HIT-OMA

User's Guide

Document Version 3.4.00

Standard Object MAnager
of the
Hierarchical Evaluation Tool

HIT

Version 3.4.000

HIT-OMA USER'S GUIDE

Norbert Weißenberg (Editor)
Achim Wilde (Editor)

Written and typed by the editor, based upon the German predecessor, the "HIT-OMA Benutzerhandbuch".

Abstract:

OMA is the standard object manager of the hierachical evaluation tool, HIT, when the graphical HIT user interface HITGRAPHIC and its related data base are not used. This guide describes how to use OMA.

This document is released for internal and external use. Corrections, comments, criticism and suggestions for improvement of this document are welcome.

Address:

Universität Dortmund
Informatik IV
Prof. Dr.-Ing. H. Beilner

D-44221 Dortmund

Telefon: (Germany)-(231) 755-2570
Telefax: (Germany)-(231) 755-2386
E-Mail: hit@jojo.informatik.uni-dortmund.de

Contents:

0. Introduction

The program OMA is the standard object manager of the hierachical evaluation tool, HIT, when the graphical HIT user interface HITGRAPHIC and its related data base are not used. This guide describes how to use OMA.

All objects used or produced in the modelling and performance evaluation process except object modules can be stored in a modelling base (shortly called *mobase*), which is administered by OMA. Even more than one mobases can simultaneously be accessed.

Currently a mobase is implemented as a SIMULA directfile (a file type with random access) having a special format. This format is an extension of that used by the modelling base HITFMS of the HIT predecessor COPE.

The HIT system (i.e. the HI-SLANG compiler and all analyzers generated) can directly access and store objects within mobases. This is automatically done if the HIT control or source file is itself stored in a mobase, otherwise the mobase usage can be specified in the control file as described in the HI-SLANG Reference Manual.

For all kinds of object manipulation, which cannot be performed by the HIT system (and for those which can) the interactive OMA program can be used. Moreover OMA is a simple HIT user interface, since editors, the HIT system and a printer can normally be activated via OMA commands.

This guide first presents the unique format of OMA command arguments, followed by explaining all OMA commands. The commands are grouped logically and for every command some examples are presented. The first appendix even contains a total sample session. The next appendix presents a standard control (configuration) file format, stemming from experiences made with OMA. A listing of the contents of the HIT standard mobase and some special hints follow. Instead of an index the OMA help text concludes this guide.

Note that OMA is the German word for grandma. The current version of OMA is titled OMA-SAFE (OMA - Stabilized And Further Extended). This title is a little bit misleading, since the efforts for implementing and testing OMA were small compared with that of HIT. So the Universität Dortmund shall not be liable for errors contained in OMA or for incidental consequential damages of mobases.

1. Arguments of OMA Commands

All work with OMA is done interactively by entering OMA commands. All OMA commands may be abbreviated and have two, one or no arguments. The use of upper-case or lower-case letters in OMA commands and arguments is the same. The commands are explained in the next chapter. As a preparation this chapter introduces the format of command arguments. Arguments denote objects or object set, files or mobases or command modifiers.

An object is denoted by at least specifying some of its attribute values, given in parentheses (see below) and possibly prefixed by a mobase name. If there are more that one objects having the given attribute value combination, then this combination defines an object set. If on the other hand only a single identifier is given, this identifier is interpreted as object name, and thus it means all objects having this name. Moreover objects can be denoted by their link name, if the latter has been defined in a control file (see commands CONTROL, BIND). Finally an object set can be referred by '+', denoting the current constructed object set (see 2.1 below). Thus the syntax for arguments denoting objects is similar to that used in HIT control files:

obj := [object]
| [mobase] attributes
| "link"
| +
|

The *attributes* are defined below. Some commands accept arguments which are either objects or files (referred as *file_obj*). Here the same syntax applies, but a single identifier is interpreted as a file name (which may be SYSIN or SYSOUT), not as a object name. Object names can always be denoted by typing (,,*object*):

file_obj ::= **obj** | **SYSIN** | **SYSOUT**

Other commands accept file names or names of mobases as arguments. The syntax of these names is identical to that of the operating system used, but there must not be a '('-symbol contained, to be able to distinguish file names from mobase names:

file ::= <name of a (sequential) file outside OMA>
mobase ::= <name of an index sequential file containing objects>,

For mobases a default actual mobase can be set with command MOBASE, and thus the mobase name can be omitted. The initial default is the HIT standard mobase which is write protected. A mobase is an index-sequential file in a special format. Do not specify normal files as mobases to avoid OMA crashes.

Command modifiers are of type text:

text ::= <an arbitrary quoted text, e.g. "OMA">

Now we treat the object attributes in detail: There are four attribute values to exactly specify objects. The syntax is that of positional parameters, since the position within the brackets denotes the semantics of the attribute. Each may be missing, but leading commas must appear to denote the position. For the meaning of missing attribute values see the next chapter:

attributes::= ([module], [type], [object], [protection])

The first attribute *module* denotes the kind of representation. There are six alternatives: CONTROL files, HI-SLANG sources, PRECOMpiled elements, PREANAlyzed components (aggregates), SIMULA-written component control procedures (for historical reasons) and any kind of input or output DATA. Each representation name may be abbreviated:

module ::= CONTROL | HISLANG | PRECOM | PREANA | SIMULA | DATA

The second attribute denotes the type or kind of the object. There are 20 self-explaining possibilities, which may be abbreviated:

type	::=	MODEL	COMPONENT	SERVICE	PROCEDURE
	|	CONTROL	EXPERIMENT		
	|	ACCEPT	SCHEDULE	OFFER	DISPATCH
	|	FILE	DUMPFILE	TABLE	GRAPH | HISTOGRAM
	|	TRACE	LISTING	MATRIX	STATES | OTHERS

Module and type have to be consistent according to the following table:

module	allowed types
CONTROL	CONTROL, OTHERS
HISLANG	MODEL, COMPONENT, SERVICE, PROCEDURE, EXPERIMENT, OTHERS
PRECOM	MODEL, COMPONENT, SERVICE, PROCEDURE, EXPERIMENT, OTHERS
PREANA	COMPONENT, OTHERS
SIMULA	ACCEPT, SCHEDULE, DISPATCH, OFFER, OTHERS
DATA	FILE, DUMPFILE, TABLE, GRAPH, HISTOGRAM, TRACE, LISTING, MATRIX, STATES, OTHERS

Each object has an individual name *object* consisting of letters, digits and the characters '.' and '_'. It has to start with a letter or a '#'-character, and 12 characters are significant for OMA. Object names cannot simply be abbreviated, but asterisks can be used within as wild cards. Object names are always stored using upper-case letters.

object ::= <name of an object within a mobase>

There are two protection modes for objects:

protection::= P | U

The default is U. Protected objects can only be manipulated (overwritten, changed, erased) after a special confirmation. The same holds for objects added by another owner (see DIR). Generally you have to confirm manipulations on existing objects or files.

2. OMA Commands

2.0. General Remarks

Most OMA commands have a unique structure: They have none or one argument following the command name, or two arguments separated by TO:

command_name [argument1 [TO argument2]]

If the second arguments only denotes a command modifier it is separated by WITH instead of TO. All commands may be abbreviated if the abbreviation is unique, i.e. no other OMA command starts with that abbreviation.

In the case that the first argument *argument1* is an incompletely specified attribute (attribute value combination) the operation *command_name* will be executed sequentially for any object matching that attribute values. Note: If no attribute values are given (i.e. no argument is specified), all objects are denoted.

The number of matches will be given after finishing all such operations, e.g.

> DIR : 25 Objects.

In general such informations start with a '>'-character. If severe errors occur during execution or if a command sequence is stopped by the user by answering a standard query (see below) with "EXIT", the command name in the above output is surrounded by "Aborted ... after".

In the case that the second argument *argument2* is an incompletely specified attribute the missing attribute values are taken from the first argument before every single operation.

Example:

CHANGE (HISLANG) TO (,,,P)

This command protects all HI-SLANG objects within the actual mobase. If e.g. there are only the three objects

(HISLANG, COMPONENT, cpu, U),
(HISLANG, MODEL, pc386, P),
(PREANA, COMPONENT, cpu, U),

in the actual mobase, then the following two operations will subsequently be executed:

CHANGE (HISLANG, COMPONENT, cpu, U)
TO (HISLANG, COMPONENT, cpu, P)
CHANGE (HISLANG, MODEL, pc386, P)
TO (HISLANG, MODEL, pc386, P)

The missing attribute values of the specification (HISLANG) are filled from the actual matching object and the same is true for (,,,P).

