The Mjølner BETA System

Using BETA on the Macintosh

Mjølner Informatics Report
MIA 90-06(1.4)
April 1997

Contents

1 The Mjølner BETA System on Macintosh

This manual describes how to use the Mjølner BETA System release 4.0.2 on Macintosh.

The current version of the BETA compiler on the Macintosh is available as a `MPW` tool. To use the `MPW` tool compiler some additional software is required:

MPW Requirements

- `MPW` version 3.4 or later
- Interfaces&Libraries
- `MPW` PPCLink tool v1.4 or later
- `MPW` Rez tool

System Requirements

Furthermore, the compiler requires a minimum machine configuration of 40 MB of memory and a Power Mac processor (or Performa with RISC processor), and a harddisk with 60 - 100 mb of free space (depending on the size of the harddisk).

2 Installation Guide

This section describes how to unpack and install the Mjølner BETA System for Macintosh. The section also includes description of memory requirements to MPW and to BETA applications.

2.1 Contents

The distribution contains:

- BETA Compiler v5.2
- Basic libraries with text, stream, file, persistence, concurrency, etc.
- Container libraries with collections, sets, lists, hashtables, sorting, etc.
- GuiEnv, a platform independent object oriented graphical user interface library,
- BETA demo programs, including demo programs of how to use BETA and how to use GuiEnv.

2.2 Installation

2.2.1 Unpack the Distribution Floppies

Self-extracting archive

This distribution is stored as a self-extracting archive. To install the Mjølner BETA System, doubleclick the self-extracting archive.

2.2.2 Configuring the Files

UserStartup•beta
BetaStartup•Home
BetaStartup•Menu

After the beta folder is installed you need to move the file `UserStartup•beta` in the `beta` folder to the `MPW` folder (the folder containing the `MPW Shell`).

The purpose of `UserStartup•beta` is to set up the environment used by The Mjølner BETA System. When starting `MPW` the first time after you moved `UserStartup•beta` to the `MPW` folder, you will be asked to select the `beta` folder. The position of the `beta` folder will be remembered afterwards using the file `BetaStartup•Home` in the `MPW` folder. If this file is deleted or you move or rename the `beta` folder you will be prompted for the `beta` folder again.

During the first startup `UserStartup•beta` also creates the file `BetaStartup•Menu`. In `BetaStartup•Menu` the **ß** menu is defined as described below. You can modify this file to suit your specific needs. In case it is deleted a new standard `Beta-Startup•Menu` is created during the next startup of the `MPW Shell`.

2.3 Memory Requirements

2.3.1 Memory Requirements to MPW Shell

15 MB for the MPW Shell

To ensure proper workings of `MPW` and the BETA compiler, you should ensure, that enough memory is allocated for `MPW`. In order to run the BETA compiler at least 15 MB of memory must be allocated to the `MPW` Shell. The amount of memory can be set by selecting the `MPW Shell` icon and using the **Show Info** command in the Finder as illustrated in the figure below:

More memory allocated to `MPW` makes the compiler and especially the linker run faster.

2.3.2 BETA Application Memory Requirements

1,5 MB for a BETA application

You may have to increase the memory requirements of a compiled BETA application. This is done by selecting the BETA application, and then use the **Show Info** command in the Finder.

3 Using MPW—Macintosh Programmers Workshop

This section briefly describes how to use `MPW` for compiling and running BETA applications.

`MPW` is a program development environment for the Macintosh computers with tools for editing, compilation and execution of e.g. BETA programs. `MPW` is a command driven interface to the Macintosh operative system (analogous to the Unix shell).

MPW WorkSheet

`MPW` is centered around the concept of a worksheet, where the commands are entered and thereafter executed. A worksheet saves its contents from session to session and can therefore be used to contain the most often used commands such that they can be easily executed in later sessions. The commands in the worksheet can be edited using the usual Macintosh editing facilities.[1]

Commands in MPW

Commands can be executed from the worksheet in two different ways:

- The text cursor is placed somewhere in the command line and the Enter key[2] is pressed. The entire line will then be executed as one command.
- Some text may be selected and the Enter key is pressed. The selected text will then be executed as one command.

3.1 Using **MPW** for BETA programming

When starting `MPW` after installation, an environment is set up to make it easy to use the BETA system. The BETA compiler can be executed either using the script `beta` or by using the special **ß** menu.

Notice that you only need to activate the BETA compiler on the program fragment that constitutes the application. If the program fragment uses other fragments (libraries), these are automatically included by the compiler and linker. If some fragments have been changed since the last compilation, these fragments will automatically be recompiled.

---

1 Please se your Macintosh manuals for how to use your Macintosh.

2 The Enter key is located in the lower right corner of the numeric keypad. Pressing the Return key will just insert a carriage return

3.2 The ß Menu

The BETA environment defines a **ß** menu containing items which makes it easy to use the BETA compiler.

