here too. This mechanism is not terribly powerful and so may need to be improved later. When this file is altered it will be automatically re-read.

### 2.7.4 SNMP

Currently monitoring is done by sending separate requests for the host and each one of its interfaces. A few variables are read such as traffic in and up time from which load and reachability are derived.

The name of an interface is given as <host.name.xy>+<#>interface, so obviously each interface could have more than one name.

It would be nice to query all the interfaces at the one time rather than sending out request for each separately.

Code has been included for listening to Snmp Traps but this has not been tested yet.

### 2.7.5 RIPE Database

This is an IP Network database maintained by the European IP coordinating body RIPE, using the *whois protocol* for access to information. It might be nice to have monetd work as a whois server and so merge these sets of information. This would provide such useful information as technical contacts and their addresses.

### 2.7.6 X.500

This would be a good source of contact information one day. It could be accessed via scripts or by maybe using the ISODE stuff in here directly. It should be a source of contacts and addresses but it is also possible to draw maps of the with the links representing primary and secondary servers between directory service agents and the parts of the tree they cover.

others and move the nodes accordingly. This is at least n-squared but is quite a good untangle heuristic.

**Regional Reduction Algorithm** Find the regions in a network. Treat these as nodes in a new network, links represent the places where regions touched. If multiple links between two nodes are reduced to one the number of regions should reduce at each step. Finally we have a heirarchy of subnets that can be drawn quite easily with the minimum number of overlaps.

**Finishing** As a final measure for an untangled network would be an algorithm to shift nodes so that they are aligned and the links between them are draw straight.

In addition if networks can be reduced to a group of trees or single cycles. A good tree/cycle layout algorithm can be used.

**Strength of links** Some links can be given priority over each other.

## 2.7 Information Sources

### 2.7.1 IP

A lot of information can be obtained from the nameserver. Status information is found by using ICMP pings. Currently, an IP address and a host are treated as separate objects but they could be queried together.

### 2.7.2 TCP

The system can ping tcp servers using the port number and hostname to define them. The request is done simply by trying to bind to the port. For example, monitoring a nameserver could be done using,

*ping = "tcp:&+53"*

xmonet would replace *&* by the name of the object to be pinged.

### 2.7.3 X.25

The reachability of an X.25 host is done simply by opening a virtual circuit to that host. This means we could only ever send X.25 queries out one at

- A system using only a tiny proportion of its supposed capacity.

### 2.6.3 Auto-discovery

Getting the network topology is trivial with IP networks, but difficult or impossible for other types of network. It would be possible for SNMP if the names of neighbours were given in the MIB.

There is also the difficulty of determining when data gained from the network overrides that entered by hand.

### 2.6.4 Automatic Layout

With computer networks as big and complex as they are becoming it is not really practical to keep them up to date by hand. Assuming the topology is available it would be very helpful to have them laid out in a pleasing manner.

**Layout / Untangle** Just getting the nodes roughly in position is a big help, which can be finished of by hand.

**Links** It is pleasing for links to always be either straight or at 45 degrees. It is not desirable for links to be made up of many segments, since it is difficult to keep one line straight a compromise is to use up to two segments. [4]

**Geography** Sometimes nodes should stay in a particular region.

**Zoom** Some systems are more important than others and should be drawn larger, with a zoom for smaller stuff.

**Shape** Graphs where nodes are drawn as single points are the simplest, rectangles are a bit more difficult.

**Batch/Incremental** With batch layout, all topology is known at once and an optimal layout can be performed. When nodes are added incrementally/ interactively, then it is undesirable if the whole layout changes in front of the users eyes.

**Magnetic Heuristic** Treat linked nodes as attractors and unlinked nodes as repulsors. From this work out the force each node exerts on all the

[4] This is especially useful when the link itself carries information.

## 2.6 Issues

This section discusses some of the issues, problems and solutions involved in monet. It is far from complete or extensive but should serve as an illustration.

### 2.6.1 Reachability

This could be looked at as if the system is up or down. It depends to a large extent on where the monitoring is done from. A system might be up but not reachable. A broken network might be only partially reachable. An overloaded system may not respond to queries.

There is no realiable way to acertain if a system is up or down. Even for protocols with an explicitmeans of querying the status one can never be sure. For protocols that don't some arbitary means must be found.

### 2.6.2 Behaviour

It is nice for to know if some system is behaving in a strange way, then it can be fixed directly without having to seach for the problem. Monet uses a general behaviour indicator for this purpose, a system can be behaving normally, oddly or chaotically. The difficulty is in deciding what constitutes these behaviours for different types of system. In addition they must be made given only partial information.