If the command SET VERBOSE is given, which is the default, any single operation is protocolled un-abbreviated and with the actual parameters used before execution. If SET CONFIRM is given (not by default) every single operation has to be confirmed. For this purpose queries are generated having the following standard format:

? question [obj] ? (YES, NO, EXIT)

e.g. ? More ? (YES, NO, EXIT)
or ? Erase mob(DATA, TABLE, pc368, U) ? (YES, NO, EXIT)

There are always three possibilities for the answer, which can be abbreviated (y,n,e):

YES : The command is performed.
NO : This command is not performed, but the same operation will be applied to the next object , if the first argument of the command denotes a set of objects, and thus the command implied a command sequence.
EXIT : The total command sequence is terminated

All commands of a session are listed on a protocol file (see appendix D.). Moreover for most installations the terminal output will be protocolled, too.

We now present all OMA commands in a logical ordering. For every command first the syntax is given in bold style, followed by an informal explanation of the semantics and some examples. For more examples see appendix A.

2.1 Object Sets

Object sets allow to perform an operation (OMA command) with a group of object sequentially by entering only one command, with the object set occuring as a first parameter. There are two possibilities for specifying object sets:

- *Implicite* object sets are defined by using incompletely specified object attribute values as defined above. Every missing attribute value means the set of objects having any value for this attribute. With this mechanism only special kinds of object sets can be built.

- *Explicite* object sets allow assembling an arbitrary set of objects.

There are three commands for the manipulation of explicite object sets, which are denoted by the operator symbols '+' (union), '-' (difference) and '*' (intersection). The current object set itself is also denoted by '+'. Thus '+' can be the first argument of most OMA commands.

Note that implicite object sets are often used to build the explicite set (the current object set), but the latter can only be used implicitely for that purpose.

All of the following three commands have an optional text parameter. If it is provided, the corresponding set operation is only performed for those objects which contain the given text, either in original writing or upper-case. The first occurance found is displayed on the terminal. The text argument allows for simple data base queries. For examples see command '*'.

2.1.1.Command +. Object Set Union

+ obj [WITH text]

Add all objects denoted by *obj* and containing *text* (lowcase or upcase) to the current object set, not to the mobase. This set can be used within every command having a first *obj*-argument by writing '+' for that argument, e.g. 'DIR +'. For examples see command '*'.

2.1.2.Command -. Object Set Difference

- obj [WITH text]

Substract all objects denoted by *obj* and containing *text* (lowcase or upcase) from the current object set. This set can be used within every command having a first *obj*-argument by writing '+' for that argument, e.g. 'DIR +'. Note: '-' alone makes the current object set empty. For examples see command '*'.

2.1.3.Command *. Object Set Intersection

*** obj [WITH text]**

Leaves all objects denoted by *obj* and containing *text* (lowcase or upcase) within the current object set. This set can be used within every command having a first *obj*-argument by writing '+' for that argument, e.g. 'DIR +'.

Examples:

-	The actual object set becomes empty (no argument given, i.e. all objects are denoted).
+ (HI)	Add all HI-SLANG objects to it.
- (,MODEL)	Eliminate models from the actual set.
- (,,P)	Eliminate protected objects.
* () WITH request	Which of these objects use a service named *request*?
* S*	Intersection with objects, which name start with 'S'.
DIR +	Display the actual object set after all these modifications.
SH +	Show the contents of these objects.
-	Delete the actual object set again.
+ mob () WITH get	In which objects of mobase *mob* does the identifier *get* occur?

2.2. Display Commands

The display commands inform about the state of a mobase respectively the commands possible.

2.2.1.DIR. Display Mobase Contents

DIR obj

Displays the directory contents (module, type, object, protection, size, version, last update, owner) for those objects denoted by *obj*. The directory output is sorted alfabetically and affected by SET LINES/LENGTH. It has the following format:

No	Module	Type	Object	P	Size	Ver	Last Update	Owner
n	(module,	type,	object	,P)	256	1	19 APR 1989 15:59	HIT

The *n* is a running number for the set of objects denoted by *obj*. All attribute values of the object follow within the brackets. Next the *size* of the object is given in Lines of Code, and the *ver*sion number is increased whenever an object is modified. Moreover the date, time and the performer/new *owner* of the last update is given.

If more than LINES objects are to be displayed a query in a standard format is submitted whether and when to continue the output.

Examples:

		Display ...
DIR mob	(HISLANG)	all HISLANG objects within *mob*
D	(,COMPO)	all available component types within actual mobase
D	(,,P)	all protected objects
DIR	(,SCHED,,P)	all protected *schedule* procedures
DIR	(,,S*)	all objects with names starting with S
DIR S*		short form of DIR (,,S*)
DIR mob	()	all objects in mobase *mob*
D		short form for DIR ()

Hint:

Some terminals allow to use the DIR output to enter a new command, e.g. by substituting the number by a new command or by a cut and paste technique.

2.2.2.SHOW. Display Object Contents

SHOW obj [WITH text]

Displays the contents of all objects denoted by *obj*. The first line on screen is the actual line (AL). If *text* is given the AL is set to the first line containing that text. Within SHOW the following sub-commands can be given (the last line displayed summarizes this command list):

END	: Quit SHOW, the next object specified by *obj* is shown.
EXIT	: Quit SHOW totally.
+ [n]	: Set AL forward *n* lines or one screen, if *n* is omitted.
- [n]	: Same backwards
++	: Set *AL* to one screen before end of object
--	: Set *AL* to 1.
SET n	: Set *AL* to *n*..
FIND t	: Set *AL* to next line containing text *t*
LINES n	: Number of displayed line becomes *n.*
LENGTH n	: Number of displayed columnes becomes *n.*
NUMBERS	: Switching on/off line numbering.
EDIT	: *AL* and following lines (until only '.' entered) may be overwritten
CHANGE t TO u	: Change all occurences of *t* to *u*, each change has to be confirmed.

All sub-command names may be abbreviated. Here *n* is a natural number and *t* is a string, which must be quoted if it contains blanks. SHOW output is affected by SET LINES/LENGTH/NUMBERS, defining the defaults for the corresponding local commands.

After every correct command the screen is refreshed, and up to LINES lines of the object are displayed, beginning with the new AL. Then the command summary is given again, indicating a wait for input.

SHOW can be terminated by END or EXIT. Navigation is supported by means of the *+n* and *-n* commands, stepping *n* lines forward or backwards.The commands ++ and -- can be used to reach a border of the object, and SET n absolutely addresses line *n* of the object. With LINES and LENGTH the number of lines and columns to be displayed can be set, i.e. the terminal or window size. With NUMBERS the line numbering of the output can be toggled.

SHOW can also be used for simple object editing in the case that the line structure of the object is not concerned: by EDIT the next keybord input lines overwrite the object's lines starting with the actual line until a single dot is entered, and CHANGE serves to replace some next occurances of the firrst text argument by the second, if no line gets too long.

Examples:

SHOW (DATA, TABLE)	Display result tables.
FIND evaobj	Search for the results concerning *evaobj*.
F	Search for next occurance of evaobj.
+	See the next page.
NUM	Set line numbering.
END	Display the next table object, if there are more.

2.2.3.HELP. Display Help Text

HELP [text]

Appendix D. is contained in the HIT standard mobase. This text is displayed by the command 'SHOW <standard mobase> (,,help) [WITH *text*]', i.e. the first line shown is the first line containing *text.* If text is not given the help text is displayed from the beginning.

HELP must be terminated with END, and all SHOW sub-commands are available to navigate in the OMA help text.

Examples:

HELP file_obj	What is a *file_obj?*
+10	Read the next 10 lines.
F copy	How to copy objects?
END	Quit HELP.

2.3. Object Management

The main task of OMA is to manage objects, i.e. to add, select, copy, modify and delete objects. The following commands serve for this purpose.

