The Mjølner System
Using on UNIX
Reference Manual

Mjølner Informatics Report
MIA 90-04(1.4)
October 1997

Table of Contents

1 The Mjølner System on UNIX

This document describes the usage of the Mjølner System on UNIX workstations. The Mjølner System is currently available for the following UNIX platforms: Sun4 SPARC Solaris, HP 9000 series 700 workstations running HP-UX, Silicon Graphics MIPS workstations running IRIX, and PC's running ELF based Linux.

1.1 Search Path

The Mjølner System is often installed in the directory:

```
/usr/local/lib/beta
```

The binary executables of the Mjølner System, including the BETA compiler are then installed in the directory

```
/usr/local/lib/beta/bin
```

The user should include this directory in his or her search path. The file `.login` should contain, e.g.

```
set path=($path /usr/local/lib/beta/bin)
```

But the system can be placed anywhere (see chapter 2).

1.2 Calling the Compiler

Assume that the file `foo.bet` contains a valid BETA fragment. This fragment may be translated by executing the command

```
beta foo
```

This command will invoke the BETA compiler and link the output from the BETA compiler. Linking will be done using the standard UNIX linker. The final object code will be put on the file `foo`, which may be executed by typing:

```
foo
```

1.3 System Directories

Some UNIX installations consist of a network of workstations from different vendors. For instance, the Computer Science Department, Aarhus University has a system consisting of SGI, Sun-4, Linux, and HP-9000 workstations. In order to be able to use the same BETA source files for the different systems on the same file system, the object code are placed in different subdirectories.

Assume that the file `foo.bet` resides in the directory:

```
/usr/smith/test
```

If the BETA compiler is invoked on a SPARC Solaris workstation, the object code for `foo` is placed in the directory:

```
/usr/smith/test/sun4s
```

This means that the directory will contain the file:

```
foo.o
```

If `foo.bet` is translated on a HP-9000 series 700, the object code file is placed in the directory:

```
/usr/smith/test/hpux9pa
```

If the `foo.bet` has been translated at a SPARC, subsequent translations on e.g., a HP-9000 will only perform code generation, i.e. no parsing and semantic checking will be performed.

Note that the *executable* file `foo` is by default placed in

```
/usr/smith/test/foo
```

This means that if `foo.bet` has been translated on SPARC, then `foo` may only be executed at a SPARC. The user should, e.g., move `foo` to the directory `/usr/smith/test/sun4s` or rename the file in order to save the SPARC executable before translating `foo.bet` on another system. Alternatively, the `-o` option of the compiler may be used, as in

```
beta -o sun4s/foo foo
```

1.4 Environment Variables

The following environment variables are used in the Mjølner System on UNIX platforms. For most users, only the BETALIB variable is relevant:

BETALIB

Specifies where *~beta* is located. If not set, *beta* is assumed to be a user name, and *~beta* being the home directory of that user. If *beta* is not a user, then `/usr/local/lib/beta` will be used. It is used by many tools in the Mjølner System.

BETAOPTS

Specifies options that the beta compiler should be invoked with by default.

BETALINKOPTIONS

Specifies the linker options to be used by the BETA compiler when linking (using standard UNIX linker). If set, it totally overwrites the default link options, the compiler would have used otherwise.

CC

Set by the compiler in the jobfiles on all UNIX platforms. Thus `$(CC)` can be used in BUILD property commands (see [MIA90-02]) and Makefiles invoked using the MAKE property.

LD_LIBRARY_PATH

This is a colon separated list of directories to search for external libraries during linking. Notice that not all standard UNIX linkers supports this variable directly, but the `..job` files generated by the beta-compiler will still use this variable.

LD_LIBRARY_PA on Silicon Graphi

On Silicon Graphics machines, this variable is especially important: If a BETA program is larger than a certain limit, the program will be linked using *shared object files.* This means, that a part of the linking process is postponed until runtime, and in order for this to work, the runtime loader must be able to locate the shared object files generated by the compiler. So if the output from a compilation ends with

```
Note! Object program uses shared object files in directory sgi
```

the directory `sgi` must be included in your `LD_LIBRARY_PATH` before attempting to run the program. Otherwise you will get a loadtime error like

```
793:./foo: rld: Fatal Error: cannot map soname 'foo1..so'
using any of the filenames
/usr/lib/foo1..so:/lib/foo1..so:
/lib/cmplrs/cc/foo1..so:
/usr/lib/cmplrs/cc/foo1..so:
 -- either the file does not exist or the file is not
mappable (with reason indicated in previous msg)
```

TMPDIR

Normally, the link-directives in the `..job` files will use `/tmp` for temporary files. If another directory is to be used (e.g. because `/tmp` is full), setting TMPDIR to the name of a directory, prior to compilation, will cause the link-directives to place temporary files in this directory.