ß menu items

The items are:

Compile {Active}	Compile the front most window
Recompile	Compile the previously compiled file again
Compile File...	Prompts the user for a file, and compiles the selected file
Execute	Execute the last compiled file
Open Fragment	Tries to open the selection. The selection is treated as a BETA file name, e.g. `~beta/guienv/v1.4/guienv`
Directory {Active}	Changes directory to the location of the file shown in the front most window
-Application	Compile the BETA program as an application (default)
-Tool	Compile the BETA program as an `MPW` Tool

beta script

In most cases, the **ß** menu defines the necessary interface to the BETA compiler. However, the advanced user may prefer to use the `beta` script instead, please see [MIA 90-02] for legal options etc.

When selecting **Compile {Active}** the `beta` script is invoked with the active window as argument. The script first executes the BETA compiler and then executes the job file generated by the compiler. The job file links the compiled application. Output during compilation is directed to a separate `MPW` window.

3.2.1 Application and Tool

The options **-Application** and **-Tool** specify whether the generated application should be a Macintosh stand-alone application or a `MPW` tool. The default is that the application is linked as a Macintosh stand-alone application. If a BETA program uses the input/output streams keyboard and screen in `betaenv` and is executed as a stand-alone Macintosh application, a simple console window is opened. The input/output stream is redirected to this window. The console also defines the standard , **File**, and **Edit** menus. The **Cut**, **Copy**, and **Paste** items in the **Edit** menus are available.

The execution can be stopped by selecting **Quit** in the **File** menu.

Notice, that if the program is using **GuiEnv**, it must be compiled as a stand-alone application. The input/output is in this case redirected to a special **GuiEnv** window.

3.3 Environment Variables

The following `MPW` environment variables are used by the Mjølner BETA System.

> **Warning: Except for the Verbose, Time, and BETART you should not change these variables yourself.**

`{Verbose}`

To analyse the BETA compiler memory usage is could be useful to set the `Verbose` option by issuing the following command in the `MPW WorkSheet`:

```
Set Verbose 1
Export Verbose
```

Output from the garbage collector of the BETA compiler will then be directed to a special window. The following command will turn off the garbage collector output:

```
Unset Verbose
Export Verbose
```

{Time}

To analyse the time used by the BETA compiler and the linker set the `Time` option by:

```
Set Time 1
Export Time
```

Time usage of the compiler and the linker will be printed in the compiler output window. The following command will turn off the time usage output:

```
Unset Time
Export Time
```

{betalib}

Specifies the location of the BETA folder (*~beta*). It is used by many tools in the Mjølner BETA System. It is set in the `MPW` startup sequence (`UserStartup•beta`).

{betaopts}

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

{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*200Kb, AOABlock=200Kb, LVRABlock=200Kb,
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 200Kb)
```

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 200Kb)
```

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: 200).

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: 200).

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: 50).

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 AOAPer-

centage, 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: 200).

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: 50).

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 new block 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

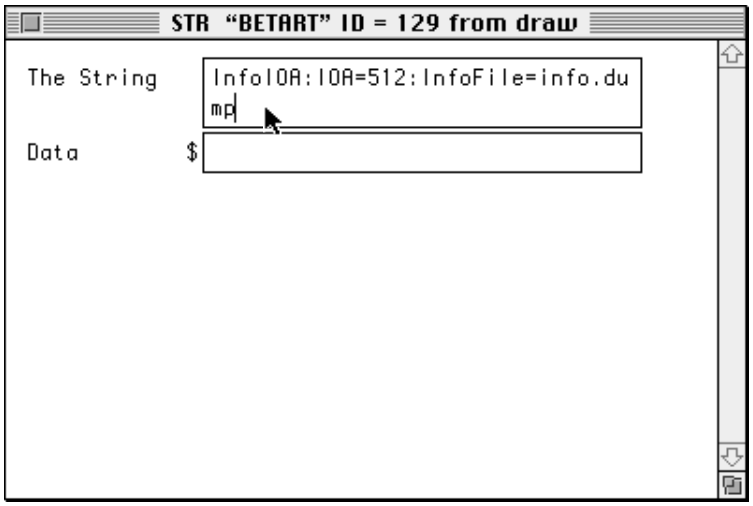
Name of file on which to write all this information (Default: standard error file `stderr`).

Example (for a `MPW` tool):

```
set BETART "InfoIOA:IOA=512:InfoFile=info.dump"
export BETART
```

Example (for an application):

For applications the BETART environment variable is read from the resource of type `'STR '` with number 129 and name BETART. To change this resource use a resource editor, e.g. ResEdit. Specify the value as The String:



The specify the number and name using the Resource Info dialog.

`{betaLinkLibs}`
: Internal linking variable. Specifies the linker libraries to be used by the BETA compiler when linking (using standard MPW linker). If set, it totally overwrites the default link options, the compiler would have used otherwise.

`{betaLinkOptions}`
: Internal linking variable. Specifies the linker options to be used by the BETA compiler when linking (using standard MPW linker). If set, it totally overwrites the default link options, the compiler would have used otherwise.

`{BetaLinkCreator}`
: Specifies the application creator, default “BETA”

`{BetaLinkType}`
: Specifies the application type, default “APPL”

By using the **`-Application`** and **`-Tool`** items in the **ß** menu `BetaLinkCreator` and `BetaLinkType` is changed to “`MPS `” and “`MPST`”, respectively.