2.3.1.ADD. Add File as Object

ADD file [TO obj]

Insert a copy of the given *file* into the mobase and with the attribute values given by *obj*. Some attribute values may be left out. OMA can determine them: *object* is set to *file*, and the *modules* PREANA, PRECOM, CONTROL and HISLANG as well as the *types* MODEL, COMPONENT, SERVICE and EXPERIMENT may be set from the first line of the file contents. If the *file*name contains wild card symbols ('*'), all files corresponding to the pattern are added, but this facility is installation dependent. Moreover the attribute defaults described in the HI-SLANG Reference Manual, chapter 8.2.3., apply.

If an object with the same attribute values is already contained in the mobase (an older version), a standard query is submitted whether to overwrite it (note that there is currently no version management). Other queries appear for protected objects and for objects created by someone else but the current user.

Examples:

ADD pc386 TO mob (HI,MOD)	Add file *pc386* as HI-SLANG model with the same name.
ADD pc386	Add it to the actual mobase, attribute values determined automatically.
ADD sysin TO (CON, , exp1)	Create a new object via keyboard.

2.3.2.SELECT. Select Object on File

SELECT obj [TO [EXTEND] file]

Select the object denoted by *obj* to the given file. If *obj* specifies more than one mobase object, all those objects are selected. If *file* is omitted it is set to the *object* name. If *file* is '#' the filenames are generated due to a fixed pattern (see appendix D.).The argument *file* can be prefixed by EXTEND to extend an existing file by the object's contents.

Examples:

SEL S*	Select all objects starting with letter 'S' to files with the same name. If there are more than one with the same name the last one overwrites the previous after a query.
SEL * TO #	Select all objects to unique file names being generated.
SEL * TO EXTEND all	Concatenate all objects on file *all.*
SEL cpu TO nws.cpu	The *cpu* is written to the named file.
SEL cpu TO sysout	The *cpu* is displayed on terminal, but better use SHOW.

2.3.3.COPY. Copy Objects

COPY obj1 TO obj2

Copy all objects denoted by *obj1* to destinations denoted by *obj2*. Free positions in *obj2* are filled from *obj1*, while free positions in *obj1* mean FOR ALL those objects, as explained in chapter 2.0.

Examples:

COPY mob1 (,,,P) TO mob2 ()	Copy all protected object to another mobase, all objects get the former attributes.
COPY * TO mob2 (DATA,OTH)	Copy all objects of the actual mobase to *mob2*, but as data objects.
COPY cpu TO (,COMPO)	A model *cpu* is copied to transform it to a component.

2.3.4.ERASE. Erase Objects

ERASE obj [!]

Erase all objects denoted by *obj*. If '!' is not given, every single object deletion has to be confirmed after a standard query. Protected objects can only be erased after another confirmation. The same holds for objects created by another user.

Examples:

ER	Delete all objects within the actual mobase after separate confirmations.
ER (DATA) !	Delete all data objects without confirmation.
ER +	Erase the objects assembled as object set '+'.
ER cpu	Erase object *cpu*.

2.3.5.CHANGE.Change Object Attribute Values

CHANGE obj1 TO obj2

The attribute values of object *obj1* are changed to those specified by *obj2*. Free positions in *obj2* are filled from *obj1*, while free positions in *obj1* mean FOR ALL those objects. Both arguments have to refer to the same mobase, and the consistency of the new attribute value combination is tested.

The main application is to change either the object name or the protection mode. A modification of module and type of an object is seldomly useful.

Note that the user is responsible that the attribute values of the object fit to the contents of the object. Only the command ADD can determine attribute values from the contents of the object to initially set the attribute values.

Examples:

CHANGE + TO (,,,P)	Protect all objects of the object set '+'.
CH random TO rand	Change name of object(s) *random.*
CH (,MODEL, cpu) TO (,COMPO)	Change type of *cpu.*

2.3.6.NEWOBJECT. Type a New Object.

NEWOBJECT obj

The next lines read from terminal (until a line only containing a single '.' is entered) are added as a new object with attribute values denoted by *obj*. So an editor call is not necessary. If *obj* already exists it will be overwritten after a standard query. Similar to ADD this command can obtain *type* and *module* of the new object in most cases, otherwise an error message is generated.

Examples:

NEWO cpu	Enter a new object called *cpu*
TYPE cpu COMPONENT; ...	line by line (the first line contains the
...	words TYPE and COMPONENT, so the
END TYPE;	missing attribute values can be set),
.	until a single dot is entered.

2.4. Mobase Management

This group of commands does not concern single objects, but a mobase in total. It is advisable to compress a mobase after having performed some write or delete operations to minimize the gaps in the mobase. To copy a mobase completely COPYMOBASE can be used. The copy is implicitely compressed, since unused blocks are not copied. By NEWMOBASE a new empty mobase can be created. Moreover there exist some CONVERT commands to convert a mobase into a sequential file and vice versa.

2.4.1.COMPRESS. Move Gaps to the End

COMPRESS [mobase]

Moves gaps within the index sequential file *mobase* to the end and reserves new space for at least 50 entries in the directory. Thus if the directory is full COMPRESS has to be called before a new object can be added. If *mobase* is omitted, the actual mobase is compressed. The mobase must exist and be in HIT mobase format.

Examples:

COMPRESS	Compress the actual mobase.
COM my.mobase	Compress a named mobase.

2.4.2.COPYMOBASE. Copy a Mobase

COPYMOBASE [mobase1 TO] mobase2

The *mobase1* (or the actual mobase) is copied to a file named *mobase2*. Unused blocks are not copied, so *mobase2* becomes smaller in most cases. This command is identical to CONVERTMM (convert mobase to mobase) below. If *mobase1* is omitted the actual mobase is copied to *mobase2*. A file named *mobase2* must not exist before, while *mobase1* must exist, otherwise error messages are generated.

Examples:

COPYM	my.mob TO his.mob	Make a compressed copy.
MOBASE	my.mob	These two commands ...
COPYM	his.mob	... have the same effect.

2.4.3.NEWMOBASE.Create an Empty Mobase

NEWMOBASE mobase

A new empty mobase with name *mobase* is generated. The file *mobase* must not exist before. Note: *mobase* does not become the actual mobase automatically.

Examples:

NEWM empty.mob	Create a new named mobase.

2.4.4.CONVERTMF. Convert Mobase to File

CONVERTMF [mobase TO] file

The suffix MF means Mobase to File. The index sequential file *mobase* is converted to a sequential and therefore portable and smaller file named *file*. Only in this format a mobase can be transferred to a computer of another type. If omitted, *mobase* is the actual mobase. The *file* must not exist before.

Examples:

CONVERTMF mob TO mob.init	The mobase *mob* is copied in sequential format to file *mob.init.*
MOBASE mob	These two commands ...
CONVERTMF mob.init	... have the same effect.

2.4.5.CONVERTFM. Convert File to Mobase

CONVERTFM file TO mobase

The suffix FM means File to Mobase. The sequential file *file* is assumed to be in a format to convert it to a mobase named *mobase*, i.e *file* has been generated by CONVERTMF before. Otherwise OMA may fail.

Examples:

CONVERTFM mob.init TO mob	Make *mob.init* a mobase named *mob* again.

2.4.6.CONVERTMM. Copy Mobase to Mobase

CONVERTMM [mobase1 TO] mobase2

The suffix MM means Mobase to Mobase. The modelling base *mobase1* (or the actual mobase) is copied to a mobase file named *mobase2*. Unused blocks are not copied, so *mobase2* becomes smaller in most cases. This command is identical to COPYMOBASE above. If *mobase1* is omitted the actual mobase is copied to *mobase2*. A file named *mobase2* must not exist before.

Examples:

CONVERTMM my.mob TO his.mob	Mobase *my.mob* is totally (but without gaps) copied to *his.mob*.
MOBASE my.mob	These two commands ...
CONVERTMM his.mob	... have the same effect.

2.5. OMA as HIT User Interface

The program OMA is not only an object management system, but a simple HIT user interface as well. Via OMA you can call some editors of the underlying operating system to create or modify HI-SLANG sources and other texts comfortably, and you can activate the HIT system to compile and analyze such models. Moreover the results and sources can be printed, and most other existing operating system commands can be executed via OMA. Note however that the extend to which all this is possible is installation defined, with BS2000 and UNIX installations supporting all facilities.

2.5.1.CMD and /. Operating System Access

CMD text and **/ rest_of_line**

For activating an arbitrary operating system command there are two formats due to historical reasons: The CMD command takes a text argument, while the / command submits the rest of the line to the operating system. The last format is preferable.