If an network object goes up and down three or four times in an hour we might think it odd, more than 10 we could even use the word chaotic. Currently this is how behaviour if defined. The precise reason is writen in the log file as a comment, but the idea is really for a network manager to look at a network map and see something blinking (or whatever) and see if they can find out what is happening.

It would be nice to use other events to change the behaviour parameter, simple (easily implementable) examples would be,

- If the traffic going into a system on an interface is very high and similar to that going out. Here there may be packets bing bounced back and forth.
- Very veriable times in responding to queries.
- One interface overloaded and the others almost empty.

available to clients; and secondly to send out queries and cook the results into a set of status attributes.

The database follows a simple object oriented model that allows dynamic definition of new classes and allows individual entities to inherit the attributes of their parent class.

Users can view parts of the network database by defining a *map* which is basically a list of entities with some geometry information.

The status attributes listed below are a subset of those available for each entity [3]. Some of these can be quite accurately determined, such as the speed of an SNMP interface, but others, such as reachability are more shakey.

**Reachability** Is the system up or down, or has it been down for a long period of time and thus become *critical.*

**Future** Is this just a temporary system or about to be taken down. Is is being planned by has not yet been installed or is its future pretty certain.

**Behaviour** No details, just some unusual events.

**Capacity** For a network, its average speed. Not really definable for a host.

**Official Name** monet uses domain names to uniquely identify all objects. Fine for some objects but problematic for others.

**Load In** Incoming traffic for some recent period expressed as a percentage of its total capacity.

**Load Out** Same as for Load In but for outgoing traffic.

**Number of external Connections**

**List of Connected Systems** This has not really been implemented yet. It is used for the Explore Command described in the chapter of xmonet.

**Contact Address** E-mail address of someone to contact if there is problem with the network entity.

[3] A full list, with explanation, is available with the command *monetsh> attributes*

**Interactive, Automatic and Visual** Maps provide an overview of the network, they are also useful for planning and troubleshooting. Interesting situations should be marked clearly and very interesting events should be notified to managers via bleeps or automatic mail.

All the data needed is already there in the network. The amount of data the user has to enter themselves must be minimised. This is especially important for large databases and for consistency.

Uses existing databases, such as the nameserver, snmp agents, whois database for information. This obtained dynamically or via scripts.

Many interesting situations can be analysed easily by software.

**Distributed Monitoring and Logging** Monitoring is done via local agents which passes on requests to other agents for remote monitoring status. This means that the monitoring information need be gathered only once without doing queries over long distances, and that the same information is available, interactively to anyone who can connect to an agent.

Maps can be stored locally or at aremote site which is accessable by anyone for depositing or retrieving maps. This allows the work of many people to be shared, and to help build a total map of the world.

The very large number of entities monitorable requires that maps be drawn at least semi-automatically [2].

Some logging is done, but is rather primitive. A much more powerful system is planned which would automatically compress logs and allow remote interactive access and automatic production of graphs and reports.

Monet is not designed for monitoring low level traffic, the contents of packets, etc. It is more oriented towards higher level management, although if someone out there has such a package they could be merged.

## 2.5 Model

The system consists of a set of communicating agents. These have two tasks, which are to act as database servers making monitoring information

[2] The very simple drawings on maps are an effort to use the minimum space possible

protocols, and the many management tools developed without interoperability built in.

SNMP at least provides a common interface but requires proxy agents to work with non IP networks. It has no history mechanism and you cannot start at one agent and explore outwards from it [1]. Lastly it can only deal with physical hosts, you could maybe have a proxy to monitor an ethernet but it cannot describe higher level entities like servers such as mail transfer agents or nameservers.

These two problems, of inconsistent tools and no really common interface for network management are what monet tries to solve.

## 2.4 Features and Goals

Really the main design goal was to produce a common framework for all network monitoring. So that managers need write only the low level monitoring code, already having reporting, logging and visualisation tools avalilable.

**Generic and Extensible** Can be used to draw all sorts of networks, IP, SNMP, X.25, DNS, MX/SMTP, X.500. New classes of network map can be created quite easily by the user without additional code although some programming effort may be put in if they are to be monitored. It is very easy to add monitoring modules for new protocols, or to avoid writing new code, the output from existing monitor programs can be piped directly to the monitoring agent.

Access to the networking database is available from the command line, X or from scripts. In addition an C language API is planned.

