The Mjølner System
Using on Windows NT and Windows 95
Reference Manual

Mjølner Informatics Report
MIA 94-32(1.4)
October 1997

Table of Contents

1 Installation Guide for PCs running Windows NT / 95

This chapter contains the installation guide for the Mjølner System on Windows NT and Windows 95 using the Microsoft , Gnu or Borland SDK.

1.1 Minimum Requirements

Requirements for using The Mjølner System, release 4.1 (compiler v5.3) on Windows NT/95:

- CPU: 386/486/Pentium.
- RAM: At least 16Mb RAM. 32Mb (or more) recommended.
- Free disk space: Approximately 50Mb.
- Linker, and standard C libraries from either Microsoft (recommended) or Borland.
- Windows 95 or Windows NT 3.5.1 or later.

1.2 Distribution Contents

The distribution contains:

- **Compiler v5.2**
- **Libraries**

  basic libraries, persistence, GUI libraries, etc.
- **demo programs**

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

  The Hyper Structure Editor.

- **Frigg**

  Graphical Interface Builder

- **Valhalla**

  The Mjølner Debugger

1.3 Installation

1.3.1 Run the setup program

To install The Mjølner System, run the Setup Program found on the disk labelled[1]

```
The Mjølner System
      Setup
```

The Setup Program installs The Mjølner System on your system in the directory you specify.

Note, that if you are installing on *Windows NT* you must have Administrator privileges. Otherwise no program group will be created.

Note also, that you must *not* install the Mjølner System in a directory that contains spaces in the path.

1.3.2 Set up Environment Variables

When the installation is complete you should make sure that the batch file to start the compiler is in your path. If you installed The Mjølner System in e.g. C:\BETA, the batch file is located in

```
C:\BETA\bin\beta.bat
```

which should then be part of your PATH environment variable.

You will also need to set (some of) the environment variables BETALIB, SDK, ASM_VERSION, LIB and LD_LIBRARY_PATH. See section 2.3 for details on these variables.

The setup for Microsoft could look something like:

```
PATH=C:\BETA\BIN;%PATH%
PATH=C:\MSDEV\BIN;%PATH%
set LIB=C:\MSDEV\LIB;
set BETALIB=C:\BETA
set SDK=ms
set ASM_VERSION=6        (Needed only is makefiles are used)
```

[1] If you fetched the system by FTP, the `setup` program will be located in the directory named `Disk1`

The setup for Gnu could look something like:

```
PATH=C:\BETA\BIN;%PATH%
set BETALIB=C:\BETA
set SDK=gnu
set ASM_VERSION=0
```

Furthermore, you should setup whatever is needed by the Gnu SDK. This includes environment variables like PATH, C_INCLUDE_PATH, LIBRARY_PATH, and probably others, depending on the Gnu SDK.

The setup for Borland could look something like:

```
PATH=C:\BETA\BIN;%PATH%
PATH=C:\BC45\BIN;%PATH%
set LD_LIBRARY_PATH=C:\BC45\LIB;
set BETALIB=C:\BETA
set SDK=bor
set ASM_VERSION=4
```

You can make Windows NT set the environment variables automatically each time you log in via the `System tool` in the `Control Panel`.

You can make Windows 95 set the environment variables automatically each time you log in by setting them in the `autoexec.bat` file.

1.3.3 SDK Requirements

In order to use the Mjølner System you need to have a linker, a make utility and C libraries from either Microsoft or Borland. The Microsoft utilities are recommended.

These utilities must satisfy the following:

1.3.3.1 Microsoft

- *Linker*:

  LINK.EXE from e.g. Microsoft Visual C++.

- *Make*:

  NMAKE.EXE from e.g. Microsoft Visual C++.

  If you do not plan to use the BETA MAKE facility (see section 2.4), you can do with a dummy NMAKE program, that simply exits 0, or no NMAKE at all.

- *C libraries*:

  from e.g. Microsoft Visual C++.

If you have installed Microsoft Visual C++ 2.0 or later for Windows 95 or Windows NT, you do not need any more software to run the Mjølner System. If you do not have Visual C++, you will have to get a linker and C libraries from Microsoft (http://www.microsoft.com/).

1.3.3.2 Gnu

- *Linker*:

  GCC.EXE and LD.EXE.

• *Make*:

MAKE.EXE

If you do not plan to use the BETA MAKE facility (see section 2.4), you can do with a dummy MAKE program, that simply exits 0, or no MAKE at all.

• *C libraries*:

from GCC.

1.3.3.3 Borland.

• *Assembler*: **NOTE** you *must* have an assembler when using the Borland SDK.