Examples:

CMD "fstat dat.*"	For UNIX write *ls* instead of *fstat.*
/fstat dat.*	This is the same.

2.5.2.EDxx. Edit Objects

EDxx obj

All objects denoted by *obj* can sequentially be edited by the editor denoted by *xx.* The command ED always invokes the standard editor. The list of available editors is given by the command SET without parameters. For BS2000 and UNIX the following EDxx commands are available:

UNIX:		**BS2000:**	
EDT:	textedit	EDOR, ED:	EDOR
EDV, ED:	vi	EDT:	EDT (supported by an erroneous assembly procedure)
EDJ:	jove		

Before rewriting a changed object to the mobase (denoted by *obj*) OMA submits a standard query and asks for a confirmation.

Examples:

EDOR S*	Edit all objects with names starting by 'S' sequentially
...	enter arbitrary EDOR commands.
HR!H	Leave EDOR for this object.
...	Possibly a next object is loaded.
HR!H	Leave EDOR again.

2.5.3.HIT. Compile and Analyze Objects

HIT obj [WITH text]

Call HIT for (the first) object denoted by *obj* and with additional arguments *text.* The object must have module CONTROL or HISLANG. It is not selected, but the HIT-system reads it directly from the mobase. All standard link names of the HIT system have a default binding to the mobase denoted by *obj*, i.e. listing and results are automatically written to this mobase if they are not explicitely bound. The call given is displayed by entering command SET. After the call OMA is activated again.

Examples:

HIT mymodel WITH "SYMBDUMP=4"	Transmit a BS2000 HIT parameter.
HIT office.lib (CON,CON, experiment1)	HIT call for another mobase.

After the second command e.g. the listing is contained in the *office.lib* as (DATA, LISTING, experiment1, U) and the table output is (DATA, TABLE, experiment1, U).

2.5.4.PRINT. Print Objects

PRINT obj [WITH text]

All objects denoted by *obj* are printed with arguments given by *text.* The print command can be displayed by command SET. If *text* is not given the arguments are "optimally" selected (concerning paper size and print control characters) dependent on *module* and *type* of every objects. This can be affected by SET P3/P4/PP.

Examples:

PR *2	Print all object with ending '2' (version number).
PR (,LI)	Print all listings.

2.6. Other Commands

The rest of commands provided by OMA cannot be grouped again. They in general ease OMA handling. By entering the command END the program OMA is terminated.

2.6.1.BIND. Define Session Alias for Objects

BIND [link TO file_obj]

With no arguments all active bindings are displayed. Otherwise *link* is bound to the given file or object and can be used for that in commands with *obj*-arguments. For the normal user this command is of no interest.

Examples:

BIND	Displays the bindings of all standard link names and user link names.
BIND "EDIT" TO #edit	Defines a new OMA help file, which has link name "EDIT".
BIND "X" TO mob2(HI,MO,cpu)	Define an alias for that object.
SHOW "X"	Use the alias in OMA commands.

2.6.2.CONTROL. Interpret a Control File

CONTROL file_obj

The denoted files or objects (having module CONTROL) are interpreted as HIT control files, and all segments of that control file are interpreted (%COMMON, %COMPILER and %ANALYZER). All mentioned bindings can then be used, e.g. SHOW "TABLE". You must not remind the object names in this case. All actual bindings can be displayed by command BIND. Moreover the %MOBASE commands in the control file define the actual mobase.

Examples:

CONT control_file	Interprete the named HIT control file.

2.6.3.CP. Common Copy

CP file_obj1 TO file_obj2

Copy all files (or SYSIN/SYSOUT) or objects denoted by *file_obj1* to destinations denoted by *file_obj2*. The commands ADD, SELECT and COPY are special cases and use this command internally, since CP equals

SELECT, if only *file_obj2* is a file
ADD,if only *file_obj1* is a file
COPY, if no argument denotes a file and a
file copy, if both arguments denote files.

Examples:

CP	myfile	TO yourfile	File	-> File
CP	myfile	TO (HI, SERVICE, ST)	File	-> Object
CP	(„cpu)	TO #cpu	Object	-> File
CP	+	TO second.mobase ()	Object set	-> Object set

Note that the brackets must be given in the last example, otherwise you copy the object set to a normal file named *second.mobase*. However, OMA will ask you if you want to overwrite that file. If you forget the brackets you can simply answer NO and nothing will happen.

2.6.4.END. Quit OMA

END

Quits the OMA program. As usual for the HIT system the number of all errors and warnings occured and the cputime used will be given. END cannot be abbreviated.

Examples:

END	Terminate the OMA session.

2.6.5.MOBASE.Define a Default Mobase

MOBASE [READONLY] mobase

The *mobase* becomes the new current mobase, to which *obj* arguments of other commands refer, if they don't explicitly refer to another mobase. READONLY mobases cannot be changed or erased, but can however be read simultaneously by different users.

The HIT standard mobase is the initial actual mobase of OMA; it has READONLY mode. The mode can't be abbreviated and can't be changed in the same OMA session.

Examples:

MOBASE READONLY modelling.base	Define a default mobase.
DIR	Display directory of that mobase.
SEL cpu	Select a object.
ADD newobj	This yields an error, because of the READONLY mode.

2.6.6.READ. OMA Commands from File

READ file

Interprets the *file* contents as OMA commands, if the file exists. This helps to adapt OMA to different users and systems, since every user can thus have his own OMA startup file containing e.g. a default mobase and settings possible by the command SET. Moreover by this facility OMA can be used in batch mode. Note: all necessary confirmations should also be in that file.

Examples:

READ oma.startup	Interprete all OMA commands found in file *oma.startup.*

2.6.7.SET. The OMA State

SET [variable [value]]

With no arguments the setting of all variables, the list of build in editors and some other self-explaining information is displayed (see example in appendix A.). The defaults are installation defined, but often VERBOSE, NOCONFIRM, NOWARN, NONUMBERS, LINES 18, LENGTH 80 is set. With arguments the *variable* is set to the given *value*, with the following meaning:

SET [NO]**VERBOSE**	: All commmands on objects are echoed before execution in a non-abbreviated style.
SET [NO]**CONFIRM**	: The execution of every single command listed by VERBOSE must be confirmed.
SET [NO]**ALL**	: All command are only executed for first object matching the given attribute value combination, if NOALL is set.
SET [NO]**WARN**	: Warnings are suppressed by NOWARN.
SET [NO]**NUMBERS**	: SHOW output is numbered by default.
SET **LINES int**	: Default number of lines of a screen (for SHOW/ DIR).
SET **LENGTH int**	: Default number of columns of a screen (SHOW/ DIR).
SET **HIT text**	: Default argument for HIT call.
SET **ED text**	: Default argument for EDitor call.
SET **Px text**	: Print argument for DIN A3 objects (*x*=3) or DIN A4 objects (*x*=4) or PRINTFILE objects (*x*=P).

Examples:

SET	Display all status informations of OMA.
SET LI 15	Set the number of lines for SHOW/DIR to 15.
SET NOWARN	Suppress warnings.
SET NUMBERS	Give line numbers in SHOW.
SET HIT ",task=exp"	Only perform experiments from now on (BS2000 style).

3. Literature

/BeMW88/ Beilner, H./Mäter, J./Weißenberg, N.:
Towards a Performance Evaluation Environment: News on HIT,
in: Proc. "Modelling Techniques and Tools for Computer Performance Evaluation", Palma de Mallorca, Spain, 1988

/Heck91/ Heck, E.:
HITGRAPHIC User's Guide,
Universität Dortmund, Informatik IV, 1991

/LeWe89/ Lengewitz, P.; Weißenberg, N.:
HIT User's Guide for UNIX Systems,
Universität Dortmund, Informatik IV, 1989

/Weis92a/ Weißenberg, N.:
HIT and HI-SLANG. An Introduction,
Universität Dortmund, Informatik IV, 1992

/Weis92b/ Weißenberg, N.:
HI-SLANG Reference Manual
Universität Dortmund, Informatik IV, 1992

Appendix A. A Sample Session

In this example three kinds of texts have to be distinguished: Comments start with an empty line and are prefixed by '#'-characters. OMA inputs are written in **bold** style, and OMA outputs are given in a smaller fond.