4 A Session Using the BETA Compiler

The following steps are usually performed when handling BETA programs under MPW:

1. MPW is started by double clicking the MPW Shell icon or on a BETA source file (.bet). This opens MPW with the worksheet document in a window.
2. We now edit the BETA program text. This may be initiated in two different ways:
   a) We open an existing BETA file by selecting the **Open** entry in the **File** menu.
   b) We create a new file for the BETA program by selecting the **New** entry in the **File** menu. The file name must end with .bet.
3. Now edit the file using the ordinary Macintosh editing facilities in the window containing the program.
4. Save the changed BETA program using the **Save** entry in the **File** menu.
5. Select **Compile {Active}** in the **ß** menu.
6. Messages will now be written in the {MPW}:compilerOutput window. These messages informs about the progress of the compilation, and eventually about syntactic and semantic errors.
7. The program may now be executed by selecting **Execute** in the **ß** menu.
8. Close MPW using the **Quit** entry in the **File** menu.

Steps 3, 4, 5 and 7 may be repeated over and over, while making changes to the same program.

5 Errors and Configuration

5.1 Memory Allocation Errors

When compiling programs or running BETA applications, memory is allocated when needed. During compilation memory is allocated in the `MPW` heap and during execution of BETA applications memory is allocated in the application heap. In case the execution runs out of memory, one of the following messages can appear:

Possibly memory error messages

- IOA heapspace full
- Couldn't allocate ToSpace
- Couldn't allocate IOA

You will then have to increase the memory heap as described in section 2.3.

5.2 Virus detectors

Some virus detectors (e.g. Vaccine and Gatekeeper) do not allow `MPW` to operate correctly. Gatekeeper complains during compilation of a BETA program. Vaccine does not complain; but causes `MPW` to run erroneously, typically to hang. Therefore Vaccine and Gatekeeper must be configured to accept `MPW`.

5.3 Problems with Memory

In case the system is unable to launch an application from `MPW`, the problem might be lack of memory. A solution could be to exit `MPW` and activate the application from the Finder by double-clicking on the application icon.

5.4 Directories in Fragments

Please notice, that directories in ORIGIN, INCLUDE etc. in fragments must be specified using the Unix directory syntax. That is, the Macintosh file:

Unix directory syntax

```
{betalib}guienv:v1.4:guienv
```

must e.g. in an ORIGIN property of a fragment be specified as:

```
ORIGIN '~beta/guienv/v1.4/guienv'
```

Please also note, that `~beta/` is used instead of `{betalib}`.

6 Accessing the Macintosh Toolbox from BETA

Programming the Macintosh is done through the **Toolbox**. The Toolbox includes a large number of routines and they are all documented in **Inside Macintosh**. There exists a BETA interface to most of these routines such that they can be used in a BETA program.

The fragment `maclib` in file `{betalib}maclib:v3.0:maclib` contains this toolbox interface.

6.1 BETA extensions to the Toolbox

Object oriented extensions to the interface to the Macintosh Toolbox have been built in order to make it possible to program the Macintosh in "the BETA way." Using purely Toolbox interface routines from BETA forces you to program in a Pascal-like fashion.

The fragments in the folder `{betalib}guienv:v1.4:` contain an environment (**GuiEnv**) where windows, text editors, dialogs, menus, etc. have been lifted to "real" BETA patterns with an object oriented interface.

The **GuiEnv** environment are documented in **The Mjølner BETA System—Lidskjalv: User Interface Framework**, Mjølner Informatics Report MIA-95-30.

The Toolbox interface is documented in **The Mjølner BETA System—Macintosh Libraries**, Mjølner Informatics Report MIA-90-10.

The folder `{betalib}demo:maclib:` contain demo programs using the Toolbox interface and the folder `{betalib}demo:guienv:` contain demo programs using the object oriented extensions.

6.2 Adding Resources to Applications built with the BETA Compiler

You may use macintosh resource files in your application. You do this by specifying a RESOURCE property, e.g.

```
ORIGIN '~beta/basiclib/v1.5/betaenv';
RESOURCE ppcmac 'foo.r;
-- program: descriptor --
(# do ... #)
```

The BETA compiler will automatically compile all the resources in "foo.r" into the application. The extension `'.r'` tells the system that the resource file is a textual

description that must be compiled using the MPW Rez tool. If the extension is `'.rsrc'` the system knows that the resource is a compiled resource file, and it will include the compiled resources without calling the Rez tool.

You may need to use the lowlevel Toolbox interface to utilize these resource in the current version of the libraries.

References

[Madsen 93] O. L. Madsen, B. Møller-Pedersen, K. Nygaard: *Object-Oriented Programming in the BETA Programming Language*, Addison-Wesley, 1993, ISBN 0-201-62430-3

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

[MIA 90-10] Mjølner Informatics: *The Mjølner BETA System – The Macintosh Libraries*, MjølnerInformatics Report MIA 90-10.

[MIA 94-27] Mjølner Informatics: *The Mjølner BETA System – Lidskjalv User Interface Framework, Reference Manual*, MjølnerInformatics Report MIA 94-27.

Index