TASM32.EXE, which is sold as a separate tool from Borland. Notice, that version 4.0 of TASM32.EXE does not work on Windows 95. A patch is available from the Borland "Patchs Available" WWW Page at

```
http://loki.borland.com/cpp/Patchs.htm.
```

The patch is directly available by FTP from

```
ftp://ftp.borland.com/pub/techinfo/techdocs/language/
  tools/turboasm/ta4p01.zip.
```

• *Linker*:

TLINK32.EXE (version 1.0), which is included in the above Assembler package from Borland, does not work under Windows 95. It aborts with an "Illegal Instruction" error.

You need to obtain the TLINK32.EXE linker (version 1.50), which is part of the Borland C++ (version >= 4.5).

• *Make*:

MAKE.EXE, which is included in the above Assembler package from Borland, and is also part of Borland C++ (version >= 4.5).

If you do not plan to use the BETA MAKE facility (see section 2.4), you can do with a dummy MAKE program, that simply exits 0, or no MAKE at all.

• *C libraries*: must be obtained from e.g. Borland C++ (version >= 4.5).

If you have installed Borland C++ for Windows 95/NT, version 4.5 or later, all you need to install additionally is Borland Assembler, which is available from Borland (http://www.borland.com/).

1.4 Compiling a Demo Program

After installation you may try using the BETA compiler using the following steps:

1. Copy a demo program from demo directory to your working directory:
   ```
   copy C:\BETA\demo\beta\record.bet
   copy C:\BETA\demo\beta\recordlib.bet
   ```
2. Compile the `record` program (`record.bet`):
   ```
   beta record
   ```
3. Execute the `record` program:
   ```
   record
   ```

For further details, see [MIA 90-2].

2 Using BETA on Windows NT / 95

This chapter contains various useful information for your everyday use of BETA on Windows NT and/or Windows 95.

2.1 Using the tools.

2.1.1 Running the BETA compiler

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

```
beta foo
```

at the prompt in your favorite shell, e.g. "4DOS for Windows NT". This command will invoke the BETA compiler and assemble and link the output from the BETA compiler. Assembly and linkage will be done using either the Borland or the Microsoft assembler and linker. The final object code will be put on the file `foo.exe`, which may be executed by typing:

```
foo
```

Typing

```
beta -h
```

will show a list of legal options for the compiler.

You can also run the BETA compiler by double clicking the compiler icon in the program group that was created during the installation.

See [MIA 90-2] for details.

2.1.2 Running Sif

Execute as follows:

```
sif [file]
```

See [MIA 90-11] for details.

2.1.3 Running Frigg

Execute as follows:

```
frigg [projectname]
```

See [MIA 96-33] for details.

2.1.4 Running Valhalla

Execute as follows:

```
valhalla [projectname]
```

See [MIA 92-12] for details.

2.2 System Files and Directories

The assembly code (Borland SDK only) and object code are placed in different sub directories.

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

```
C:\USERS\smith\test
```

When the BETA compiler is invoked, the assembly code and object code for `foo` is placed in the directory:

```
C:\USERS\smith\test\nti\%SDK%
```

Where `%SDK%` is either `ms`, `gnu` or `bor` depending on whether you are using the Microsoft, Gnu or Borland software development tools.

This means that the directory will contain the files:

```
foo.asm
```

```
foo.obj
foo.db
```

Note that `foo.asm` is usually deleted after the compilation.

The executable file `foo.exe` will always reside as

```
C:\USERS\smith\test\foo.exe
```

The file `DelsLog.1` located in `%BETALIB%` is a file used by the `uninstall` mechanism. It should not be deleted.

See [MIA 90-2] for a complete list of files created by the compiler during compilation.

2.3 Environment Variables

The following environment variables are used in the Mjølner System on Windows NT / 95. For most users, only the BETALIB and SDK variables are relevant:

BETALIB

Specifies how *~beta* is expanded in BETA fragment properties. I.e. it should be set to the path indicating where you installed the Mjølner System. 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 either Borland TLINK32 or Microsoft LINK). If set, it totally overwrites the default link options, the compiler would have used otherwise.

TMPDIR

Normally, the link-directives in the `.bat` files will use `C:\TEMP` for temporary files. If another directory is to be used (e.g. because `C:\TEMP` 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. If TMPDIR is not set, the environment variables TMP and TEMP are searched.

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 behavior 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 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 (above) and AOAPercentage, because they specify conflicting behavior 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 behavior 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 behavior 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:

```
set BETART=InfoIOA:IOA=1024:InfoFile=info.dump
```

SDK

Can be set to either `bor`, `gnu` or `ms` to signal whether the Gnu, Borland or Microsoft development tools are to be used with the Mjølner System.

ASM_VERSION

specify what assembler version to use with the SDK. Use:

ASM_VERSION=6 for Microsoft MASM6.11 (ML.EXE)

ASM_VERSION=386 for Microsoft MASM386.EXE

ASM_VERSION=4 for Borland TASM32.EXE

For SDK=ms ASM_VERSION=386 is default.