The example was executed on a SUN/4 workstation under UNIX. For OMA installations on other machines it looks similar:

First oma has to be activated. For UNIX this is simply:
oma
```
> You may enter HELP [what] to get information (concerning 'what')
HIT    Version 3.1.000       OMA SAFE Object MAnager       of  92-03-15 13:00
```

Above the start message of OMA is shown. How to create a new mobase? We try

help MOBASE
```
> SHOW HIT/SYSTEM/MOBASE(DATA,OTHERS,HELP,P) WITH "MOBASE"
                                                    V
                a default can be set with command MOBASE.
link    ::=     linkname of an object or a file, as defined in a HIT
                control file or with command BIND
text    ::=     an arbitrary quoted text, e.g. "OMA"
...
> Please enter END,EXIT,+n,-n,++,--,SET n,FIND t,EDIT,LENGTH n,LINES n,NUMBERS
```

The 'V'-character after the VERBOSE output points to the occurence of the word
MOBASE in the help text. But this is not interesting. We search on by entering the
command FIND. Although we get the command MOBASE now, the command
NEWMOBASE is also visible. Herafter we exit HELP:

find
```
V
MOBASE [READONLY] mobase
        'mobase' becomes the new current mobase, to which 'obj' arguments
        of other commands refer, if they don't explicitly refer to another
        mobase. READONLY mobases can't be changed or erased, but can
        therefore be read simultaneously by different users.

NEWMOBASE mobase
        A new empty mobase with name 'mobase' is generated.

 NEWOBJECT obj
...
> Please enter END,EXIT,+n,-n,++,--,SET n,FIND t,EDIT,LENGTH n,LINES n,NUMBERS
```
end
```
> HELP :            Okay.
```

We want to create a mobase named *testlib* and make it the actual mobase:

newmobase testlib
```
> NEWMOBASE :   Okay.
```

mobase testlib
```
> MOBASE  :       Okay.
```

Just before we have written a component type *cpu* on a file, which shall now be added. # Here we can see the VERBOSE output and a completion message. *Type* and *module* of # the object was automatically determined, as the DIR output proves.

add cpu
```
> COPY cpu TO testlib(,,CPU,U)
> ADD        :       1 Objects.
```

d
```
> Mobase 'testlib' totally contains  1 objects.
```

No	Module	Type	Object	P	Size	Ver	Last Update	Owner
1	(HISLANG,	COMPONENT,	CPU	,U)	12	1	1992-03-16 11:00	HIT

```
> DIR:       1 Objects.
```

Now we add online a control file for performing a precompilation of *cpu*, and submit # the control file to the HIT system:

newobj (con,,cpuprecom)
```
> COPY SYSIN TO testlib(CONTROL,CONTROL,CPUPRECOM,U)
Please enter lines until you only enter a '.'
```
%compiler
%parm=precom,nowarn
%mobase testlib
%bind "precom" to (,,cpu)
%end
%copy "cpu"
.
```
> NEWOBJECT :    1 Objects.
```

hit cpup*
```
> HIT testlib(CONTROL,CONTROL,CPUPRECOM,U) WITH " "
HIT    Version 3.1.000       HI-SLANG Compiler        of 92-03-15 13:00
Please enter name of Compiler SOURCE or CONTROL file:
testlib(CONTROL,CONTROL,CPUPRECOM,U)

> FAN        :       Only   2 Warnings.  Cpu Time used :      0.080 Seconds.
> PASS 1     :       Only   1 Warning .  Cpu Time used :      1.080 Seconds.
> PASS 2     :       Okay.               Cpu Time used :      0.520 Seconds.
> PRE_WRITE :        Okay.               Cpu Time used :      0.500 Seconds.
> T O T A L :        Only   3 Warnings.  Cpu Time used :      2.860 Seconds.
                                         Compile Rate :       6.690 Lines/Sec.
> HIT :      1 Objects.
```

One can ignore the precompilation warnings.
As a result we now have two more objects contained in the mobase: The listing of the # precompilation and the precompiled representation of the cpu. For this type the first # compiler passes will always be skipped from now on, since the compiler automatically # selects the most efficient representation available.

d
```
> Mobase 'testlib' totally contains  4 objects.
```

No	Module	Type	Object	P	Size	Ver	Last Update	Owner
1	(CONTROL,	CONTROL,	CPUPRECOM	,U)	5	1	1992-03-16 11:01	HIT
2	(DATA,	LISTING,	CPUPRECOM	,U)	34	1	1992-03-16 11:01	
3	(HISLANG,	COMPONENT,	CPU	,U)	12	1	1992-03-16 11:00	HIT
4	(PRECOM,	COMPONENT,	CPU	,U)	12	1	1992-03-16 11:01	

```
> DIR:       4 Objects.
```

Now we will do our first performance analysis. An experiment to analyze our *cpu* shall
be created, but we don't want to leave OMA for this purpose, but try a "buildin" editor.
Lets see which editors are available:

set
```
Variable settings
VERBOSE, CONFIRM, ALL, WARN, NUMBERS  : T, F, T, F, F,
LINES  , LENGTH                       :   18,   80
HIT    :
ED     :
P3     :
P4     :-o std4gs
PP     :

> Other Informations
HIT  file pattern: <S><-/><1>t.<+>.<L3>
obj->file pattern: t.<O>.<T>.<M>
HIT  call command  : hit '#' #
PRINT   command  : prt -c siemlp -b o4 -o std3gs # #

> Available EDITORs: (With ED you get the first editor)
EDV   : vi
EDT   : textedit
EDJ    : jove
else :
```

The last lines contain the desired informations. We take we standard UNIX editor *vi.*
But first we must create the object to edit. We start with a copy of the control file,
modify it and add the experiment in the same object. Then we activate HIT again:

copy (con,,cpup*) to cpueval
```
> COPY testlib(CONTROL,CONTROL,CPUPRECOM,U) TO testlib(CONTROL,CONTROL,CPUEVAL,U)
> COPY          :          1 Objects.
```

ed cpueval
```
> ED testlib(CONTROL,CONTROL,CPUEVAL,U)
```

< vi editing >
:wq

```
? Update  testlib(CONTROL,CONTROL,CPUEVAL,U) ? (YES,NO,EXIT)
```
y
```
> ED            :          1 Objects.
```

hit cpueval

```
> HIT testlib(HISLANG,EXPERIMENT,CPUEVAL,U) WITH " "
HIT    Version 3.1.000       HI-SLANG Compiler         of 92-03-15 13:00
Please enter name of Compiler SOURCE or CONTROL file:
testlib(HISLANG,EXPERIMENT,CPUEVAL,U)

> PRE_READ  :     Only   1 Warning .   Cpu Time used :     1.500 Seconds.
> PASS 1    :     Only   1 Warning .   Cpu Time used :     0.820 Seconds.
> PASS 2    :     Only   2 Warnings.   Cpu Time used :     0.720 Seconds.
> EXT_REF   :     Okay.                Cpu Time used :     0.320 Seconds.
> LINKER    :     Okay.                Cpu Time used :     0.140 Seconds.
> SCG       :     Okay.                Cpu Time used :     1.100 Seconds.
> T O T A L :     Only   4 Warnings.   Cpu Time used :     5.180 Seconds.
                                       Compile  Rate :     6.977 Lines/Sec.

compiling : t.hitcode.sim

End of pass 1 0.540 seconds
End of pass 2 1.800 seconds
End of pass 3 4.000 seconds
End of pass 4 0.000 seconds
End of pass 5 0.020 seconds
End of pass 6 3.580 seconds
     No errors and no warnings.
     End of Simula Compilation.

linking : t.hitcode with asimul

HIT              Version 3.1.000    HIT Analyzerof 92-03-15 13:00
Please enter name of Analyzer CONTROL file:
testlib(CONTROL,CONTROL,CPUEVAL,U)

Number      Line    : Description of Errors or Warnings
W.0375      1 : Obj. testlib(DATA,LISTING,CPUEVAL,U) already exists. It will be extended.

> SIMULATIVE    :      Only   1 Warning .  Cpu Time used :      1.140 Seconds.
> T O T A L     :      Only   1 Warning.   Cpu Time used :      8.580 Seconds.

> HIT :       1 Objects.
```