The system is designed to be configurable, with as little as possible built in to the code. For example the X interface is almost totally configurable, through its applications defaults file, more so than most other X programs; and tools like the visual traceroute are implemented without building direct support into the system.

Monet is portable, free, public domain software. Written for *unix* in K&R C with *X11 Release 4* and tested on various machines including apollo, sun, gould and a previous version under silicon graphics and ultrix.

[1] A connectivity map could be built into snmp.

# Chapter 2

# Introduction

## 2.1 Introduction

Monet is a generic tool for distributed mapping, monitoring and logging of computer networks. Many network management tools are are often specific to one set of protocols or are not designed to work with other software. The basic design goal of monet was to build a tool that could be used with all kinds of network and also allowed interaction with existing tools.

The basic philosophy behind monet is to build a model of the networking universe that keeps itself up to date and produces maps, reports, graphs and warnings without involving people more than necessary.

## 2.2 What's in a name?

The name monet required some alternative lunch time conversations as all the obvious names have already been used up (net watch/view/visualise etc). It comes from *Mo(nitor) net(work)* and is also the surname of the french painter *Claude Monet* which should give a hint at something visual.

## 2.3 Rational

Networks are increasing in size and complexity not only due to purchases but also because of the increase in interconnectivity between them. There are quite often many bodies and people involved which makes it very hard to find out what is actually going on. This is further complicated by different

# Chapter 1

# This Release

This is the second beta release of the **monet** software, Version 0.2-Beta. It is intended as free, open software and has the GNU license. The idea is to be open and useful, so please think of ways to improve it. The software was written by James Barr of *NIKHEF-H* and you can send him e-mail at *a38@nikhef.nl*. In addition there is general discussion list for network mapping *ripe-map@nic.eu.net*, to subscribe send to *ripe-map-request@nic.eu.net*.

As the program is a beta release, please keep the following in mind. The monitor program sends out a lot of queries. It should not overload any networks of reasonably high capacity, but it could fill up a few X.25 log files or a 9.6kbps line. So be a bit careful and don't leave multiple copies monitoring Portugal from Japan if you can. Later versions will try to do some checking but the current stuff isn't really dangerous.

This guide is rather rough, it is assumed the reader knows unix and has had previous experience installing software. It also assumes some practical networking knowledge, especially knowledge of the internet. The first part includes a discussion of the design goals, problems and solutions. The manual pages included in the appendix document how to install and use the software.

Please, use, comment and if your disposition allows, enjoy.

# List of Figures

3.1 Object Names . . . . . 19
3.2 Map Symbols . . . . . 25

3.1.3 Namespace . . . . . 16
3.2 Maps . . . . . 16
3.3 Politics . . . . . 17
3.4 Definitions . . . . . 17
3.5 Basic Operations . . . . . 20
3.5.1 Map Symbols . . . . . 24
3.5.2 Shapes . . . . . 24
3.6 Standard Classes . . . . . 24

**A Monetsh Commands** **28**

**B Entity Attributes** **33**

**C Xmonet Applications Defaults** **35**

**D Configuration Options** **36**

**E Monetd default requests** **39**

**F To be done** **42**

**G Installation** **45**

**H Application Manual Pages** **52**

**9 Library Manual Pages** **72**

**Glossary** **84**

# Contents


**1 This Release** **3**

**2 Introduction** **4**

2.1 Introduction . . . . . 4
2.2 What's in a name? . . . . . 4
2.3 Rational . . . . . 4
2.4 Features and Goals . . . . . 5
2.5 Model . . . . . 6
2.6 Issues . . . . . 8
2.6.1 Reachability . . . . . 8
2.6.2 Behaviour . . . . . 8
2.6.3 Auto-discovery . . . . . 9
2.6.4 Automatic Layout . . . . . 9
2.7 Information Sources . . . . . 10
2.7.1 IP . . . . . 10
2.7.2 TCP . . . . . 10
2.7.3 X.25 . . . . . 10
2.7.4 SNMP . . . . . 11
2.7.5 RIPE Database . . . . . 11
2.7.6 X.500 . . . . . 11
2.7.7 Domain Name Server . . . . . 12
2.8 Interpreting Queries . . . . . 12
2.8.1 Pragmatics . . . . . 13

**3 Concepts** **14**

3.1 Entity Database . . . . . 14
3.1.1 View from the outside . . . . . 14
3.1.2 View from the inside . . . . . 15

# The Monet Manual

by James Barr

NIKHEF-H
Kruislaan 409
1009DB Amsterdam
The Netherlands

Version 0.2-Beta

20 July 1991