MACHINETYPE

Is set automatically by the compiler during the execution of the `..job` files and `make` files. It may be necessary to set this variable manually, if these command files are executed manually.

BETART

Is used to set various characteristics of the BETA runtime system. The specification consists of a list of entries, separated by ':' (colon). Colon in the beginning and at the end of the BETART value is optional. The specification ignores case, except for the case of string-entry values.

Entries may appear more than once in BETART. The last specification will in this case be used. The semantics of the different BETART entries are given below.

There are three types of entries: *boolean* , *integer*, and *string* entries:

Boolean entries: The default value for all boolean entries is false. Mentioning the boolean entries in BETART sets its value to true. Boolean entries have the form <entry>, where <entry> is one of:

Info

Print information about heap sizes etc. at startup.

The following is an example of the output produced when *Info* is set:

```
#(Heap info: IOA=2*512Kb, AOABlock=512Kb, LVRABlock=512Kb,
CBFABlock=1Kb)
```

which reports the sizes of the two Infant Object Area (scavenging) heaps, the size of each Adult Object Area block, the minimum size of each Large Value Repetition Area block, and the size of each Call Back Function Area block.

InfoIOA

Print information on garbage collection in the infant object area during execution.

If set, then after each IOA garbage collection, a line like the following will be printed:

```
#(IOA-7 12? 4%)
```

which tells that this was the seventh IOA garbage collection, that it was triggered by a request to allocate an object of size 12 long words (48 bytes), and that after the garbage collection, the IOA heap is 4% filled. In special situations, other information may be printed if InfoIOA is set. These will also be marked `#(IOA`.

InfoAOA

Print information on garbage collection in the adult object area during execution.

If set, then after each AOA garbage collection, a line like the following will be printed:

```
#(AOA-3 1024Kb in 2 blocks, 23% free)
```

which tells that this was the third AOA garbage collection, that 1024 Kb is used by 2 AOA blocks, and that after the garbage collection 23% of AOA is unused.

Another message that may be printed if InfoAOA is set is

```
#(AOA: new block allocated 512Kb)
```

which tells that an object was moved to the AOA heap, and that there was not enough room for it. In this case the best solution was to allocate a new block. Other times this situation may trigger an AOA garbage collection. The heuristics for this may be controlled by some of the other properties mentioned in this section.

In special situations other messages may be printed if InfoAOA is set, these will also be marked `#(AOA`.

InfoLVRA

Print Information on garbage collection in the large value repetition area during execution.

If set, then after each LVRA garbage collection, a line like the following will be printed:

```
#(LVRA-2 1536Kb in 2 blocks, 17% free)
```

which tells that this was the second LVRA garbage collection, that 1536 Kb is used by 2 LVRA blocks, and the after the garbage collection 17% of LVRA is unused.

Another message that is often printed if InfoLVRA is set is

```
#(LVRA: make free list  1024Kb in 2 blocks, 243Kb free)
```

which tells that a large value repetition object was attempted allocated in the LVRA heap, and that there was not enough room for it. In this case

the best solution was to rebuild an internal free-list. The numbers reported tell that there was currently two blocks allocated, after the rebuilding of the free list, 243Kb was found free.

Another message that may be printed if InfoLVRA is set is

```
#(LVRA: new block allocated 512Kb)
```

which tells that a large value repetition object was attempted allocated in the LVRA heap, and in this case the best solution was to allocate a new block. Other times this situation may trigger rebuilding of the internal free-list or an LVRA garbage collection. The heuristics for this may be controlled by some of the other properties mentioned in this section.

InfoLVRAAlloc

Print information on allocation in the large value repetition area during execution.

If set, then when a large value repetition is attempted allocated in LVRA, a line like the following will be printed:

```
#(LVRAAlloc integer repetition, range=300, size=1216)
```

which tells that it is an integer repetition of range 300 which is requested, and that it occupies 1216 bytes in the LVRA heap.

InfoCBFA

Print information about the callback function area during execution.