Now we have four additional data objects generated: a listing and the results in form of
a table, dumpfile and a trace, as demanded in our experiment (not shown here):

d

```
> Mobase 'testlib' totally contains  9 objects.
```

No	Module	Type	Object	P	Size	Ver	Last Update	Owner
1	(CONTROL,	CONTROL,	CPUEVAL	,U)	34	1	1992-03-17 13:39	HIT
2	(CONTROL,	CONTROL,	CPUPRECOM	,U)	5	1	1992-03-16 11:01	HIT
3	(DATA,	DUMPFILE,	CPUEVAL	,U)	136	4	1992-03-17 13:24	
4	(DATA,	LISTING,	CPUEVAL	,U)	203	2	1992-03-17 13:24	
5	(DATA,	LISTING,	CPUPRECOM	,U)	34	1	1992-03-16 11:01	
6	(DATA,	TABLE,	CPUEVAL	,U)	276	4	1992-03-17 13:24	
7	(DATA,	TRACE,	CPUEVAL	,U)	112	4	1992-03-17 13:24	
8	(HISLANG,	COMPONENT,	CPU	,U)	12	1	1992-03-16 11:00	HIT
9	(PRECOM,	COMPONENT,	CPU	,U)	12	1	1992-03-16 11:01	

```
> DIR:        9 Objects.
```

Let's do some queries. First we delete the actual object set, add all data objects and
substract listings. The resulting set is then displayed, but only the attributes are given in
this case:

```
-
> -     :       No Objects.
```

+ (data)

```
> + testlib(DATA,DUMPFILE,CPUEVAL,U)
> + testlib(DATA,LISTING,CPUEVAL,U)
> + testlib(DATA,LISTING,CPUPRECOM,U)
> + testlib(DATA,TABLE,CPUEVAL,U)
> + testlib(DATA,TRACE,CPUEVAL,U)
> +     :       5 Objects.
```

-(,lis)

```
> - testlib(DATA,LISTING,CPUPLOT,U)
> - testlib(DATA,LISTING,CPUPRECOM,U)
> -     :       2 Objects.
```

dir +

```
No| Module        | Type          | Object          |P| Size |Ver|Last Update          | Owner
  1 (DATA,        DUMPFILE,       CPUEVAL         ,U)
  2 (DATA,        TABLE,          CPUEVAL         ,U)
  3 (DATA,        TRACE,          CPUEVAL         ,U)

> DIR:        3 Objects.
```

We create the intersection * with all mobase objects () containing the word *population*.
The corresponding object line will always be displayed before the VERBOSE outputs:

*** () with population**

```
        POPULATION      ALL             0.000000E00
> * testlib(DATA,DUMPFILE,CPUEVAL,U) WITH "population"
Hierarchy   | Esti |    POPULATION      |       THROUGHPUT      |
> * testlib(DATA,TABLE,CPUEVAL,U) WITH "population"
> *     :       2 Objects.
```

We now protect both objects of the set and look into them to see the produced results:

change + to (,,,p)

```
> CHANGE testlib(DATA,TABLE,CPUEVAL,U) TO testlib(DATA,TABLE,CPUEVAL,P)
> CHANGE testlib(DATA,TRACE,CPUEVAL,U) TO testlib(DATA,TRACE,CPUEVAL,P)
> CHANGE  :       2 Objects.
```

show +

```
> SHOW testlib(DATA,DUMPFILE,CPUEVAL,P)
...
```

end

```
> SHOW testlib(DATA,TABLE,CPUEVAL,P)
...
```

end

```
> SHOW      :       2 Objects.
```

We now print and erase the listings and compress the mobase:

print (,lis)
```
> PRINT testlib(DATA,LISTING,CPUEVAL,U) WITH " "
> PRINT testlib(DATA,LISTING,CPUPRECOM,U) WITH " "
> PRINT       :       2 Objects.
```

er (,li)
```
> ERASE testlib(DATA,LISTING,CPUEVAL,U)
? Erase  testlib(DATA,LISTING,CPUEVAL,U) ? (YES,NO,EXIT)
y
> ERASE testlib(DATA,LISTING,CPUPRECOM,U)
? Erase  testlib(DATA,LISTING,CPUPRECOM,U) ? (YES,NO,EXIT)
y
> ERASE     :       2 Objects.
```

com
```
> COMPRESS testlib
> COMPRESS      :       Okay.
```

Finally we copy our tested *cpu* to an already existing project mobase in a protected
mode, make sure that it arrived and exit OMA:

copy cpu to project.lib(,,,P)
```
> COPY testlib(HISLANG,COMPONENT,CPU,U) TO project.lib(HISLANG,COMPONENT,CPU,P)
> COPY testlib(PRECOM,COMPONENT,CPU,U) TO project.lib(PRECOM,COMPONENT,CPU,P)
> COPY      :       2 Objects.
```

dir project.lib (,,cpu)

```
> Mobase 'project.lib' totally contains      33 objects.

No| Module       | Type           | Object          |P|   Size | Ver| Last Update          | Owner
1 (HISLANG,      COMPONENT,  CPU                    ,P)    12       11992-03-17 13:50      HIT
2 (PRECOM,       COMPONENT,  CPU                    ,P)    12       11992-03-17 13:50      HIT

> DIR:       2 Objects.
```

end
```
> T O T A L  :       Okay.                Cpu Time used :      5.460 Seconds.
```

Appendix B. Experiences Made With OMA

At the Universität Dortmund OMA was used for the management of the series of experiments described in /BeMW88/, i.e. for storing all objects resulting from that modelling and analysis process. Thereby a "standard" control file format was developed.

One of the purposes of the control file is to define all input files or mobase objects for the HI-SLANG compiler. Thus the control file serves to define configurations of models without modifying the source text of the model. Depending on the version and representation (e.g. HISLANG, PREANA, PRECOM) for instance of a component type which is to be added to the model by %COPY "link name", i.e. depending on the "environment" defined in the control file, a different efficiency and/or performance will be reached when the analyzer is run.

It is advisable to use the control file as documentation for every experiment performed. Apart from providing configuration information, such documents informally describe the changes with respect to the previous experiment and the observations and conclusions from performing the experiment.

Example:

```
%COMMON      ------------------------   For both COMPILER and ANALYZER ---------
%MOBASE office.lib
%BIND    "evaluation"      TO (PREANA,    COMPONENT,    evaluation2)

%COMPILER    ------------------------   Configuration Segment ----------------------------
-
%BIND    "office"          TO (HISLANG,   MODEL,        office1)
%BIND    "handling"        TO (HISLANG,   COMPONENT,    handling4)
%BIND    "experiment"      TO (HISLANG,   EXPERIMENT,   expsep2)

%ANALYZER    ------------------------   Result Directing-----------------------------------
% Usage of DEFAULT-bindings to the mobase containing this control file.

%END     -------------------------------   Model Structure -----------------------------------
%TITLE   Office Model
%COPY    "office"
%COPY    "handling"
%COPY    "evaluation"
%COPY    "experiment"
% The source consists of 4 modules with versions and representations as described above

%EOF     -------------------------------   Experiment Description ---------------------------

EXPERIMENT   :    exp5        AUTHOR: N. Weissenberg          Date: 25.2.88

BASED ON     :    exp3, exp4

CHANGES      :    The DISPATCH strategy of server link was changed to
                  EQUAL (100) (exp3) and evaluation was aggregated (exp4).
-------------------------------------------------------------------------------------------
OBSERVATION:      The results are the same as before, except for server link.

CONCLUSION   :    Both steps could be performed.
```

All object names in this example end with a digit. This is a simple way for version management.

Appendix C. The HIT Standard Mobase

The standard mobase of the HIT system is the default actual mobase. It contains all predefined elements for modelling, as described in the HI-SLANG Reference Manual.

The first object is the OMA help text. Eight predefined component types and a service type follow. Moreover all standard component control procedures of HIT are contained, written in SIMULA for efficiency reasons.

When entering DIR after an OMA call the following display results (ignoring intermediate queries):

> Mobase 'HIT/SYSTEM/MOBASE' totally contains 32 objects.