For SDK=gnu ASM_VERSION is not in use currently.

For SDK=bor ASM_VERSION=4 is default.

LD_LIBRARY_PATH

When using the Borland SDK, this variable should point to the directory containing the Borland libraries to link with.

LIB

When using the Microsoft SDK, this variable should point to the directory containing the Microsoft libraries to link with.

2.3.1 Automatic Setting of Environment Variables

You can make Windows NT set the BETALIB, SDK, LD_LIBRARY_PATH, ASM_VERSION and Path environment variables automatically each time you log in via the System tool in the Control Panel.

You can make Windows 95 set the BETALIB, SDK, LD_LIBRARY_PATH, ASM_VERSION and Path environment variables automatically each time you log in by setting it in the `autoexec.bat` file.

2.4 Make Files

Make files have been made mostly obsolete by the addition of the BUILD property. It is recommended *not* using Make files. Please refer to 2.5 for more information on how to use BUILD.

Make files are used in the Mjølner System to keep externally linked object files up-to-date. E.g. you may imagine having a `foo.bet` file containing:

```
ORIGIN '~beta/basiclib/v1.5/betaenv';
OBJFILE nti '$/foo.obj';
MAKE nti 'external/foo.make';
-- PROGRAM:descriptor--
(# c_foo: external
     (# ... #);
do ...
#)
```

The OBJFILE property specifies that an external object file should be included in the link directive (located in either the `nti\ms`, `nti\gnu` or `nti\bor` sub directory – this is what the dollar sign expands to). This external object file could, e.g. contain the definition of the external function `c_foo`, that is referenced. The MAKE property specifies that on the `nti` platform (i.e. Windows NT/95 for Intel), the makefile `foo.make` in the `external` subdirectory should be invoked.

Makefiles will invoke c compilers and assemblers. But the options and names of the compiler and assembler are very different depending on the SDK used. As a result most of the commands that normally go into a make file has been extracted into the following three files:

1. `%BETALIB%\bin\ms\rules6.make` for MASM 6.11 compatibility (Microsoft)
2. `%BETALIB%\bin\ms\rules386.make` for MASM386 compatibility (Microsoft)
3. `%BETALIB%\bin\bor\rules4.make` for TASM32 compatibility (Borland)

Please consult the file for specific information.

Some variables are involved in the Make process:

SRCDIR

: Is set automatically by the compiler. The value is the directory in which the external source files reside.

OBJDIR

: Is set automatically by the compiler. The value is the directory in which the external object files are to be generated.

BASEDIR

: Must be set by the user (is used by the rules$(ASM_VERSION).make files). The value must be the directory in which the BETA source file containing the MAKE property resides.

MAKEOBJS

: Must be a list of objects to build.

As a result of moving the product specific commands to another file, the make files in the Mjølner System for Windows NT/95 are extremely simple, and will always consist of three parts:

1. Specify the BASEDIR and MAKEOBJS variables.
2. Include the rules$(ASM_VERSION).make file
3. Specify dependencies.

Here is how `foo.make` would look like, assuming that the `foo.c` resides in the `external` directory:

```
BASEDIR = $(SRCDIR)\..
MAKEOBJS= $(OBJDIR)\foo.obj
!include <$(BETALIB)\bin\$(SDK)\rules$(ASM_VERSION).make>
$(OBJDIR)\foo.obj: $(SRCDIR)\foo.c
```

How a c file is compiled is embedded in the `rules$(ASM_VERSION).make` file. One should always remember to create make files that are not referring to a specific compiler or assembler.

2.5 BUILD

Please refer to the compiler manual [MIA 90-2] for details on the BUILD property in general. Using BUILD with the Mjølner System on Windows could be difficult, as your BUILD statements must take into account all (currently) three SDK values. This is currently handled by the `betacc` program in `%BETALIB%\bin`. A typical BUILD property would be something like: (Taken from regexplib.bet in basiclib/private)

```
BUILD nti     '$$/regexpr.obj' 'external/regexpr.c'
              'external/regexpr.h'
              'betacc $0 $1'
      ppcmac  ':$$:regexpr.obj' ':external:regexpr.c'
              ':external:regexpr.h'
              'MrC -D MAC -D __STDC__ -proto strict -o $0 $1'
      default '$$/regexpr.o' 'external/regexpr.c'
              'external/regexpr.h'
              '$CC -c -o $0 $1';
```

2.6 Using Windows Resource Files

You may use Windows resource files in your BETA programs. You do this by specifying a RESOURCE property, e.g.

```
ORIGIN '~beta/basiclib/v1.5/betaenv';
RESOURCE nti 'foo.res';
-- PROGRAM:descriptor--
(# do ...
#)
```

This specifies, that when linking, the `foo.res` precompiled resource file should be included. You may also specify inclusion of a non-compiled resource file (`foo.rc`). This will make the compiler generate appropriate calls of the resource compiler for the SDK used. If you are using Microsoft SDK, this requires, that you have the RC.EXE tool installed, and that it can be found in your path. If you use the Borland SDK, it requires that BRCC32.EXE is installed and is in your path.