If set, then when enough new callbacks have been installed that a new block is needed in the CBFA area, a line like the following will be printed:

```
#(CBFA: new block allocated 1Kb)
```

which tells that a new block 1 kilobyte in size was allocated to extend the CBFA heap.

In special situations other messages may be printed if InfoCBFA is set, these will also be marked `#(CBFA`.

InfoAll

Sets all Info entries: Info, InfoIOA, InfoAOA, InfoLVRA, InfoCBFA, and InfoLVRAAlloc.

QuaCont

Continue execution after runtime detection of qualification error in reference assignments.

Integer entries: These have the form <entry>=<value>, where <entry> is one of the following, and <value> is any positive integer. The default values are noted in parenthesis below:

IOA

The size in Kb of the infant object area (Default: 512).

IOAPercentage

The minimum free fraction in percent of the infant object area. If less than this fraction is free in IOA after an IOA garbage collection, then, in the current version of the runtime system, the execution of the program is terminated. This limitation will be eliminated in a future version of the runtime system. (Legal range: 3 to 40, default: 10).

AOA

The size in Kb of one block in the adult object area (Default: 512).

AOAMinFree

The minimum free area in Kb in the adult object area. If less than this size is free after an AOA garbage collection, then the next allocation in AOA will cause a new block to be allocated. Please note that it is only meaningful to specify *one* of AOAMinFree and AOAPercentage (below), because they specify conflicting behaviour for allocation in AOA. (Default: 100).

AOAPercentage

The minimum free fraction in percent of the adult object area. If less than this fraction is free after an AOA garbagecollection, then the next allocation in AOA will cause a new block to be allocated. Please note that it is only meaningful to specify *one* of AOAMinFree (above) and AOAPercentage, because they specify conflicting behaviour for allocation in AOA. (Legal range: 3 to 97, default: 0, i.e., AOAMinFree is used).

LVRA

The default size in Kb of one block in the large value repetition area (Default: 512).

LVRAMinFree

The minimum free area in Kb in the large value repetition area. If less than this size is free after an LVRA garbage collection, then the next allocation in LVRA will cause a new block to be allocated. Please note that it is only meaningful to specify *one* of LVRAMinFree and LVRAPercentage (below), because they specify conflicting behaviour for allocation in LVRA. (Default: 200).

LVRAPercentage

The minimum free fraction in percent of the large value repetition area. If less than this fraction is free after an LVRA garbage collection, then the next allocation in LVRA will cause a newblock to be allocated. Please note that it is only meaningful to specify *one* of LVRAMinFree (above) and LVRAPercentage, because they specify conflicting behaviour for allocation in LVRA. (Legal range: 3 to 97, default: 0, i.e., LVRAMinFree is used).

CBFA

The size in Kb of one block in the callback function area (Default: 1).

String entries: These have the form <entry>=<value>, or <entry># <value>, where <entry> is one of the following, and <value> is any string. The default values are noted in parenthesis below:

InfoFile

Name of file on which to write all this information (Default: standard error file `stderr`).

Example (using `csh`):

```
setenv BETART "InfoIOA:IOA=1024:InfoFile=info.dump"
```

1.5 Usage of the BETA Compiler

The BETA compiler can be used, using the following steps:

1. Copy a demo program from demo directory to your working directory:

   ```
   cp $BETALIB/demo/current/beta/record.bet .
   cp $BETALIB/demo/current/beta/recordlib.bet .
   ```

2. Compile the `record` program (`record.bet`):

   ```
   beta record
   ```

3. Execute the `record` program:

   ```
   record
   ```

For further details, see [MIA 90-02].

1.6 Usage of Valhalla

The Mjølner BETA debugger `valhalla` uses the X window system. To use it start X the way you are used to. Then start the debugger by issuing the command (to debug the `record` program):

```
valhalla record
```

For further details, see [MIA 92-12].

1.7 Usage of Sif

The Mjølner Hyper Structure Editor `sif` uses the X window system. To use Sif start X the way you are used to. Sif must be activated from an `xterm` window, or a corresponding shell program. Just type:

```
sif record
```

or just

```
sif
```

For further details, see [MIA 90-11].

1.9 Usage of Freja

The Mjølner CASE tool `freja` uses the X window system. To use freja start X the way you are used to. Freja must be activated from an `xterm` window, or a corresponding shell program. Just type:

```
freja record
```

or just

```
freja
```

For further details, see [MIA 93-31].