No	Module	Type	Object	P	Size	Ver	Last Update	Owner
1	(DATA,	OTHERS,	HELP	,P)	267	219	NOV 1989 17:00	HIT
2	(HISLANG,	COMPONENT,	COUNTER	,P)	32	119	APR 1989 15:59	HIT
3	(HISLANG,	COMPONENT,	FTSERVER	,P)	15	119	APR 1989 15:59	HIT
4	(HISLANG,	COMPONENT,	NOWAITSEND	,P)	55	214	MAR 1991 17:29	HIT
5	(HISLANG,	COMPONENT,	OBSERVER	,P)	56	114	MAR 1991 16:52	HIT
6	(HISLANG,	COMPONENT,	PRIOSERVER	,P)	14	119	APR 1989 15:59	HIT
7	(HISLANG,	COMPONENT,	SEMAPHOR	,P)	19	119	APR 1989 15:59	HIT
8	(HISLANG,	COMPONENT,	SERVER	,P)	12	119	APR 1989 15:59	HIT
9	(HISLANG,	COMPONENT,	SYNCHSEND	,P)	52	214	MAR 1991 17:30	HIT
10	(HISLANG,	COMPONENT,	TOKENPOOL	,P)	35	229	AUG 1991 11:38	HIT
11	(HISLANG,	SERVICE,	WATCHER	,P)	9	119	APR 1989 15:59	HIT
12	(SIMULA,	ACCEPT,	ALWAYS	,P)	29	119	APR 1989 15:59	HIT
13	(SIMULA,	ACCEPT,	LIMITED	,P)	39	119	APR 1989 15:59	HIT
14	(SIMULA,	ACCEPT,	RESTRICT	,P)	13	119	APR 1989 15:59	HIT
15	(SIMULA,	DISPATCH,	AGGRDISP	,P)	117	119	APR 1989 15:59	HIT
16	(SIMULA,	DISPATCH,	EQUAL	,P)	29	119	APR 1989 15:59	HIT
17	(SIMULA,	DISPATCH,	SDEQUAL	,P)	102	119	APR 1989 15:59	HIT
18	(SIMULA,	DISPATCH,	SDSHARED	,P)	102	119	APR 1989 15:59	HIT
19	(SIMULA,	DISPATCH,	SHARED	,P)	30	119	APR 1989 15:59	HIT
20	(SIMULA,	OFFER,	ALL	,P)	13	119	APR 1989 15:59	HIT
21	(SIMULA,	SCHEDULE,	CPRIO	,P)	103	119	APR 1989 15:59	HIT
22	(SIMULA,	SCHEDULE,	CRANDOM	,P)	97	119	APR 1989 15:59	HIT
23	(SIMULA,	SCHEDULE,	FCFS	,P)	31	119	APR 1989 15:59	HIT
24	(SIMULA,	SCHEDULE,	IMMEDIATE	,P)	14	119	APR 1989 15:59	HIT
25	(SIMULA,	SCHEDULE,	LCFS	,P)	31	119	APR 1989 16:00	HIT
26	(SIMULA,	SCHEDULE,	LCFSPR	,P)	20	119	APR 1989 16:00	HIT
27	(SIMULA,	SCHEDULE,	PRIONP	,P)	74	119	APR 1989 16:00	HIT
28	(SIMULA,	SCHEDULE,	PRIOPREP	,P)	107	119	APR 1989 16:00	HIT
29	(SIMULA,	SCHEDULE,	PRIOPRES	,P)	83	119	APR 1989 16:00	HIT
30	(SIMULA,	SCHEDULE,	RANDOM	,P)	33	119	APR 1989 16:00	HIT
31	(SIMULA,	SCHEDULE,	SEMSCHED	,P)	35	119	APR 1989 16:00	HIT
32	(SIMULA,	SCHEDULE,	TOKSCHED	,P)	50	229	AUG 1991 11:38	HIT

> DIR : 32 Objects.

Appendix D. Special Hints

The OMA system consists of the following files:

1) Load module *oma*, or the three SIMULA sources for *mobase*, *fan* and *oma*
2) Operating system procedure *oma*
3) HIT message library
4) HIT standard mobase
5) HIT startup file

The STARTUP file of the HIT system has a special %OMA segment. Moreover the %COMMON segment is valid. For BS2000 it contains the following data:

```
%COMMON       Implementation dependent filenames
%MOBASE        READONLY      $I4IC10.HIT.STANDARD.MOBASE
%BIND          "MESSAGE" TO  $I4IC10.HIT.TEXTE

% Construction of Default Filenames: <C>ControlfileName, <L>Linkname,...
%DEFAULT       "#<S>.<L>"
...
%OMA
%BIND          "LISTING"   TO #OMA.LISTING
%BIND          "EDIT"      TO #OMA.EDIT
%DEFAULT       "#<O>.<T>.<M>"
%DEFAULT       "DO $I4IC10.HIT,CONTROL='#',OMA=Y,#"
%DEFAULT       "PRINT #,LOCK=YES,#"
%END
```

The %DEFAULT records denote name patterns. The first one (within %COMMON) defines default file names consisting of the control file name (without suffix) <S> and the link name <L>, while the second is OMA specific and defines the default file name for selected objects, consisting of object name <O>, type <T> and module <M>.

The last two %DEFAULTS denote the pattern of how to call HIT and how to print files in the respective operating system. The '#'-characters will be substituted by OMA. In both cases the first one is a file name (control file object name or file name constructed according to the second %DEFAULT) and the second one may give additional parameters.

The other records define the names of the standard mobase (default mobase for OMA), the HIT message library, the OMA listing and a standard temporary output file of OMA.

The user interface commands of OMA are implemented as portable as possible in the following way (if a command can't be ported its activation has no effect):

CMD: OMA uses the FAN system which already implements the CMD command if possible.

EDxx: The procedure EDIT_OBJ of the OMA source contains external procedure declarations and calls, if the editor can't be activated by the CMD command.

HIT: The HIT call is contained in the third %DEFAULT record (see above) and is executed via the CMD command. After the termination of the HIT run OMA should be activated again.

PRINT: The PRINT command is contained in the fourth %DEFAULT record (see above) and is executed via the CMD command.

Currently there is no explicite version control in OMA (besides counting object updates). The access to older versions is no more possible after updating an object. The user can however control versions himself by adding a version number to the object name (see appendix B.).

Due to historical reasons and implementation restrictions mobases require a hugh amount of space, since they are implemented as SIMULA directfiles, which necessary have a fixed record length of 133(contents)+4(index) bytes. Shorter object lines (even empty lines) are filled up with blanks. The storing of more than one object line within one record would remove this drawback, but would increase the access times; thus this is not implemented.

Therefore it is advisable to archive mobases in sequential representation (see command CONVERTMF) and/or in a compressed format (UNIX compress or BS2000 FMS libraries or the like). A compression of factor 5 can in most cases be achieved.

Appendix E. The OMA Help Text

Instead of an index we conclude with the text displayed by entering the HELP command of OMA. It contains a summary of all OMA commands listed in alphabetical order, and starts by explaining the format of OMA command arguments.

All work with OMA is done interactively by entering OMA commands as explained here. All OMA commands may be abbreviated and have 0..2 arguments. In commands with two arguments these arguments are separated by TO, if both denote objects or files, or by WITH, if the second argument is a command modifier.

Arguments of OMA commands:

The following kind of arguments occur in the commands:

obj := [object]
| [mobase] attributes
| "link"{the object bound to *link*}
| + {the current object set, see command '+'}

file_obj ::= same as *obj*, but a single token is interpreted as a *file* name (which may be SYSIN or SYSOUT), not as *object* name. Objects can be denoted by („*object*).

file ::= name of a (sequential) file outside OMA

mobase ::= name of an index sequential file containing objects, a default can be set with command MOBASE

link ::= link name of an object or a file, as defined in a HIT control file or with command BIND

text ::= an arbitrary quoted text, e.g. "OMA"

Object Attributes:

There are four attributes to access objects, each may be missing:

attributes::= (module, type, object, protection)

module ::= CONTROL | HISLANG | PRECOM | PREANA | SIMULA | DATA
(kind of representation, may be abbreviated)

type ::= MODEL | COMPONENT | SERVICE | PROCEDURE
| CONTROL | EXPERIMENT
| ACCEPT | SCHEDULE | OFFER | DISPATCH
| FILE | DUMPFILE | TABLE | GRAPH| HISTOGRAM
| TRACE | LISTING | MATRIX | STATES| OTHERS
(kind of object, may be abbreviated)

object ::= name of an object within a mobase, 12 significant characters, asterisks within are treated as wild cards.

protection::= P | U

Protected objects can only be manipulated (overwritten, changed, erased) after a special confirmation. The same holds for objects added by another owner (see DIR). Generally you have to confirm manipulations on existing objects or files.