2.7 Calling External Functions

As shown in, e.g. [MIA 94-24], functions and procedures produced by the C compilers can be called from within BETA. Normally you will just prefix your pattern with `external` to achieve this. On Windows NT/95 there are, however, two different ways to call C:

1. If the C function is declared as `__stdcall`, you must specify `callStd` in the do-part of the BETA external declaration, e.g.

   In C:
   ```
   int __stdcall foo1();
   ```
   In BETA:
   ```
   foo1: external
     (# result: @integer;
     do callStd;
     exit result
     #);
   ```
   Notice, that many C header files has defined `WINAPI` as an alias for `__stdcall`.

2. Otherwise you can specify callC in the do-part. This is default, so leaving out the do-part is equivalent.

2.8 Linking without Console

By default the programs are linked in such a way that they always include a console. To avoid this you must set the BETALINKOPTIONS environment variable before compiling the application. This may especially be relevant for GUI programs.

2.8.1 Borland

If you use the `bor` SDK, you must set `BETALINKOPTIONS` as follows

```
set BETALINKOPTIONS=-aa -c -n -Tpe
```

2.8.2 Microsoft

If you use the `ms` SDK, you must set `BETALINKOPTIONS` as follows

```
set BETALINKOPTIONS=/NOLOGO /MACHINE:ix86 /SUBSYSTEM:WINDOWS
                    /ENTRY:WinMainCRTStartup /INCREMENTAL:no
```

(all in one line).

You may combine this with incremental link by using `/INCREMENTAL:yes`, please see the following section.

2.8.3 Gnu

Currently unknown. Please refer to the BETA FAQ for up to date information.

2.9 Microsoft Incremental Link

If you are using the Microsoft SDK, you may speed up linking by using *incremental link*. This can be done by setting the BETALINKOPTIONS environment variable.

> **NOTICE: If you use incremental link, programs using persistence will not work, and if your program results in any runtime-error, the BETA runtime system will not be able to correctly display the call chain.**

To use incremental link specify `BETALINKOPTIONS` as follows

```
set BETALINKOPTIONS=/NOLOGO /MACHINE:ix86 /SUBSYSTEM:CONSOLE
                    /ENTRY:WinMainCRTStartup /INCREMENTAL:yes
```

(all in one line).

You may combine this with linking without console by using `/SUBSYSTEM:WINDOWS` instead.

2.10 Using the GNU Emacs Editor

If you want to use the very popular GNU Emacs text editor as an alternative to the Sif hyper structure editor (see [MIA 90-11]) included in the Mjølner System, you may benefit from the beta-mode for Emacs located in the file

`%BETALIB%\emacs\v1.7\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\v1.7` 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.

2.11 Documentation on the World Wide Web

Various information about BETA and The Mjølner System can be found on the Internet at the URL

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

which contains the BETA home page. This home page contains links to the BETA FAQ, Newsletters, etc., and it is the intent, that the Mjølner System manuals will be made available through this home page too.

2.12 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-language-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.

2.13 Error Reports

The following email address can be used to send error reports and comments:

```
support@mjolner.dk
```

The following classification of errors can be used to indicate which priority an error should have in the maintenance process.

B Serious bug

The tool crashes or produces a wrong result and the error cannot be circumvented.

b Minor bug

The tool produces a wrong result but the error can be circumvented

I Serious inconvenience

E.g. lack of important functionality.

i Minor inconvenience

E.g. lack of functionality that "would be nice to have", but is not crucial.

An error report should include

- a small description the steps that lead to the the error situation (if possible)
- output from the console (if available)
- dumps (if available)

3. Bibliography

[MIA 90-2] 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*: Valhalla - The Mjølner BETA Debugger, Tutorial and Reference Manual.* Mjølner Informatics Report MIA 92-12.

[MIA 94-24] Mjølner Informatics: *The Mjølner System – The Mjølner System Tutorial.* Mjølner Informatics Report MIA 94-24.

[MIA 94-26] Mjølner Informatics: *The Mjølner System – BETA Language Introduction.* Mjølner Informatics Report MIA 94-26.

[MIA 96-33] Mjølner Informatics: *The Mjølner System – Frigg User Interface Builder, Tutorial and Reference Manual.* Mjølner Informatics Report MIA 96-33.

4. Index