1.10 Usage of Frigg

The Mjølner Graphical Interface Builder `frigg` uses the X window system. To use frigg start X the way you are used to. Frigg must be activated from an `xterm` window, or a corresponding shell program. Just type:

```
frigg record
```

or just

```
frigg
```

For further details, see [MIA 96-33].

1.11 The comp.lang.beta Newsgroup

The USENET newsgroup `comp.lang.beta` is intended for discussions about the BETA language and the programs and systems written in or supporting BETA. Discussions concerning object-oriented programming principles based on the concepts known from BETA will also take place in `comp.lang.beta`, possibly cross-posted to `comp.object`. The BETA language *Frequently Asked Questions* (`beta-langauge-faq`) will be posted to `comp.lang.beta`, and the most frequently asked questions from `comp.lang.beta` will be included in the subsequent versions of this FAQ.

1.12 Internet Resources

There are two main starting points to find information about the Mjølner System and BETA. These are:

`http://www.mjolner.com/`	The Mjølner Homepage
`http://www.daimi.aau.dk/~beta/`	The BETA Language Homepage

2 Installation Guide for UNIX Workstations

This chapter contains a guide for installing BETA on UNIX platforms.

The Mjølner System is currently available for the following UNIX platforms: Sun4 SPARC running Solaris, HP 9000 series 700 workstations running HP-UX, Silicon Graphics MIPS workstations running IRIX, and PC's running ELF based Linux.

2.1 Minimum Requirements

2.1.1 Sun workstations

Solaris 2.4, 16 Mbytes, X window system (Rel. 11.5 or later). 150 Mbytes of available disk space (the full distribution occupies about 100 Mbytes). OSF/Motif 1.2 runtime libraries.

2.1.2 Silicon Graphics workstations

IRIX 5.3 or 6.2 (32 bit), X window system (Rel. 11.5 or later). 150 Mbytes of available disk space (the full distribution occupies about 100 Mbytes). OSF/Motif 1.2 runtime libraries.

2.1.3 HP workstations

HP 9000 series: HP-UX 9, 16 Mbytes, X window system (Rel. 11.5 or later), 150 Mbytes of available disk space (the full distribution occupies about 100 Mbytes). OSF/Motif 1.2 runtime libraries.

2.1.4 PC's running Linux

ELF based Linux, CPU: Intel 386/486/586/Pentium, 16 MB RAM, X window system (Rel. 11.5 or later), LessTif/Motif 1.2 or later[1], 120 Mbytes of available disk space (the full distribution occupies about 80 Mbytes).

---

[1] Motif is not required to compile simple programs, but programs using the platform independent GUI libraries as well as the graphical tools require motif. Most graphical BETA programs are known to work with LessTif, but at the time of writing this manual, the exact version of LessTif required is not known. Please consult the Internet Resources (section 1.9) for latest news.

2.2 Distribution Contents

The distribution contains:

- **BETA Compiler v5.3**
- **Libraries**

  basic libraries, persistence, the BETA interface to the X window system, a platform independent GUI environment etc.
- **BETA demo programs**

  including demo programs of how to use BETA and how to use the basic libraries and the BETA GUI libraries.
- **BETA debugger**

  Source level debugger with graphical interface.
- **Sif**

  Hyper Structure Editor.
- **Frigg**

  Graphical Interface Builder .
- **Freja**[2]

  The Mjølner BETA CASE Tool.

2.3 Installation

The easiest way to install the system is to place it in `/usr/local/lib/beta`, but it can be placed anywhere (see below). The location of the BETA system in your file directory is in the following referred to as `$BETALIB`. By default `$BETALIB` is `/usr/local/lib/beta`.

1. **Unpack the tapes** (skip this if tar-files have been obtained by ftp)

   ```
   cd $BETALIB
   ```

   The details below may vary on different systems.

   ```
   tar -xvfb device-file 2000
   ```

   where *device-file* is the UNIX file used for interfacing the tape station, e.g. `/dev/rst8`.
2. **Uncompress and untar the system:**

   The above command will have extracted a file called `system.tar.Z` or `system.tar.gz` and a number of other compressed tar-files. Refer to the release notes for your package for the exact list of files.

   Each of the compressed files should now be unpacked; use either

---

[2] Currently only available on SPARC Solaris.

```
uncompress system.tar.Z
tar -xvf system.tar
```

or (if `zcat` is available)

```
zcat < system.tar.Z | tar -xvf -
```

On Linux systems (or elsewhere, if GNU tar is available) use:

```
tar -xzvf system.tar.gz
```

Repeat this for the other files too.