Module and type have to be consistent according to the following table:

Module	Type
CONTROL	CONTROL, OTHERS
HISLANG	MODEL,COMPONENT,SERVICE,PROCEDURE,EXPERIMENT, OTHERS
PRECOM	MODEL,COMPONENT,SERVICE,PROCEDURE,EXPERIMENT, OTHERS
PREANA	COMPONENT,OTHERS
SIMULA	ACCEPT,SCHEDULE,DISPATCH,OFFER,OTHERS
DATA	FILE,DUMPFILE,TABLE,GRAPH,HISTOGRAM, TRACE,LISTING,MATRIX,STATES,OTHERS

OMA commands:

If the first argument of a command is of type *obj*, it can be specified incompletely: All or some attribute values may be missing and object may contain wild card symbols '*'. The command will then be executed for all objects matching the given attribute values within the denoted mobase.

The second argument of such commands may also be specified incompletely. Then the missing attribute values are filled each time from that object currently matching the first argument. Example: CHANGE (HI) TO (,,,P) protects all HISLANG objects.

The following list of OMA commands is alfabetically sorted:

ADD file [TO obj]
Insert a copy of the given *file* into the mobase and with the attribute values given by *obj*. Some attribute values may be left out. OMA can determine them: object is set to *file*, the modules PREANA, PRECOM, CONTROL and HISLANG as well as the types MODEL, COMPONENT, SERVICE and EXPERIMENT may be set from the first line of the file contents. If the *file*name contains wild cards, all files corresponding to the pattern are added.

BIND [link TO file_obj]
With no arguments all active bindings are displayed.Otherwise *link* is bound to the given file or object and can be used for that in commands with obj-arguments.

CHANGE obj1 TO obj2
The attribute values of object *obj1* are changed to those specified by *obj2*. Free positions in *obj2* are filled from *obj1*, while free positions in *obj1* mean FOR ALL those objects.

CMD text
Execute *text* as operating system command.

COMPRESS [mobase]
Moves gaps within the index sequential file *mobase* to the end and reserves new space for the directory.

CONTROL file_obj
The denoted file or objects (having module CONTROL) are interpreted as HIT control files. All mentioned bindings can then be used, e.g. SHOW "TABLE". All actual links can be displayed by BIND.

CONVERTMF [mobase TO] file
MF means Mobase to File. The index sequential file *mobase* is converted to a sequential and therefor portable and smaller file named *file*. If omitted, *mobase* is the actual mobase.

CONVERTFM file TO mobase
FM means File to Mobase. The sequential file *file* is assumed to be in a format to convert it to a mobase named *mobase*, i.e *file* has been generated by CONVERTMF before.

CONVERTMM [mobase1 TO] mobase2
MM means Mobase to Mobase. *mobase1* (or the actual mobase) is copied to a file named *mobase2*. Unused blocks are not copied, so *mobase2* becomes smaller in most cases.

COPYMOBASE [mobase1 TO] mobase2
Same as CONVERTMM.

COPY obj1 TO obj2
Copy all objects denoted by *obj1* to destinations denoted by *obj2*. Free positions in *obj2* are filled from *obj1*, while free positions in *obj1* mean FOR ALL those objects.

CP file_obj1 TO file_obj2
Copy all files or objects denoted by *file_obj1* to destinations denoted by *file_obj2*. The commands ADD, SELECT and COPY use this command internally. See there.

DIR obj
Directory contents (module, type, object, protection, size, version, last update, owner) displayed for those objects denoted by *obj*. Directory output is affected by SET LINES/LENGTH.

EDxxx obj
All objects denoted by *obj* can be edited by editor denoted by *xxx*. The list of available editors is given by SET.

END
Quit OMA.

ERASE obj [!]
Erase all objects denoted by *obj*. If '!' is not given, every single object deletion has to be confirmed. Protected objects can only be erased after a confirmation.

HELP [text]
This information is given with 'SHOW help [WITH text]', i.e. the first line shown is the first line containing *text*. HELP must be terminated with END.

HIT obj [WITH text]
Call HIT for (the first) object denoted by *obj* and with additional arguments *text.* The object must have module CONTROL or HISLANG. It is not selected, but the HIT-system reads it directly from the mobase. The call given is displayed by SET.

MOBASE [READONLY] mobase
mobase becomes the new current mobase, to which *obj* arguments of other commands refer, if they don't explicitly refer to another mobase. READONLY mobases can't be changed or erased, but can therefore be read simultaneously by different users.

NEWMOBASE mobase
A new empty mobase with name *mobase* is generated.

NEWOBJECT obj
The next lines read from terminal (until a line only containing a '.' is entered) are added as a new object with attribute values denoted by *obj*.

PRINT obj [WITH text]
All objects denoted by *obj* are printed with arguments given by *text.* The print command can be displayed by SET. If *text* is not given the arguments are selected dependent on module and type of the objects. This can be affected by SET P3/P4/PP.

READ file
Interpret *file* contents as OMA commands, if file exists. Helps to adapt OMA to different users and systems.

SELECT obj [TO [EXTEND] file]
Select the object denoted by *obj* to the given file. If *obj* specifies more than one mobase object, all those objects are selected. If *file* is omitted it is set to object name. If *file* is '#' the filenames are generated due to a pattern.

SET [variable [value]]
With no arguments the setting of all variables, the list of build in editors and some other information is displayed. The defaults are installation defined, but often VERBOSE, NOCONFIRM, ALL, NOWARN, NONUMBERS, LINES 18, LENGTH 80. With arguments the *variable* is set to the given *value,* with the following meaning:

SET [NO]**VERBOSE**	: All commmands on objects are echoed before execute
SET [NO]**CONFIRM**	: After VERBOSE the execution must be confirmed
SET [NO]**ALL**	: Command only executed for first object matching
SET [NO]**WARN**	: Warnings are suppressed
SET [NO]**NUMBERS**	: SHOW output is numbered
SET **LINES int**	: Number of lines of a screen (for SHOW and DIR)
SET **LENGTH int**	: Number of columns of a screen (for SHOW and DIR)
SET **HIT text**	: Default argument for HIT call
SET **ED text**	: Default argument for EDitor call
SET **Px text**	: Print argument for DIN A3 objects (*x*=3) or DIN A4 objects (*x*=4) or PRINTFILE objects (*x*=P)

SHOW obj [WITH text]

Displays the contents of all objects denoted by *obj*. The first line on screen is the actual line (AL). If *text* is given the AL is set to the first line containing that text. Within SHOW the following commands can be given (the last line displayed summarizes this command list):

END	: quit SHOW, the next object specified by *obj* is shown
EXIT	: quit SHOW totally.
+ [n]	: set AL forward *n* lines or one screen, if *n* omitted
- [n]	: same backwards
++	: set *AL* to one screen before end of object
--	: set *AL* to 1.
SET n	: set *AL* to *n*.
FIND t	: set *AL* to next line containing text *t*
LINES n	: number of displayed line becomes *n*
LENGTH n	: number of displayed columnes becomes *n*
NUMBERS	: switching on/off line numbering
EDIT	: *AL* and following lines (until only '.' entered) may be overwritten
CHANGE t TO u	: change all occurences of *t* to *u*, each change has to be confirmed.

After every command the screen is refreshed. SHOW output is affected by SET LINES/LENGTH/NUMBERS, defining the defaults for the corresponding local commands.

+ obj [WITH text]

Add all objects denoted by *obj* and containing *text* (lowcase or upcase) to the current object set, not to the mobase. This set can be used within every command having a first obj-argument by writing '+' for that argument, e.g. 'DIR +'.

- obj [WITH text]

Substract all objects denoted by *obj* and containing *text* (lowcase or upcase) from the current object set. This set can be used within every command having a first obj-argument by writing '+' for that argument, e.g. 'DIR +'. '-' alone makes current object set empty.

*** obj [WITH text]**

Leaves all objects denoted by *obj* and containing *text* (lowcase or upcase) within the current object set. This set can be used within every command having a first obj-argument by writing '+' for that argument, e.g. 'DIR +'.

/ rest_of_line

Execute *rest_of_line* as operating system command.