3. **Include `$BETALIB/bin` in your search path**
   by including

   ```
   set path=($path $BETALIB/bin)
   ```

   in the `.login` file.[3]

4. **Set `$BETALIB` environment variable**

   Set the environment variable `BETALIB` to point to the top directory of your BETA installation, e.g.:

   ```
   setenv BETALIB /usr/local/lib/beta
   ```

   or when using bash on Linux:

   ```
   bash$ BETALIB=/usr/local/lib/beta
   bash$ export BETALIB
   ```

   See also the section on environment variables in the previous chapter.

5. **Scripts in `$BETALIB/configuration`**

   The scripts located in `$BETALIB/bin` use another script to perform some tests for, e.g., machine type. This script is placed in `$BETALIB/configuration/r3.0/env.sh`. If you have a very special installation, it may be necessary to modify the `env.sh` script according to your system.

6. **Make Current Links**

   Many of the examples in the system manuals use the notation `~beta/<library>/current/…` to describe the version of libraries in the current release of the Mjølner System. In each directory `current` is a link to the current version of the library. The current links can be generated using the `make_currentlinks` script supplied with the Mjølner System.

7. **Online manuals**

   Standard UNIX manual pages for the tools in the Mjølner System are placed in `$BETALIB/man/current/man1`. To include these in the search path of the `man` program, either include `$BETALIB/man/current/` in your `MANPATH` environment variable, or have your system administrator copy the files in `$BETALIB/man/current/man1` to the directory containing local manual pages (usually `/usr/man/manl`).

   The Internet also contains various sites with documentation on the Mjølner System. Use the URL

   ```
   http://www.mjolner.dk/
   ```

   which contians the BETA home page. This home page contains links to the BETA FAQ, Newsletters, etc., and the Mjølner System manuals.

---

[3] Remember to do a `rehash` if you `source` your `.login` file manually.

8. **The Emacs Editor**

   If you want to use the very popular GNU Emacs text editor as an alternative to the Sif hyper structure editor included in the Mjølner System, you may benefit from the beta-mode for Emacs located in the file `$BETALIB/emacs/current/beta-mode.el`. A large comment in the start of this file describes how to make use of beta-mode. Also you may want to byte-compile `beta-mode.el` from within Emacs for improved performance. The directory `$BETALIB/emacs/current` also contains various other contributions for using Emacs to edit BETA programs. For instance, the file `beta-hilit19.el` contains a setup for syntactic coloring of your BETA programs, when using Emacs version 19.

9. **Motif on Linux**

   If you are running Linux, then Motif is normally not included in the standard libraries on your machine. You will then have to buy a separate Motif package (version 1.2 or later) in order to use the graphical libraries included in the Mjølner System. Alternatively you may try using LessTif - a non-commercial implementation of Motif available at

   ```
   http://www.hungry.com
   ```

   At the time of writing this manual, the complete status of using LessTiff with BETA is not known, but most Lidskjalv are known to work.

   After you have installed such a package, you should change the two environment variables `MOTIFINC` and `MOTIFHOME` set in the

   ```
   $BETALIB/configuration/env.sh
   ```

   file to reflect the placement of your Motif/LessTif installation.

3. Bibliography

[MIA 90-02] Mjølner Informatics: *The Mjølner System: BETA Compiler Reference Manual* Mjølner Informatics Report MIA 90-2.

[MIA 90-11] Mjølner Informatics: *Sif – A Hyper Structure Editor, Tutorial and Reference Manual* Mjølner Informatics Report MIA 90-11.

[MIA 92-12] Mjølner Informatics: *The Mjølner System – The BETA Source-level Debugger – Users's Guide*, Mjølner Informatics Report MIA 92-12

[MIA 91-13] Mjølner Informatics: *The Bifrost Graphics System: Reference Manual*, Mjølner Informatics Report MIA 91-13

[MIA 91-16] Mjølner Informatics: *The Mjølner System—X Window System Libraries*, MjølnerInformatics Re port MIA 91-16.

[MIA 91-19] Mjølner Informatics: *The Bifrost Graphics System, Tutorial*. Mjølner Informatics Report MIA 91-19.

[MIA 93-31] Mjølner Informatics *Freja - An Object-Oriented CASE Tool.* Mjølner Informatics Report MIA 93-31.

[MIA 96-33] Mjølner Informatics: *MIA 96-33: Frigg - User Interface Builder*. Mjølner Informatics Report MIA 96-33.

4. Index